BlindCons: A Consensus Algorithm for Privacy Preserving Private Blockchains

No Author Given

No Institute Given

Abstract. Private blockchains are becoming gradually more sought-after in many organisations. Predominately because they allow organisations to overpass the traditional limitations of public blockchains in terms of energy consumption, efficiency, and control of the system. However, such systems are in some ways losing their blockchain essence, particularly their unique feature of not being controlled by a central authority that governs many functionalities of the chain including user enrolment. Our main contribution in this paper is to propose a user ID privacy preserving consensus algorithm for private blockchains, called BlindCons. The main security properties of our protocol guarantee: consistency of the data stored in the blockchain, liveness of the system, and users' privacy by blinding their signatures to keep the user identity private. BlindCons blind signature scheme ensures that it is not possible to link a transaction to a user using the data stored in the blockchain. Our consensus BlindCons uses Okamoto-Schnorr Blind Signature proposed in 1992 and a Byzantine Fault Tolerance (BFT) consensus for the blockchain replication process.

Keywords: blind signature, transactions unlinkability, privacy-preserving consensus, permissionned blockchain.

1 Introduction

Currently, blockchain is one of the most popular distributed systems. One of the main characteristics of blockchains is their resistance to malicious modifications. Such resistance is reached by using block timestamp and hash pointers that link the last block of the chain to the previous one. The blockchain design forces that any modification in one block compels the regeneration of the following blocks in the chain. A consensus algorithm controls the blockchains data storage process, ensuring that any update on the chain is valid and committed by all the network members.

Many groups of companies have proposed *Private Blockchains* or *Permissioned Blockchains*, like for instance the MyHealthMyData consortium [**?**], which propose to connect hospitals and research centers across Europe to share medical data through a private blockchain network. These blockchains aim to manage data storage on a few nodes controlled by a company or a consortium in order to validate the transactions fastest, consume less energy and reduce the cost of mining. Another example is the project PlasticTwist [**?**] that creates a new circular economy based on an ERC20 token implemented on the top a permissioned ledger to encourage plastic recycling in Europe.

Moreover, Maersk and IBM have developed TradeLens [**?**], a supply chain system supported by the permissioned blockchain Hyperledger Fabric.

Use cases, like the examples presented above, are proliferating due to the blockchain adoption in closed ecosystems and business applications. Due to this, the European Blockchain Observatory and Forum has published a technical report [**?**] where it recommends to use private or permissioned blockchains for storing sensitive data. Therefore, the absence of mechanisms to keep the user's privacy in a permissioned scheme is turning highly relevant.

Contributions: The main contribution of this paper is the design of a new privacy-preserving consensus algorithm for private blockchains. We aim to ensure the unlinkability between a transaction recorded in the blockchain and the user that generated it. With our algorithm, we increase the privacy level of such blockchains. For this purpose, we designed a privacy-preserving consensus called BlindCons. Our algorithm relies on Okamoto-Schnorr's blind signature scheme [**?**] to ensure transaction unlinkability. Moreover, we use a Byzantine Fault Tolerance (BFT) consensus algorithm for the generation and commitment of new blocks.

Related Work: Blockchain platforms such as Bitcoin [**?**] and Ethereum [**?**] are popular due to the "anonymity" that their cryptocurrencies offer. However, Bitcoin is still showing issues with transaction linkability and traceability. During the last years some improvements have been proposed to increase Bitcoin anonymity [**?**], and also new cryptocurrencies have been developed to overcome these issues like Zcash and Monero. In the case of Zcash [**?**], the protocol achieves anonymity by using ZK-SNARKs as cryptographic proof. In the same line, Bulletproofs [**?**] proposes a new NIZKP protocol with short proofs without relying on trusted setup for confidential transactions or privacy preserving smart contracts. On the other hand, Monero is a protocol based on Cryptonote [**?**] that achieves unlinkability and untraceability by using a one-time random address and ring signatures. Although all these protocols aim to improve transactions' privacy, they are designed for permissionless blockchains, where the transaction linkability differs from permssioned architecture where the users are not anonymous. Hence, there is a strong link between the transaction signature and the user that is triggering it.

A different approach to ensure anonymity is the eCash [**?**] [**?**] model proposed for Bitcoin in [**?**]. However, the idea to use a third party to generate a blind voucher is not aligned with the principle of decentralisation that Bitcoin or any other permissionless blockchains have.

Notwithstanding several works on blockchain privacy for permissionless ledgers have been published, as far as we know, there is no formal protocol that addresses the transaction linkability issue in permissioned ledgers

Outline: Section 2 recalls Permissioned Blockchains and Okamoto-Schnorr's blind signature scheme, then in Section 3 we introduce the BFT consensus mechanism. Section 4 describes our privacy-preserving consensus algorithm BlindCons. Before concluding, we explicit the security properties of our consensus algorithm in Section 5.

2 Preliminars

We start by introducing permissioned blockchains, and then we present a blind version of the Schnorr's signature scheme proposed by Okamoto et al. in [**?**].

2.1 Permissioned Blockchain

TODO

2.2 Okamoto et al. Signature

This scheme allows the user to obtain a blind signature of the message $M \in \{0,1\}^*$ issued by an authority (A).

Definition 1 (Okamoto-Schnorr Blind Signature [?]). *Let p and q be two large primes with $q|p-1$. Let G be a cyclic group of prime order q, and g and h be generators of G. Let $H : \{0,1\}^* \to \mathbb{Z}_q$ be a cryptographic hash function.*

Key Generation: *Let $(r,s) \xleftarrow{r} \mathbb{Z}_q$ and $y = g^r h^s$ be the A's private and public key, respectively.*

Blind signature protocol:

1. *A chooses $(t,u) \xleftarrow{r} \mathbb{Z}_q$, computes $a = g^t h^u$, and sends a to the user.*
2. *The user chooses $(\beta,\gamma,\delta) \xleftarrow{r} \mathbb{Z}_q$ and computes the blinded version of a as $\alpha = ag^{-\beta}h^{-\gamma}y^{\delta}$, and $\epsilon = H(M,\alpha)$. Then calculates $e = \epsilon - \delta \bmod q$, and sends e to the A.*
3. *A computes $S = u - es \bmod q$ and $R = t - er \bmod q$, sends (S,R) to the user.*
4. *The user calculates $\rho = R - \beta \bmod q$ and $\sigma = S - \gamma \bmod q$.*

Verification: *Given a message $M \in \{0,1\}^*$ and a signature (ρ,σ,ϵ), we have $\alpha = g^{\rho}h^{\sigma}y^{\epsilon} \bmod p$.*

The Okamoto-Schnorr blind signature scheme is suitable for private blockchain architecture due to the blinding process can be performed by the same authority responsible for the enrollment process (see Figure 1, where the authority A blinds the signature of the message proposed by the user). Therefore, by using this scheme, we ensure user-transaction unlinkability.

3 Byzantine Fault Tolerant based Consensus for Private Blockchain Architecture

The blockchain is a decentralised database organised in blocks that are appended one behind the other by using a hash pointer. Each block contains records that include the data to be stored into the chain. The blockchain design makes it suitable to be used as a distributed ledger, due to the record organisation inside a block (i.e., financial transactions) and the hash chain between blocks that makes it resistant to malicious modifications. The blockchain uses a consensus algorithm to replicate the chain in each node member of the network. Nevertheless, the selection or design of the consensus

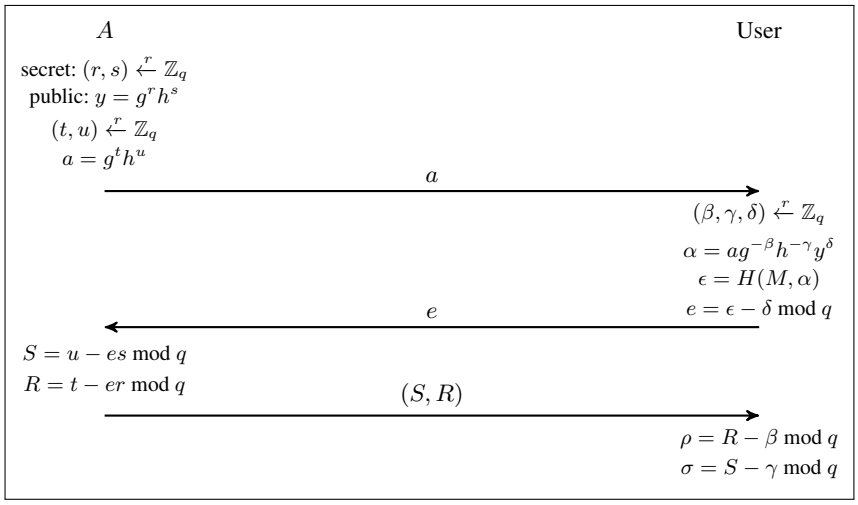

Fig. 1. Okamoto-Schnorr blind signature diagram, where $y \xleftarrow{r} \mathbb{Z}_q$ means that y is randomly chosen in $\mathbb{Z}_q$.

algorithm must be consistent with the blockchain architecture and the openness of the system.

In permissioned blockchain architectures, every user must be enrolled into the system through Certificate Authorities (CA) or user administrators before joining the network. These authorities are responsible for generating the user credential for each new client or peer. This model provides a base of knowledge of the network members, making PBFT [**?**] or BFT-Smart [**?**] based algorithms suitable for the blockchain replication process.

Our consensus approach for permissioned ledger has an opposite aim of the permissionless replication protocol. The security of the consensus algorithm in a permissionless architecture is achieved by proving the node's honesty by expending its resources (i.e., computing capacity, power consumption, stakes, among others). In contrast, a BFT-based protocols reach the consensus accepting as valid the result proposed by the nodes majority.

Our consensus algorithm is based on BFT and the *execute-order-validate* process used in Hyperledger Fabric [**?**] since version 1.0. In this protocol, each transaction passes through these three stages. The *execution* phase triggers the transaction and then it is validated by endorsement. In the *order* phase, the transactions are ordered according to the BFT consensus protocol. Finally, the **validation** phase checks the security policies per operation and application types.

In the case of our BFT-based consensus algorithms, we start with a client c_{bc} that has the same role than a client in the PBFT protocol. The client c_{bc} sends a transaction proposal with the instructions to be executed in the blockchain to the endorsing peers in order to validate the transaction. This process is analogous than the executed by the primary in the original PBFT algorithm, that sends the message with the instructions

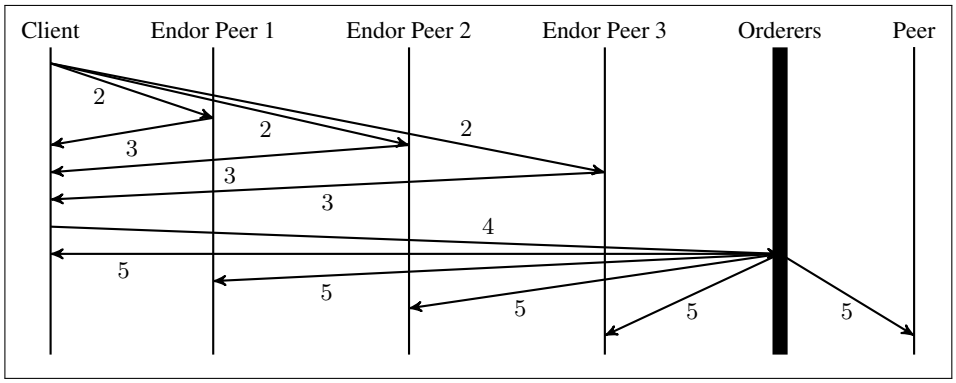

Fig. 2. BlindCons Transaction Flow.

to the backups to validate it. As for the transaction endorsement, the endorsing peers ep_{bc_i} send their responses to the client, which collects it after the validation process is concluded. If the transaction is valid, the client c_{bc} sends the transaction to the orderers $ordp_{bc_i}$ to add it into the new block that will be committed by the nodes members of the network. The orderes are responsible to propose and agreed the new block to be appended to the chain. This process is analogous to PBFT, where one orderer is the leader and the other are the backups. The orderer leader will collect a group of transactions and will organise them by order inside of a new block. Then the orderer leader sends the block proposal to the orderers backups to validate it. Each orderer backup will validate the proposal and responds to the rest of the orderer with a **prepare** message. Once the orderers have received enough valid **prepare** responses, they will submit a **commit** messages to the orderers, client and the nodes responsible to keep a local version of the blockchain.

To exemplify this process, consider that we have a client c_{bc} that needs to execute an operation o_{bc} on our BFT-based blockchain protocol. To process this request, we need to execute the following four steps: Transaction Proposal, Transaction Endorsement, Broadcasting to Consensus and Commitment (see Figure 2). Nevertheless, before we start describing the protocol, we introduce the notation used in this paper. In any distributed system, the information is passing through the peers or nodes within the network by using messages. Hence, we denote a message by using $< \cdot, ..., \cdot >$, where $\cdot$ corresponds to the message arguments. Moreover, we use the notation $(\cdot, ..., \cdot)$ to group arguments inside a message.

1) ***Transaction Proposal:*** The client c_{bc} generates a message to execute specific operations o_{bc} to be resolved by the network. This message corresponds to a transaction proposal message to be sent to the endorsing peer. The transaction proposal consists in invoking the operations o_{bc} and computing the state update and version dependency denoted by *stateUpdate* and *verDep*, respectively. The version dependencies relate the variables involve in the transaction with the local version of the variable in the client's

node and their respective operations. For example: if a client wants to read and write in the blockchain, the version dependency is a tuple $(readset, writeset)$ where:

- for every variable k read by the transaction, the pair $(k, s(k).version)$ is added to readset,
- for every variable k modified by the transaction, the pair $(k, s(k).version)$ is added to $writeset$.

Once the execution of the operations o_{bc} triggered by the client c_{bc} is completed, the client c_{bc} generates the *transaction proposal* message and then submits it to the endorsing peers. The transaction proposal message is defined as:

$<$ *PROPOSAL*, $trans_{prop}$ $>$, where:
- $trans_{prop}$:= (c_{bc}, o_{bc}, $txPayload$, $stateUpdate$, $verDep$, $retryFlag$, σ_c).
  - c_{bc}: is the client ID,
  - o_{bc}: refers to the operations implemented in the distributed system,
  - $txPayload$: is the payload of the submitted transaction,
  - $retryFlag$: boolean variable to identify whether to retry the transaction submission in case of the transaction fails,
  - σ_c: is the client signature.

2) ***Transaction Endorsement:*** The endorsing peers (ep_{bc_i}) verifies the client signature σ_c coming in transaction proposal message. If the client signature is valid, the endorser simulates the transaction proposal by executing the operation o_{bc} with the corresponding $txPayload$, and then validates that the $stateUpdate$ and $verDep$ are correct. If the validation process is successful, the endorsing peer generates a *transaction valid* message to be sent to the client c_{bc}. The message has the following structure:

$<$ *TRANSACTION-VALID*, tx_{id}, σ_{ep_i} $>$, where:
- tx_{id}: is the transaction identifier generated with the client ID and a nonce,
- σ_{ep_i}: is the signature of the message signed by the client c_{bc}.

In the case of the simulation process ending unsuccessfully, the endorsing peer generates a *transaction invalid* message:

$<$ *TRANSACTION-INVALID*, tx_{id}, $Error$, σ_{ep_i} $>$, where *Error* can be:
- *INCORRECT-STATE*: when the endorser obtains a different *state update* than the one coming in the *transaction proposal*,
- *INCORRECT-VERSION*: when there is the newest version of the variable referred in the *transaction proposal*,
- *REJECTED*: for any other reason.

3) ***Broadcasting to Consensus:*** The client c_{bc} waits then for the response from the endorsing peers. When it receives enough *Transaction Valid* messages adequately signed, the client stores the endorsing signatures into packaged called *endorsement*. Once the transaction is considered endorsed, the client invokes the consensus services by using *broadcast(blob)*, where *blob:=(trans$_{prop}$, endorsement)*.

The number of valid responses needed to consider that the *transaction proposal* is endorsed depends on the configuration of the permissioned blockchain. If the trans-

action has failed to collect enough endorsements, the client abandons this transaction proposal.

4) ***Commitment:*** Once the client c_{bc} broadcasts the transaction properly endorsed to consensus, the ordering services collect this transaction and organise it into a block. The ordering service is sorted in *views*, where each *view* represents how the orderders are organised for the consensus service. In each *view*, the orderer $ordp_{bc_{leader}}$ acts as the primary or leader, and the rest orderers $ordp_{bc_i}; i = 1, \cdots, n-1$ corresponds to the backups. In case, that the primary is being faulty or acting maliciously, the view will change and the next backup of the list will turn in the new leader. The consensus process begins when the orderer leader $ordp_{bc_{leader}}$ collects enough transactions to create a new block. The leader will propose this new block to the backups by sending a *propose* message. Each backup validates the *propose* and will respond to the other backups in the service with a message called *prepare*. Once each backup has received enough valid *prepare* responses, it will send a *commit* message to nodes connected to the network to confirm that the block has been validated successfully.

From the ordering services, we get a hash-chained sequence of blocks with the transactions endorsed. The block, prepared by the ordering services, has the form $B = ([tx_1, tx_2, ..., tx_k], h)$, where h corresponds to the hash value of the previous block.

The nodes receive the blocks from the ordering services through a direct connection or by using a gossip protocol. Once the peers have the new block, they check if the endorsement of each transaction is valid according to the policy configured in the network. The peers then verify the *verDep* in order to ensure there are no conflicts between the operation o_{bc}; the variables involved in the transaction and the current blockchain *state*. If this process finishes successfully, the transactions are committed. In the case that one of the validation fails, the peers consider the transaction as invalid, and it is dropped. Finally, the invalid transactions are informed to the client c_{bc}, and according to the *retryFlag*, the client may retry the transaction.

The nodes can change the local state only when the transactions are committed. Hence, each node connected to the network will update their local version of the ledger once all the transactions in the block are committed. Finally, the state update is done by appending the new block to their local version of the blockchain.

4 Privacy Preserving BFT for Permissioned Ledgers

Considering a permissioned PBFT based consensus protocol like the one introduced in Section 3, in this protocol, we use the digital signature as a user authentication method without protecting the user privacy. Hence, for a privacy-preserving consensus protocol, we need to have the following properties:

- Alice sends a newly signed transaction to the CA,
- Alice' signature is validated only by the CA,
- the CA signs the transaction proposed by Alice and anonymizes Alice's identity,
- all the nodes member of the transactions validation process can validate the CA's signature,
- the CA signature cannot be duplicated.

Now, to keep the privacy of the client and peers involved in the transactional process, we need to hide his ID and blind his signature. However, we do not address the ID hiding process with any particular mechanism. Therefore, we consider that the ID is replaced by a value corresponding to the anonymized user ID, and this process can be performed by using different schemes.

To address the issue related to the digital signature, we replace the signing mechanism used in the original protocol by the Okamoto-Schnorr blind signature scheme [**?**]. By recapitulating the consensus protocol above mentioned, the transactional process consists of the follows steps:

1. **Initiating Transactions:** The client c_{bc} generates a signed message to execute an operation o_{bc} in the network.
2. **Transaction Proposal:** The submitting peer sp_{bc} receives the message coming from the client c_{bc}, validates the client signature and proposes a blockchain transaction with the client instruction o_{bc}.
3. **Transaction Endorsement:** The endorsing peers ep_{bc} validate the client c_{bc} signature and verify that the transaction is correctly simulating the operation o_{bc} by using his local version of the blockchain. Then, each endorsing peer generates a signed transaction with the result of the validation process and sends it to the submitting peer sp_{bc}.
4. **Broadcasting to Consensus:** The submitting peer sp_{bc} collects the endorsements coming from the endorsing peers connected to the network. Once sp_{bc} collects enough valid answers from the endorsers, he broadcasts the transaction proposal with the endorsement to the ordering service.
5. **Commitment:** All the transactions are ordered within a block and are validated with their own endorsement. The new block is then spread through the network to be committed by the peers.

In order to maintain consistency and liveness, we keep the transactional flow. However, the steps are modified in order to accept the new blind signature scheme to authenticate the clients and the peers.

We define three new functions to represent the signing process and the operation execution inside the protocol. Let $BlindSign(M, (\beta, \sigma, \gamma), y)$ and *VerifyBlindSign(M,*$(\rho, \delta, \epsilon), y)$ be the functions to sign blind and to verify the blinded signature, respectively. Where M corresponds to the message to be signed, (β, σ, γ) to the secret values randomly chosen $(\xleftarrow{r} \mathbb{Z}_q)$ by the client or peer, (ρ, δ, ϵ) to the blinded signature; and y to the CA public key. The result obtained from the function *BlindSign* corresponds to the blinded signature (ρ, σ, ϵ). On the other hand, the function *VerifyBlindSign* returns a *valid* or *invalid* response. The third function corresponds to *EXEC(o,Payload)*, and represents the execution process of the operation o on the *Payload* performed by the peer. Additionally, we use the variables *tid* as the transaction ID, *SecurityPolicies* to define the set of security parameters configured in the node for the operation o_{bc} (i.e., read and write rights), and *TotEndorPeers* for the total number of endorsing peers in the network.

Now that the functions and the variables used in the protocol have been defined, we need to integrate it inside of each step of the transactional flow. The transactional process starts with the function **Initiating Transactions** described in Algorithm 1. In this

step, we replace the client ID by a random number, we then concatenate the arguments $(crand, o_{bc}, Payload, retryFlag)$ to be signed, and we use the function *BlindSign* to generate the blinded signature according to the protocol described in Section 2.2. Finally, we generate the message *SUBMIT* to be sent to the submitting peer sp_{bc}.

Algorithm 1 InitTx(o_{bc},$Payload$,$retryFlag$,y)

1: $crand_{bc} \xleftarrow{r} \mathbb{N}$
2: $M \leftarrow (crand||o_{bc}||Payload, retryFlag)$
3: $(\beta, \gamma, \delta) \xleftarrow{r} \mathbb{Z}_q$
4: $(\rho, \sigma, \epsilon) \leftarrow BlindSign(M, (\beta, \gamma, \delta), y)$
5: **return** $<$ *SUBMIT*, $crand_{bc}, o_{bc}, Payload, retryFlag, (\rho, \sigma, \epsilon) >$

The second step corresponds to the **Transaction Proposal** (see Algorithm 2). We start by validating the blinded signature using the function $VerifyBlidnSign(M, (\rho, \sigma, \epsilon), y)$, where M is the signed message, (ρ, σ, ϵ) is the blinded signature, and y is the CA public key. In the case that the validation process fails, the function rejects the transaction; otherwise, the peer executes the operation o_{bc} on the *Payload*, and finally gets $verDep$ and $stateUpdate$. Finally, the submitting peer generates the *PROPOSE* message to be sent to the endorsing peers.

Algorithm 2 TxProp($crand_{bc}, sp_{bc}, o_{bc}, Payload, retryFlag, (\rho, \sigma, \epsilon), y$)

1: $m \leftarrow (crand_{bc}, o_{bc}, Payload, retryFlag, (\rho, \sigma, \epsilon))$
2: $M \leftarrow (crand, o_{bc}||Payload||retryFlag)$
3: **if** $VerifyBlindSign(M, (\rho, \sigma, \epsilon), y) == invalid$ **then**
4: **return** $invalid$
5: **else**
6: $(verDep, stateUpdate) \leftarrow EXEC(o_{bc}, Payload)$
7: $trans_{prop} \leftarrow (sp_{bc}, o_{bc}, Payload, stateUpdate, verDep)$
8: **return** $<$ *PROPOSE*, $m, trans_{prop} >$
9: **end if**

The next step corresponds to the **Transaction Endorsement** detailed in Algorithm 3. We start this process validating the blinded signature using the function *VerifyBlindSign* for the message $M = (crand||o_{bc}||Payload||retryFlag)$, the signature (ρ, σ, ϵ), and the CA public key y. If we get a valid response, we continue validating the *verDep*, *stateUpdate*, and the *securityPolicies*. If *verDep* corresponds to the last version that the endorsing peer has locally, and the *stateUpdate* is the same that its local state, and the *securityPolicies* authorizes the instruction o, we endorse the transaction; otherwise, we reject. To endorse the transaction, we use the function *BlindSign* to sign the message *TRANSACTION-VALID* with his corresponding tid. Finally, once the transaction is endorsed, the endorser peer creates the *TRANSACTION-VALID* message to be sent to the submitting peer sp_{bc}.

Algorithm 3 TxEndors($m, trans_{prop}, securityPolicies, y$)

$tid \leftarrow m.Payload.tid$
$M \leftarrow (m.crand||m.o_{bc}||m.Payload||m.retryFlag)$
if $VerifyBlindSign(M, (m.\rho, m.\sigma, m.\epsilon), y)$ == *invalid* **then**
  return *invalid*
else
  ($verDep_{endorser}$,$stateUpdate_{endorser}$) $\leftarrow$ *EXEC($m.o_{bc}$,Payload)*
end if
$(\beta, \gamma, \delta) \xleftarrow{r} \mathbb{Z}_q$
if $trans_{prop}.stateUpdate \neq stateUpdate_{endorser}$ **then**
  result $\leftarrow$ *(TRANSACTION-INVALID||tid||INCORRECT-STATE)*
  $(\rho, \sigma, \epsilon)_{ep} \leftarrow BlindSign(result, (\beta, \gamma, \delta), y)$
  return *<TRANSACTION-INVALID,tid,INCORRECT-STATE,* $\sigma_{ep} >$
else if $trans_{prop}.varDep \neq varDep_{endorser}$ **then**
  result $\leftarrow$ *(TRANSACTION-INVALID||tid||INCORRECT-VERSION)*
  $(\rho, \sigma, \epsilon)_{ep} \leftarrow BlindSign(result, (\beta, \gamma, \delta), y)$
  return *<TRANSACTION-INVALID,tid,INCORRECT-VERSION,* $\sigma_{ep} >$
else if $securityPolicies = invalid$ **then**
  result $\leftarrow$ *(TRANSACTION-INVALID||tid||REJECTED)*
  $(\rho, \sigma, \epsilon)_{ep} \leftarrow BlindSign(result, (\beta, \gamma, \delta), y)$
  return *<TRANSACTION-INVALID,tid,REJECTED,* $\sigma_{ep} >$
else
  $result \leftarrow$ (*TRANSACTION-VALID*$||tid$)
  $(\rho, \sigma, \epsilon)_{ep} \leftarrow BlindSign(result, (\beta, \gamma, \delta), y)$
  return $<$ *TRANSACTION-VALID*$, tid, \sigma_{ep} >$
end if

Then, the submitting peer sp_{bc} receives the responses from the endorsing peers during the **Broadcasting to Consensus** step (see Algorithm 4). These responses are collected inside of an array called *endorsement*. Once the submitting peer has collected enough valid responses in the *endorsement* array (at least $50\% + 1$), the peer sends the transaction to the orderers to be included into the next block by using the function *broadcast*.

Algorithm 4 TxBrodCons($trans_{prop}$, $(\rho, \sigma, \epsilon_{ep})$, $endorsement_{tid}$)

$endorsment_{tid}[endorsement_{tid}.length + 1] \leftarrow (\rho, \sigma, \epsilon)_{ep}$
if $(endorsment_{tid}.length >= (\frac{TotEndorPeers}{2} + 1))$ **then**
  $blob \leftarrow (trans_{prop}, endorsment_{tid})$
  return $broadcast(blob)$
end if

Finally, during the **Commitment** (see Algorithm 5) the transaction is validated with its respective endorsement. If the $verDep$ has not changed during the validation process and the $endorsement$ is valid according to the security policies for the operation o_{bc}; the transaction is added into the new block. The *NewBlock* is an array where we store

the new transactions to be settled, and once we reach the maximum block size (*BlockLengthMax*) configured in the network, we spread the new block to be committed by the peers using the function *Commit2Network*. In the case that the validation process fails, the transactions in the block are rejected, and the *retryFlag* for those transactions are set to 1.

Algorithm 5 BlockCommitment(*blob, NewBlock, securityPolicies, retryFlag)*

1: **if** $(blob.endorsment_{tid} \notin securityPolicies) \parallel (blob.trans_{prop}.verDep \neq valid)$ **then**
2: **return** $retryFlag \leftarrow 1$
3: **else**
4: $NewBlock[NewBlock.length + 1] \leftarrow trans_{prop}$
5: **end if**
6: **if** $NewBlock.lenght == BlockLengthMax$ **then**
7: **return** $Commit2Network(B)$
8: **end if**

5 Protocol Properties

BlindCons has the following security properties.

5.1 Consistency

A protocol is said to be *consistent* if it ensures that a transaction generated by a valid user stays immutable in the blockchain.

Definition 2. *A protocol $\mathbb{P}$ is $T-consistent$ if a transaction tx generated by an honest client c_{cb} to execute a valid operation o_{bc}, it is confirmed and stays immutable in the blockchain after $T - round$ of new blocks.*

Theorem 1. *BlindCons protocol is 1-consistent.*

Proof. The protocol described in Section 4 is a BFT based consensus algorithm. Consistency is achieved by agreeing with the validity of the transaction through a Byzantine Agreement process. Hence, for a transaction tx that has reach a majority of valid endorsements for a operation o_{bc}, the probability to not settling it in a new block and having forks in the chain is neglected if we have at most $\lfloor \frac{an-1}{3} \rfloor$ out of total n malicious peers, as it has been shown in [**?**, **?**] under the terminology of safeness. It is 1-consistent because we do not have any fork; hence only one block is needed to wait in order to have a transaction validated.

5.2 Liveness

The liveness property means that a consensus protocol ensures that if an honest client submits a valid transaction, a new block will be appended to the chain with the transaction in it. Hence, the protocol must ensure that the blockchain growths if valid clients generate valid transactions.

Definition 3 (Liveness). *A consensus protocol $\mathbb{P}$ ensures* liveness for a blockchain C *if $\mathbb{P}$ ensures that after a period of time t, the new version of the blockchain C' is $C' > C$, if a valid client $c_{i_{bc}}$ has broadcasted a valid transaction tx_i during the time t.*

Theorem 2. *BlindCons achieves liveness.*

Proof. BlindCons is a PBTF-based consensus protocol. Thus, liveness is achieved if after the transaction validation process, the ordering services propose a new block B with the transactions broadcasted by the clients during a period of time t. Hence, for valid transactions tx_i, where $i \in \mathbb{N}_0$, issued by valid a client $c_{i_{bc}}$ during a period of time t, the probability that $C' = C$ is neglected if we have at most $\lfloor \frac{n-1}{3} \rfloor$ out of total n malicious peers [**?**].

5.3 Blindness

We use the definition of *blindness* defined by Schnorr in [**?**]. A signature is properly blinded if the signer cannot get any information about the signature if the receiver follows the protocol correctly.

Definition 4 (Blind signature). *A signing scheme is* blind *if the signature $(m, \rho, \sigma, \epsilon)$ generated by following correctly the protocol, is statistically independent of the interaction (a, e, R, S) with that provides the view to the signer.*

Theorem 3. *Okamoto-Schnorr signature $(m, \rho, \delta, \epsilon)$ is statistically independent to the interaction (a, e, R, S) between the authority A and the user.*

Proof. We recall how the protocol works. To generate a blind signature $(m, \rho, \sigma, \epsilon)$ the user chooses randomly $(\beta, \gamma, \delta) \in \mathbb{Z}_q$ to respond to the CA's commitment a with the challenge $c = H(m, ag^{\beta}h^{\gamma}y^{\delta}) - \delta \bmod q$. The CA then sends $(R, S) = (t - er, u - es)$ to finally obtain the signature by calculating $(\rho, \sigma) = (R - \beta, S - \gamma)$. Hence, for the constant interaction (a, e, R, S) and a unique set (β, γ, δ) randomly chosen per signature, we generate a signature $(m, \rho, \delta, \epsilon) = (m, R - \beta, S - \gamma, e + \gamma)$ that is uniformly distributed over all the signatures of the message m due to the random $(\beta, \gamma, \delta) \leftarrow \mathbb{Z}_q$ [**?**].

5.4 Non-forgeability

The non-forgeability property used in this paper refers to the impossibility for a user to compute a $k + 1$ signature by his own after k interactions with a signer; this proves that the only method to generate a valid signature is contacting the CA. This property was proved by Pointcheval and Stern in [**?**].

Definition 5 (One more forgery). *For any integer k, a PPT Turing machine A can calculates $k + 1$ signatures after k interactions with the signer Σ with non-negligible probability.*

Theorem 4. *Let Okamoto-Schnorr blind signature be a cryptographic scheme in the random oracle model. If there is a PPT Turing machine able to calculate a "one more" forgery with non-negligible probability (even with a parallel attack), then the discrete logarithm can be solved in polynomial time.*

Proof. The proof's outline is detailed in [**?**]. Consider that the public values p, q, g and h; the secret keys (r, s) and the public key y are the same as described in Section 2.2. Let *Q* be a polynomial number that represents the queries asked to the random oracle $(\mathcal{Q}_1, ..., \mathcal{Q}_Q)$.

Let *A* be the attacker. *A* is a PPT Turing machine which success in computing $l + 1$ valid forgeries $(m_i, \alpha_i, \rho_i, \sigma_i, \epsilon_i)$ after (a_i, e_i, R_i, S_i) interactions (see Section 2.2) for $i \in \{i, ..., k\}$ with non-negligible probability ϵ.

We want to calculate the discrete logarithm of h relatively to g by using oracle reply [**?**], with the CA and the attacker colluded. We start running the attack with random keys, random tape Ω and oracle *f*. Then we choose a random index *j*', and we replay the attack using the same keys and tape but with different oracle (f') such that the first $j - 1$ answers do not change. We expect to obtain from both processes a common α_i from the j^{th} query with different representations regarding g and h, with non-negligible probability. Finally, the probability of success is greater than (for more details see [**?**]):

$$\frac{1}{2(k+1)} \left(\frac{1}{6(k+1)} \frac{\epsilon}{Q^{k+1}} \right)^3 \tag{1}$$

6 Conclusion

In this paper, we propose a new consensus algorithm for user identity privacy preserving by using the Okamoto-Schnorr blind signature scheme for transactions unlinkability. This algorithm is based on the PBFT and Hyperledger Fabric consensus algorithms. Our protocol allows the users to issue a blinded signature that keeps their identity private during the blockchain transaction validation process and after the transaction is settled. The signing scheme is based on the Okamoto-Schnorr blind signature that is a provably secure [**?**] and efficient signing scheme [**?**]. The protocol was designed to implement a blind signature scheme to achieve transactions unlinkability by using the components that any permissioned or private blockchain architecture has. Hence, our algorithm performs the signature blinding process by using the CA that already exists in the permissioned scheme, making it efficient to be implemented in these kinds of blockchain architectures.

One open issue is the network privacy. Our protocol is vulnerable to user tracking through the network. Hence, the network privacy is an issue that is not addressed by BlindCons. However, as future work, we would like to integrate Mixnets [**?**] as a decentralised application to hide the transaction source and accomplish privacy at a network level.

References