BlindCons: A Consensus Algorithm for Privacy Preserving Private Blockchains

1st Mirko Koscina
Département d'Informatique
École normale supérieure
Paris, France
Be-Studys
Geneva, Switzerland
mirko.koscina.bobadilla@ens.fr

2nd Pascal Lafourcade
LIMOS
Université Clermont Auvergne
Clermont-Ferrandm France
pascal.lafourcade@uca.fr

3rd David Manset
Research and Development
Be-Studys
Geneva, Switzerland
david.manset@g2s-group.com

4th David Naccache
Département d'Informatique
École normale supérieure
Paris, France
david.naccache@ens.fr

***Abstract*—Private blockchains are becoming gradually more sought-after in many organizations. Predominately because they allow organizations 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 enrollment. Our main contribution in this paper is to propose a user ID privacy preserving consensus algorithm for private blockchains, called BlindCons. The main security guarantees of BlindCons are: consistency of the data stored in the blockchain, liveness of the system and privacy of the users by blinding their signatures to keep the user identity private. We ensure that it is not possible to link a transaction to somebody using the data stored in the blockchain. Our consensus BlindCons uses Okamoto-Schnorr Blind Signature proposed in 1992 and is based on Practical Byzantine Fault Tolerance (PBFT) consensus proposed by Castro et al. in 1999.**

I. 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 consortium of companies have proposed *Private Blockchains* or *Permissioned Blockchain*, like for instance the MyHealthMyData consortium [7], which propose to connect hospitals and research centers across Europe to share medical data through a private blockchain network. These blockchains aims to manage data storage on a few nodes controlled by the company or the consortium in order to validate the transactions fastest, consume less energy and reduce the cost of mining. Another example is the project PlasticTwist [8] that creates a new circular economy based on a ERC20 cryptocurrency implemented on the top a permissioned ledger to encourage plastic recycling in Europe. Moreover, Maersk and IBM have developed TradeLens [11], a supply chain system supported by the permissioned blockchain Hyperledger Fabric.

Use cases like the above explained are proliferating due to the blockchain adoption in closed ecosystems and/or business applications. Due this, the European Blockchain Observatory and Forum has publish a technical report [23] where recommend, in case of needing to store sensitive data, to use private or permissioned blockchains. Therefore, the absence of mechanisms to keep the users privacy in 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. Our aim is to ensure the unlinkability between a transaction recored in the blockchain and the user that generated it. With our algorithm, we increase the privacy level of such blockchains. For this purpose, we design a privacy-preserving consensus called BlindCons. BlindCons relies on Okamoto-Schnorr's blind signature scheme [16] to ensure transaction unlinkability. Moreover, we based our protocol on Practical Byzantine Fault Tolerance (PBFT) consensus algorithm proposed in [3] for the machine replication process.

Related Work:: Blockchain platforms such as Bitcoin [15] and Ethereum [25] are popular due to the anonymity that their cryptocurrencies offer. However, Bitcoin still having issues with transaction linkability and traceability. During the last years some improvements have been proposed to increase Bitcoin anonymity [20], and also new cryptocurrencies have been developed to overcome these issues like Zcash and Monero. In the case of Zcash [10], the protocol achieves anonymity by using ZK-SNARKs as cryptographic proof. On the other hand, Monero is a protocol based on Cryptonote [24] that achieves unlinkability and untraceability by using a one-time random address and ring signatures.

A different approach to ensure anonymity is the eCash [4] [5] model proposed for Bitcoin in [9]. However, the idea to use a third party to generate a blind voucher is not aligned with the main principle of decentralisation that Bitcoin has. Although there are several works on blockchain privacy for permissionless ledgers, as far as we know, there is no formal protocol that addresses the anonymity issue that

permissioned ledgers have.

Outline:: Section II recalls Okamoto-Schnorr's blind signature scheme and PBFT consensus algorithm. Section IV describes our privacy-preserving consensus algorithm BlindCons. Before concluding, we explicit the security properties of our consensus algorithm in Section V.

II. Background

We start by recalling a blind version of the Schnorr's signature scheme proposed by Okamoto et al. in [16], then we present PBFT.

A. 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 [16]): 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 $\alpha = a g^{-\beta} h^{-\gamma} y^{\delta}$, 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 of the enrollment process (see Figure 1, where the authority A blinds the signature of the message proposed by the user). Furthermore, by using this scheme we can implement a privacy model just for user unlinkability, keeping the information stored in the ledger open to everybody.

B. Practical Byzantine Fault Tolerance (PBFT) Consensus Protocol

Practical Byzantine Fault Tolerance (PBFT) is a high-performance state machine replication protocol introduced by Miguel Castro and Barbara Liskov in 1999 [3]. This algorithm can be implemented for any replicated service within a distributed architecture, based on a local replica and some operations. These operations can be simple (i.e., read and write), or any other more sophisticated deterministic operations. The algorithm achieves liveness and safety with at most $\lfloor \frac{n-1}{3} \rfloor$ out of n faulty replicas [3]. This protocol is a practical implementation of a consensus algorithm capable of tolerating Byzantine Failure [12] on an asynchronous network. Although PBFT was implemented as a replication library for a file system, the design is practical enough to be considered as a faster and simpler alternative to others traditional consensus algorithm like Paxos [14]. However, this consensus algorithm assumes common knowledge of the participant of the network, i.e., identities, as explained in [17], making it unsuitable for some open decentralised systems, i.e., Permissionless Ledger.

1) PBFT general description.: The state machine replication algorithm purposed in PBFT is based on a consensus process between the nodes within a network arranged as a distributed system. The resiliency of the algorithm is optimal when the network has at least $3f+1$ replicas or nodes with up to f of them faulty [2]. The outnumber of replicas is necessary because we can reach consensus with the response of $n-f$ of them, where f can be non-responding or faulty replicas, and n the total number of replicas into the network. Moreover, it is possible that some of the responses considered in the consensus process coming from faulty replicas. Nevertheless, if the number of non-faulty replicas is outnumbering (i.e., $n - 2f > f$), there are enough responses to consider that the protocol remains resilient against faulty replicas.

The algorithm is modelled so that each machine state replica has a local state and supports special operations to process the requests received from the network. The replicas or nodes are identified by enumerating them $\{0,...,|R|-1\}$, where $|R| = 3f+1$ and f is the maximum number of faulty replicas. The value of R is selected to maintain the optimality of the algorithm because additional replicas can degrade the performance of the replication process. The replicas are organized in views, which correspond to a succession of configurations that are changing according to the liveness of the distributed system. The view configuration consists of one primary replica, and the rest of them are backups. The views are sorted by using consecutive numbers, and the primary replica p is identified as $p = v \mod |R|$, where v is the view number. The view changing holds if the primary begins to fail.

Before we start describing the protocol, we introduce the notation used in this paper. In any distributed system, the information is passing through the replicas 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.

The PBFT protocol consists of five steps. In other to keep the coherence with the original paper [3], we use the same names of each step.

1) **REQUEST:** In this stage, a client c sends a *REQUEST* message to the primary to execute an operation o on the distributed system.
2) **PRE-PREPARE:** After the primary receives the *REQUEST* message, it validates the assign a sequence number n, then signs the message and finally sends it to the other backups in a *PRE-PREPARE* message.
3) **PREPARE:** The backups validate the operation o indicated in the *PRE-PREPARE* message, and then send its response in a signed message called *PREPARE*.

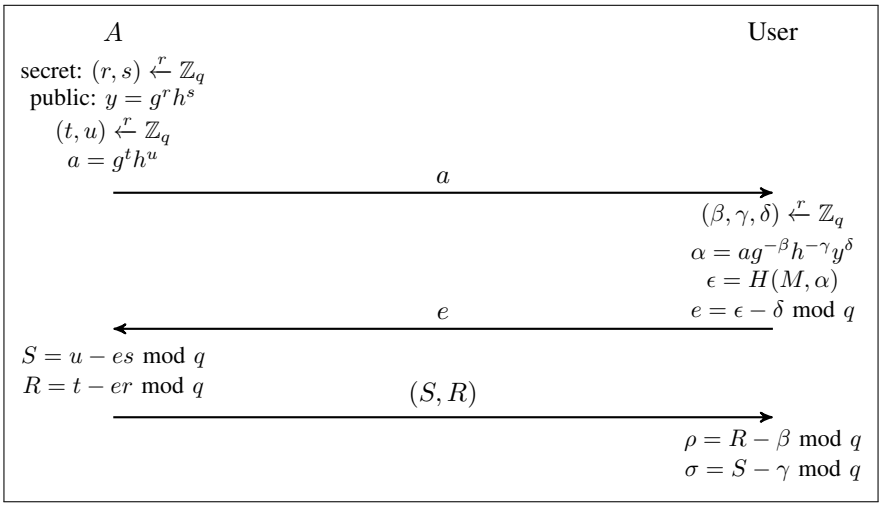

Figure 1. Okamoto-Schnorr blind signature diagram, where $y \xleftarrow{r} \mathbb{Z}_q$ means that y is randomly chosen in $\mathbb{Z}_q$.

4) **COMMIT:** Replicas send a *COMMIT* message when they receive enough valid *PREPARE* messages for the operation o.
5) **REPLY:** Once the replicas have committed the operation o they send the result r directly to the client c. Then the client c accepts the result once it receives enough *REPLY* messages with the same r from different replicas.

The process begins with a request sent by the client c to execute an operation o on the state machine. This process is carried out by sending a request message $m =<$ *REQUEST*$, o, t, c, \sigma_c >$, where t is a timestamp, and σ_c is the signature of the message signed by the client c. Once the primary received the message m starts the first phase of the algorithm called *pre-prepare*. In this phase the primary assigns a reference number n to the client request and creates a new message with structure $<$ (*PRE-PREPARE*$, v, n, d, \sigma_p), m >$, where v is the view number, d is the digest of the message m, and σ_p is the signature of the *pre-prepare* message. Then the primary sends the pre-prepared message to the backups to continue the process. The backups accept the *pre-prepare* message if:

- σ_p and σ_c are correct;
- d corresponds to the hash of the message m;
- *pre-prepare* message is in the view v;
- The sequence number is within the range $[h, H]$ (to prevent that a faulty primary exhaust the sequence number selecting a large n);
- the backup has not been accepted by another *pre-prepare* message with sequence n and view v with a different d.

Once the backup i has accepted the *pre-prepare* message, it executes the *prepare* phase broadcasting a message $<$ *PREPARE*$, v, n, d, i, \sigma_i >$ to the other replicas (including the primary).

A replica accepts the prepare message if:

- The signatures σ_c, σ_p and σ_i are correct;
- The view number v corresponds to the replica's view;
- The sequence number n is within $[h, H]$.

After the *prepare* phase, a replica k will enter to phase *commit* after it has included in their log the request m, a *pre-prepare* message for the request m in the view v with sequence number n, and $2f$ *prepare* messages from different backups. In the *commit* phase, the replica k broadcasts a message $<$ *COMMIT*$, v, n, d, k, \sigma_k >$ to the other replicas. The commit message is accepted by the replicas if:

- The signatures σ_c, σ_p, σ_i and σ_k are correct;
- The view number v corresponds to the replica's view;
- The sequence number n is within $[h, H]$.

The replica l executes the operation o requested by the client c in the message m once the *prepare* phase has concluded in $f + 1$ non-faulty replicas and has accepted $2f + 1$ commit messages from other replicas with the same view v, sequence number n, and digest d. After the execution of the operation o, the replica l generates a reply message $<$ *REPLY*$, v, t, c, l, r, \sigma_l >$, where r is the operation result, and then it is sent to the client.

Finally, the client accepts the result once it receives $f + 1$ reply messages with valid signatures from different replicas with the same timestamp t and result r.

III. PBFT 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. As any other distributed system, blockchain needs a consensus algorithm to replicate the chain in each node member of the network. Nevertheless, the selections or design of the consensus 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 a Certificate Authority (CA) before joining the network. The Certificate Authority is responsible for generating the user credential for each new client and/or peer. This model provides a base of knowledge of the network members, making PBFT based algorithms suitable for the blockchain replication process. Our consensus approach for permissioned ledger aims to the opposite of the permissionless replication protocol. The security of the consensus algorithm in a permissionless architecture is achieved proving the node honesty by expending their resources (i.e., computing capacity, power consumption, stakes, among others). In contrast, a PBFT-based protocol reaches the consensus accepting as valid the result proposed by the nodes majority.

Our consensus algorithm is based on PBFT and the *execute-order-validate* process used in Hyperledger Fabric [1] since version 1.0. In this protocol, each transaction passes through these three stages. The *execution* phase triggers the transaction and then validates by endorsing it. In the *order* phase, the transactions are ordered according to the consensus protocol. Finally, the **validation** phase checks the security policies per operation and application types.

In the case of our blockchain PBFT-based consensus algorithms, we start with a client c_{bc} that has the same role than a client in the PBFT procotol. Then we have submitting peers $sp_{i_{bc}}$ that send the transaction proposal with the instructions indicated by the client to the endorsing peers in order to validate the transaction. This process is analogous than the executed by the primary in the PBFT algorithm, that sends the message with the instructions to the backups to validate it. Then, for the transaction endorsement, the endorsing peers send their responses to the submitting peer, which is responsible to collect the responses after the validation process is done. Finally, the submitting peer sends the transaction to the orderers to generate the new block to be committed by the nodes members to the network. This process differs from the PBFT, where the backups respond to the other backups, and all of them waits to have enough **prepare** and **commit** messages to then reply directly to the client, without pass through the primary.

To exemplify this process, consider that we have a client c_{bc} that needs to execute an operation o_{bc} on our PBFT-based blockchain protocol. To process this request, we need to execute the following five steps: Initiating Transactions, Transaction Proposal, Transaction Endorsement, Broadcasting to Consensus and Commitment (see Figure 2).

1) ***Initiating Transactions:*** In the initiating transactions, the client c_{bc} generates a message to invoke specific operations o_{bc} to be resolved by the network. The message is sent to a submitting peer, which is the responsible to continue with the next step called *Transaction Proposal*. In the case that the submitting peer is offline or misbehaving, the client sends the transaction to the next submitting peer. The structure of the message is presented below:

$< SUBMIT, c_{bc}, o_{bc}, txPayload, \sigma_c, retryFlag >$, where:

- 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,
- σ_c: is the client signature,
- $retryFlag$: boolean variable to identify whether to retry the transaction submission in case of the transaction fails.

2) ***Transaction Proposal:*** The submitting peer (sp_{bc}) receives the transaction and verifies the client signature σ_c. Then prepares the transaction proposal message to be sent to the endorsing peer. The transaction proposal consists of invoking the operations o_{bc} and computing the state update (*stateUpdate*) and version dependency (*verDep*). The version dependencies relates the variables involve in the transaction with node local version of the variable 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 done, the submitting peer generates the *transaction proposal* message and then submit it to the endorsing peer. The transaction proposal message is defined as:

$< PROPOSAL, m_c, trans_{prop} >$, where:

- m_c:= $(c_{bc}, o_{bc}, txPayload, \sigma_c)$,
- $trans_{prop}$:= $(sp_{bc}, o_{bc}, txContentBlob, stateUpdate, verDep)$

3) ***Transaction Endorsement:*** The endorsing peer (ep_{bc}) verifies the client signature σ_c coming in m_c and checks that the operations o_{bn} in m_c and $trans_{prop}$ are the same. Then, the endorser simulates the transaction proposal and 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 submitting peer. The message has the following structure:

$< TRANSACTION\text{-}VALID, tx_{id}, \sigma_{ep} >$, where:

- tx_{id}: is the transaction identifier generated with the client ID and a nonce,
- σ_ep: is the signature of the message signed by the submitting peer sp_{BC}.

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

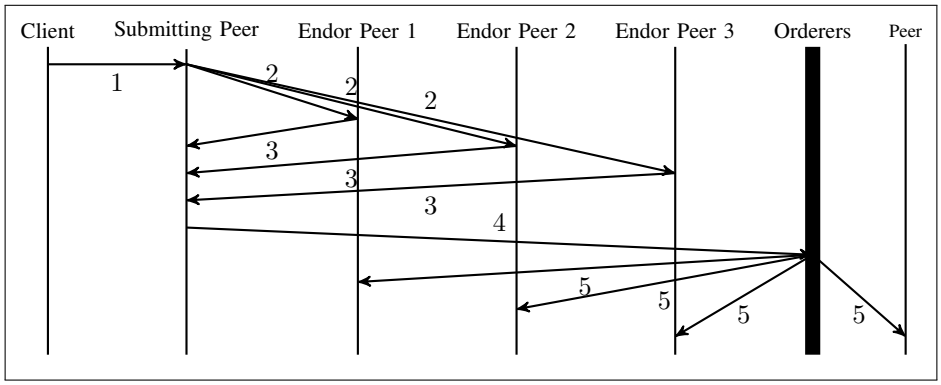

Figure 2. Transaction Flow [1].

$< \textit{TRANSACTION-INVALID}, tx_{id}, Error, \sigma_{ep} >$, 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.

4) ***Broadcasting to Consensus:*** The submitting peer waits for the response from the endorsing peers. When it receives enough *Transaction Valid* messages adequately signed, the peer stores the endorsing signatures into packaged called *endorsement*. Once the transaction is considered endorsed, the peer 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 transaction has failed to collect the enough endorsements, the peer abandons this transaction and notifies to the client.

5) ***Commitment:*** Once a submitting peer broadcast transactions properly endorsed to consensus, the ordering services collect these transactions and organize them into a block. 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 hash value of the previous block.

The peers 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. Then, the peers 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 drops it. Finally, the invalid transactions are informed to the client c_{bc} by the submitting peer sp_{bc}, and according to the *retryFlag*, the sp_{bc} may retry the transaction.

The peers can change the local state just once the transactions are committed. Hence, each peer 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.

IV. Privacy Preserving PBFT for Permissioned Ledgers

Considering a permissioned PBFT based consensus protocol like the one introduced in Section III, in this protocol, we use the digital signature as 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 members 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 will replace the signing mechanism used in the original protocol by the Okamoto-Schnorr blind signature scheme [16]. 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 propose 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} using his local version of the blockchain. Then, each endorsing peer generates a signed transactions 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 respective endorsement. Then, the new block is spread through the network to be committed by the peers.

In order to maintain the consistency and liveness that the protocol has, 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 response *valid* or *invalid*. 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 we already have defined the functions and the variables used in the protocol, 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, then we concatenate the arguments $(crand, o_{bc}, Payload, retryFlag)$ to be signed, and then we use the function *BlindSign* to generate the blinded signature according to the protocol described in Section II-A. Finally, we generate the message *SUBMIT* to be sent to the submitting peer sp_{bc}.

The second step corresponds to the **Transaction Proposal** (see Algorithm 2). In this process 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 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*, $crand_{bc}, o_{bc}, Payload, retryFlag$, $(\rho, \sigma, \epsilon) >$

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. In the case that the response is valid, we continue validating the *verDep*, *stateUpdate*, and the *securityPolicies*. In the case that the *verDep* corresponds to the last version that the endorsing peer has locally, and the *stateUpdate* is the same that it has as 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}.

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*.

Finally, during the **Commitment** (see Algorithm 5) the transaction is validated with his 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.

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,* σ_{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,* σ_{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,* σ_{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

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

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 block are rejected and the *retryFlag* for those transactions are set to 1.

V. Protocol Properties

BlindCons has the following security properties.

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

if $(blob.endorsment_{tid} \notin securityPolicies)$ || $(blob.trans_{prop}.verDep \neq valid)$ **then**
  return $retryFlag \leftarrow 1$
else
  $NewBlock[NewBlock.length + 1] \leftarrow trans_{prop}$
end if
if $NewBlock.lenght == BlockLengthMax$ **then**
  return $Commit2Network(B)$
end if

A. Consistency

A protocol is said to be *consistent* if 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 1: The protocol described in Section IV 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 endorses for a operation o_{bc}, the probability to not settle it in a new block and to have 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 [3], [13] under the terminology of safeness. It is 1-consistent because we do not have any fork so only one block is needed to wait in order to have a transaction validated.

B. 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 the 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 [3]. ■

C. Blindness

We use the definition of *blindess* defined by Schnorr in [22]. 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 the protocol, correctly, 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 response to the CA's commitment a with the challenge $c = H(m, ag^{\beta}h^{\gamma}y^{\delta}) - \delta \bmod q$. Then the CA 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 signature of the message m due to the random $(\beta, \gamma, \delta) \leftarrow \mathbb{Z}_q$ [22]. ■

D. 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 [18].

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 parallel attack), then the discrete logarithm can be solved in polynomial time.

Proof: Outline of the proof is detailed in [18]. 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 II-A. 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 II-A) 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 [19], 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 bigger than (for more details see [18]):

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

■

VI. Conclusion

In this paper, we propose a new consensus algorithm for user identity privacy preserving by using a Okamoto-Schnorr blind signature for transactions unlinkability. This algorithm is based on 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 Okamoto-Schnorr blind signature that is a provably secure [18] and efficient signing scheme [21]. 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 a future work, we would like to integrate Mixnets [6] as decentralised application to hide the transaction source and accomplish privacy at a network level.

References

[1] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. Hyperledger fabric: A distributed operating system for permissioned blockchains. *arXiv preprint arXiv:1801.10228*, 2018.

[2] Gabriel Bracha and Sam Toueg. Asynchronous consensus and broadcast protocols. *Journal of the ACM (JACM)*, 32(4):824–840, 1985.

[3] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[4] David Chaum. Blind signatures for untraceable payments. In *Advances in cryptology*, pages 199–203. Springer, 1983.

[5] David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. In *Conference on the Theory and Application of Cryptography*, pages 319–327. Springer, 1988.

[6] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90, February 1981.

[7] MyHealthMyData Consortium. My health my data. http://www.myhealthmydata.eu/, 2016. [Online; accessed Sept. 2018].

[8] Ptwist Consortium. Plastictwist. http://www.myhealthmydata.eu/, 2018. [Online; accessed Sept. 2018].

[9] Ethan Heilman, Foteini Baldimtsi, and Sharon Goldberg. Blindly signed contracts: Anonymous on-blockchain and off-blockchain bitcoin transactions. In *International Conference on Financial Cryptography and Data Security*, pages 43–60. Springer, 2016.

[10] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash protocol specification. Technical report, Technical report, 2016–1.10. Zerocoin Electric Coin Company, 2016.

[11] IBM. Tradelens. http://www.tradelens.com/, 2016. [Online; accessed Sept. 2018].

[12] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[13] Jinyuan Li and David Maziéres. Beyond one-third faulty replicas in byzantine fault tolerant systems. In *NSDI*, 2007.
[14] J-P Martin and Lorenzo Alvisi. Fast byzantine consensus. *IEEE Transactions on Dependable and Secure Computing*, 3(3):202–215, 2006.
[15] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.
[16] Tatsuaki Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In *Annual International Cryptology Conference*, pages 31–53. Springer, 1992.
[17] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 643–673. Springer, 2017.
[18] David Pointcheval and Jacques Stern. Provably secure blind signature schemes. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 252–265. Springer, 1996.
[19] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli Maurer, editor, *Advances in Cryptology — EUROCRYPT '96*, pages 387–398, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.
[20] Amitabh Saxena, Janardan Misra, and Aritra Dhar. Increasing anonymity in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 122–139. Springer, 2014.
[21] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of cryptology*, 4(3):161–174, 1991.
[22] Claus Peter Schnorr. Security of blind discrete log signatures against interactive attacks. In *International Conference on Information and Communications Security*, pages 1–12. Springer, 2001.
[23] Ken Timsit Tom Lyons, Ludovic Courcelas. Blockchain and gdpr. Technical report, The european union blockchain observatory and forum, oct 2018.
[24] Nicolas Van Saberhagen. Cryptonote v 2.0, 2013.
[25] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 151:1–32, 2014.