SHARVOT: secret SHARe-based VOTing on the blockchain

Silvia Bartolucci
nChain
London, UK
silvia@ncrypt.com

Pauline Bernat
nChain
London, UK
pauline@ncrypt.com

Daniel Joseph
nChain
London, UK
daniel@ncrypt.com

ABSTRACT

Recently, there has been a growing interest in using online technologies to design protocols for secure electronic voting. The main challenges include vote privacy and anonymity, ballot irrevocability and transparency throughout the vote counting process. The introduction of the blockchain as a basis for cryptocurrency protocols, provides for the exploitation of the immutability and transparency properties of these distributed ledgers.

In this paper, we discuss possible uses of the blockchain technology to implement a secure and fair voting system. In particular, we introduce a secret share-based voting system on the blockchain, the so-called SHARVOT protocol[1]. Our solution uses Shamir's Secret Sharing to enable on-chain, *i.e.* within the transactions script, votes submission and winning candidate determination. The protocol is also using a shuffling technique, *Circle Shuffle*, to de-link voters from their submissions.

KEYWORDS

Voting, secret sharing, asymmetric encryption, blockchain network.

1 INTRODUCTION

A blockchain is a public ledger, managed by a peer-to-peer network, offering security, data immutability and transparency to its users [1]. The advent of blockchain technology via the cryptocurrency Bitcoin [2] has paved the way for the exploitation of the appealing ledger's properties for the most diverse applications. From healthcare to finance, blockchain-based solutions are created, using digital identification and smart contracts.

In the sector of electronic voting and rating, numerous voting platforms and startups focusing on building election systems on the blockchain have emerged and attracted high attention and funding [2]. Indeed, features such as *irrevocability* and *transparency* of information stored on chain make the technology ideally suited for the development of secure voting systems.

Decentralisation is one of the biggest challenges. Usually existing solutions rely on a central authority to validate and correctly count votes or feedback. The central authority may use different cryptographic primitives, such as blind signatures and homomorphic encryption, to make the users' submissions anonymous and verifiable [3]. In Chaum's protocol [4], participants send their votes to mixing authorities that reshuffle and encrypt the votes before broadcasting them, in order to disguise the link between votes and voters. Among the existing decentralised cryptographic solutions, *Secure Multi-Party Computations* protocols enable a set of users to compute a function of their joint private inputs (which they want

to keep secret) without the need of a trusted third party, allowing the parties to jointly compute the average of their ratings privately and securely [5].

Focusing on how to preserve votes secrecy while guaranteeing integrity of the process is also fundamental. In [6], participants submit binary votes $v_i \in \{0, 1\}$ masked by adding a conveniently created random variable together with a proof that the vote is either 0 or 1. Despite the votes being concealed, one can verify the vote format and prevent double submissions at the price of using computationally expensive zero-knowledge proofs [7].

In this paper, we present solutions using the blockchain technology to announce and build an election and determine a winning candidate. In particular, we focus on how to broadcast the votes within the transactions script and correctly count them while (i) protecting voters privacy and anonymity (without the need of zero-knowledge proofs), (ii) allowing only eligible users to cast their preference, (iii) preventing attacks aimed at invalidating the ballot.

The paper is organised as follows. In Sec. 2, we describe the main challenges faced when building a secure and fair voting protocol. In Sec. 3, we present the *SHARVOT* protocol, a blockchain-based solution using the technology presented in Sec. 2. Sec. 4, 5 are devoted to discussion and conclusive remarks.

2 E-VOTING: BUILDING BLOCKS

In this section, we describe how to exploit the features of the ledger and blockchain transactions to cast and correctly count ballots. We discuss possible ways to store the ballot in-script and the vote format respectively in Sec. 2.1 and 2.2. In Sec. 2.4 and Sec. 2.5 we describe how to prevent submission from non-eligible voters and how to protect vote's secrecy.

2.1 Vote recording and data storage

Data storage on the blockchain is controversial. A conspicuous fraction of developers believe that data may overload the full nodes by increasing (i) disk storage costs and (ii) memory of the *unspendable transaction output* (UTXO) set. Indeed, some applications create pseudo-payments to use the recipient's Bitcoin address as a 20-bytes field, hence creating unspendable UTXO. According to the Bitcoin protocol, data can instead be stored in script using the opcode *OP_RETURN < data >*, which allows for the storage of 80 bytes of non-payments related data [1, 8].

An alternative solution exploits the pay-to-script-hash (P2SH) script[3] of a *m-of-n* multisignature transaction. According to these scripts, n public keys P_i, $i = 1, \ldots, n$ are recorded, and in order to redeem the script, at least $m \leq n$ signatures S_j, $j = 1, \ldots, m$,

---

[3]P2SH scripts allow a user to lock UTXOs in a transaction in such a way that the recipient must provide data to complete the script, therefore unlocking the UTXOs sent to the Bitcoin address derived from the P2SH script.

(corresponding to any subset of n public keys) must be provided [1]. The m-of-n multisignature script is of the format:

$$OP\_0\, S_1 \ldots S_m < m\, P_1 \ldots P_n\, n\, OP\_CHECKMULTISIG > . \quad (1)$$

The space devoted to public keys can be used to store metadata. Indeed, a multisignature address can be of the form q-of-n with $q \geq 1$, where the output can be spent by providing the signatures associated to the q public keys while the remaining $n - q$ public key slots are used to encode data of a maximum size of 64 bytes each. This eliminates the need for an additional unspendable UTXO and allows anyone to access and verify the metadata stored in the multisignature script. Consider for instance a *1-of-3* multisignature script, where two of the three data elements reserved for public keys, are used to store metadata instead. The script takes the following format:

$$OP\_0\, S_1 \; < OP\_1\, m_1\, m_2\, P_1\, OP\_3\, OP\_CHECKMULTISIG > , \quad (2)$$

where m_1, m_2 are the metadata stored in the script. In voting protocols, the metadata elements could correspond to the set of encrypted/shuffled votes $\{v_i\}, i = 1, \ldots, m$, of m users. Note the caveat that, for the current version of the Bitcoin protocol, the maximum number of public keys allowed in a multisignature output is 15, hence limiting the maximum number of votes that can be stored per multisignature script.

2.2 Vote commitment format

In general, a vote can be submitted in different formats: as a string to indicate the name of a candidate, as a binary variable, an integer, etc. In this paper, we express the vote as a 64-bytes key, denoted by $k_{i,j}$, where the index i relates to the candidate selected by the user while j identifies a particular voter. We describe how to construct these keys in the next section.

A voter U_i wishing to vote for candidate C submits their key share $k_{C,i}$. The key can also be concatenated with an identification value for candidate C indicated with IdC. As an example, a vote submission may take the following form: $v_i = k_{(C,i)}||\text{IdC}$ where $||$ represents strings concatenation. The vote v_i can also be encrypted with the public key Pk_C of the candidate before being embedded in the script, *e.g.* $\hat{v_i} = Enc_C(v_i)$. In the event that the candidate C attempts to decrypt the vote, the component IdC serves as an indicator to the candidate that the decryption was successful.

2.3 Voting using Shamir's Secret Sharing

Secret Sharing schemes are examples of t-of-n threshold cryptosystems, whereby a secret k is divided among n participants, such that at least $t + 1$ participants are required to collaborate to reconstruct k. The knowledge of any t pieces of the secret k leaves the latter undetermined.

Shamir's secret sharing [9] is based on *Lagrange Polynomial Interpolation* and the secret k is assumed to be an element of a finite field F of size p (p is a prime number). The scheme comprises a dealer (dealer-less versions also exist) and a set of n participants $U_1, U_2, \ldots, U_n$.

In the presence of a dealer, the latter chooses an arbitrary t-degree polynomial $f(x) = \sum_{j=0}^{t} a_j x^j$ and defines the secret as $k = f(0) = a_0$. The dealer then computes n points $p_i = (x_i, f(x_i)), i = 1, \ldots, n$, and distributes the point p_i to the participant U_i, i.e. each

participant receives their share $k_i = f(x_i)$ of the secret. Using the *Lagrange Polynomial Interpolation*, if $t + 1$ out of the n participants collaborate, they can reconstruct any point on $f(x)$ with their shares of the key k.

In Sec. 3 we show how key shares can be used to express a preference and determine the winner in an election process.

2.4 Eligibility

E-voting systems are often targeted by *Sybil attacks*, where a single user attempts to gain access to multiple identities and modify the outcome of the election. In particular, on the blockchain users can create multiple pseudonyms, *i.e.* public keys. To prevent this kind of attacks, one could (i) introduce a fee per vote, which would discourage an attacker from disrupting the process (the attack might be less cost-effective); (ii) introduce an identification system to allow only registered voters to submit their ballot.

In the second case, a personal secret key $k_i, i = 1, \ldots, n$, may be given to a set of n endorsed users by a trusted authority, *e.g.* a company announcing an internal election, a website etc. Endorsed voters use their private key to encrypt their vote. The list of keys is kept private by the verifier who later validates the submission: only allowed users will have their vote correctly decrypted and counted.

A possible submission of vote v_i is of the form $v_i \oplus k_i$, where $\oplus$ represents the XOR of the two strings. Since $v_i \oplus k_i \oplus k_i = v_i$, the verifier decrypts the vote by XOR-ing the string with the key of the users k_i. The decryption would only be available to users whose key is known to the verifier.

Alternatively, the verifier can create a pair of public/private key per user, (Pk_i, Sk_i), and communicate to the eligible user the public key only. The user can encrypt their ballot using Pk_i. The vote submission, of the form $Enc_{Pk_i}(v_i)$, is sent to the verifier who decrypts it using the private key Sk_i, $Dec_{Sk_i}(Enc_{Pk_i}(v_i)) = v_i$ [4].

2.5 De-linking users and submissions

Research on de-anonymisation techniques shows that, despite the use of pseudonyms in cryptocurrencies networks such as Bitcoin, users may be exposed to attacks aimed at uncovering real identities [10, 11]. To address this vulnerability, a variety of *coin mixing* solutions have been developed to enhance the untraceability of coins flows and the unlinkability of transactions with the coins owners [1]. These solutions typically disguise the links between users' addresses by mixing the individual input and output of users in one single transaction, so-called *CoinJoin* transaction [12].

We developed a new decentralised solution, *Circle Shuffle*, for randomizing the position of the output addresses that compose a *CoinJoin* transaction. This protocol is designed in such a way that no participant signing the transaction knows which of the output addresses corresponds to a specific Bitcoin input address - therefore addressing a limitation present in the *CoinShuffle* protocol, where at least one party in the construction of the transaction has knowledge of the input and output address of another party [13]. In the context of voting protocols, shuffling techniques are useful to de-link users from their votes, as we describe in Sec. 3.

---

[4]The functions Dec_{Sk_i}, Enc_{Pk_i} vary depending on the cryptosystem used. For instance, in the RSA [7], given a public key $(n, e) = k$, the private key d and the plaintext to encrypt x, $Enc_k(x) = x^e \bmod n$ and $Dec_d(Enc_k(x)) = (x^e \bmod n)^d$.
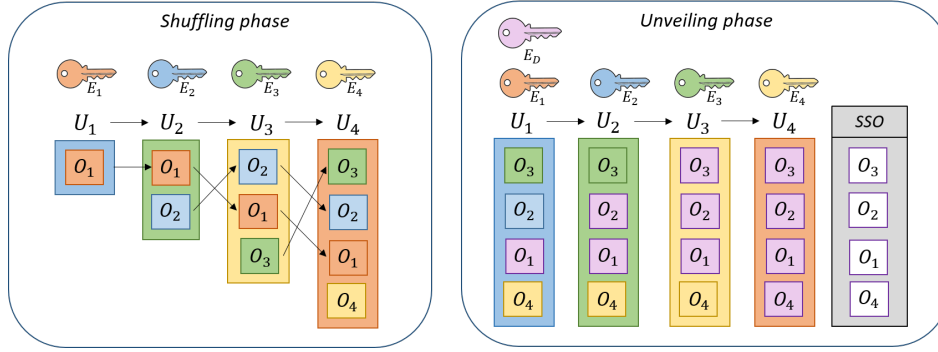
Figure 1: *Circle Shuffle* **technique: (left) Shuffling of the encrypted output addresses. (right) Unveiling of the shuffled set of output addresses.** O_i **and** $E_i, i = 1, \ldots, 4$**, represent, respectively, the output and the public key used to encrypt it of the** *i*-**th participant.**

2.5.1 Initialisation. A random sequence S of n participants $U_1, U_2, \ldots, U_n$ is decided. Each participant U_i has a corresponding public/private key pair: (E_i, k_i) where $E_i = k_i \times G$ (G is the generator of the Elliptic Curve chosen in the protocol [7]). The public keys $E_1, E_2, \ldots, E_n$ are made available to all participants. Additionally, the first participant publishes the public key of an ephemeral pair: (E_D, k_D).

2.5.2 Shuffling Phase. The shuffling phase begins when the first participant, U_1, encrypts their output address O_1 with their public key E_1. The encrypted output address then comprises the 'Set of Shuffled Outputs' (*SSO*).

U_1 encrypts the *SSO* with E_2, the public key of the next participant in S, U_2, and forwards the encrypted *SSO* to U_2. U_2 then decrypts the *SSO* using k_2, encrypts their output address O_2 using E_2, and adds the newly encrypted address to the *SSO*. U_2 shuffles the order of the encrypted output addresses within the *SSO*, encrypts the shuffled *SSO* with E_3, and then forwards the encrypted *SSO* to the third participant U_3. This process continues until it reaches U_n. This phase is illustrated in Fig.1 (left) for a set of $n = 4$ participants.

Once the encrypted output address of the last participant is added to the *SSO*, U_n performs a final shuffle, encrypts the *SSO* with E_1, and sends the *SSO* back to the first participant, U_1.

2.5.3 Unveiling Phase. At this point, the *SSO* is once again sent to each of the n participants. The ephemeral public key E_D, generated by the first participant U_1, is used to ensure that no participant gains information about the others during the protocol directly.

At the end of the first encryption-shuffle loop, the first participant U_1 is in possession of the set of shuffled outputs. U_1 decrypts the *SSO*, searches for their encrypted output address in the *SSO* and, when found, decrypts the address using the associated private key k_1. The first participant then encrypts O_1 with E_D. U_1 then encrypts the new *SSO* with the public key of the second participant, P_2, and forwards the encrypted set to U_2. The second participant in turns decrypts the *SSO* using k_2, finds their output address, decrypts it using k_2 and re-encrypts it with E_D. U_2 then encrypts the new *SSO* with the public key P_3 and forwards the encrypted set to U_3. The process continues until each participant has found their encrypted

output address in the *SSO* and replaced it with the corresponding decrypted (and encrypted with E_D) value, see Fig. 1 (right).

The last participant encrypts the *SSO* with E_1 (or E_D) and sends the encrypted *SSO* to the first participant U_1. The first participant decrypts the *SSO*, and each encrypted address contained within it, using the ephemeral private key k_D. The permutation of the output addresses in the *SSO* at that point represents the final order in which the outputs are included in a *CoinJoin* transaction.

3 THE SHARVOT PROTOCOL

Consider n voters $U_1, \ldots, U_n$, wishing to express their preference between several candidates. For simplicity, we will assume a choice between two candidates, A and B. The *SHARVOT* protocol consists in a voting solution utilising the Bitcoin blockchain; the protocol is based on the Shamir's secret sharing scheme and a shuffling technique, the *Circle Shuffle*.

Private key shares (based on the scheme presented in Sec. 2.3) are assigned to the voters who commit fees in a n inputs - 1 output transaction that stores, and therefore permanently records, the votes. Each input corresponds to the fees paid by a voter. The winning candidate must receive both a majority of votes and a number of votes past a specified threshold to collect the fees.

The *SHARVOT* protocol is designed with a failsafe such that any Bitcoins committed by the voters is recoverable if no candidate collects enough key shares to spend the UTXO in the transaction signed by the voters. A dealer-based key share distribution scheme is expected to be the most popular implementation of the *SHARVOT* protocol, as having a dealer assume the role of an authority control over the list of eligible voters.

Keys Generation. A public/private key pair is assigned by the dealer to each of the two candidates, A and B, respectively $P_A = k_A \times G$ and $P_B = k_B \times G$. The candidates also possess a public/private key pair each, $M_A = s_A \times G$ and $M_B = s_B \times G$ respectively, and publish their public key. The dealer also publishes the public keys P_A and P_B while keeping k_A and k_B private. For each secret, the dealer computes n key shares, $k_{A,i}, i = 1, \ldots, n$, and $k_{B,i}, i = 1, \ldots, n$, and distributes a pair of key shares $(k_{A,i}, k_{B,i})$ to each voter U_i. The key shares are obtained using the t-of-n threshold scheme presented in Sec. 2.3. As a result, the k_A and k_B values can only be determined
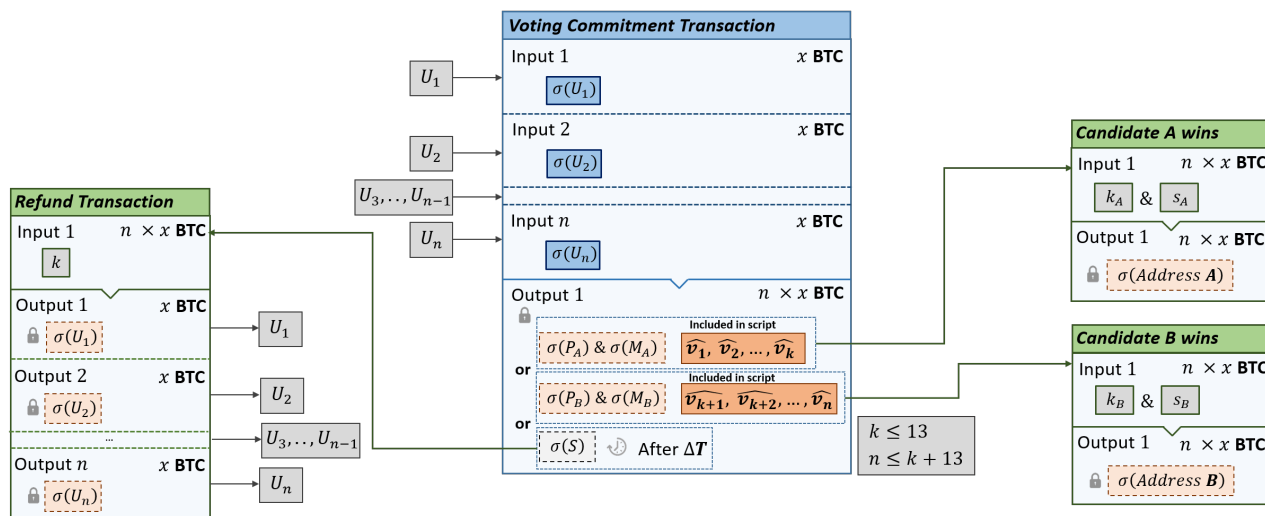
Figure 2: Transactions created in the *SHARVOT* protocol: the *Voting Commitment Transaction* casting the votes and spending the voters' fees, the transactions owned by each candidate, and the *RefundTransaction* returning the fees to the voters in case no candidate wins.

when voters submit a sufficient number of votes $t+1$ for a particular candidate.

Votes Submission. Each voter U_i encrypts their ballot for the candidate of their choice, as explained in Sec. 2.2. For instance, assume U_i gives their vote to a candidate B and therefore composes $\hat{v}_i = Enc_B(k_{B,i}||\text{IdB})$. The list of shuffled votes (obtained using the shuffling technique described in Sec. 2.5) is sent to the dealer.

P2SH address. The dealer, upon receiving the selected key shares from the n voters, generates a P2SH address. The maximum number of public keys allowed in a multisignature output is 15 (see Eq.1). Hence, 13 fields in the multisignature script can be used to store the vote of participants, while the two remaining fields can be used to store the public key given by a candidate (M_A or M_B) as well as the public key assigned to the latter by the dealer (P_A or P_B).

The P2SH script is therefore based on *if − else statements*, where each statement is a multisignature script incorporating up to 13 votes and the aforementioned keys of a candidate. A final statement, in the form of a *scriptPubKey* ($S = k \times G$), is added in case no candidate obtains a sufficient number of votes after a period of time ΔT.

Voting Commitment Transaction. The dealer then creates the *Voting Commitment Transaction* (VCT) which includes one output where $n \times x$ Bitcoins are sent to the P2SH address. The transaction, illustrated in Fig. 2, is sent to the voters who add their input (of x Bitcoins) and sign the *VCT*. The last participant to sign the *VCT* sends it to the dealer who will submit it for inclusion in the blockchain. Note that, while signing the *VCT*, the voters can verify in perfect anonymity the correctness of their vote included in the public record.

Refund Transaction. Before the *VCT* is actually broadcast on the network, the dealer creates a *Refund Transaction* (RT) that spends the output in the *VCT* and redistributes x Bitcoins to each voter.

The transaction is signed by the dealer using the private key k that unlocks the third option (although the latter includes a locking time preventing the UTXO to be spent before ΔT). The dealer sends the signed *RT* to the voters. Then and only then the *VCT* is submitted for inclusion in the blockchain.

Ideally, the secret k should be controlled by both the dealer and the voters so that under no circumstance the dealer can steal the voting fees. One could, for instance, add a 1-of-n multisignature in the script so that both the dealer's signature and at least one voter's signature is needed to spend the UTXO of the Voting Commitment Transaction.

Election result. In case one candidate successfully decrypts $t + 1$ or more key shares in the shuffled list included in the P2SH script, the candidate can reconstruct the secret key needed to unlock the $n \times x$ Bitcoins in the *VCT*. If no candidate obtains sufficient key shares, the voters can broadcast the *RT* and recover their fees.

4 DISCUSSION

In the *SHARVOT* protocol, transactions I/O shuffling and voting encryption techniques are implemented to guarantee the de-linking of the users from their vote submission, while conveniently constructed transactions ensure a permanent and immutable record of the ballot on the blockchain.

The dealer in *SHARVOT* certifies the voters' eligibility through the distribution of key shares. The concatenation of the votes with an identifier for each candidate and the encryption of the vote prevents the dealer (or the voters) from manipulating the vote. Nevertheless, this does not completely prevent from the submission of multiple and/or erroneous votes by the same participant(s) with the malicious aim to disrupt the voting protocol. Such behaviour is however discouraged by attaching a cost, *i.e.* the voting fee, to the voter's submission itself. Dealer-less versions exist although these imply further rounds of communication among the voters

and therefore would be more popular for an internal voting process with a limited number of participants. Finally, a natural limitation of the protocol arises from the size of the script used to generate the P2SH output address in the *VCT*.

5 CONCLUSION

In this paper, we presented the *SHARVOT* protocol that uses the blockchain technology to announce and build an election and determine a winning candidate by collecting voters' ballots in an immutable, storage-efficient and anonymous manner.

Indeed, we showed how the use of multisignatures in P2SH scripts allows for efficient storage of votes, which are encrypted using candidates identifiers and de-linked from the voters using shuffling techniques, and preserves the voters anonymity. Finally, the use of the blockchain guarantees data immutability and transparency.

We also discussed the vulnerabilities associated to the SHARVOT protocol. Note that this solution remains secure, transparent and completely anonymous, without the need for complex cryptographic methods such as zero-knowledge proofs.

REFERENCES

[1] Andreas M. Antonopoulos. 2014. *Mastering Bitcoin: unlocking digital cryptocurrencies* (1st. ed.). O'Reilly Media, Inc.
[2] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008).
[3] Robert Riemann and Stéphane Grumbach. 2017. Distributed Protocols at the Rescue for Trustworthy Online Voting. *ArXiv preprint* 1705.04480 (2017).
[4] David L. Chaum. 1981. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM* 24, 2 (1981), 84–90.
[5] Andrew C. Yao. 2015. Protocols for secure computations. *International Conference on Information and Communications Security,* (2015), 82–96.
[6] Zhichao Zhao and T-H. Hubert Chan. 2015. How to vote privately using bitcoin. *International Conference on Information and Communications Security,* (2015), 160–164.
[7] Jonathan Katz and Yehuda Lindell. 2014. *Introduction to modern cryptography.* CRC press.
[8] Massimo Bartoletti and Livio Pompianu. 2017. An analysis of Bitcoin OP_RETURN metadata. *ArXiv preprint* 1702.01024 (2017).
[9] Adi Shamir. 1979. How to share a secret. *Commun. ACM* 22, 11 (1979), 612–613.
[10] Dmitry Khovratovich Biryukov, Alex and Ivan Pustogarov. 2014. Deanonymisation of clients in Bitcoin P2P network. *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (2014), 15–29.
[11] QingChun ShenTu and JianPing Yu. 2015. Research on Anonymization and De-anonymization in the Bitcoin System. *ArXiv preprint* 1510.07782 (2015).
[12] Greg Maxwell. 2013. CoinJoin: Bitcoin privacy for the real world. *Post on Bitcoin Forum* (2013). Retrieved Feb 4, 2018 from https://bitcointalk.org/index.php?topic= 279249.0
[13] Pedro Moreno-Sanchez Ruffing, Tim and Aniket Kate. 2014. CoinShuffle: Practical decentralized coin mixing for Bitcoin. *European Symposium on Research in Computer Security* (2014), 345–364.