UCoin: An Efficient Privacy Preserving Scheme for Cryptocurrencies

Mohammad Reza Nosouhi, Shui Yu, *Senior Member, IEEE,* Keshav Sood, Marthie Grobler, Raja Jurdak, Ali Dorri, and Shigen Shen *Member, IEEE*

Abstract—In cryptocurrencies, privacy of users is preserved using pseudonymity. However, it has been shown that pseudonymity does not result in anonymity if a user's transactions are linkable. This makes cryptocurrencies vulnerable to deanonymization attacks. The current solutions proposed in the literature suffer from at least one of the following issues: (1) requiring a trusted third–party entity, (2) poor performance, and (3) incompatible with the standard structure of cryptocurrencies. In this paper, we propose Unlinkable Coin (UCoin), a secure mix–based approach to address these issues. In UCoin, the link between the input (payer) and output (payee) addresses in a transaction is broken. This is done by mixing the transactions of multiple users into a single aggregated transaction in which the output addresses have been secretly shuffled. In our protocol design, we first develop HDC–net, a secure shuffling protocol that enables a group of users to anonymously publish their data. Then, we deploy the proposed HDC–net protocol in the UCoin architecture (as a mixing unit) to generate the aggregate transactions. We show that UCoin (1) does not rely on a trusted third–party, (2) can mix 50 transactions in 6.3 seconds that is 18% faster than the current solutions, and (3) is fully compatible with the architecture of cryptocurrencies.

Index Terms—Anonymity; Bitcoin; Cryptocurrencies; Privacy; Unlinkability.

1 Introduction

Since their emergence in the last decade, cryptocurrencies (e.g., Bitcoin) have attracted a lot of attention from both academia and industry. According to the latest records, 2224 different cryptocurrencies have been listed in CryptoCurrency Market Capitalizations web site with $250 billion total market cap [1]. The main characteristic of cryptocurrencies that makes them popular is that unlike e–cash systems [2–3], they do not require any trusted central party to work. Instead, cryptocurrencies store users' transactions in a distributed ledger known as the blockchain which is publicly accessible. Use of the blockchain technology also enables cryptocurrencies to guarantee data integrity which is absolutely critical for financial databases. The reason is that blockchains are inherently immutable and completely resistant to data modification, i.e., once data is added to the ledger, it cannot be changed or deleted [4–6].

Although there is a common perception that cryptocurrencies provide anonymous transactions, the majority of cryptocurrencies still struggle to guarantee anonymity for their users [7–8]. In cryptocurrencies, users may use many different pseudonyms (e.g., Bitcoin addresses) to preserve their privacy. Thus, they believe that their transactions are anonymous. However, this is not correct because pseudonymity does not always result in anonymity. The results of some research studies show that since users' transactions are generally linkable, they can be deanonymized by using information available in the public ledger [9–12]. This may make cryptocurrencies vulnerable to deanonymization attacks.

To address this issue, a number of privacy protection schemes have been proposed in the literature. In some of these proposals, a new cryptocurrency has been proposed that provides anonymity by design [10], [13–14]. In other proposed solutions, privacy–preserving protocols have been developed specially for Bitcoin to provide anonymity for its users [15–19]. There are also some centralized mixing services that have been implemented to provide anonymity for Bitcoin users [20–22]. They receive Bitcoin transactions from different users and mix them together into an aggregated transaction in such a way that it is infeasible to link each bitcoin address (pseudonym) in the aggregated transaction to its owner.

However, these solutions suffer from at least one of the following issues: (1) requiring a trusted third-party entity [17], [18], [21], [22], (2) poor performance (in terms of speed or storage) [13], [15], [22], and (3) lack of compatibility with the standard structure of cryptocurrencies [5]. Moreover, some of these proposals are vulnerable to DoS attacks [16], [19] or do not support blockchain pruning [10], [13], [14]. The latest approach (mixing services) also needs relying on a third party that is not always secure [7–8], [23].

Motivated by this, we present UCoin, an efficient protocol for anonymous payment in cryptocurrencies. Our protocol design consists of two phases, i.e. (1) development of a mixing protocol, and (2) integration of the mixing protocol into the architecture of cryptocurrencies. In the first phase,

- *Mohammad Reza Nosouhi and Shui Yu are with the School of Computer Science, University of Technology Sydney, Australia. E-mail: mohammad.r.nosouhi@student.uts.edu.au; shui.yu@uts.edu.au*
- *Keshav Sood is with the School of IT, Deakin University, Melbourne Australia. E-mail: keshav.sood@deakin.edu.au*
- *Marthie Grobler is with the Commonwealth Scientific and Industrial Research Organization (CSIRO), Australia. E-mail: Marthie.Grobler@data61.csiro.au*
- *Raja Jurdak and Ali Dorri are with the School of Computer Science, Queensland University of Technology. E-mail: {r.jurdak and ali.dorri}@qut.edu.au*
- *Shigen Shen is with the Departement of Computer Science, and Engineering, Shaoshing University, China. E-mail: shigens@usx.edu.cn*

we modify the Dining Cryptographers Network protocol (DC–net) [26] and propose *Harmonized DC–net (HDC–net)* as a new decentralized and self–organizing mixing protocol. The original DC–net protocol suffers from two drawbacks that make it impractical, (1) *collision possibility*, and (2) security issues. We propose a solution for each drawback and develop the final HDC–net mixing protocol.

HDC–net is a distributed and non–interactive mixing protocol. It enables N peers to mix their messages in such a way that an attacker who monitors the peers' traffic cannot determine which message belongs to which peer (note that *message* here refers to any sort of information, not necessarily payment information. In other words, HDC–net can work with non–monetary applications as well). The main characteristic of HDC–net protocol is that it does not require any trusted third party for mixing, such as a mix server or an onion router [24], [25]. In other solutions for anonymous communication (e.g. *Tor* [24] and *onion routing* [25]), a number of trusted mixing servers are needed, otherwise, the protocols become vulnerable against traffic analysis attacks [16].

In the next phase, we employ the proposed HDC–net protocol as the core module of UCoin and use it to mix the individual transactions of a group of users into a single aggregated transaction in such a way that the link between the payers and payees is unknown to any observer. We apply the proposed mechanism on Bitcoin [27] as the most popular cryptocurrency [4], [6], [8] to provide anonymous Bitcoin payments. The same approach can be used for other cryptocurrencies since our HDC–net protocol is fully–independent of the applied cryptocurrency settings. Using the UCoin protocol, N peers can non–interactively mix their transactions into a single aggregated transaction (without using a third–party mixing server) to ensure that the input and output accounts in the aggregated transaction are unlinkable (see fig. 1).

We use the Deterlab [28] infrastructure to perform a prototype implementation of UCoin and test its performance. The results of our experiments show that UCoin can mix 50 Bitcoin transactions in 6.3 seconds which is faster than the current solutions. The following are our contributions:

- We develop solutions to address the security drawbacks of the original DC–net protocol. These drawbacks make it feasible for an adversary to deanonymize the sender of each message.
- We propose HDC–net, a decentralized and self–organizing anonymous mixing protocol that does not require any trusted–third party. It enables a group of peers to anonymously and securely mix their messages such that it is not feasible to determine which message belongs to which peer.
- Further, we develop UCoin by deploying the proposed HDC–net protocol as the mix approach to provide anonymous Bitcoin payments. UCoin is simpler and faster than the current solutions. Moreover, it is fully compatible with the Bitcoin protocol.

The remainder of this paper is organized as follows. After reviewing the related literature in Section 2 and presenting some preliminaries in Section 3, we introduce the HDC–net and UCoin protocols in Section 4 and 5, respectively. Then, in Section 6, we present and discuss performance analysis and experimental results. Finally, we conclude the paper in Section 7.

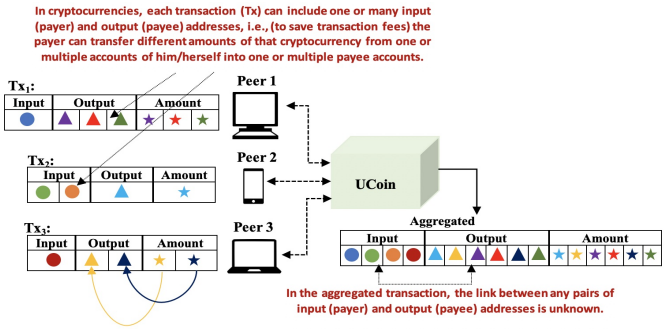

Fig. 1: Using the UCoin protocol, the peers can jointly aggregate their transactions into a single transaction in which the link between input and output addresses is known to neither an observer nor the peers (i.e., each peer is aware of the addresses of his/her own Tx only). Note that the amount fields of the individual transactions experience the same shuffle. This ensures us that a correct amount is transferred to each output account (as it would be transferred using the associated individual transaction).

2 Related Work

In this section, we present a brief literature review of anonymity and privacy in cryptocurrencies. We review the related research work in three main categories.

(1) Cryptocurrencies that provide anonymity by design: A number of cryptocurrencies have been proposed in the literature that inherently provide anonymity for users, i.e. privacy preserving issues have been considered in their design. In CryptoNote [13], ring signatures are used to group the users' addresses. The users in the group employ non–interactive zero–knowledge proofs to generate a ring signature. To verify this ring signature, all the public keys of the users in the group are required. Thus, it is not feasible to determine which address belongs to a specific user. However, in CryptoNote, a larger group size results in a transaction with a larger size [8], [16].

In Zerocoin [14], the link between transactions are hidden using non–interactive zero–knowledge Proofs. It works based on a decentralized setting, thus, it does not require any trusted party. The Schnorr [29] technique is used for the zero–knowledge proofs. However, Zerocoin does not hide amounts and destinations of payments [10]. To address these issues, Ben-Sasson et al. [10] proposed Zerocash, a fully-developed ledger–based cryptocurrency. In Zerocash, Zero Knowledge Succinct Non–interactive ARguments of Knowledge (zk–SNARK) [30] is employed to hide not only inputs but also outputs and amount of a transaction. Due to the use of zkSNARKS, Zerocash needs a one–time parameter setup performed by a trusted party. This makes it dependant on a trusted party [16], [8].

(2) Privacy–preserving protocols developed for Bitcoin: These protocols are added as an extension to the Bitcoin architecture to provide anonymity for the users. CoinShuffle++ proposed by Ruffing et al. [16] is a decentralized

mixing protocol designed for Bitcoin users specifically. CoinShuffle++ works based on DiceMix, a peer–to–peer mixing protocol proposed in the same paper. In fact, DiceMix is applied on Bitcoin to make transactions unlinkable. CoinShuffle++ is simpler and more efficient than its predecessor, CoinShuffle [15].

In CoinParty [17, 18], a number of peers are selected to perform as mixing peers to provide unlinkability for the Bitcoin transactions. It uses threshold variant of the Elliptic Curve Digital Signature Algorithm (ECDSA) to aggregate coins by a threshold transaction in the commitment phase before the final address shuffling is performed. SecureCoin [19] is another privacy preserving protocol that is fully compatible with Bitcoin. It shuffles the destination addresses of Bitcoin transactions using public key encryption. The employed technique is similar to the onion routing [25] protocol. Unlike CoinParty, SecureCoin does not require any mixing peer. Thus, it is not vulnerable to disruptions that may be conducted by mixing peers. However, SecureCoin needs its users to trust the protocol since it requires the peers to deposit their coins in a temporary aggregation bitcoin address before the main stage of the protocol is performed.

(3) Mixing Services: A number of centralized mixing services [20] have been proposed to provide anonymity for Bitcoin users. By using these services, users can send their coins to a trusted mixing service and receive them back at a new address. In fact, a mixing service removes the link between a bitcoin address used in a transaction from the owner of the transaction. However, users must fully trust the mixing service provider. These services provide limited anonymity because the mixing service provider is still able to link the new addresses to the real addresses [32].

Mixcoin proposed by Bonneau et al. [21] is a centralized mixing service that tries to address the first issue by using an accountable mechanism. It makes the mixing service provider accountable and ensures that any dishonest mixing operation results in poor reputation. However, it does not guarantee that the mix service always performs the protocol honestly. To enhance the Mixcoin protocol, Valenta et al. proposed Blindcoin [22]. It uses a blind signature scheme [33] to prevent the mixing server from accessing the output addresses of transactions. Thus, the link between input and output addresses is removed for the mixing server. However, the risk of coin theft is still present.

3 Preliminaries

In this section, we present the foundation for the next sections. We first present a brief overview of Bitcoin (refer to [4], [6], and [8] for a comprehensive review). Then, we review the DC–net protocol and introduce its challenges.

3.1 Overview of Bitcoin

Bitcoin [27] is the first and most popular digital currency and payment system. In Bitcoin, payments are conducted by creating transactions that transfer bitcoins between users. To do this, users create their transactions and broadcast them into the Bitcoin peer–to–peer network without the need for an intermediary (e.g., a central bank). A published transaction is then picked up by miners for verification to ensure its authenticity, validity, and integrity. A miner can be any resourceful Bitcoin user that performs the following computations: (1) verifying newly issued transactions, (2) creating new blocks of transactions by embedding the newly verified transactions into a new block, and (3) adding these blocks to the blockchain (the public ledger). These computations require a high level of processing power, therefore, miners are rewarded for their contribution. For example, a data center in China, designed specifically for Bitcoin mining, demands up to 135 megawatts of power [31].

In Bitcoin, every block includes a SHA–256 cryptographic hash of the previous block. This links every block to its previous block and hence creates the whole blockchain. Thus, the blockchain (the public ledger) is kept consistent and immutable. Inside a Bitcoin transaction, the payer and payee accounts are inserted using their Bitcoin addresses (pseudonyms). The payer's and payee's addresses are called input and output addresses of a transaction, respectively. These Bitcoin addresses are created using a series of immutable cryptographic hash computations on the public key of users. There is a standard four–step procedure for this: (1) the user creates a random 256–bit private key, (2) the Elliptic Curve DSA algorithm is used to generate a 512–bit public key from the private key, (3) the public key is hashed down to 160 bits using the SHA–256 and RIPEMD hash algorithms, and (4) the resulting 160–bit is encoded in ASCII using Bitcoin's custom Base58Check encoding and published by the user as his/her Bitcoin address. Other users will use this 160–bit address to transfer bitcoins to his/her wallet.

In Bitcoin, users can create and publish as many Bitcoin addresses as they want for themselves. To spend their bitcoins, users must sign their transactions using their private key (generated in step 1). Therefore, the private key must be kept secret otherwise your bitcoins can be stolen and spent by a malicious user. A Bitcoin transaction is allowed to include multiple inputs and outputs addresses. In this case, the transaction must be signed with the private keys associated with every input address.

Bitcoin also mitigates the risk of *double spending* problem of digital currencies in which digital coins can be copied and paid again to a different payee [8]. In traditional payment systems, this is prevented by the central authority (i.e., a bank) that governs the currency. However, in Bitcoin, this is prevented by keeping users' transactions public (in the ledger or the blockchain) and also using the confirmation mechanism that miners follow, i.e., miners do not allow the second transaction to be added to the blockchain. This confirmation mechanism includes some consensus rules (or validation rules) that one of them prevents double spending. By following this rule, a miner ensures that every user can only spend his/her unspent coins that have been specified in the output fields of his/her previous transaction.

3.2 The DC–net Protocol

The Dining Cryptographers network (DC–net) [26], [34] is a decentralized protocol for anonymous communication. It enables a group of users to broadcast their messages in the group anonymously. The protocol was titled DC–net by Chaum [34] due to the example scenario that he presented in his paper to explain the protocol:

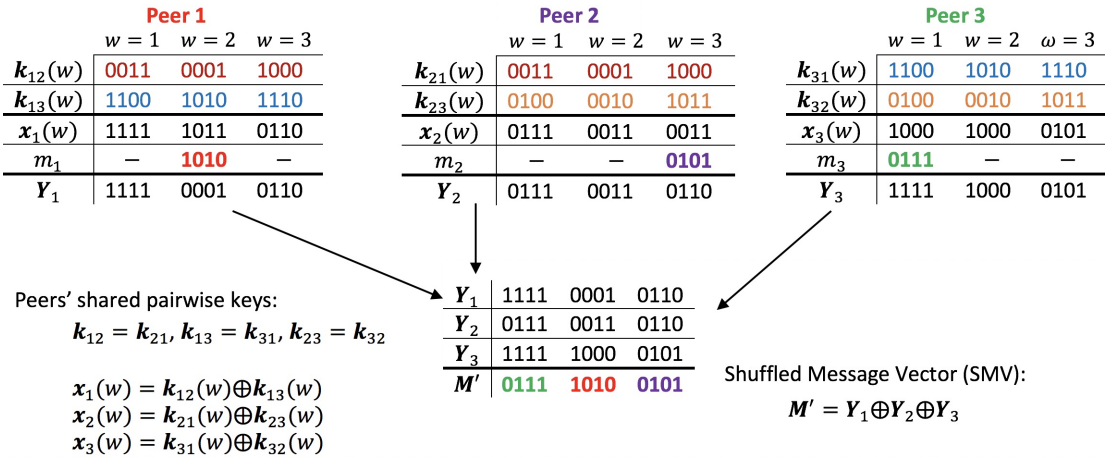

Fig. 2: Illustration of the DC–net protocol. Three peers want to anonymously publish their four–bit messages $\mathbf{m}_1$, $\mathbf{m}_2$, and $\mathbf{m}_3$. They first compute vector XK by XORing their shared pairwise keys. After every peer selects a random slot to insert his/her message, vector Y is computed and broadcast in the group by each peer. Finally, if there is no collision (i.e., each peer has selected a unique slot), the Shuffled Message Vector $\mathbf{M}^{'}$ (obtained by every peer) shows the messages $\mathbf{m}_1$, $\mathbf{m}_2$, and $\mathbf{m}_3$ that have been shuffled.

The bill of three cryptographers who are dining in a restaurant has been paid anonymously. It might be paid either by one of the three cryptographers or the office of National Security Agency (NSA). They discuss about a method to find out if their bill has been paid by NSA. However, if NSA has not paid the bill, the method should not reveal the identity of the cryptographer who has paid it (they respect their right of making an anonymous payment). Thus, they decide to run the following protocol:

Every cryptographer shares a pairwise secret bit with each one of the other two cryptographers. Therefore, the first cryptographer, let's say C_1, shares k_{12} and k_{13} bits with C_2 and C_3, respectively. Note that $k_{12} = k_{21}$, $k_{13} = k_{31}$, and $k_{23} = k_{32}$. In the next step, every cryptographer calculates the XOR of his keys (for example, C_1 obtains $k_{12} \oplus k_{13}$) and XORs the result with bit 1 if he has paid the bill. Otherwise, the result is XORed with bit 0. Then, every cryptographer reveals his own result to the group. Finally, they XOR the three announced bits. If NSA has paid the bill, the result of this XOR operation is 0.

In [26], the extended version of this protocol has been presented. It considers a group of N peers $P_1, P_2, P_3, \ldots, P_N$ who want to anonymously broadcast their L–bit messages $\mathbf{m}_i$ $(i = 1, 2, \ldots, N)$ in the group. Each pair of peers (P_i, P_j) generates and shares an L–bit key $\mathbf{k}_{ij}(w)$ such that $\mathbf{k}_{ij}(w) = \mathbf{k}_{ji}(w)$ for $i, j, w \in \{1, 2, \ldots, N\}$. In addition, every peer calculates the following vector which is called the *XORed Keys* (XK):

$$\mathbf{X}_i = [\mathbf{x}_i(1) \;\; \mathbf{x}_i(2) \;\; \mathbf{x}_i(3) \;\; \ldots \;\; \mathbf{x}_i(N)],$$

where

$$\mathbf{x}_i(w) = \oplus_{\substack{j=1 \\ j \neq i}}^{N} \mathbf{k}_i j(w), \quad w = 1, 2, \ldots, N. \tag{1}$$

In the next phase, the N peers perform the following steps to publish their messages in the group:

- Every peer P_i chooses a random position (slot) $s_i \in \{1, 2, \ldots, N\}$ in the $\mathbf{X}_i$ vector.
- Using the following equation, vector $\mathbf{Y}_i = [\mathbf{y}_i(1) \;\; \mathbf{y}_i(2) \;\; \mathbf{y}_i(3) \;\; \ldots \;\; \mathbf{y}_i(N)]$ is obtained and published in the group:

$$\mathbf{y}_i(w) = \begin{cases} \mathbf{x}_i(w), & w \neq s_i \\ \mathbf{m}_i \oplus \mathbf{x}_i(s_i), & w = s_i \end{cases}$$

Due to $\oplus_{i=1}^{N} \mathbf{x}_i(w) = 0$ for every w in $1, 2, \ldots, N$, if a unique slot has not been selected by more than one peer, we have:

$$\oplus_{i=1}^{N} \mathbf{Y}_i = \mathbf{M}^{'},$$

where $\mathbf{M}^{'}$ is the *Shuffled Messages Vector* (SMV), i.e. it is the vector of peers' messages $\mathbf{M} = [\mathbf{m}_1 \;\; \mathbf{m}_2 \;\; \mathbf{m}_3 \ldots \mathbf{m}_N]$ with shuffled elements.

Thus, the peers can broadcast their messages in the group while the sender of each message remains anonymous (see fig. 2).

3.3 Challenges of DC–net

Although DC–net guarantees users' anonymity, it suffers from the following critical problems [15–16], [35], [41–42]:
Collision possibility: Assume two peers P_i and P_j select the same slot of their XK vector, i.e. $s_i = s_j$. In this case, the peers recover $\mathbf{m}_i \oplus \mathbf{m}_j$ in the related slot of $\mathbf{M}^{'}$ and one of the $\mathbf{M}^{'}$'s elements is recovered as **0**). Thus, both messages $\mathbf{m}_i$ and $\mathbf{m}_j$ can not be recovered.
Security issues: The DC–net protocol is vulnerable against disruptions made by dishonest users. It needs all the peers to execute the protocol honestly. To disrupt the protocol, a dishonest peer can fill his $\mathbf{Y}_i$ vector with random irrelevant bits and broadcast it. In this case, all the peers' messages become unrecoverable. The other security issue of the DC–net protocol enables an adversary to deanonymize the sender of each message. We explain this using the following example:

Assume a group of four peers P_1, P_2, P_3, and P_4 want to perform the DC–net protocol. They have 5–bit messages to broadcast in the group. In the first three cycles of the

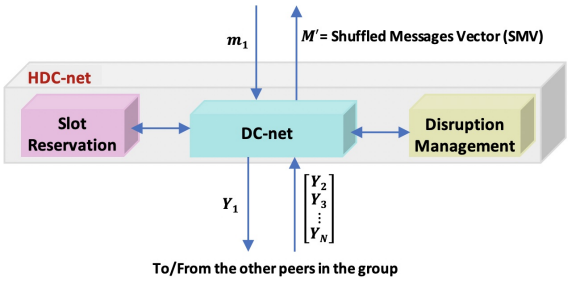

Fig. 3: The proposed system architecture for HDC–net. In this figure, peer P_1 is anonymously broadcasting $\mathbf{m}_1$ in a group of N peers.

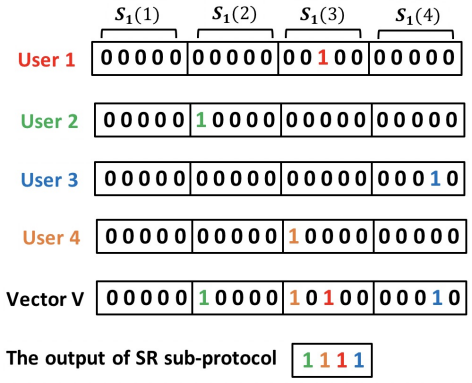

Fig. 4: An example that shows performance of the SR sub–protocol.

protocol (to publish their first three messages), every peer publishes $\mathbf{Y}_i^{(1)}, \mathbf{Y}_i^{(2)}$, and $\mathbf{Y}_i^{(3)}$ $(i = 1, 2, 3, 4)$.

Assume $\mathbf{X}_1 = [10100 \quad 11100 \quad 00111 \quad 10001]$ is the XK vector of peer P_1 who has selected slots 2, 4, and 1 for the first three cycles of message publishing. Suppose $\mathbf{m}_1^{(1)} = 10011, \mathbf{m}_1^{(2)} = 11001$, and $\mathbf{m}_1^{(3)} = 10101$ are his/her messages in the first three cycles. Therefore, for vector $\mathbf{Y}_1$ we have:

$$\mathbf{Y}_1^{(1)} = [10100 \quad 01111 \quad 00111 \quad 10001]$$
$$\mathbf{Y}_1^{(2)} = [10100 \quad 11100 \quad 00111 \quad 01000]$$
$$\mathbf{Y}_1^{(3)} = [00001 \quad 11100 \quad 00111 \quad 10001]$$

If an adversary carefully analyzes the above vectors, he can obtain the XK vector of peer P_1. There are two possible cases to consider:

- Assuming P_1 has selected a different slot in each cycle, the element of $\{\mathbf{y}_1^{(l)}(w)\}$ $(l = 1, 2, 3)$ that has been repeated at least twice is identified as $\mathbf{x}_1(w)$ (for every $w \in \{1, 2, 3, 4\}$). Using the obtained $\mathbf{X}_1$, the adversary can calculate $\mathbf{X}_1 \oplus \mathbf{Y}_1^{(j)}$ and identify P_1 as the sender of messages $\mathbf{m}_1^{(1)}$, $\mathbf{m}_1^{(2)}$, and $\mathbf{m}_1^{(3)}$.
- If P_1 has selected the same slot for at least two cycles, the elements of $\{\mathbf{y}_1^{(l)}(w)\}$ are totally different (provided that the peer's messages are not the same in each cycle). Thus, the adversary concludes that this is the slot which P_1 has selected during at least two cycles.

Therefore, using DC–net, users remain anonymous only for a limited time. In the next section, we propose a solution to enable users to regularly change their pairwise keys in a non–interactive way.

4 The Proposed HDC–net Protocol

Fig. 3 shows the proposed system architecture for HDC–net. We develop *Slot Reservation* and *Disruption Management* sub–protocols, and integrate them into DC–net to address the collision possibility and security challenges. In the following subsections, we describe the three phases of HDC–net, i.e. *Initialization*, *Scheduling*, and *Message Publishing*.

4.1 Initialization

In this phase, every peer (let's consider peer P_1 as an example) generates the matrix of pairwise symmetric keys

$$\mathbf{K}_1^{(0)} = \begin{bmatrix} \mathbf{K}_{12}^{(0)} \\ \mathbf{K}_{13}^{(0)} \\ \dots \\ \mathbf{K}_{1N}^{(0)} \end{bmatrix}, \text{where } \mathbf{K}_{1j}^{(0)} = [\mathbf{k}_{1j}^{(0)}(1) \ \ \mathbf{k}_{1j}^{(0)}(2) \ \dots \ \mathbf{k}_{1j}^{(0)}(N)].$$

In this vector, $\mathbf{k}_{1j}^{(0)}(w)$, $(w = 1, 2, \dots, N)$ is an L–bit symmetric key that is shared secretly by P_1 and P_j $(j = 2, 3, \dots, N)$.

In addition, P_1 signs $\mathbf{K}_{1j}^{(0)}$ (i.e. , row j of matrix $\mathbf{K}_1^{(0)}$) and send it to the other peers.These signatures are utilized by the disruption management sub–protocol to handle disruptions made by dishonest peers. We assume that peers use a decentralized public key infrastructure (DPKI) that works based on a decentralized identifier (DID) to generate and manage public/private keys and digital certificates. DIDs ([43], [44]) do not rely on centralized registration and certificate authorities to manage the keys and digital certificates. This makes them an effective PKI solution to handle the key generation and management in the UCoin protocol which is a decentralized peer–to–peer application. Note that the deployment of a DID has no impact on the de–anonymization of users. This is because the target of UCoin is to anonymize the payments of users not to make the users anonymous to each other. In other words, the users in an anonymity group know the identity of each other, but after making an aggregated transaction they are not able to identify the source of every individual payment.

4.2 Scheduling Phase

In this phase, the SR sub–protocol is performed by every peer P_i (see Algorithm 1):
(1) vector $\mathbf{S}_i = [\mathbf{S}_i(1) \quad \mathbf{S}_i(2) \dots \mathbf{S}_i(N)]$ is created by every peer P_i. In this vector, every element has L zero bits $(\mathbf{S}_i(w) = \mathbf{0}, w \in \{1, 2, \dots, N\})$.
(2) Every peer P_i selects two random integers $l \in [1, L]$ and $n \in [1, N]$. Then, the lth bit in $\mathbf{S}_i(n)$ is set to 1 to obtain $\mathbf{S}_i^{'}$.
(3) Every peer P_i computes and broadcasts the following vector:

$$\mathbf{Z}_i = [\mathbf{Z}_i(1) \quad \mathbf{Z}_i(2) \dots \mathbf{Z}_i(N)],$$

where $\mathbf{Z}_i(w) = [\oplus_{\substack{j=1 \\ j \neq i}}^{N} \mathbf{k}_{ij}^{(0)}(w)] \oplus \mathbf{S}_i^{'}(w)$, $w \in \{1, 2, \dots, N\}$.
(4) After $N - 1$ vector $\mathbf{Z}_j$ $(j = 1, 2, \dots, N, j \neq i)$ are received from the other $N - 1$ peers, P_i calculates vector $\mathbf{V} = [\mathbf{V}(1) \quad \mathbf{V}(2) \dots \mathbf{V}(N)]$, where $\mathbf{V}(w) = \oplus_{i=1}^{N} \mathbf{Z}_i(w)$. In the XOR operation, the terms associated with the pairwise

Algorithm 1: Slot Reservation Sub–Protocol

Input: $\mathbf{P} = [P_1\ P_2\ \dots\ P_N]$, $\mathbf{K}_i^0$, $\mathbf{R}_i = [r_{i1}\ r_{i2}\ \dots\ r_{iN}]$, L, $p = 0$
Output: $slt_i^{(p)}$
1: $\mathbf{S}_i = [\mathbf{S}_i(1)\ \mathbf{S}_i(2)\ \dots\ \mathbf{S}_i(N)] = \mathbf{0}$,
  where $|\mathbf{S}_i(w)| = L$ **for** $w = 1, 2, \dots, N$
2: $n = randi[1, N]$, $l = randi[1, L]$
3: $\mathbf{S}_i(n) = bitset(\mathbf{S}_i(n), l)$
4: $\mathbf{Z}_i = [\mathbf{Z}_i(1)\ \mathbf{Z}_i(2)\ \dots\ \mathbf{Z}_i(N)]$,
  where $\mathbf{Z}_i(w) = [\oplus_{\substack{j=1\\ j\neq i}}^{N} \mathbf{k}_{ij}^{(p)}(w)] \oplus \mathbf{S}_i(w)$ **for** $w = 1, 2, \dots, N$
5: **broadcast** $(\mathbf{Z}_i)$
6: **receive** $(\mathbf{Z}_j)$ **from** $P_j \in \mathbf{P}$, **for** $j = 1, 2, \dots, N,\ j \neq i$
7: $\mathbf{V} = [\mathbf{V}(1)\ \mathbf{V}(2)\ \dots\ \mathbf{V}(N)]$,
  where $\mathbf{V}(w) = \oplus_{j=1}^{N} \mathbf{Z}_j(w)$ **for** $w = 1, 2, \dots, N$
8: Consider $\mathbf{V}$ as a single bit–string with LN bits
9: $H = hamming(\mathbf{V})$
10: **if** $H = N$
11: $slt_i^{(p)} = 0$, $b = 1$
12: **while**$(b \leq (n-1)L + l)$
13: **if** $\mathbf{V}(b) = 1$
14: $slt_i^{(p)} = slt_i^{(p)} + 1$,
15: $b = b + 1$
16: **return**$(slt_i^{(p)})$
17: **else**
18: $\mathbf{K}_i^{(p+1)} = h(\mathbf{K}_i^{(p)})$, $p = p + 1$
20: **Jump** to line 1

keys are cancelled out. Thus, we have $\mathbf{V}(w) = \oplus_{i=1}^{N} \mathbf{S}_i^{'}(w)$. Therefore, if we consider vector $\mathbf{V}$ as a single bit stream, it shows all the 1s that the peers have already set in step 2 (see Fig. 4).
(5) Every peer P_i obtains the hamming weight (H) of $\mathbf{V}$ (it shows the number of 1s in $\mathbf{V}$). $H = N$ indicates that there is no collision, i.e. the peers have selected unique slots. Thus, P_i only keeps the 1s and deletes all the 0s of $\mathbf{V}$. In addition, the 1 associated to P_i is highlighted by this peer (Fig. 4). The result is an N–bit vector consisted of 1 only. In this vector, the position of the highlighted 1 is the slot that is assigned to peer P_i. However, $H \neq N$ means there is at least one collision, i.e. the same numbers l and n have been selected by two or more peers in step 2. In this scenario, the SR sub–protocol should be restarted. To perform this, every peer P_i updates his/her keys from $\mathbf{K}_i^{(0)}$ to $\mathbf{K}_i^{(1)}$. Note that for the next invoke of SR, the keys of every peer P_i should be updated. The update mechanism is explained in the next subsection.

Therefore, in the scheduling phase, vector $\mathbf{S}_i^{'}$ is considered as a single LN–bit vector where bits represent slots. Thus, there are LN available slots for the N peers to select. This reduces the probability of collision to a very small level since L is a large number (see section 6.4). Moreover, those peers that need higher communication rates can reserve $B > 1$ slots. For these peers, step 2 of SR should be repeated B times.

4.3 Message Publishing (MP) Phase

After the N peers have been allocated a unique slot, they can publish their messages in the group. To address the second security issue of the original DC–net protocol (see Section 3.3), we need to have unequal and distinctive elements in $\mathbf{Y}_i^{(p)}$ vector in every cycle of the protocol ($p = 0, 1, 2, \dots$). To do this, peers can renew their pairwise keys before they start a new cycle of message publishing, i.e. an updated set of keys is used to publish a new message.

Assume every peer P_i wants to broadcast message $\mathbf{m}_i^{(p)}$ in cycle p of the message publishing phase. To update the keys at cycle p, every peer P_i calculates the hash of the key that has been used in cycle $p-1$, i.e., $\mathbf{K}_i^{(p)} = h(\mathbf{K}_i^{(p-1)})$, $i = 1, 2, \dots, N$, where $h(.)$ is a one–way hash function that generates digests of size L. The result is $\mathbf{K}_i^{(p)}$ that is different from $\mathbf{K}_i^{(p-1)}$. Thus, every peer P_i applies a different set of keys in equation (1) that results in a different XK vector $\mathbf{X}_i$ in every cycle of message publishing. This makes the elements of $\mathbf{Y}_i^{(p)}$ and $\mathbf{Y}_i^{(p-1)}$ unequal.

Now we explain the MP phase. In cycle p of this phase, peer P_i broadcasts message $\mathbf{m}_i^{(p)}$ by calling the $MP(P, \mathbf{K}_i^{(p)}, \mathbf{m}_i^{(p)}, slt_i)$ algorithm. Vector P indicates the other $N-1$ peers and slt_i shows the slot number dedicated to P_i. This algorithm generates $\mathbf{Y}_i^{(p)}$ based on equation 2 and 3.

$$\mathbf{y}_i^{(p)}(w) = \oplus_{\substack{j=1\\ j\neq i}}^{N} \mathbf{k}_{ij}(w),\ for\ \ w = 1, 2, \dots, N, w \neq slt_i, \quad (2)$$

and

$$\mathbf{y}_i^{(p)}(slt_i) = \mathbf{m}_i^{(p)} \oplus (\oplus_{\substack{j=1\\ j\neq i}}^{N} \mathbf{k}_{ij}(slt_i)) \quad (3)$$

Finally, peer P_i can compute the SMV vector by XORing $\mathbf{Y}_i^{(p)}$ with the other $N-1$ vector $\mathbf{Y}_j^{(p)}$ ($j = 1, 2, \dots, N, j \neq i$) received from the other peers:

$$SMV^{(p)} = \oplus_{i=1}^{N} \mathbf{Y}_i^{(p)} \quad (4)$$

SMV contains all the peers' messages shuffled in such a way that the sender of each message remains unknown.

4.4 Disruption Management

Upon the calculation of the SMV vector, every peer P_i inspects it to make sure his/her message $\mathbf{m}_i$ (at slot slt_i) has not been corrupted, i.e., $\mathbf{m}_i \neq \mathbf{m}_i^{'}$ indicates a corruption, where $\mathbf{m}_i^{'}$ is the message extracted from slot slt_i of the calculated SMV vector. If a disruption is detected in cycle p, every peer P_i performs the following steps to handle the disruption:
(1) P_i announces in the group that his/her message in slot slt has been corrupted.
(2) After the announcement of P_i is received, other peers publish their keys associated with this slot, i.e., P_j ($j = 1, 2, \dots, N, j \neq i$) publishes $\{\mathbf{k}_{jl}^{(p)}(slt)\}_{\substack{l=1\\ l\neq j}}^{N}$.
(3) Every peer P_j examines the published keys of other peers to check if a peer (say P_l) has published a key different than their agreed pairwise key $\mathbf{k}_{jl}^{(p)}$.
(4) If in the previous step P_j detects that P_l has not published their agreed pairwise key, the identity of P_l is announced in the group as the disruptor. To prove this claim, P_j publishes the signature of P_l on the real $\mathbf{k}_{jl}^{(p)}$ and r_{jl}.
(5) $D_j = \oplus_{\substack{l=1\\ l\neq j}}^{N} \mathbf{k}_{jl}^{(p)}(slt)$ is computed by P_i for $j \neq i$.
(6) If $D_j \neq \mathbf{Y}_j^{(p)}(slt)$, the identity of P_j is published in the group as the disruptor.

After the disruptor's identity is announced in the group, other peers resume the MP phase without involving the disruptor. They also need to update their keys $\mathbf{K}_i^{(p)}$ by removing the row related to the disruptor.

5 The UCoin Framework

In this section, we present UCoin, an efficient coin mixing protocol for Bitcoin peers. It employs the proposed HDC–net protocol as a mixing module that enables users to anonymously mix their transactions perior to publishing. Before explaining the UCoin protocol, we present some assumptions related to the used trust and threat model.

5.1 Threat and Trust Model

As discussed before, UCoin is developed by applying the HDC–net mixing protocol to the Bitcoin setting. It ensures that the input and output accounts in a jointly created mixing transaction are unlinkable. Note that the links between input and output addresses in an aggregated transaction are unknown to not only an external observer (an adversary who monitors the transactions to deanonymize users), but also to the peers who have created that transaction, i.e., among all the shuffled output addresses, they are able to recognize only the output addresses that have been added by themselves. In other words, the only difference between the deanonimizing power of an adversary from outside the group and the peers in the group is that the peers can recognize the output addresses added by themselves. Thus, the anonymity set is smaller for them.

We assume that the N peers in the group communicate through a reliable broadcast communication protocol [37], [38]. Since our HDC–net protocol works based on the original DC–net protocol, it is inherently resistant to traffic analysis attacks [39]. The reason is that in the slot reservation and message publishing phase of the HDC–net, every peer P_i broadcasts vector Y_i only, in which the peer's message $\mathbf{m}_i$ has been encrypted by all the pairwise keys (all the pairwise keys are required to decrypt it). Thus, the peers can relaxedly broadcast their messages in the group.

It is assumed that after all the output addresses have been shuffled, one peer is randomly selected to create the final (aggregated) transaction and broadcast it to the Bitcoin network for verification. Moreover, to make the aggregated transaction valid, all the peers must sign it using their Bitcoin private key, otherwise the aggregated transaction is not verified by the miners.

5.2 Protocol Description

In the first step, every peer P_i calls the $MP(P, K_i^0, L, q, 0)$ sub–protocol, where $P = [P_1\ P_2\ ...\ P_N]$ is the set of peers, and K_i^0 is the matrix of pairwise symmetric keys. L is the length of the message which is 160 (bitcoin output address has 160 bits). q is the number of slots that the peer wants to reserve. Note that Bitcoin users usually require at least two output addresses because they usually have more bitcoins in their account than what they want to spend. Thus, the first output address is the payee's address and the other one is a change address that belongs to the original user (payer). Thus, the users' remaining balance is transferred to the second address. In this case q can be set to 2 since the peer has two messages (output addresses) for mixing.

The $slot\_reservation()$ sub–protocol returns slt_i, i.e., the slot reserved for this peer (i.e., P_i). If the peer calls the sub–protocol with $q > 1$, it returns a list of the reserved slots.

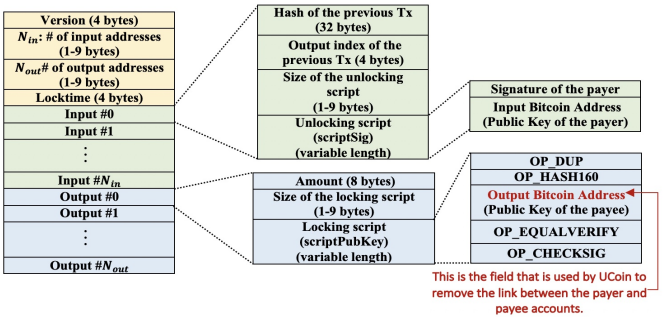

Fig. 5: The general format of an aggregated transaction based on the Bitcoin setting. The input fields of each individual transactions are copied into the input fields of the aggregated transactions. The output addresses of the individual transactions (that have been secretly shuffled by the UCoin protocol) are placed in the output fields of the aggregated transaction.

In the next step, P_i invokes the $MP(P, K_i^{(p)}, m_i^{(p)}, slt_i)$ algorithm with $p = 0$ where $K_i^{(p)}$ is the matrix of pairwise keys in round p, $m_i^{(p)}$ is the peer's message (i.e., the Bitcoin output address), and slt_i is the slot number assigned to P_i. p denotes the current call number of the *MP()* algorithm. It starts with 0 and increases by 1 every time the algorithm detects a disruption (the disruptor is excluded from the protocol and the *MP()* algorithm is invoked again with the updated keys). Note that if all the peers perform the protocol honestly, no disruption is detected and only a single run of the *MP()* algorithm is required. Thus, in this case, p remains at 0 and never increases.

Finally, the *MP()* algorithm returns a list of all the output addresses that have been shuffled. One peer is randomly selected to create the final (aggregated) transaction. Note that in Bitcoin, a transaction with multiple input addresses is successfully verified by miners only if it has been signed with all the private keys associated with the input addresses. Therefore, the aggregated transaction must be signed by all the peers. They can check whether the correct number of bitcoins is transferred to their associated output accounts before they sign the aggregated transaction. At the end, the selected peer broadcasts the aggregated transaction signed by all the peers into the Bitcoin peer–to–peer network (see Fig. 5).

In the UCoin application, the group size can change over time such that the peers can either remain in a group (if they have other transactions to publish) or leave it once they publish their transaction. In this case, the remaining peers can participate to mix their other transactions. New peers can also join the group to increase the size of the anonymity set.

There might be situations where the number of available peers in the group is not enough to achieve the required size for the anonymity set. To handle this challenge, the peers that join early can agree on a specific size and start the protocol as soon as an enough number of peers join the group. However, this solution might be an infeasible approach for users who need to make a quick payment. In the following, we propose an efficient and feasible solution to address this problem.

Assume there are N_1 available peers $P_1, P_2, \ldots, P_{N_1}$ in

the group where each peer P_i has n_i^o different output address in his/her transactions. If the peers agree on S_a as the required size for the anonymity set, the peer unavailability problem occurs when $S_a > \sum_{i=1}^{N_1} n_i^o$. In fact, the available peers need a set of λ additional output addresses to achieve the required anonymity set, where $\lambda = S_a - \sum_{i=1}^{N_1} n_i^o$. In this case, every peer p_i generates a random number and publishes it in the group. Then, the UCoin application sorts the published numbers in a descending way and selects λ peers who have published the largest numbers. Every selected peer should create a separate transaction (in addition to his/her main transaction) in which the peer transfers a specific amount of the cryptocurrency between his/her own accounts (i.e., it can be regarded as a dummy transaction). This approach is quite feasible since in cryptocurrencies, users usually create multiple addresses (accounts) for themselves. Thus, using this technique, the required λ output addresses are quickly provided, and the peers can start the UCoin protocol.

6 Performance Evaluation

In this section, we evaluate the performance of HDC–net. Moreover, we present the results of our prototype implementation to show that UCoin is a practical protocol.

6.1 Security Analysis

There are different security and privacy challenges that UCoin needs to address. The target of these threats can be either deanonymizing users or disrupting the protocol. We first examine collusion attacks that target to deanonymize an honest peer in the group.

Theorem. *A coalition of N_C peers results in the deanonymization of an honest peer with the probability of no more than $1/N_H$, where N_H is the number honest peers in the group.*

Proof. Consider N_C colluding peers $\{P_1^C, P_2^C, \ldots, P_{N_C}^C\}$ join a group that consisted of N_H honest peers to deanonymize an honest peer P_h. To do this, every colluding peer P_i^C has to participate in the HDC–net protocol by performing the following steps.

1) P_i^C generates the secret pairwise keys $\mathbf{K}_i^{(0)} = \{\mathbf{K}_{i,j}^{(0)}\}_{j\in[1,N],j\neq i}$, where $N = N_C + N_H$.
2) The colluding peers $\{P_i^C\}_{i\in[1,N_C]}$ share their secret pairwise keys $\mathbf{K}_i^{(0)}$with each other and compute $A_1(w) = \oplus_{i=1}^{N_C}\mathbf{k}_{i,h}^{(0)}(w)$ for $w = 1, 2, \ldots, N$, where $\mathbf{k}_{i,h}^{(0)}$ is the secret key shared between P_i^C and P_h.
3) The $slot\_reservation()$ algorithm is invoked by P_i^C to obtain a dedicated slot $slt_i^{(0)}$ of vector $\mathbf{Y}_i^{(0)}$(see equations 2 and 3).
4) The colluding peers $\{P_i^C\}_{i\in[1,N_C]}$ share their secret slots $\{slt_i^{(0)}\}_{i\in[1,N_C]}$ with each other.
5) The dummy messages $\{\mathbf{m}_i^C\}_{i\in[1,N_C]}$ are applied to vectors $\{\mathbf{Y}_i^{(0)}\}_{i\in[1,N_C]}$ using equations 2 and 3. The results are published in the whole group.
6) Upon the receipt of $\{\mathbf{Y}_i^{(0)}\}_{i\in[1,N]}$, each peer computes $SVM^{(0)} = \oplus_{i=1}^{N}\mathbf{Y}_i^{(0)}$ (i.e., the vector of shuffled messages).

The colluding peers start to deanonymize P_h. To do this, they need to (i) compute the XK vector of P_h (i.e., $\mathbf{X}_h^{(0)}$), (ii) compare it with the received $\mathbf{Y}_h^{(0)}$, (iii) identify the slot $slt_h^{(0)}$at which $\mathbf{Y}_h^{(0)}$ is different than $\mathbf{X}_h^{(0)}$, and (iv) obtain $\mathbf{m}_h^{(0)} = SVM^{(0)}(slt_h^{(0)})$. Using equation 1, the colluding peers start to build $\mathbf{X}_h^{(0)}$, i.e., $\mathbf{X}_h^{(0)} = A_1(w) \oplus A_2(w)$, where $A_2(w) = \oplus_{j=N_C+1}^{N}\mathbf{k}_{j,h}^{(0)}(w)$. Since the colluding peers have no knowledge of $\{\mathbf{k}_{j,h}^{(0)}\}_{j\in[N_C+1,N]}$, it is infeasible for them to compute $A_2(w)$. This forces them to make a guess on $slt_h^{(0)}$. There are $N = N_C + N_H$ slots in $SVM^{(0)}$. Among them, N_C slots $\{slt_i^{(0)}\}_{i\in[1,N_C]}$ are known to the colluding peers. Thus, they need to make a guess on a set of N_H slots which is successful with the probability of no more than $\frac{1}{N_H}$. By employing a group entry control mechanism (like the one proposed in [41]), we can prevent malicious users from setting up large size collusion groups.

Moreover, to further limit the chance of having a large N_c (i.e., a small N_H, equivalently), we suggest preventing users from joining any group that they like. Instead, they can first submit their request to the anonymity application. Then, the application randomly allocates every user to each group. Moreover, the approach proposed in Section 5.2 can be used to handle situations where the number of available peers in the group is not enough to achieve the required size for the anonymity set.

DoS attacks on SR sub–protocol: Suppose a malicious peer P_D reserves many slots by setting the majority (or all) of vector $\mathbf{Z}_D$'s bits to 1. This results in many collisions during the scheduling phase. According to our experimental results, using the SR sub–protocol, the maximum number of SR restarts is two, which can be considered as the threshold to determine the existence of a DoS attack. Thus, if the peers have to restart the SR sub–protocol more than this threshold, a DoS attack is detected.

When a DoS attack is detected during the scheduling phase, the DM sub–protocol is invoked which outputs the identity of disruptor(s). Then, after P_D is excluded from the list of peers, the honest peers can resume the protocol.

Passive Disruptions: In passive disruptions, a malicious peer P_D, pretending to become offline, leaves the group to prevent other peers from computing SMV. In this case, the remaining peers do not receive $\mathbf{Y}_D$ to compute SMV using Equation 4 since P_D no longer broadcasts $\mathbf{Y}_D$. In other words, P_D does not play his role in performing the protocol. However, the remaining peers can easily resume the protocol as soon as they detect P_D's departure. If the number of failures for a peer exceeds a threshold, his identity can be reported to the application provider on suspicion of passive disruption.

Active Disruptions: The target of an active disruptor P_D is to corrupt the peers' messages in SMV. To do this, he fills vector $\mathbf{Y}_D$ with random bits before publishing it. As discussed in Section 4.4, when an active disruption is detected by the peers in HDC–net, they perform the DM sub–protocol that outputs the identity of the disruptor. Then, the remaining honest peers can resume the MP phase after they removed P_D from the list of peers.

Revocation: In UCoin, the DM sub–protocol is invoked whenever a disruption attack is detected. It identifies the disruptor and broadcasts his identity to other peers who can resume the protocol by revoking the malicious peer

from the protocol. This prevents an adversary from jamming the protocol. Therefore, assuming there are at least three honest peers in the group, UCoin successfully terminates for every honest peer at the end with the minimum possible anonymity set 3.

If all the peers are honest, it only needs two rounds of the HDC–net protocol to mix the output addresses, i.e., one round for slot reservation and another for message publishing. In case of d dishonest peers, it needs at most $2+d$ rounds. In the worst case, each dishonest peer disrupts a separate round of the protocol. In this case, we need to perform an additional round of the message publishing phase of the protocol per dishonest peer. However, if two or more dishonest peers disrupt the same round of the protocol, it needs less than $2+d$ rounds to mix the output addresses.

Accountability: HDC–net is developed based on the original DC–net protocol. Hence, it is inherently resistant to traffic analysis attacks [39]. In other words, regarding a group of N peers, it is not feasible for an adversary (from either inside or outside the group) to determine the sender of a specific Bitcoin output address by exploring the traffic of the peers in the network. This enables the peers to explicitly broadcast their protocol messages in the group using the employed broadcast communication protocol [37], [38]. Thus, the honest peers in the group are able to obtain the identifying network information of malicious peers (e.g., IP address) and use it to exclude such peers from the future coin mixtures and make them accountable for their behavior in the network.

6.2 Anonymity Analysis

Considering a group of N honest peers $P = \{P_1, P_2, \ldots, P_N\}$, the size of the anonymity set provided by UCoin is equal to $\sum_{i=1}^{N} n_i^o$ where n_i^o is the number of Bitcoin output addresses sent by peer P_i. A larger anonymity set results in a higher level of anonymity because an attacker who is conducting a deanonymization attack on peer p_v has to guess an address among a larger set of addresses. As discussed before, Bitcoin users usually place at least two output addresses. The first address is the Bitcoin address of the payee and the second one is a change address belonging to the payer to receive the remaining balance. Thus, for $N = 50$ and assuming every peer sends at least two output addresses (i.e., $n_i^o = 2$ for $i = 1, 2, \ldots, N$), an adversary can successfully link a specific peer to any of the shuffled output addresses with a probability no more than 0.01.

The above logic can also be used for an adversary from inside the group. Note that in this case, the size of the anonymity set for the adversary A is $\sum_{\substack{i=1 \\ i \neq A}}^{N} n_i^o$ because he recognizes his own Bitcoin output addresses among all the output addresses. In case of facing a deanonymization attack by D different attacker, i.e. $\Delta = \{A_1, A_2, \ldots, A_D\}$, the anonymity set is limited to $\sum_{\substack{i=1 \\ i \notin \Delta}}^{N} n_i^o$.

6.3 Practical Considerations

In this section, we present some evaluations on the practicability of UCoin and discuss why it is a practical protocol.

Full Compatibility: UCoin does not require any modification in the Bitcoin protocol or in the standard format of a Bitcoin transaction and can immediately be deployed without any change to the Bitcoin protocol. The reason is that Bitcoin supports transactions with multiple input and output addresses. In fact, the Bitcoin protocol is used only in the last step of the UCoin protocol to create the aggregated transaction which is valid according to the Bitcoin protocol and can be decoded and interpreted for verification by Bitcoin miners. Moreover, since our HDC–net protocol is not specific to the Bitcoin setting, it can be used for other cryptocurrencies as well. In other words, the peers invoke the MP() algorithm inputting it with their intended Bitcoin addresses. Then, MP() mixes these addresses of all the participating peers and outputs the resulting SVM to every peer. MP() can be invoked by inputting it with a different message (i.e., it does not need to be a Bitcoin address necessarily). Hence, HDC–net can be employed to provide anonymity for other cryptocurrencies.

Cost Efficiency: While in the centralized mixing services [20–22], users have to pay a mixing fee to the service provider [15], [16], in UCoin, they do not need to pay any mixing fee. The reason is that UCoin is performed by the users jointly, thus, there is no need for a trusted third–party mix server. However, the participating peers are charged with the transaction fee for the single aggregated transaction according to the standard transaction fee in Bitcoin [40] that is shared between them.

Latency: To investigate the effect of number of peers (N) on latency, we consider the peer with the slowest device. The reason is that in UCoin (which is a distributed protocol), the required computation and communication procedures are performed by the peers in parallel. In fact, the other devices can not proceed the protocol until the slowest device performs the relevant computation procedures and publishes the result in the group. Therefore, for the latency we have

$$T = t_{max}^{init}(N) + t_{max}^{SR}(N) + t_{max}^{MP}(N) + \sum_{i=1}^{C} t_{max,i}^{Com}(N),$$

where t_{max}^{init}, t_{max}^{SR}, and t_{max}^{MP} are maximum latency of the initialization, slot reservation, and message publishing phases of the protocol, respectively, that are caused by the slowest device. $t_{max,i}^{Com}$ denotes the latency related to the ith communication and C is the number of communications required to obtain the aggregated transaction. Note that C will be 3, if (1) the slot reservation sub–protocol is finalized in a single round, and (2) all the output addresses of the peers are XORed with a single XK vector. Because the size of XK vector is a function of N, t_{max}^{init} may increase with N. In fact, the maximum possible size of the XK vector is $\sum_{i=1}^{N} LB_i$, where L and B_i are the length and number of output addresses that peer P_i wants to mix. Assuming $B_i = B \forall i \in \{1, 2, \ldots, N\}$, the maximum size of XK becomes NLB. If the peers connect to the internet with an average speed of R bps, we have

$$t_{max}^{MP} = \frac{NLB}{R}$$

For example, for N=1000, B=2 , R=100 Mbps, and considering the Bitcoin setup (i.e., L=160 bit), t_{max}^{MP} is 3.2 msec.

Regarding t_{max}^{init}, t_{max}^{SR}, and t_{max}^{MP}, a similar approach can be adopted to obtain their relationship with N (see "Computation Complexity" for more information).

Computation Complexity: Regardless of the digital signatures that are a part of the Bitcoin protocol, UCoin mostly requires XOR operations. This makes it an efficient protocol in terms of computation overhead that can be run on the systems with limited computational resources.

Considering N honest peers in a group, it only requires single invocations of *slot_reservation()* and *MP()* sub–protocols for a peer to obtain the shuffled messaged vector SMV. The *slot_reservation()* sub–protocol itself requires $N+1$ and N XOR computations in steps 4 and 7, respectively (see algorithm 1). The *MP()* sub–protocol requires $(N-1)$ and $(N-1)+1$ XOR computations for equations (2) and (3), respectively. Thus, it requires $2N-1$ XOR computations. Since N is in range 10 to 50 for a bitcoin mixing protocol [15–19], UCoin is quite efficient in terms of computation complexity (see Section 6.5 for the results of protocol implementation).

Communication overhead: In UCoin, the peers in a group anonymously mix the Bitcoin addresses (output addresses) that belong to their payees. In the Bitcoin protocol, users have 160–bit addresses. Thus, in UCoin, every peer in the group invokes the HDC–net protocol with 160–bit messages as input. We assume there are at most 50 peers in a group which is quite realistic (Ruffing et al. [16] have also assumed that there are at most 50 peers in a group while Ibrahim has considered 25 peers in a group in SecureCoin [19]). In this case, the maximum possible size of the XK vector is obtained as $\sum_{i=1}^{N} LB_i$ where B_i is the number of output addresses that peer P_i wants to be mixed, i.e., the number of slots that P_i wants to reserve. Considering $N = 50$, $L = 160$ bit (size of a Bitcoin address), and $B_i = 2$ for $i = 1, 2, \ldots, N$, the maximum size of the XK vector (that every peer broadcasts in the group) is 2 KB that is quite practical considering the level of storage and network resources that are currently available to users. This makes UCoin very efficient in terms of communication overhead.

6.4 Collision Analysis

In this subsection, we formally calculate the probability of collision in the SR sub-protocol of HDC–net and provide the relevant analysis to ensure that the collision possibility issue of the original DC–net protocol is effectively addressed. We first assume that every peer wishes to reserve a single slot during the slot reservation procedure (i.e., $B = 1$). In this case, if $\bar{\rho}$ is the probability of no collision and N users want to select a single slot from NL slots, based on the theory of Birthday Paradox problem, we have:

$$\begin{aligned}\bar{\rho} &= (\frac{NL}{NL})(\frac{NL-1}{NL})(\frac{NL-2}{NL})\ldots(\frac{NL-(N-1)}{NL}) \\ &= \frac{N!\binom{NL}{N}}{(NL)^N} = \frac{(NL)!}{(NL-N)!(NL)^N}\end{aligned}$$

This is because the first user has NL slots to select one from, while the second user has $NL-1$, the third user has $NL-2$, etc. We can further simplify the obtained formula by re–writing the above equation as

$$\bar{\rho} = (1-\frac{1}{NL})(1-\frac{2}{NL})\ldots(1-\frac{(N-1)}{NL})$$

in which we can replace the $(1-\frac{i}{NL})$ terms with $e^{-\frac{i}{NL}}$ since $e^x \approx (1+x)$ for $x \ll 1$. In this case, we have

$$\bar{\rho} \approx e^{-\frac{1}{NL}}.e^{-\frac{2}{NL}}\ldots e^{-\frac{(N-1)}{NL}} = e^{-\frac{(N-1)}{2L}}$$

Finally, for the probability of collision we have

$$\rho = 1-\bar{\rho} \approx e^{-\frac{(N-1)}{2L}}$$

If we assume the typical values of $N = 50$ and $L = 160$ bit (the size of a Bitcoin address), ρ will have an approximate value of 0.15 which is still unacceptable. To further decrease the collision probability, we propose to use a larger value for L during the slot reservation phase. For example, the selection of $N = 50$ and $L = 3200$ results in $\rho = 0.008$ which is quite negligible. In this case, the peers publish vectors of size $N = 20$ KB during the slot reservation phase. Note that this phase is run by the peers once only. Thus, the increase of L does not cause a significant effect on the communication overhead. To obtain a minimum value for L in order to achieve $\rho < 0.01$, we have $1 - e^{-\frac{(N-1)}{2L}} < 0.01$ which results in $L > 50(N-1)$. Thus, for $N = 50$, L should be greater than 2450 to achieve $\rho < 0.01$.

6.5 Implementation

We developed a prototype implementation of HDC–net protocol to evaluate its deployment in the UCoin protocol. The implementation is written in *Python* and uses OpenSSL 1.1.0 for elliptic curve DSA signatures and PKI operations. We used the Deterlab [28] infrastructure to test the prototype under realistic network conditions. Deterlab provides a shared testbed for distributed protocols and enables us to easily change network topology and node configurations.

Setup: The testbed topology that we used in Deterlab consists of three 100 Mbps LANs with 10 ms latency between the core switches and clients. The three LANs are connected together using 10 Mbps links with 20 ms latency. We executed the protocol for 5 to 50 clients. We assume no more than 50 peers are collaborating in the group. As we discussed in Section 6.3, this assumption is quite realistic. The other coin mixing protocols have considered the same assumption. For example, Ruffing et al. [16] have also assumed that there are at most 50 peers in a group while Ibrahim has considered 25 peers in a group in SecureCoin [19]. Two types of client machines were used for the experiments: 3GHz Intel Xeon Dual Core with 2GB of RAM and 2.13 GHz Intel Xeon CPU X3210 Quad Core with 4GB of RAM.

Collisions: Fig. 6 and 7 show the probability of collision for different values of the scheduling overhead efficiency factor B. The results show that collisions are more likely to occur for larger values of B that shows a sensible trade–off between collision probability and the efficiency factor B. Note that the values of collision probability shown in the figure have been obtained based on only a single run of the scheduling phase. However, we noticed that even for larger values of B, 100% of the slots are successfully allocated in at most two runs of the scheduling phase. We adopted $L = 3200$ that results in Z_i of maximum size 20 KB (for every peer P_i) which is quite practical. Better results are obtained for the probability of collision if a larger L is adopted. This is because it provides more bits in vector S_i by

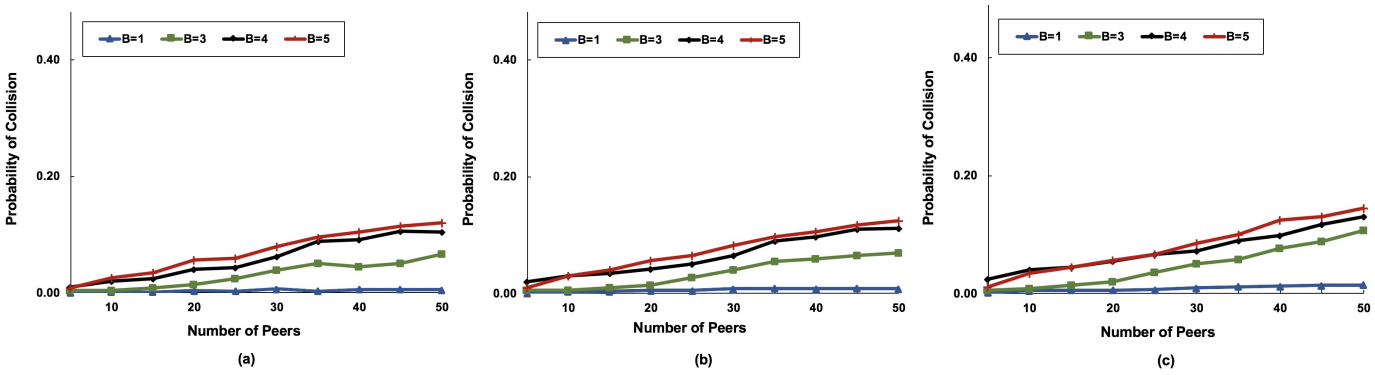

Fig. 6: Collision probability for different values of B and message sizes of (a) $L = 3200$ (b) $L = 2880$ (c) $L = 2400$.

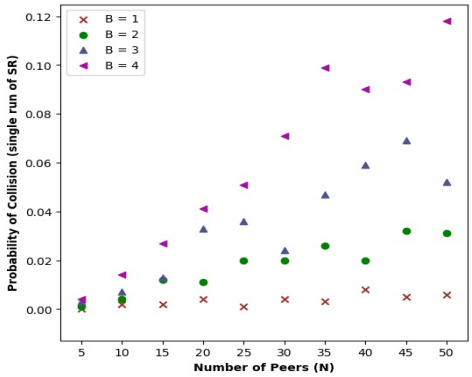

Fig. 7: Collision probability for a single run of the Slot Reservation sub–protocol for different valuse of the scheduling overhead efficiency factor B

which peers can reserve slots in the scheduling phase. Thus, lower collision rate is offered at the cost of publishing larger Z_i vectors. Fig 5, shows the effect of message size L on the probability of collision for different values of the scheduling overhead efficiency factor B. It shows that collisions are more likely to occur for the smaller message sizes since in this case, the number of available slots to select is smaller.

Latency: Fig. 8–a shows the average time required to perform the three phases of HDC–net, i.e. initialization, slot reservation and message publishing. In this figure, the shown results are for the scenario in which the clients publish messages of length 160 bit (the size of a Bitcoin address) since in UCoin the peers in a group mix the Bitcoin addresses by invoking the HDC–net protocol. The figure shows that large values of N result in larger XK vectors that make the system slower. Note that the illustrated time required for performing the message publishing phase includes the time needed for a single run of the SR sub–protocol. For example, considering $N = 50$, it takes 0.88 sec for the clients to anonymously publish a 160–bit message (a Bitcoin address) in the group. 0.49 sec of this time is spent on the slot reservation phase. Note that performing the slot reservation phase takes a bit longer than the message publishing phase. The reason is that we consider a larger value for the message length L in the scheduling phase to avoid collisions (larger than 160 bit that is used in the message publishing phase) as discussed in this section.

The end–to–end latency of HDC–net and some of the most well–known anonymity protocols are compared in Fig. 8–b. As you see, HDC–net outperforms Dissent [42] and Verdict [35] protocols in terms of speed. The reason is that the slot reservation and message publishing phases in HDC–net are performed using simple and lightweight operations, i.e. XOR and SUM. However, in Dissent and Verdict, heavy–duty tasks are performed for public–key encryption/decryptions and zero–knowledge proofs. Note that the latency of HDC–net will be shorter if we have $B > 1$ because in this case only a single run of the SR is required for B consecutive cycles of the message publishing.

The average time required to reserve B slots in a single run of the SR sub–protocol has been shown in fig. 8–c for different values of B. As the figure indicates, for $B = 1$ and $B = 3$ only a single run of SR is required to reserve B slots. However, some collisions have occurred for $B = 5$ and $B = 7$ in larger values of N. As the figure shows, in these situations, the SR sub–protocol has been restarted that resulted in a larger latency. Finally, in Fig. 9, the end–to–end latency of UCoin and DiceMix [16], the state–of–the–art coin mixing protocol are compared. As seen in the figure, UCoin outperforms DiceMix in terms of speed. In fact, UCoin requires three phases to work while DiceMix is performed in four different rounds that require some heavy–duty operations. For example, in round three, every peer needs to solve and compute the power sums. In addition, in DiceMix, the peers in a group commit to their DC–net vector using a hash function while in UCoin no heavy–duty task is required and the address shuffling is performed using simple and lightweight operations. Moreover, the slot reservation phase has been inspired and designed using the original DC–net protocol which makes it simple, lightweight, and fast.

In the solutions that work based on zero–knowledge proofs [10], [14], the anonymization task is performed using a two–step procedure. In the first step, users must generate a secure cryptographic commitment, i.e., the coins, which needs to be recorded in the blockchain in order to be validated by other users. Then, to spend their coins, users must publish a zero–knowledge proof for the relevant coins. This two–step mechanism makes these solutions inefficient in terms of latency [4], [6], [8], [19]. Specifically, the double–discrete logarithm proofs result in long verification times. For example, in Zerocash [10], the average time required to verify the commitments is in the rage of a few hundred

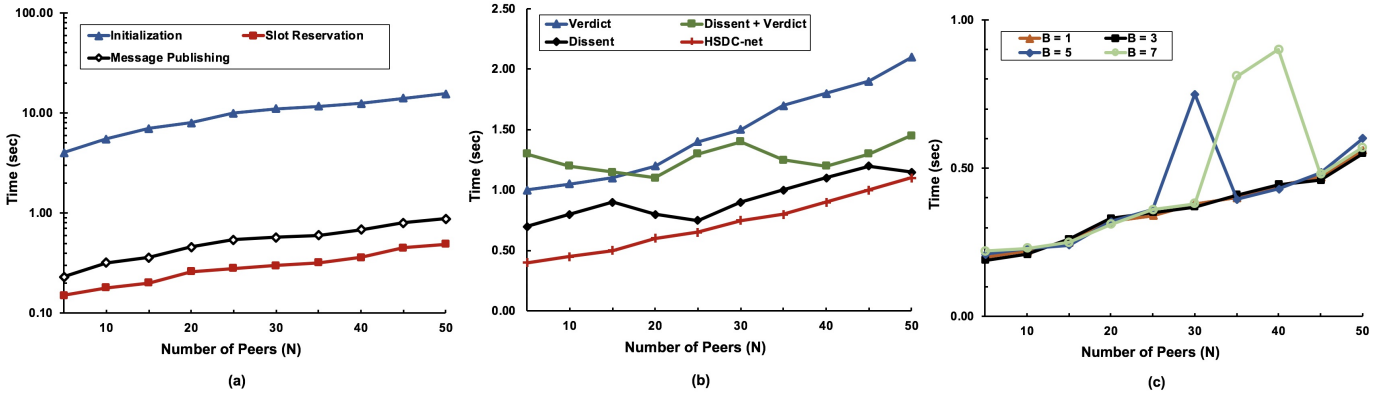

Fig. 8: (a) The average time required to initialize the HDC–net protocol, schedule a slot, and execute one cycle of message publishing (b) Comparison of HDC–net and some well–known anonymity schemes in terms of latency(c) The average time needed to schedule B slots.

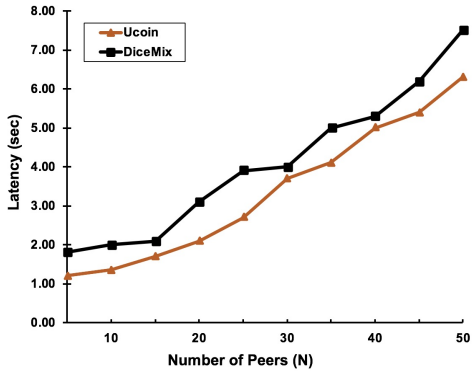

Fig. 9: Comparison of UCoin and DiceMix protocol in terms of latency

seconds while in UCoin, an aggregated transaction can be generated in a few seconds. Note that although in UCoin a group of peers need to participate in the protocol, the required computation and communication procedures are performed by the peers in parallel. This reduces the overall latency which is an advantage of distributed protocols in which the burden of protocol is shared between a group of users.

7 Summary and Future Work

This paper proposed Unlinkable Coin (UCoin), a secure and efficient mixing approach to protect cryptocurrencies against deanonymization attacks. It enables users to make anonymous payments by generating unlinkable transactions. To develop the UCoin protocol, we first proposed HDC–net a decentralized mixing protocol that enables a group of peers to anonymously mix their messages. Then, we added HDC–net as an extension to the Bitcoin architecture to provide anonymous Bitcoin payments. The main strengths of the proposed scheme are: (1) no dependency on a trusted third–party entity, (2) fully compatible with architecture of the current cryptocurrencies, and (3) simpler and faster than the current solutions. Our prototype implementation shows that UCoin is a practical protocol that provides anonymity for Bitcoin users.

An interesting direction for future work is to apply our HDC–net mixing protocol to other applications in which lack of anonymity is a concern. This results in developing different applications that provide anonymity as a feature for their users.

References

[1] CryptoCurrency Market Capitalizations. [Online]. Available: https://coinmarketcap.com. Accessed: 5–Jun–2019.
[2] J. Camenisch, S. Hohenberger, and A. Lysyanskaya, "Compact e–cash", in *EUROCRYPT '05*.
[3] T. Sander and A. Ta–Shma, "Auditable, anonymous electronic cash", in *CRYPTO '99*.
[4] M. Conti S. Kumar E, C. Lal, and S. Ruj, "A Survey on Security and Privacy Issues of Bitcoin", in *IEEE Communications Surveys & Tutorial* , vol. 20, no. 4, pp. 3416–3452, 2018.
[5] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cyptography and privacy–preserving smart contracts", in *IEEE S&P*, 2016.
[6] F. Tschorsch and B. Scheuermann, "Bitcoin and Beyond: A Technical Survey on Decentralized Digital Currencies", in *IEEE Communications Surveys & Tutorial*, vol. 18, no. 3, pp. 2084–2123, 2016.
[7] H. Halpin and M. Piekarska, "Introduction to security and privacy on the blockchain", *2017 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, 2017.
[8] M. C. K. Khalilov and A. Levi, "A Survey on Anonymity and Privacy in Bitcoin–like Digital Cash Systems", in *IEEE Communications Surveys & Tutorial*, vol. 20, no. 3, pp. 2543–2585, 2018.
[9] C. Lin, D. He, X. Huang, X. Xie and K. K. R. Choo, "PPChain: A Privacy–Preserving Permissioned Blockchain Architecture for Cryptocurrency and Other Regulated Applications," in *IEEE Systems Journal*, doi: 10.1109/JSYST.2020.3019923.
[10] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized Anonymous Payments from Bitcoin", in *S&P*, 2014.
[11] C. Lin, D. He, X. Huang and K. -K. R. Choo, "OBFP: Optimized Blockchain-Based Fair Payment for Outsourcing Computations in Cloud Computing," in *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 3241–3253, 2021, doi: 10.1109/TIFS.2021.3073818.
[12] D. Ron and A. Shamir, "Quantitative analysis of the full Bitcoin transaction graph", in *Financial Cryptography and Data Security*, 2013.
[13] N. van Saberhagen, "CryptoNote v 2.0 Whitepaper", Oct. 2013. [Online]. Available: https://cryptonote.org/whitepaper.pdf. Accessed: 5–Jun–2019.
[14] I. Miers, C. Garman, M. Green, and A. D. Rubin, "Zerocoin: Anonymous Distributed E–Cash from Bitcoin", in *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, pp. 397–411, 2013.
[15] T. Ruffing, P. Moreno–Sanchez, and A. Kate, "CoinShuffle: Practical decentralized coin mixing for Bitcoin", *Computer Security ESORICS 2014*, Lecture Notes in Computer Science, Springer, vol. 8713, pp. 345–364, 2014.

[16] T. Ruffing, P. Moreno–Sanchez, and A. Kate, "P2P Mixing and Unlinkable Bitcoin Transactions", In *24th Annual Network and Distributed System Security Symposium, NDSS*, 2017.
[17] J. H. Ziegeldorf, F. Grossmann, M. Henze, N. Inden, and K. Wehrle, "CoinParty: Secure Multi–Party Mixing of Bitcoins", in *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy–CODASPY '15*, pp. 75–86, 2015.
[18] J. H. Ziegeldorf, R. Matzutt, M. Henze, F. Grossmann, and K. Wehrle, "Secure and anonymous decentralized Bitcoin mixing", *Future Generation Computer Systems*, 2016.
[19] M. H. Ibrahim, "SecureCoin: A Robust Secure and Efficient Protocol for Anonymous Bitcoin Ecosystem", *International Journal of Network Security*, vol.19, no.2, pp. 295–312, Mar. 2017.
[20] Bitcoin Wiki, https://en:bitcoin:it/wiki/Category: Mixing Services.
[21] J. Bonneau, A. Narayanan, A. Miller, J. Clark, J. A. Kroll, and E. W. Felten, "Mixcoin: Anonymity for Bitcoin with accountable mixes", *Financial Cryptography and Data Security*, Lecture Notes in Computer Science, Springer, vol. 8437, pp. 486–504, 2014.
[22] L. Valenta and B. Rowan, "Blindcoin: Blinded, accountable mixes for Bitcoin", *Financial Cryptography and Data Security*, Lecture Notes in Computer Science, Springer, vol. 8976, pp. 112–126, 2015.
[23] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage, "A fistful of bitcoins: characterizing payments among men with no names", In *Proceedings of the 2013 conference on Internet measurement*, pp. 127–140. ACM, 2013.
[24] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The Second–Generation Onion Router", In *Proceedings of the 13th USENIX Security Symposium*, USENIX Association, pp. 303–320, 2004.
[25] D. Goldschlag, M. Reed, and P. Syverson, "Onion Routing for Anonymous and Private Internet Connections", *Communications of the ACM*, vol. 42, issue 2 , pp.39–41, 1999.
[26] P. Golle and A. Juels, "Dining Cryptographers Revisited", In *Advances in Cryptology – EUROCRYPT 2004*, Springer, Berlin, pp. 456–473, 2004.
[27] S. Nakamoto, "Bitcoin: A Peer–to–Peer Electronic Cash System", 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf.
[28] www.isi.deterlab.net/
[29] C. P. Schnorr, "Efficient signature generation for smart cards", *Journal of Cryptology*, vol. 4, no. 3, pp. 239–252, 1991.
[30] E. Ben–Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, "SNARKs for C: Verifying program executions succinctly and in Zero knowledge", *Advances in Cryptology – CRYPTO 2013*, Lecture Notes in Computer Science Springer, vol. 8043, pp. 90–108, 2013.
[31] https://www.energymatters.com.au/renewable-news/mining-bitcoin-solar-em5737/
[32] C. Lin, D. He, X. Huang, M. K. Khan and K. R. Choo, "DCAP: A Secure and Efficient Decentralized Conditional Anonymous Payment System Based on Blockchain," in *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 2440–2452, 2020, doi: 10.1109/TIFS.2020.2969565.
[33] G. Fuchsbauer, "Automorphic signatures in bilinear groups and an application to round–optimal blind signatures", *IACR Cryptology*, ePrint Archive, 2009:320, 2009.
[34] D. L. Chaum, "The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability", *Journal of Cryptology*, vol. 1, pp. 65–75, 1988.
[35] H. Corrigan–Gibbs, D. Isaac Wolinsky, and B. Ford, "Proactively Accountable Anonymous Messaging in Verdict", In *Proceedings of the 22nd USENIX Security Symposium*, pp. 147–162, 2013.
[36] E. Syta, H. Corrigan–Gibbs, S. C. Weng, D. Wolinsky, B. Ford, and A. Johnson, "Security Analysis of Accountable Anonymity in Dissent", *ACM Transactions on Information and System Security (TISSEC)*, vol. 17, issue 1, pp. 1–35, 2014.
[37] P. M. Melliar–Smith, L. E. Moser, and V. Agrawala, "Broadcast protocols for distributed systems", *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, no. 1, pp. 17–25 , 1990.
[38] B. Bellur and R. G. Ogier, "A reliable, efficient topology broadcast protocol for dynamic networks", *IEEE INFOCOM '99*.
[39] S. Angel, and S. Setty, "Unobservable Communication Over Fully Untrusted Infrastructure", In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation*, pp. 551–569, 2016.
[40] http://bitcoinfees.com/
[41] S. Goel, M. Robson, M. Polte, and E. Gun Sirer, "Herbivore: A Scalable and Efficient Protocol for Anonymous Communication", *Technical Report. Cornell University*, Ithaca NY 14850, USA, 2003.
[42] H. Corrigan–Gibbs, B. Ford, "Dissent: Accountable Anonymous Group Messaging", In *Proceedings of the 17th ACM conference on Computer and communications security (ACM CCS' 10)*, pp. 340–350, 2010.
[43] Decentralized Identifiers (DIDs) v1.0. *World Wide Web Consortium.* Accessed: 27-06-2021. Available:https://www.w3.org/TR/did-core/Overview.html.
[44] C. Farmer, S. Pick and A. Hill, "Decentralized identifiers for peer–to–peer service discovery," *2021 IFIP Networking Conference (IFIP Networking)* 2021, doi:10.23919/IFIPNetworking52078.2021.9472201.