CoinLayering: An Efficient Coin Mixing Scheme for Large Scale Bitcoin Transactions

Ning Lu*,†, Yuan Chang*, Wenbo Shi*, and Kim-Kwang Raymond Choo‡, Senior Member, IEEE
*College of Computer Science and Engineering, Northeastern University, Shenyang, China
†School of Computer Science and Technology, Xidian University, Xi’an, China
‡Department of Information Systems and Cyber Security, Department of Electrical and Computer Engineering, and Department of Computer Science, The University of Texas at San Antonio, San Antonio, TX 78249, USA

Abstract—Coin mixing can be used to preserve identity privacy of Bitcoin owners, by engaging a set of middlepersons (i.e., Mix) to temporarily hold the transacting Bitcoins and remove the linkage between the transacting parties. However, existing schemes are generally not scalable due to limitations associated with the anonymity set, and self-credibility. In this paper, we propose an efficient coin mixing scheme (hereafter referred to as CoinLayering). To achieve strong anonymity, CoinLayering randomly selects two sets of middlepersons to respectively execute Bitcoin holding and Bitcoin trading. The seller can also select lower-loaded sets of middlepersons in the shortest time possible. We also design two coin mixing protocols, CoinLayering-PA and CoinLayering-PB, to mitigate the risk due to misbehaving middlepersons and Supervisor. We then mathematically prove that CoinLayering achieves both strong anonymity and self-credibility, and evaluate its performance to demonstrate its scalability.

Index Terms—Blockchain, Bitcoin, Identity privacy, Coin mixing, Large scale Bitcoin transactions

1 INTRODUCTION

The interest in cryptocurrency, and particular Bitcoin, is partly evidenced by the increasing number of such currencies and the trading volume [1]. For example, as of Dec 5, 2020, there are reportedly 7,863 cryptocurrencies, in 33,925 markets, with a market capitalization of USD 571,589,849,765 (and Bitcoin dominates approximately 62.46% of the market)1. In other words, the volume of Bitcoin transactions is significant. Similar to other consumer technologies, there are underlying security and privacy challenges in Bitcoin and other cryptocurrencies [2], [3]. For example, since all Bitcoin transactions can be publicly audited in the blockchain, one can perform an analysis of the distributed ledger, using the heuristic cluster to analyze transaction data, and infer the true identities of transaction parties. The exposure of the user’s identity can lead to other attacks, such as stealing of the user’s Bitcoins [4], [5]. One high profile incident occurs in July 2017, where the leakage of nearly 31800 users’ information on Bithumb (e.g., email address and mobile phone number) facilitated the exfiltration of billions of South Korean won, the official currency of South Korea [6]. This necessitates the protection of identity privacy of transacting parties in the Bitcoin marketplace.

Manipulating the ownership of Bitcoins to obfuscate the interlinkage of transacting parties (also known as coin mixing) is one approach used to protect user identity privacy. Specifically, in such an approach, coin mixing usually allows Bitcoin sellers to engage a set of middlepersons to temporarily hold on to their coins and further blind the transaction. As shown in Fig. 1, all sellers send their Bitcoins and buyers’ identity information to the middlepersons and entrust them to complete the transactions. Consequently, the sellers and buyers are not linkable to each other.

![Fig. 1. Coin mixing: A brief overview](image)

There are, however, several challenges in the implementation of coin mixing, particularly if we also take into consideration the constantly evolving threat landscape and the scale of Bitcoin trading.

1) Strong anonymity. In an attempt to violate the identity privacy of transacting parties, an adversary can guess the buyer-seller relationship to bypass the coin mixing system. This is an attack that affects most of the existing schemes. Normally, the increase in the mixing scale can improve the difficulty of guessing the relationship and thus effectively defend against such an attack. Here mixing scale is also regarded as the anonymous set, which is mainly associated with the maximum number of simultaneous acceptable anonymous transactions during the interval that the system completes a coin mixing. Apparently, the larger the anonymous set, the stronger the anonymity could achieve. For example,
merging multiple transactions into one transaction can obscure the seller-buyer relationships to some extent, but its effectiveness is subjected to the constraints of Bitcoin’s maximum transaction size (e.g., 100KB). In other words, existing schemes generally can only take as input few transactions and this reduces the difficulty of correctly guessing the mapping between both buyer and seller [7], [8], [9], [10]. Alternatively, we can choose to direct all transactions to an explicit middleperson, say Mix, in order to efficiently separate the buyer from the seller. However, the compromise of Mix would make it easier to guess buyer-seller relationships in not one, but many transactions [11], [12], [13]. Therefore, how to achieve enhanced anonymity against such an adversary is a challenge, and this is the one we seek to address.

2) **High scalability.** A practical coin mixing scheme needs to be able to scale up (significantly) when needed, and it does not appear to be the case in existing schemes. For example, randomly selecting middlepersons to perform mixing tasks can reduce the risk of colluding peers, but it comes at the cost of execution efficiency and consequently scalability [14]. In the case of a significantly large number of transactions, the schemes in [7], [8] can only take a transaction and serially execute the mixing tasks. Such a design has time and performance implications. Confined to the limited processing capacity in terms of bandwidth and computation resources, the middleperson in the schemes of [12], [13] becomes a performance bottleneck, which can also lead to denial of service (DoS). Therefore, CoinLayering is designed to achieve high scalability.

3) **Self credibility.** Coin mixing allows the middleperson to take control of the users’ Bitcoins, which in itself is a risk. For example, to improve efficiency, existing schemes such as those in [15], [16] introduced a third-party to act as the middleperson. However, this requires blind trust in this middleperson to be doing the right thing (e.g. not to steal the users’ Bitcoins, not to collude with an adversary and/or leak information about the transaction) [12], [13]. Therefore, CoinLayering includes a mechanism to penalize misbehaving middlepersons, and consequently, achieve self-credibility.

Specifically, in our proposed CoinLayering (see also Table 1), we introduce a User – Mix – Supervisor based system model, in which the Supervisor is authorized by the government (e.g. a central bank, banking regulator, or financial intelligence unit), and responsible for the middlepersons’ (Mixes) task assignments. We assume Mix to be some organization (e.g. a financial institution), which profits by hosting the sellers’ Bitcoins and trading them with the buyers. To achieve strong anonymity, and motivated by the observation that the leakage of seller-buyer relationship can potentially occur during the holding and trading actions, these actions are delegated to two different Mixes, and the User can randomly select both Mixes and utilize different identities interact with them, which secures the transaction’s privacy and further facilitates the growth of anonymous set. Also, to achieve high scalability, we design an efficient Mixes selection algorithm, which can determine the Mixes that meet the User’s requirements (e.g., privacy and efficiency) in the shortest time possible. We also consider that in a real-world deployment, either the Mix or the Supervisor may attempt to steal the User’s Bitcoins. Thus, to be able to penalize a misbehaving Mix, we design a coin mixing protocol under a semi-trusted Mix (hereafter referred to as CoinLayering-PA), which uses group signature to disclose the identities of misbehaving Mixes to facilitate subsequent penalties. To penalize a misbehaving Supervisor, we design a coin mixing protocol under a semi-trusted MixSupervisor (hereafter referred to as CoinLayering-PB), which employs the security threshold signature to replace the supervisor and make up the cost difference.

In the next section, we will introduce relevant background materials and the related literature. In Sections 3 and 4, we will give an overview of CoinLayering and the secure coin mixing protocol, respectively. Then, we will present our security and performance evaluations in Sections 5 and 6. The last section concludes this paper.

### 2 Relevant Background and Literature

#### 2.1 Background

In a typical coin mixing scheme, there exists a middleperson set \( M \), a seller \( s \) and a buyer \( b \). The coin mixing procedure \( F(s, b) \) can be formalized as follows:

\[
F(s, b) = f_1(s, M) \cdot f_2(M, b),
\]

where \( f_1(s, M) \) is used to remove the link between the seller and transaction Bitcoins. This compounds the challenge of a middleperson in inferring the origin of these Bitcoins, and \( f_2(M, b) \) is used to ensure accurate delivery to the right buyers.

A practical coin mixing scheme should satisfy the following requirements, even when dealing with large-scale Bitcoin transactions:

- **Strong anonymity.** To improve the difficulty of guessing, the anonymity set should be as large as practical.
- **DoS resilience.** Under normal circumstances, the middlepersons would be available to provide mixing

<table>
<thead>
<tr>
<th>Functions</th>
<th>[7], [8]</th>
<th>[9]</th>
<th>[10]</th>
<th>[11], [12], [13]</th>
<th>[14]</th>
<th>[15], [16]</th>
<th>CoinLayering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong anonymity</td>
<td>( x )</td>
<td>( x )</td>
<td>( x )</td>
<td>( x )</td>
<td></td>
<td>( x )</td>
<td>( x )</td>
</tr>
<tr>
<td>Scalability</td>
<td>Low execution time</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
<td>( \times )</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
</tr>
<tr>
<td></td>
<td>Less bandwidth overheads</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
</tr>
<tr>
<td>Self credibility</td>
<td>Anti-denial service</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
</tr>
<tr>
<td></td>
<td>Preventing collusion</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
</tr>
<tr>
<td></td>
<td>Preventing theft</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
</tr>
</tbody>
</table>
services to users, failing which the middlepersons must be held accountable.

- Low execution time. The execution time should be minimized.
- Minimal bandwidth overhead. To avoid service degradation due to network congestion, the bandwidth overhead should be as minimal.
- Preventing collusion. In the event that middleperson collude, either among themselves or an external adversary, to disclose the seller-buyer relationship, there must be a mechanism to identify and penalize these misbehaving middlepersons.
- Preventing theft. The middlepersons cannot steal the users’ Bitcoins.

2.2 Related Work

There have been attempts to design anonymous cryptocurrencies, such as Zerocash [17] and Monero [18]. Although such anonymous cryptocurrencies are promising, they are not as widely adopted as Bitcoin. Hence, in this paper we will only focus on coin mixing that can be deployed in Bitcoin (or other similar cryptocurrency). According to the system structure, existing Bitcoin mixing schemes are either completely centralized or completely decentralized.

Completely decentralized based schemes. Maxwell et.al [9] proposed a coin mixing scheme (Coinjoin), in which a large number of peer nodes in the blockchain are engaged as middlepersons. To remove the link between the seller and the buyer, a middleperson is required to combine multiple transactions into one transaction. However, the middleperson may be able to infer relevant transaction information and collude with each other during the node negotiation process. Hence, Ruffing et.al [7] proposed CoinShuffle, which shuffles the output address. Such an approach prevents the middleperson from learning information about the buyer associated with the transaction. To reduce the number of communication rounds, they proposed CoinShuffle++ [8]. To ensure the resilience of the system in the event of attacks or node failure, Ziegeldorf et.al [19] proposed CoinParty. The latter uses both secure multiparty computing protocol and threshold signature technology to improve robustness. However, it requires the middleperson to be online all the time, and it is vulnerable to DoS attacks. Moreover, subject to the constraints of Bitcoin’s maximum transaction size, it only allows one to input few transactions. In other words, the anonymous set is small. To overcome these limitations, Maxwell et.al [14] proposed Xim, which allows the seller to randomly and anonymously select middleperson so as to conceal the real task execution position. Such an approach increases the difficulty of guessing the mapping between buyer and seller, and is resistant to DoS attacks. However, it needs take several hours to complete a coin mixing task, and clearly is not scalable.

Completely centralized based schemes. Bonneau et.al [15] proposed the centralized MixCoin scheme, in which all transactions are handled by a middleperson (Mix) in order to separate the buyer from the transaction Bitcoins. While it can prevent the Mix from stealing the User’s Bitcoins, it does not prevent the Mix from leaking transaction information. Thus, Valenta et.al [16] used blind signature to remove the relationship between the buyer and the transaction. However, in their approach, the Mix can steal the User’s Bitcoins. Inspired by eCash, Heilman et.al [11], [12] designed an anonymous cryptocurrency (TumbleBit), which is compatible with Bitcoin. TumbleBit uses both blind signatures and smart contracts to ensure security during transactions between Users and Mixes. The Mix in TumbleBit uses multi-party secure computing’s cut-and-choose method to remove the link between the seller and the Mix. Ferretti et.al [20] improved TumbleBit, in order to be used for anonymous payments on private chains. In a separate work, Liu et.al [13] respectively adopted group transaction to reduce the possibility of Bitcoins stolen by Mixes and ring signature to accurately deliver the transaction to the buyer. However, the exposure of Mix will ease the correct guessing of the buyer-seller relationships. In addition, for large scale transactions, they are also vulnerable to DoS attacks due to the performance bottleneck of Mix.

Unlike the above discussed approaches, our proposed CoinLayering adopts the User → Mix → Supervisor based system model. In the model, the Supervisor (a central node) is only responsible for lightweight task assignment and regulation, and thus removes the risk of being a performance bottleneck. Also, Mixes are randomly selected to implement coin mixing so as to improve anonymity.

<table>
<thead>
<tr>
<th>TABLE 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A summary of notations</strong></td>
</tr>
<tr>
<td><strong>Notation</strong></td>
</tr>
<tr>
<td>Mix</td>
</tr>
<tr>
<td>Supervisor</td>
</tr>
<tr>
<td>BB</td>
</tr>
<tr>
<td>(x_i,y_i)</td>
</tr>
<tr>
<td>ID (i)</td>
</tr>
<tr>
<td>(E_i)</td>
</tr>
<tr>
<td>(K_i)</td>
</tr>
<tr>
<td>(E)</td>
</tr>
<tr>
<td>(p_i)</td>
</tr>
<tr>
<td>(A^k_i)</td>
</tr>
<tr>
<td>(w)</td>
</tr>
<tr>
<td>DA</td>
</tr>
<tr>
<td>CL</td>
</tr>
<tr>
<td>RS</td>
</tr>
<tr>
<td>(k_1,k_2)</td>
</tr>
<tr>
<td>(h)</td>
</tr>
<tr>
<td>(T_1,T_2,T_3,T_4)</td>
</tr>
<tr>
<td>(I,O)</td>
</tr>
<tr>
<td>(V_i)</td>
</tr>
<tr>
<td>(W)</td>
</tr>
<tr>
<td>(nonce_i)</td>
</tr>
<tr>
<td>(f_i)</td>
</tr>
<tr>
<td>(b)</td>
</tr>
<tr>
<td>(U,U^*)</td>
</tr>
<tr>
<td>(tx_i)</td>
</tr>
<tr>
<td>(b_i)</td>
</tr>
<tr>
<td>(T)</td>
</tr>
<tr>
<td>(\rho)</td>
</tr>
<tr>
<td>(LQ)</td>
</tr>
<tr>
<td>(L_i)</td>
</tr>
<tr>
<td>(BC)</td>
</tr>
</tbody>
</table>

3 OUR PROPOSED COINLAYERING

In this section, we present the system model and the respective system components and features, the Mixes selection approach to guarantee execution efficiency, and two potential threats faced by CoinLayering. Table 2 summarizes the notations used in this paper.
3.1 System model

As is previously discussed, to increase the difficulty of guessing the seller-buyer relationships and further achieve the strong anonymity in large scale Bitcoin transactions, CoinLayering allows for the random selection of Mixes and separation of holding and trading action assignment to the different Mixes. Also, to achieve scalability, CoinLayering allows User to select multiple available Mixes in the shortest time possible. Specifically, we introduce a User - Mix - Supervisor based coin mixing scheme. It requires all Mixes to compete for the coin mixing task. Supervisor is tasked with Mixes assignments and regulation, which is responsible for recommending the most appropriate \( k \) candidate Mixes that are able to satisfy the User's requirements. Then, User selects two lightly loaded Mixes as the ultimate performers in a random fashion. Moreover, to minimize communication overhead, we introduce a Bulletin Board to broadcast relevant information. The system model in CoinLayering is represented in Fig. 2.

```
① Mixing request
② Audit
③ Service request
④ Register
⑤ Voucher
⑥ Transaction 2
⑦ BB

Input address I
Output address O

User
Mix
Supervisor

Escrow address E1
Private address K1
chosen Mix1

Escrow address E2
Private address K2
chosen Mix2

User1
User2

Supervisor's record.

Under ideal conditions, CoinLayering works as follows:

Step 1: Mix sends a registration request to the Supervisor. Upon successful registration, Mix can provide mixing services for users.

Step 2: User makes a mixing request to Supervisor, which recommends \( k \) candidate Mixes to the User. On being accepted, User selects two Mixes from \( k \) candidates. Let \( Mix_1 \) and \( Mix_2 \) respectively denote these two chosen Mixes.

Step 3: User makes service requests to the two Mixes. Mixes receive the requests and then send the commitment \( V \) as the reply. User transfers Bitcoins from address I to escrow address \( E_i \) of \( Mix_i \), and then builds a transaction \( tx_1: I \rightarrow E_i \) (recorded in BB).

Step 4: \( Mix_1 \) confirms the transaction from \( BB \) and sends a voucher \( W \) to User.

Step 5: User receives voucher \( W \) and sends it to \( Mix_2 \).

Step 6: \( Mix_2 \) receives voucher \( W \) and builds a transaction \( tx_2: K_2 \rightarrow O \), where \( K_2 \) is the private address of \( Mix_2 \).

Step 7: After the mixing is completed, the Supervisor audits the Mixes by reviewing the BB and recycles the amount in all the escrow addresses \( E_i \) to its total escrow address \( E \). It also transfers the same Bitcoins to the private address of \( Mix_2 \) according to \( tx_2 \)'s record. Supervisor audits once within a certain time.

One may argue that, once the two Mixes collude with each other, the User's identity privacy may still be leaked. However, the possibility of such an event occurrence is relatively low. The reasons can be stated as follows. Firstly, without being aware of each other, the candidate Mixes have been designated to User by the Supervisor. In this case, it is difficult for them to collude in advance. Secondly, the User can further optionally select two Mixes from the candidates according to its security requirements, which further increases the collusion between these Mixes difficulty.

3.2 Mix Selection

In CoinLayering, we adopt the multiple supplier selection strategy to improve the difficulty of guessing the seller-buyer relationships, i.e., on one hand Supervisor needs select \( k \) appropriate candidate Mixes according to User's performance and security requirements (e.g., the execution efficiency and credibility), on the other hand User needs randomly select two lightly loaded Mixes from these candidates. Obviously, in CoinLayering, the results of Mix selection not only affects the quality of coin mixing, but also its execution time. This requires that Mix selection is able to satisfy all Users' requirements in the shortest time possible. But, there are two problems to achieve this goal. Firstly, the diversity of Users' requirements makes it difficult to match. For example, some Users focus on mixing fees, while others on service efficiency (both are contradictory). Secondly, considering Supervisor cannot obtain the Mixes' status information (or underlaying network) in real time,
The article discusses the selection of Mixes based on a Dominance Graph (DG). The process involves defining the attributes of Mixes and selecting the most suitable one for a given User request. When a Mix is selected, it is placed in the system to provide mixing services. The selection process is based on minimizing waiting times and maximizing the efficiency of the system.

**Definition 1 (Mix service)**. Given a Mix, its Mix service is measured by multiple attributes that are of interest to Users (including bandwidth, acceptance rate, service efficiency, credibility, etc.), where \( A_i \) denotes the identity of Mix, \( f_i \) denotes its mixing fee, and \( A_i^k \) denotes its attribute vector. The attribute vector is \( A_i = [A_i^1, A_i^2, ..., A_i^n] \), where \( A_i^k \in [0, 100] \) denotes the score of \( k \)th attribute.

**Definition 2 (User's preferences)**. Each User may have different preferences. For example, some Users may focus on the mixing price, while others on service efficiency. Given a User \( j \), we formally define the top \( k \) query as \( Q_j = \{f_j, w_j^m\} \), where \( f_j \) denotes the highest mixing fee accepted by User \( j \), and \( w_j^m \) denotes the weight of User \( j \)'s preferences and \( \sum_m w_j^m = 1 \).

**Definition 3 (Aggregate function)**. Given a Mix \( i \) and a User \( j \), the matching degree \( MD_i \) between Mix \( i \) and User \( j \) can be computed through the following aggregation function. After receiving the User's mixing request, Supervisor refers to Mixes' service information in BB, and returns the top \( k \) Mixes with the highest aggregation value to User \( j \).

\[ MD_i = \vec{A}_i \cdot \vec{w}_j \]  \hspace{1cm} (2)

**Algorithm 1 Mix selection based on DG**

```
BEGIN
1. Input: DG, User query parameter \( w_j \), aggregate function \( f_j \)
2. CL ← \( f_j(DA, w_j) \); // the dominance graph first-level data \( DA \)
3. RS ← r from the CL; // the ordered candidate list \( CL \)
4. For \( k1 < k2 \) do length
   5. For each child node of \( r \) do
     6. If all parent nodes of \( C \) are \( CL \) Then
     7. CL ← \( k(C, w_j) \);
     8. End If
    9. RS ← r;
 10. End For
 11. End For
13. END
```

The process involves several steps, including inputting the data, calculating the dominance graph, and selecting the top \( k \) Mixes based on the User's query parameter and the aggregate function. The algorithm aims to efficiently select the most suitable Mix for each User, ensuring that the system remains stable and efficient.

```
1545-5971 (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: University of Texas at San Antonio. Downloaded on March 16, 2021 at 12:18:21 UTC from IEEE Xplor. Restrictions apply.
```
above factors, Supervisor eventually searches a feasible \( k \).

After \( k \) has been determined, the waiting decision process can be stated as follows: when User sends the service request to a busy Mix, Mix would inform the User of the current queue length \( L_i \). If \( L_i \leq LQ \), we recommend User to wait; otherwise, it should choose some other Mixes, where \( LQ \) denotes the average queue length. It’s worth noting that, although the operations of Mix_1 (responsible for performing group signatures) and Mix_2 (responsible for verifying group signature) are different, their computation costs are almost the same, which would be proved in the experimental evaluation section. This means that we do not have to distinguish them in design waiting decision strategy.

### 3.3 Potential Threats

For simplicity, the above strawman design assumes all components to be honest and well behaved. Once relax these assumptions, CoinLayering would face the following potential threats. To fix these threats and enhance its self-credibility, we respectively devise the corresponding coin mixing protocols in Section 4.

**Semi-trusted Mixes.** To maximize their self-interest, Mix may record Users’ transaction information in the background, and sell them to the adversary. Moreover, it is likely to steal Users’ Bitcoins without providing any service. Furthermore, the lazy Mix would deliberately delay the service time. Therefore, it is necessary to disclose the identities of misbehaved Mixes and further punish them.

**Semi-trusted Supervisor.** Because “enemy within” exists, Supervisor may be not completely honesty. For example, the misbehaved insider may steal Users’ Bitcoins and make false accounts to cover up its behavior. More complicated, it may some compromise Mixes to obtain their private keys and further steal Bitcoins. Therefore, it is desired to limit the Supervisor’s behaviors and thus prevent from its stealing.

### 4 Secure Coin Mixing Protocol for CoinLayering

In this section, we first describe a coin mixing protocol to prevent Mix’s semi-honest, termed as CoinLayering-PA. And on this basis, to further solve the Supervisor’s semi-honest behavior, we design a more secure coin mixing protocol termed as CoinLayering-PB.

#### 4.1 CoinLayering-PA

The primary principle of CoinLayering-PA design can be stated as follows: to secure the ownership of Bitcoins, Mixes that host Users’ Bitcoins must mark the vouchers with signatures; to prevent from Bitcoins stolen, Mixes are required to provide the deposit to Supervisor; to prevent Mix from colluding with adversaries to leak information, the interaction between Mixes should be blinded; to urge lazy Mixes, Users are allowed to set time constraint. Guiding by this principle, we combine the Schnorr signature and congruence based group signature technologies to proof the Mix’s escrow Bitcoins. The choice is due to the following: (1) it can insure the anonymous between Mixes, i.e., Mix_1 cannot see the identity of Mix_2 when verifying the signature, and further prevent them from colluding with each other. (2) It allows Mixes dynamically joining and exiting, which is more applicable for large scale transactions. (3) It has less computation, and makes the entire protocol more efficient.

As is shown in Fig. 3, there are three phases in our protocol: registration, mixing and audit. Only when Mix puts up rent deposit can eligible for rendering mixing service. When Users request coin mixing service, the time limits \( T_i \) for the service is attached. If Mixes accept the request, it must be completed within \( T_i \). After the escrow operation, Mix_1 signs the blind message as a voucher. Users can require Mix_2 to complete transaction by virtue of this voucher. During the execution of a transaction, we allows User to terminate transaction under these conditions: if Users want to terminate the transaction after received the commitments from Mixes, they just have to wait until after the \( T_i \); if Users want to terminate the transaction after constructed \( tx_1 : I \rightarrow E_1 \), it only needs to change the output address \( O \) to \( I \) and \( ID_2 \) to \( ID_1 \) in the message \( m' \). Then, certificate \( W \) is handed over to Mix_1 for verification, which can facilitate it to construct the transaction \( tx_2 : K_1 \rightarrow I \), and further recover the transaction. In addition, considering User need pay for mixing service, as the number of coin mixing increases, its financial burden would grows. In this case, we design an incentive mechanism. Its basic idea is that Supervisor authenticates Users’ applications and then provides some rebates for users who continuously mixed coins. Limited by the space, we will depict in Appendix A.

![Fig. 3. CoinLayering-PA.](image-url)
address $E$. After $\text{Supervisor}$ receives the deposit and publishes the $User$’s service information (e.g., its reputation and service efficiency) on BB, the $User$ is registered successfully. This means that it is officially upgraded to a $Mix$. Join: $\text{Supervisor}$ sends a modulus $p_i$ to each $Mix$. Each $Mix$ generates its own private key $x_i$ and calculates its own public key $y_i = g^{x_i} \mod p_i$. Then, speak its $y_i$ and $ID$ and meantime send them to $\text{Supervisor}$. If $Mix$ has a malicious move, $\text{Supervisor}$ can be held accountable according to its identity. To prevent $Mix$ from messing up, $\text{Supervisor}$ needs to prove by knowledge sign that it owns the private key $x_i$ and submits the corresponding public key $y_i$ [23]. $Mix$ selects a random number $r_1$, and computes $c_i = H(\text{Time})[y_i][|g|^s_i^2]$, $s_i = r_i - c_i \cdot x_i$, where $\text{Time}$ is a timestamp. After $\text{Supervisor}$ receives $c_i$ and $s_i$, it verifies that $c_i = H(\text{Time})[y_i][|g|^s_i y_i^{s_i}]$. If the equation is true, $User$ can prove $y_i = g^{x_i} \mod p_i$. $\text{Supervisor}$ constructs the Chinese remainder theorem congruence $c = y_i \mod p_i$, and $p_i$ and $p_1$ of each $Mix$ [24], and compute $c = \sum_{i=1}^{k} y_i \cdot P_i \cdot P_i^{-1}$ and post it on the BB. Among them, $P = \prod_{i=1}^{k} P_i$ and $P_i^{-1}$ satisfies the integer solution of $P_i \cdot P_i^{-1} = 1$.

Exit: When $Mix$ exits, $\text{Supervisor}$ conducts a transaction audit. If the audit result is correct, the public key $y_i$ of $Mix$ is changed by $\text{Supervisor}$, the new $c$ is calculated and updated on BB, so that the $Mix$ cannot perform the legal group signature.

4.1.2 Mixing Phase

In the mixing phase, $User$ reached an agreement with $Mix$. If $Mix$ agrees to provide the service, it needs to provide a commitment to $User$. When $User$ initiates the transaction $tx_1$, $Mix$ needs to give it a group-signed voucher $W$. After $Mix_2$ verifies $W$, it would build $tx_2$ to complete the coin mixing. Mixing phase including the following steps.

Step 1: $User$ wants to make a transaction $tx_0$: ($input$ address$)$→ $O$ ($output$ address). For this, it first randomly selects two from the recommended $k$ $Mixes$, and creates two identities $U$ and $U^*$. Then, as $U$, send both $T_1$ (Time limit for transaction $I \rightarrow E_1$) and $T_2$ (Time limit for signing message $m'$) to $Mix_1$. And meantime, as $U^*$, sends both $T_3$ (Time limit for sending voucher $W$) and $T_4$ (Time limit for transaction $K_1 \rightarrow O$) to $Mix_2$.

Step 2: If $Mix_1$ accepts the mixing request, it needs to send the commitment $V_i = \{\text{nonce}_1, T_1, T_2, \text{sign}(T_1 \| T_2) \| \text{nonce}_1 \| x_i\}$ to the user. The sign is the group signature $^1$ based on Schnorr signature with parameter $c$. For a message $m$, $Mix_1$ chooses a random number $r$, and calculates $s_1 = g^r \mod p_1, s_2 = H(m) \cdot x_i - r$. ($p_1, s_1, s_2$) is the signature. $Nonce$ is a random number to prevent the replay attack. The same is true for $Mix_2$.

Step 3: When $User$ authenticates $V_i$, the public keys $y_1, y_2$ of the $Mixes$ are recorded. $User$ first calculates $y_i = c \mod p_i$ according to the information disclosed by the group $c$, and then judges whether the equality $s_1 \cdot g^{y_1^2} = y_i^{H(m)}$ is true, which can determine the validity of the signature $V_i$.

Step 4: $User$ builds the transaction $tx_1 : I \rightarrow E_1$ (announced on the BB), and generates $m = \{0\|ID_2\|\text{nonce}_3\}$, a random number $b$ as the blinding factor, and calculates $m' = m \cdot b^p$. Finally, send $m'$ to $Mix_1$.

Step 5: $Mix_1$ confirms the transaction $tx_1$ and signs $W = \text{sign}(m')$, to $User$ by group signature. $W$ is the voucher used to communicate with $Mix_2$. For $Mix_1$, a transaction $tx_1$ corresponds to a signature. If $Mix$ is excessively signed, it will be discovered and punished by $\text{Supervisor}$ in audit phase.

Step 6: $User$ $U$ changes his identity to $U^*$, posts voucher $V$ on BB and sends $\{W, b, O, ID_2, \text{nonce}_3\}$ to $Mix_2$ to verify the voucher.

Step 7: $Mix_2$ first verify the group signature to obtain $m'$, remove the blindness of $b$ and $y_2$ to obtain $m^*$, and compare with the $m$. If they are consistent, $Mix_2$ build the transaction $tx_2 : K_2 \rightarrow O$, where $K_2$ is the private address of $Mix_2$. Otherwise, $Mix_2$ rejects the voucher.

Change of $User$’s identity belongs to data obfuscation at the network layer. To prevent adversary from obtaining identity and privacy information by discovering the network topology, researchers have proposed that the blockchain can be used on networks with privacy protection features, such as Tor [26]. Another type of digital currency known for privacy is Monero, which uses an anonymous communication scheme I2P [27]. Compared to the Tor protocol, the same network link is used to send and receive data. I2P uses multiple links to send data and accepting data can better hide IP and prevent transaction the traceability through network layer information [28].

4.1.3 Audit Phase

In the audit phase, $Mix$’s denial of service needs be monitored by $\text{Supervisor}$. In addition, transaction differences between $Mixes$ need be made up of auditing signatures.

Denial of service audit: For $Mix$’s denial of service behavior, we use the form of $User$ disclosure for auditing. If $E_1$ refuses to sign $m'$ message after generating $tx_1 : I \rightarrow E_1$ , $User$ only needs to take the record of $Mix_1$’s commitment $V_i$ and $tx_1$ to expose. If $tx_2 : K_2 \rightarrow O$ is refused after $Mix_2$ verifies the credential $W$, $User$ only needs to hold the commitment $V_2$ and the voucher $W$ of the $Mix_2$ to expose. $\text{Supervisor}$ confirms the disclosure. If there is a denial of service in $Mix$, $\text{Supervisor}$ will deduct the deposit and mark the corresponding $Mix$ by $ID$, and $Mix$’s reputation will decrease. Once the score of tags is too low, the $\text{Supervisor}$ has the right to force the malicious $Mix$ to exit group.

Signatures audit: $\text{Supervisor}$ compares whether the number $A$ of $Mix$’s signatures is less than or equal to the number $B$ of corresponding host transactions. If $A > B$, 

\[1545-5971$ (c) 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information. Authorized licensed use limited to: University of Texas at San Antonio. Downloaded on March 16,2021 at 12:18:21 UTC from IEEE Xplore. Restrictions apply.
Mix is considered as malicious one. Then, Supervisor would deduct the mixed consumption and its deposit, and meantime force it out of the group. Finally, Supervisor builds \( tx_3 : E_i \rightarrow E \) to recycle Bitcoin of all escrow addresses \( E_i \) to the total escrow address \( E \) of Supervisor. For honest Mixes, Supervisor sends mixing consumption to their private addresses.

### 4.2 CoinLayering-PB

When Supervisor makes up the cost difference between Mixes, it may steal Bitcoins. To prevent such behavior, CoinLayering-PB should allow the transactions between Mixes be carried out by themselves. For this, we use threshold signature technology to insure the security of this operation. The choice is due to the following: (1) secure. For a Mix’s operation, only when a majority of Mixes support can be completed. (2) Fault-tolerant. Even under the condition that 1/3 Mixes are compromised, it can normally work. (3) Efficiency. Supervisor can provide a large number of parameters for the threshold signature process in advance. Moreover, to prevent private keys from being compromised and thus protect Mixes’ Bitcoins, we also improve the group signature. To sum up, as is shown in Fig. 4, the improvements in CoinLayering-PB are mainly in the following phases, compared to CoinLayering-PA.

**Fig. 4. CoinLayering-PB.** The operations in red rectangles are ones distinct from CoinLayering-PA.

**In the registration phase,** each Mix’s escrow address should be generated by itself. We use secure Joint – RSS to achieve it. The detailed process is as follows:

**Step 1:** Each Mix gets the share \( k_i \) of the key \( k \) by secure Joint – RSS, which is based on Shamir Key Sharing (SS). It can divide a key into \( n \) key shares. As long as 2/3 participants are online, the original key can be restored through the key share.

**Step 2:** Mix computes the public key \( k_iG \)(announced on the BB), and the escrow address is \( E_i = Hash(k_iG) \). (The specific contents of Joint – RSS are as follows: each participant \( Mix_j \) takes itself as the center and selects a random secret value \( k_i^0 \). Then, construct a polynomial \( f_i(x) \), and execute SS to get the share of \( k_i^0 \). Mix_j (1 < j < n) receives the \( f_i(j) \) sent by the remaining \( n - 1 \) participants \( U_i(1 < i < n, i \neq j) \) and calculates \( k_i = \sum f_i(j) \) as the key share; to ensure the correctness of \( f_i(x) \) sent by Mix_i. Each Mix_j can get \( k_iG, a_i1^jG \). Mix_j can calculate \( k_iG = f_i(x)G = \sum f_i(j)(a_i1^jG) \). If the equation is true, the key share received by Mix_j is correct. Meanwhile, the secret by the participants is \( k = \sum k_i^0 \), the secret-sharing \( k_i = \sum f_i(j) \).

**In the mixing phase,** the group signature of the Mix is a very important step. If the private key of the Mix is leaked, not only the identity of the Mix will be forged, but the Bitcoin in the escrow address will also be at risk. It is very important to prevent the private key of Mix from leaking. We have improved the group signature as follows:

During the \( t \) time period, the private key of Mix_i is \( x_i \). At time \( t + 1 \), the private key \( x_i t+1 = x_i t^2 \mod (p_i - 1) \). At the same time, the key update algorithm will erase the key in time \( t \) immediately after the private key in time \( t + 1 \) is generated. If \( t = T \), the private key is output as an empty string. When the time slice runs out, group members need to regenerate a pair of keys.

The user’s group signature \( s_1 = g^{x_i} \mod p_i, s_2 = H(m) \cdot x_i t - r \). For group signature verification, first the verifier calculate \( y_i = c \mod p_i \) according to the information disclosed by the group \( c \) and then judge whether the equality \( s_1 \cdot g^{x_i} = y_i H(m)^2 \) is true to determine the validity of the signature.

**In the audit phase,** all escrow addresses are generated by the Mix and the Supervisor cannot operate on the escrow address. But this does not affect the Supervisor auditing Mix. If a malicious Mix privately forwards the Bitcoin in the escrow address, and it will be discovered during the audit phase. Supervisor will punish dishonest Mix, deduct the deposit and cancel its identity. During the audit phase, Supervisor removes all Mixes with malicious behavior. Mix wants to get the corresponding Bitcoin from the escrow address. Divided into the following steps:

**Step 4:** Each Supervisor initiates a transaction \( tx_i : E_i \rightarrow E \). All Bitcoins are transferred to the total escrow address.

**Step 5:** The Supervisor calculates the Bitcoins that the Mix should receive and announces the audit results on the BB. The Supervisor generates the key \( d_i, [d_i] \) and calculates the key share \( d_i \cdot [d_i] \), through the SS. Supervisor will assign them to each Mix.

**Step 6:** Mix initiates its own transaction \( tx_i : E \rightarrow E \). With \( (x, y) = (d_iG, R) = x \), each Mix computes \( s_i = ([d_i] \cdot (e + k_i \cdot R) \) and puts \( s_i \) on the BB.

**Step 7:** The Mix can get the \( s = \sum_{i \in T} b_is_i, b_i = \prod_{i \in A, i \neq j} 2^{-1} \), so it can use \( s \) to legally sign transaction \( tx_i \).

### 5 Security Analysis

This section mainly analyzes the security of Coinlayering, including strong anonymity, anti-denial service, signature unforgeability and backward security.
Theorem 1 (Strong anonymity). In CoinLayering, it is difficult for an adversary to guess the buyer-seller relationship.

Proof. To enhance the anonymity of Bitcoin owners, coin mixing system is responsible for cutting off the relationship between seller and buyer thoroughly. For this, to hide the target transaction A from the adversary, it can make other transactions that take place simultaneously with A as the missed items, which consist of the anonymous set. Apparently, the larger the anonymous set, the stronger the anonymity is. In addition, suppose that there are \( N_1 \) buyers and \( N_2 \) sellers during the interval that the system completes a coin mixing. Under ideal condition, for any pair of seller and buyer, the probability of getting it right is \( 1/(N_1 \cdot N_2) \). In other words, the upper bound of anonymous set is \( (N_1 \cdot N_2) \).

Based on the above, if the anonymous set of CoinLayering can achieve \( (N_1 \cdot N_2) \), it possesses the strong anonymity. To prove it, we take the following two steps: (1) the connection between seller and buyers in CoinLayering is less and (2) the size of anonymous set \( (N_1 \cdot N_2) \) is large enough. For the first step, in CoinLayering, because the adversary cannot detect the User’s choice and the connection between Mix1 and Mix2 is also cut off by group signature, their interactive process cannot be directly observed by the ledger, and this maximizes the difficulty of guessing the relationship between the buyer and the seller. One may argue that, once Mix1 and Mix2 collude with each other or the adversary colludes with a small number of Users, the guessing probability can be enhanced.

However, considering the Mix selection and economic cost, the occurrence probability of such situations is very low. For the second step, because CoinLayering has an effective load balance result through Mix selection, it can normally run under the large scale Bitcoin transactions as long as plentiful of Mixes are involved. To sum up, the strong anonymity in CoinLayering has been proved.

Theorem 2 (Anti-denial service). In CoinLayering, any Mixes who refuse to provide services would be exposed.

Proof. When User sends the timestamps \( T_1, T_2 \) or \( T_3, T_4 \) to a Mix \( \tau \), \( \tau \) normally requires issuing a commitment to User after accepting the request. However, once \( \tau \) is a semi-honest Mix in CoinLayering, three unexpected cases are exhaustive. (1) If \( \tau \) does not respond, User can choose another Mix after the timeout. In this case, \( \tau \) would not get any benefit from it except for the waste of time. (2) If \( \tau \) denies the mixing service after hosting Bitcoins and reject to provide a voucher for the User, Supervisor would audit the number of \( \tau \)'s vouchers during the audit phase. In this case, \( \tau \) would lose its mixing qualification and be charged the deposit. (3) If \( \tau \) provides the User with a voucher, it refuses to transfer the escrow Bitcoin. The commitment and voucher will be evidence of the denial of service. In the context of Blockchain, \( \tau \) cannot deny its behavior. For the above, if a semi-honest Mix actively refuses to serve the User, the Mix can not get any extra benefits and meantime its malicious behavior would be exposed.

Theorem 3 (Unforgeability). In CoinLayering, the adversary cannot forge any Mixes’ identities to provide users with false services.

Proof. In CoinLayering, the group signature is used to cut off the relationship between Mixes, and the threshold signature is used to secure the transactions. In this, once an adversary \( A \) forges these two signatures, it can forge Mix’s identity and defraud its Bitcoins. To prove the unforgeability of CoinLayering, we just need prove the following subproblems: the group signature is unforgeable and the threshold signature is unforgeable.

(1) Group signature unforgeability. Assume to the contrary that there exists an adversary \( A \) who can forge the group signature with a non-negligible probability \( P_0 \) under the random oracle model. In CoinLayering, we use \( s_1 \cdot g_i \cdot \sigma_i = y_i^{H(m||r_i)} \) to verify the validity of Mix’s signature. For convenience, we term \( s_1 \) and \( s_2 \) together as \( s_1 \) and let \( h_i = H(m||r_i) \). In this, to satisfy the above equation, the main work of \( A \) is to search \( s_1 \) and \( h_i \). To simulate the searching process, we construct a challenger \( B \) to respond to adversary \( A \)'s queries. The whole procedure can be divided into 3 steps. Step 1: select a Mix, as the forged object. After \( A \) sends the key query \( O_{Key} \) relevant with Mix; to \( B \), \( B \) randomly selects \( x_i \in Z_p \) to calculate \( (pk_1, sk_1) = (g^{r_i}, x_i) \), and then let Mix join the group, in which the group parameter \( c \) is updated by Supervisor. Step 2: acquire the Mix’s signature. After \( A \) sends a plaintext message \( m \) to \( B \), \( B \) uses the Schnorr signature technique \( \sigma = (m, s_1, s_2) \) to compute the \( m \)'s signature, where \( s_1 = g^r \mod p, s_2 = H(m||r_i) \cdot r_i - (x_i) \), and \( r_i \) is a random number. To acquire more of the Mix’s signatures, \( A \) can choose different messages and analyze them. Step 3: forge the Mix’s signature. After \( A \) forges the signature \( sign(m) \) and sends it to \( B \), \( B \) verifies whether such signature is valid. Once the forged signature \( sign(m) \) cannot be certified false through a series of queries, \( A \)'s forgery succeed and it can only be regarded as Mix. In this case, without loss of generality, we further assume that \( A \) can forge two signatures \((m, r_i, h_i, s_1) \) and \((m, r_i, h_i^*, s_1^*) \). Based on them, we can derive \( g_i^* = r_i \cdot y_i^{h_i^*} \) and \( y_i^{s_1^*} = r_i \cdot y_i^{h_i^*} \). Going a further step, we can use \( G_1 = g \cdot x_i = (s_1 - s_1^*) \cdot h_i^* \) to calculate the discrete logarithm \( x_i \) of \( y_i \).

However, this is the Discrete Logarithm Problem (DLP), i.e., there exists no polynomial time algorithm to search a feasible \( x_i \) under given \((g, g_i^*)\). This means that such adversary \( A \) does not exist, which contradicts the precondition.

(2) Threshold signature unforgeability. In CoinLayering, the threshold signature is a combination of ECDSA and Shamir’s Secret Sharing (SS). The ECDSA signature can be computed as \( s = d^{-1}(e + r \cdot k) \), where \( d \) is the private key of Mix, and \( k \) is the temporary key generated during signature calculating. Going a further step, by using SS technique, \( d \) and \( k \) are divided into sub-keys respectively, and then issue them to the varied Mixes. Suppose that an adversary \( A \) can control the first \( t \) Mixes, and further monitor their sub-keys. To prove the unforgeability of the threshold signature, we simulate the threshold signature process, and further certify that adversary \( A \) cannot utilize these \( t \) Mixes to recover the ECDSA signature \( s \). The simulation process is as follows. After obtaining the \( t \) Mixes’ sub-key \((d_1, d_2, \ldots, d_t) \) and \((k_1^*, k_2^*, \ldots, k_t^*) \), \( A \) can use interpolation formula to calculate \( R = k_i^*G(1 \leq i \leq t) \) and then calculate sub-signature \( s_i^* = ([d_i^{-1}]^t) \cdot (e + k_i^* \cdot R) (1 < i < t) \) through broadcasting.
$R$ to honest Mixes. Considering that ECDSA is a secure signature technique, to forge the signature $s$, A can only resort to the sub-signature $s_i$. According to Shamir’s secret sharing, $d$ and $k$ are $t$-order polynomials, i.e., only when more than $t$ sub-keys are collected can $A$ obtain $d$ and $k$. Since $s$ is generated by $d$ and $k$, it is $2t$-order polynomial, and thus $A$ requires collecting $2t$ $s_i$. Because each Mix stores a sub-key, $A$ needs compromise more than $2t$ Mixes, which contradicts the preassumption. Thus, the threshold signature cannot be forged.

**Theorem 4** (Forward security). In CoinLayering, if the adversary gets Mix’s private key, the system is still safe and trusted.

**Proof.** In CoinLayering, even if Mix reveals its current private key $x_i$ to the adversary $A$, $A$ is also unable to retrieve the previous information and further reveal User’s privacy. To prove it, we just need prove the following points: the forward security of the Mix’s private key and the forward security of the group signature.

(1) Forward security of the Mix’s private key. In CoinLayering, given a Mix, its private key $x_i$ is associated with the time period $j$, i.e., $x_i$ would change over time. For this, we utilize $x_{i,j+1} = x_{i,j}^2 \mod (p_i - 1)$ to update the private key. Because this one-way key updating function is based on large prime factorization of $p_i - 1$, in the limited time available, the adversary cannot use the current key $x_{i,j}$ to calculate the previous key $x_{i,j-1}$.

(2) Forward security of the group signature. Given a Mix, suppose that its private key $x_i$ at the time period $j$ is leaked. If the adversary $A$ tries to forge the group signature at time period $j - 1$, it needs make the equation $r_i \cdot g^{s_i} = y_i^2 H(m) \mod (p_i - 1)$ true through searching two valid values of both $r_i$ and $s_i$. According to Theorem 3, only when the private key $x_{i,j-1}$ has been acquired can $r_i$ and $s_i$ be found. Yet, due to the forward security of $x_{i,j}$, $A$ cannot calculate $x_{i,j-1}$. This means that, the signature at $j - 1$ is still secure.

6 **Performance Evaluation**

In this section, we focus on the proof of scalability in CoinLayering. For this, we first use the theoretical analysis to demonstrate its efficiency. Then, build simulations platform to perform extensive experiments so as to supplements the above analysis results.

6.1 **Theoretical Evaluation**

We use the mathematical analysis to evaluate the computation costs of encryption ($E_n$), square operation ($S$), modular multiplication ($M$), modular exponentiation ($E$), hash function ($H$) and elliptic curve scalar multiplication ($R$). Firstly, considering that the main computation work of Mix is to complete group signature, to measure the Mix’s load in CoinLayering, we conduct a theoretical evaluation on the involved group signature. Secondly, to demonstrate the efficiency, a theoretical evaluation of the entire protocol is necessary.

(1) In CoinLayering, we combine the Congruence based Group Signature with Schnorr Signature, termed as CGSSS. We choose the Forward Secure Group Signature (FSGS) [29] to compare with ours, both of which are capable of prevent Mixes from colluding, i.e., have the same security level. The former uses Mix selection to make Mixes unable to collude with each other; the latter increases the interaction with Supervisor in signature generation stage to prevent collusion.

Table 3 shows the comparison result, which contains five Stages: Key Update (KU), Member Joining (MJ), Member Revocation (MR), Signature Generation (SG) and Signature Verification (SV). Let $n$ be the number of Mix in the signature. In the key update stage, each Mix needs to update its own key $x_{i,t+1} = x_{i,t}^2 \mod (p_i - 1)$. Because each Mix is updated at the same time, it is an $(S)$ operation. In the member joining stage, Supervisor adds each registered Mix to the congruence, and compute $c = \sum_{i=1}^{k} y_i \cdot P_i \cdot P'_i$, which is $(nE)$ operations. In the member revocation stage, Supervisor modifies the public key $y_i = y_i^2 \mod p_i$ corresponding to the exit member and recalculates the public key $c$, which is $(E + nM)$ operation. In the signature generation stage, Mix chooses a random number $r$, and computes $s_1 = g^r \mod p_i$, $s_2 = (H(m) \cdot x_i - r) \mod p_i$, which is a $(E + M)$ operation. In the signature verification stage, Mix calculates $s_1 \cdot g^{y_i} = y_i^2 H(m)$ to verify the correctness of the signature, which is a $(2E + M)$ operation. Compared with FSGS, CGSSS has advantages in the calculation of the key update stage and member joining stage. Firstly, FSGS requires all Users to update their private keys simultaneously, but CGSSS could allow Users to update their private keys according to their own requirements, which can effectively reduce its computational overhead. Secondly, whenever a new Mix joins, CGSSS only requires Supervisor to recalculate the group public key $c_{new}$ and then issue them to the Mixes, but FSGS requires the new one to interact with all other Mixes, and update their parameters. In particular, even under the condition that part of Mixes cannot receive the public key $c_{new}$, the interactions among the remaining Mixes can run normally in CGSSS. This cannot only reduce the computation cost, but also minimize the bandwidth overhead. In brief, CGSSS is more efficient. Especially, as more and more Mixes join in the Bitcoin marketplace, it has better characteristic of scalability.

<table>
<thead>
<tr>
<th>Stages</th>
<th>CGSSS</th>
<th>FSGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key Update</td>
<td>$S$</td>
<td>$2S$</td>
</tr>
<tr>
<td>Member Joining</td>
<td>$nM$</td>
<td>$n(M + E)$</td>
</tr>
<tr>
<td>Member Revocation</td>
<td>$E + nM$</td>
<td>$E + nM$</td>
</tr>
<tr>
<td>Signature Generation</td>
<td>$E + M$</td>
<td>$2E + M$</td>
</tr>
<tr>
<td>Signature Verification</td>
<td>$2E + M$</td>
<td>$2E + M$</td>
</tr>
</tbody>
</table>

(2) We choose two typical schemes to compare with CoinLayering. Coinparty [10] is a decentralized structure coin mixing scheme, which also uses threshold signatures in the scheme. Zerocoin [30] is a centralized structure coin mixing scheme with excellent privacy protection.

Table 4 shows the computation costs of CoinParty, Zerocoin and CoinLayering. Let $n$ be the number of Mix in the protocol. In CoinLayering, there is no operation to encrypt the message. In the registration phase, each User generates the key $y_i = g^{x_i} \mod p_i$ and a knowledge signature $c_i = H(\text{Time} || y_i || g || g^{x_i})$, which include
6.2 Experimental Evaluation

We firstly measured the computation time of CoinLayering-PA and CoinLayering-PB respectively, including the group signature and threshold signature. Furthermore, we evaluate the overall performance of CoinLayering. Specially, investigate its computation and storage overheads, as the number of blocks increases. Calculation time refers to the actual running time of various operations, including modular multiplication $T_M$, modular exponentiation $T_E$, hash $T_H$ and elliptic curve scalar multiplication $T_R$. All of the experiments are performed on the server with Intel 2.6GHz i7-4720 CPU, 8GB RAM and Windows XP. We use JPBC (Java Pairing-Based Cryptography Library) library to implement our concerned cryptographic techniques, in which RSA modulus in the selected accumulator is 1024 bits and hash function is SHA-256.

6.2.1 On The Performance of CoinLayering-PA

As congruence-based group signature (i.e., CGSSS) is the main cryptographic technique that is simultaneously involved in the CoinLayering-PA’s registration phase and mixing phase, we focus on testing its computation time. According to Section 6.1, such group signature is divided into five stages: MJ, MR, KU, SG and SV. Fig. 5 investigates the computation costs at different stages. The experimental results are consistent with our theoretical evaluation results. Compared with the existing secure group signature $FSGS$ [29], the advantages of CoinLayering-PA are mainly reflected in the following stages. In KU stage, compared with CoinLayering-PA, $FSGS$ requires one more squaring operation, but there is no significant difference in their computation costs, because the squaring operation is lightweight. In MJ stage, because the computation cost of $E$ is relatively higher, as the $Mixes$ increase, the computation cost of $FSGS$ is about to get even larger, i.e., the difference between CoinLayering-PA and $FSGS$ would also become larger. In SG phase, to ensure security, $FSGS$ requires $Supervisor$ to perform a $E$ operation, which also increases the computation cost.

6.2.2 On The Performance of CoinLayering-PB

To secure the Bitcoin transactions between $Mixes$, CoinLayering-PB adds threshold signature to CoinLayering-PA. In this, we first test the computation costs of threshold signature in CoinLayering-PB. Then, measure the computation time of CoinLayering-PB’s entire process and compare it with other protocols (including CoinParty [10] and ZeroCoin [30]).

### TABLE 4

<table>
<thead>
<tr>
<th>Operations</th>
<th>CoinParty</th>
<th>ZeroCoin</th>
<th>CoinLayering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encryption</td>
<td>$n^2 E_n$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Modular Multiplication</td>
<td>$8nM$</td>
<td>$9nM$</td>
<td>$4nM$</td>
</tr>
<tr>
<td>Modular Exponentation</td>
<td>$8nH$</td>
<td>$12nE$</td>
<td>$8nE$</td>
</tr>
<tr>
<td>Hash</td>
<td>$4nH$</td>
<td>$nH$</td>
<td>$5nH$</td>
</tr>
<tr>
<td>Elliptic Curve</td>
<td>$(10n)_R$</td>
<td>0</td>
<td>$(3n)_R$</td>
</tr>
<tr>
<td>Scalar Multiplication</td>
<td>$nE$</td>
<td>$E$</td>
<td>$E$</td>
</tr>
</tbody>
</table>

Fig. 5. Comparison between CoinLayer-PA and $FSGS$

Fig. 6. Computation costs at different stages in threshold signature

In CoinLayering-PB, the threshold signature technique are mainly divided into the following stages: Security Secret Sharing (SSS), Escrow Address Generation (EAG), Total Escrow Address generation (TEA), Request Signature Key (RSK), Sub-Signature Generation (SSG), Threshold Signature Generation (TAG). In this, we investigate the computation costs at different stages, and the results are shown in Fig. 6. We can obviously observe that SSS and RSK stages take more time, but this does not affect the efficiency of transaction. The reasons can be stated as follows. Firstly, although Joint-SSS operation with high computation cost is required in SSS stage, it only requires performing once
during the Mix registration. Secondly, since In−SS operation with high computation cost is the main work of RSK stage, CoinLayering-PB allows the Supervisor with abundant computing resources to perform it, instead of those over-loaded Mixes. This can effectively ensure the efficiency of transactions. From the above, even in the face of large scale transactions, Coinlayering-PB is scalable.

CoinLayering-PB mainly contains the following operations: M, E, H and R. For the convenience of functional analysis, we investigate the total running time of each operation, and the results are shown in Fig. 7, in which SUM denotes the and the aggregation time of all operations. We can summarize the following interesting observations: (1) the computation cost of CoinLayering-PB is between CoinParty and ZeroCoin. This is because R operation would bring about the high computing overhead. In CoinLayering, R operation is performed 4 times, while 10 times in CoinParty. As for ZeroCoin, it is based on discrete logarithms and thus does not require R. (2) The computation cost of CoinLayering-PA is almost equal with CoinLayering-PB. This means that, the threshold signature is lightweight, and does not take up Users’ transaction time.

Fig. 7. Comparison between Coinlayering-PB and other protocols

6.2.3 On The Performance of CoinLayering

As the number of transactions increases, performance overhead of CoinLayering also grows. To ensure that CoinLayering can cope with the large scale Bitcoin transactions, here we focus on evaluating its scalability. Firstly, we investigate how the computation costs of varied entities (including User, Supervisor, Mix1 and Mix2) scale with an increasing number of transactions. We separate Mix1 from Mix2 during test execution, because their operations are different in mixing process. And we test the overall performance of CoinLayering and meantime compare it with CoinParty [10] and Zerocoin [30]. Secondly, considering that the number of network trips is an important factor affecting task execution time, we compare the communication amounts in the varied schemes. Thirdly, considering that the multiple copies of redundant storage in blockchain make it potential for the scalability issues, we aims to investigating the storage cost through varying the number of transactions. For the discrete logarithm-based signature and elliptic curve-based signature, we respectively fix 1024-bit and 256-bit.

Fig. 8 shows the computation costs of varied entities worked as a function of the number of transactions. For any entity, we measure its computation time through accumulating the running time of all its linear pair operations. From the evaluation results, we made the following observations. Firstly, it is evident that the computation cost of User is low and its curve is almost flat. The reason is that, for the User, only a few operations are required to perform in CoinLayering. In general, its total computation cost is 2M + 3E + H, which includes registration cost M + E + H (mainly involve public key generation and knowledge signature) and mixing cost E + M (involve message blinding). Secondly, though the computation cost in Mix1 is slightly higher than Mix2, their difference is not so much (belong to the same order of magnitude). This is because, judging by computational overhead only, compared to Mix1, Mix2 needs to only one more M. Thirdly, with the increasing of transactions, the computation cost of Supervisor is also rising up. The increase of transactions requires more registered Mixes, which results in the surge of M operations. From the above, we can conclude that, in the face of large scale transactions, Supervisor is more likely to be the bottleneck of the system. However, once we migrate it to cloud platform with super capacity, such issue would be well solved.

Fig. 8. Computation costs of varied entities by increasing the number of transactions

Fig. 9. Overall comparison between CoinLayering and other schemes

Fig. 9 shows how the overall computation cost varied as the number of transactions increases. For ease of comparison between the other schemes, we measure overall computation time through accumulating the trading time of each entity. In the simulation, trading time just refers to the spent time during mixing phase, considering that registration cost is produced only when the system is initialized or new User joins. The evaluation results are summarized as follows. Firstly, no matter CoinLayering, Coinparty or ZeroCoin, the
The overall trend of its computation cost is obviously rising up, with the increasing of transactions. In CoinLayering, according to the conclusion from Fig. 8, Supervisor is the major contributors to the increase of overall computation cost. For any new User, in Coinparty, other ones have to interact with it so as to generate the new signature, which adds additional $E + M$ operations. Zerocoin requires all Mixes to constantly interact with the new User and perform $E$ operation. Secondly, with the increasing of transactions, the computation cost of CoinLayering is higher than Coinparty and Zerocoin, which is slightly different from Fig. 7. This is because, CoinLayering wouldn’t require significant $M$, $E$ and $H$ operations in trading situations, and meantime its most time-consuming operation $R$ takes place in the registration phase, which would not take up the trading time.

Fig. 10 shows the evaluation results for communication amount (i.e., the number of network trips), which is worked as a function of the number of Users and the number of transactions. It is clear that, compared to CoinParty, both ZeroCoin and CoinLayering require less communication amount. The reason can be stated as follows. ZeroCoin adopts zero-knowledge proof technology and thus only requires three communications for one transaction; on this basis, CoinLayering adds one more interaction between two Mixes; CoinParty requires significant communications to generate threshold signatures and escrow addresses.

Fig. 11 shows how the storage cost changed as the number of transactions increases. In coin mixing, the signatures and other credentials would be recorded on the chain. It is evident from the figure that the storage requirements in CoinLayering is higher than CoinParty, but far lower than Zerocoin. This is because CoinParty adopts the elliptic curve based threshold signature with the smaller length, but CoinLayering still requires a group signature, besides the 256-bit threshold signature. In ZeroCoin, it adopts the knowledge signature whose length is the same with the group signature of CoinLayering. In addition, it needs to first convert Bitcoin to 1024-bit Zerocoin, which enforces the need for the storage.

7 Conclusion

In this paper, we proposed an efficient coin mixing scheme for large scale Bitcoin transactions. The building blocks of our proposed CoinLayering scheme are as follows: a User − Mix − Supervisor based system model (that allows User to randomly select two Mixes to respectively execute the Bitcoin holding and Bitcoin trading actions), a Mix selection algorithm (to ensure task completion), and two security coin mixing protocols (to mitigate the risk due to misbehaving middlepersons and Supervisor). Our security and performance evaluations demonstrated the utility of CoinLayering.

ACKNOWLEDGMENT

This work supported by the National Natural Science Foundation of China (Nos.62072093, 62072092, 61601107and U1708262); the China Postdoctoral Science Foundation (No.2019M653568); the Fundamental Research Funds for the Central Universities (No.N202302); the Natural Science Foundation of Hebei Province of China (No.F2020501013). The work of Kim-Kwang Raymond Choo was supported by the Cloud Technology Endowed Professorship, and National Science Foundation (NSF) CREST Grant HRD-1736209. Yuan Chang is the co-first author of this work.

REFERENCES

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2020.3043366, IEEE Transactions on Dependable and Secure Computing


Kim-Kwang Raymond Choo (Senior Member, IEEE) received the Ph.D. in Information Security in 2006 from Queensland University of Technology, Australia. He currently holds the Cloud Technology Endowed Professorship at The University of Texas at San Antonio (UTSA). He was included in Web of Science’s Highly Cited Researcher in the field of Cross-Field – 2020, and in 2015 he and his team won the Digital Forensics Research Challenge organized by Germany’s University of Erlangen-Nuremberg. He is the recipient of the 2019 IEEE Technical Committee on Scalable Computing Award for Excellence in Scalable Computing (Middle Career Researcher), the 2018 UTSA College of Business Col. Jean Piccione and Lt. Col. Philip Piccione Endowed Research Award for Tenured Faculty, the British Computer Society’s 2019 Wilkes Award Runner-up, the 2019 EURASIP Journal on Wireless Communications and Networking Best Paper Award, the Korea Information Processing Society’s JIPS Survey Paper Award (Gold) 2019, the IEEE Blockchain 2019 Outstanding Paper Award, the Inscrypt 2019 Best Student Paper Award, the IEEE TrustCom 2018 Best Paper Award, the ESORICS 2015 Best Research Paper Award, the 2014 Highly Commended Award by the Australia New Zealand Policing Advisory Agency, the Fulbright Scholarship in 2009, the 2008 Australia Day Achievement Medallion, and the British Computer Society’s Wilkes Award in 2008.
**APPENDIX A: INCENTIVE STRATEGY**

To attract more honest Mixes, CoinLayering pays several Bitcoins as the profit to them. For a User, with the increase of the transactions, the cost to pay for those mixing service would also grow linearly. But, this would bring severe economy burdens for this User, and further decrease the incentive to mix coins. In this, we provide an incentive mechanism termed as CoiInc, in which User would acquire extra rewards as long as it persistently utilizes CoinLayering.

Only if the User continuously purchases the mixing service can its costs be reduced by Supervisor. Given a time interval $T_d$, we term the User that continuously purchases more than two mixing services during $T_d$ as the continuous mixing User. In this, our proposed CoinInc contains two steps: applying for incentive subsidy and granting for incentive subsidy. The details can be depicted as follows:

**Step 1:** When User purchases the coin mixing service more than once, it firstly performs the operation $tx_2^2: I \rightarrow E_1$. And then calculates the difference $T_s$ between the current transaction time and the last transaction time. If $T_s \leq T_d$, the User submits the $tx_2^2$ record to Supervisor and applies for incentive subsidy.

**Step 2:** Upon receiving the request, Supervisor firstly verifies the $tx_2^2$ and record its input User’s address $I$. Then, use $I$ to search the last transaction $tx_d^1$ in $BB$. Meantime, judge whether the User $I$ is eligible for the incentive subsidy through computing the time difference between $tx_d^1$ and $tx_2^2$. If it does, Supervisor computes the amount of its incentive subsidy $f = BC \times b$ and then sends them to $I$, where $BC$ denotes the total value of the current transaction, $b$ denotes the proportion of incentive subsidy in $BC$. Normally, $b$ is proportional to the number of mixing times during $T_d$.

It is worthy of note that, although Supervisor can verify the input User’s address during the second step, it is incapable of watching the privacy of seller-buyer relationship and thus cannot decrease the anonymous set. The reason is that, during the interaction with Mix$_2$, the identity of User is different from that it interacts with Supervisor. Going a further step, Supervisor is unable to guess the buyer associated with Mix$_2$, let alone the connection between the seller and buyer.

**APPENDIX B: A MORE SECURE GROUP SIGNATURE TECHNIQUE**

In this section, we mainly illustrate how to embed the existing secure group signature technique [25] into our proposed CoinLayering. To prevent the Mix from watching the User’s information in secret, it requires the value of elements in group signature to be changed every time. The details can be depicted as follows.

**Step 1:** Supervisor generates the group’s private key $(p, q, d)$, where $p$ and $q$ are two random primes. Then, generate group’s public key $(n, e)$, where $n = p \times q$ and $e \times d = 1 \pmod{n}$. At last, assign a random prime number $p_i$ for each Mixes and meantime send it to the corresponding Mix.

**Step 2:** When Mix$_i$ receives $(g, n, p_i)$ from Supervisor, it calculates the Mix$_i$’s public key $y_i = g^{pi} \mod n$ and sends it to Supervisor, where $g$ is the generator of cyclic group, and $x_i$ is Mix$_i$’s private key.

**Step 3:** After receiving $y_i$, Supervisor constructs the congruence $c = y_i \mod p_i$ and then saves $c$’s value in a private way.

**Step 4:** Mix$_i$ uses its private key $x_i$ to sign the message $M$. Then, calculate its signature $s_i = h(M||\xi \cdot x_i) \mod n$ and send $(M, \xi, s_i, p_i)$ to Supervisor.

**Step 5:** Supervisor needs verify whether the signature is valid through $y_i = c \mod p_i$ and $h(M||\xi) = s_i^{y_i} \mod n$. If valid, it calculates group signature $C = (h(M||s_i||r_1||r_2))^d$, where $r_1 = p_i + ah(M||\xi) \mod n$ and $r_2 = (ah(M||\xi))^e \mod n$. And then, send $(M, \xi, s_i, C, r_1, r_2)$ to Mix$_i$.

**Step 6:** When a coin mixing deal starts, Mix$_1$ firstly sends the group signature (as a voucher) to User. To verify this group signature, User then sends it to Mix$_2$. If $C^e = h(M||s_i||r_1||r_2)$, it is valid.

**APPENDIX C: AN ILLUSTRATION EXAMPLE**

In this section, for easy understanding of CoinLayering, we present illustrative examples for the main primitives (including Mix selection and coin mixing protocol). Under normal circumstances, the entire workflow of CoinLayering can be stated as follows.

**TABLE 5**

<table>
<thead>
<tr>
<th>Mix,</th>
<th>$A_1$</th>
<th>$A_2$</th>
<th>$A_3$</th>
<th>$A_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>i=1</td>
<td>60</td>
<td>95</td>
<td>60</td>
<td>95</td>
</tr>
<tr>
<td>i=2</td>
<td>70</td>
<td>85</td>
<td>70</td>
<td>85</td>
</tr>
<tr>
<td>i=3</td>
<td>90</td>
<td>75</td>
<td>90</td>
<td>75</td>
</tr>
<tr>
<td>i=4</td>
<td>95</td>
<td>65</td>
<td>95</td>
<td>65</td>
</tr>
<tr>
<td>i=5</td>
<td>55</td>
<td>80</td>
<td>55</td>
<td>80</td>
</tr>
<tr>
<td>i=6</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>75</td>
</tr>
<tr>
<td>i=7</td>
<td>80</td>
<td>60</td>
<td>80</td>
<td>60</td>
</tr>
<tr>
<td>i=8</td>
<td>45</td>
<td>65</td>
<td>45</td>
<td>65</td>
</tr>
<tr>
<td>i=9</td>
<td>72</td>
<td>62</td>
<td>72</td>
<td>62</td>
</tr>
<tr>
<td>i=10</td>
<td>80</td>
<td>50</td>
<td>80</td>
<td>50</td>
</tr>
</tbody>
</table>

**Step 1:** The owner of Bitcoins demonstrates to Supervisor that it is fully qualified for a Mix. When successful, it generates a private key. Assume that there are 10 Mixes registered in Supervisor.

**Step 2:** Supervisor evaluates each Mix and the relevant attributes are shown in Table 5. And meanwhile, Supervisor records the Mixes’ keys and enables them to generate signatures.

**Step 3:** When User initiates coin mixing, it first finds Mixes that meets its requirements, i.e., sends its requirements $\vec{w} = (0.1, 0.1, 0.2, 0.6)$ to Supervisor.

**Step 4:** Supervisor firstly analyzes the current queueing situation and calculates the least number of candidate Mixes $k = 3$ and average queue length $LQ = 5$. And then, use Mix selection strategy to recommend 3 appropriate Mixes, whose attributes are $M_1 = (60, 90, 60, 95)$, $M_2 = (70, 85, 70, 85)$, $M_3 = (90, 75, 90, 75)$.

**Step 5:** User randomly selects Mix$_i$ recommended by Supervisor and obtain its queue length $L_i$. If $L_i < LQ$, Mix$_i$ can be specified as Mix$_{i1}$ or Mix$_{i2}$.
Step 6: User utilizes identity $U$ to send $T_1(23), T_2(38)$ to $Mix_1$, and meantime utilizes identity $U^*$ to send $T_3(53), T_4(68)$ to $Mix_2$. When accept the requests, $Mix_1$ and $Mix_2$ respectively send commitments $V_1 = \{\text{sign}\{23\|||38\||40\text{ibuLn6jFDn3ZVF}\}x_1$ and $V_2 = \text{sign}\{53\|||68\||\text{OBTIKydiEpkkGjzw}\}x_2$.

Step 7: User uses identity $U$ to build $tx_1 : I \rightarrow E_1$ before $T_1$ and sends $m' = (O||ID_2||tfGzh3NFY\text{H0W}DugN) : b^{e_2}$ to $Mix_1$.

Step 8: $Mix_1$ checks transaction $tx_1$ and sends $W = \text{sign}\{m'\}x_1$ to $U$ before $T_2$.

Step 9: User utilizes identity $U^*$ and sends $W$ to $Mix_2$ before $T_3$.

Step 10: $Mix_2$ verifies $W$ and builds $tx_2 : K_2 \rightarrow O$ before $T_4$. After the owner of address $O$ receives the transaction $tx_2$, the User’s transaction is completed.

Step 11: supervisor conducts an audit for each $Mix$ every time $t$, which is used to check whether the number of transactions hosted by $Mix$ is consistent with the number of issued certificates $W$.

Step 12: During the audit on supervisor, $Mix_1$ builds the transaction $tx_3 : E_i \rightarrow E$, and $Mix_2$ initiates threshold signature $s_i = (i|d_i|) \cdot (e + k_i \cdot R)$ on transaction $tx_4 : E \rightarrow K_2$ (including mixing fees).