Generic Privacy Preserving Private Permissioned Blockchains

Frédéric A. Hayek
Clermont-Auvergne University, CNRS, LIMOS, France
frederic.hayek@uca.fr

Mirko Koscina
be ys Pay
France

Pascal Lafourcade
Clermont-Auvergne University, CNRS, LIMOS, France

Charles Olivier-Anclin
be ys Pay
Clermont-Auvergne University, CNRS, LIMOS, France

ABSTRACT

Private permissioned blockchains are becoming gradually more sought-after. Such systems are reachable by authorized users, and tend to be completely transparent to whoever interacts with the blockchain. In this paper, we mitigate the latter. Authorized users can now stay unlinked to the transaction they propose in the blockchain while being authenticated before being allowed to interact. As a first contribution, we developed a consensus algorithm for private permissioned blockchains based on Hyperledger Fabric and the Practical Byzantine Fault Tolerance consensus. Building on this blockchain, five additional variations achieving various client-wise privacy preserving levels are proposed. These different protocols allow for different use cases and levels of privacy control and sometimes its revocation by an authority. All our protocols guarantee the unlinkability of transactions to their issuers achieving anonymity or pseudonymity. Miners can also inherit some of the above privacy preserving setting. Naturally, we maintain liveness and safety of the system and its data.

CCS CONCEPTS

• **Security and privacy** → *Security protocols*; **Pseudonymity, anonymity and untraceability**;

KEYWORDS

Privacy, Byzantine Fault, Blockchain, Signature.

ACM Reference Format:
Frédéric A. Hayek, Mirko Koscina, Pascal Lafourcade, and Charles Olivier-Anclin. 2023. Generic Privacy Preserving Private Permissioned Blockchains. In *The 38th ACM/SIGAPP Symposium on Applied Computing (SAC '23), March 27 - March 31, 2023, Tallinn, Estonia.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3555776.3577735

1 INTRODUCTION

Blockchains are replicated synchronized databases shared across a trustless network. They started with Bitcoin's Nakamoto consensus style [25], meaning having miners compete to create the next block with a longest-chain-win rule[1]. Anonymity in Bitcoin can be quite contentious [28], which led to more anonymous open blockchains such as Zcash [20] and Monero (based on Cryptonote [33]). Bitcoin was, and to some extent still is, open to all users and all miners. However, blockchains are undergoing an evolutionary shift towards private and permissioned architecture, and implementing BFT consensus instead of Nakamoto consensus[2]. This shift mainly aims to regulate access to sensitive information and to gain in scalability.

Distributed ledger privacy. Danezis and Gürses [13] divided privacy into confidentiality and control (over personal data). With distributed ledgers, control level depends on the openness and configuration of the ledger. Since control includes the right to erasure, and in a distributed setting there is no way to make sure that some data has indeed been deleted, then control is somewhat out of our reach. Confidentiality refers to the difficulty of extracting knowledge from data. With blockchains, the data is an operation, and it must have a specified format and must meet certain requirements. These requirements usually include a signature by its issuer. The two relevant ends of the identity confidentiality spectrum are pseudonymity and anonymity. Pseudonymity is having entities identified by pseudonyms, but not necessarily being able to link the pseudonyms to the identities behind them. A good example of pseudonymity is Bitcoin [25], where users are identified by their public keys, and it is generally infeasible to trace the public key to a real world identity. Note that in bitcoin a user can create as many pseudonyms as they wish. Anonymity, on another hand, is when it is infeasible to link anything. One such example is Monero's usage of ring signatures [29] in order to anonymize the sender's identity that is blended with a set of other identities.

Private (respectively permissioned) distributed ledgers must naturally restrict their usage (respectively their mining) to legitimate entities only. These restrictions seem, at first, incompatible with privacy, giving rise to a dilemma: how to restrict private (respectively permissioned) distributed ledgers usage (respectively mining) while ensuring user (respectively miner) privacy? Note that this apparent dilemma is applicable to virtually any type of private or permissioned distributed ledger, whether it be relying on Proof-of-Work, Proof-of-Stake, Byzantine Fault Tolerance (BFT) protocols or any other type of consensus mechanism.

Contributions. In this article, we propose a new BFT consensus blockchain, SignCons, modelled after Hyperledger Fabric [2, 15], with several constructions to gain in privacy. The blockchain itself makes a distinction between two categories of miners: endorsers who verify transactions, and orderers who order valid transactions in blocks. This grants us an *execute-order-validate* architecture. Orderers reach consensus using a modified version of the Practical

[1]Length here does not necessarily mean number of blocks (as it did it at Bitcoin's conception). Length can for example denote difficulty in the current Bitcoin protocol, or weight in Ethereum's GHOST algorithm implementation [31].

[2]Note that some intermediate options exist, such as Proof-of-Stake blockchain, which is in the Nakamoto consensus style as well as being public and permissioned.

Publication rights licensed to ACM. ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of a national government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.
SAC'23, March 27 – March 31, 2023, Tallinn, Estonia
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9517-5/23/03…$15.00
https://doi.org/10.1145/3555776.3577735

Fault Tolerance [10], yielding safety and liveness under very moderate assumptions. As for our main contributions, firstly we give three similar yet distinct ways to acquire user anonymity, then we give another two ways to acquire user pseudonymity, and finally we propose two constructions to gain endorser and/or orderer pseudonymity. Constructions aimed at users, endorsers and orderers are composable. All constructions (anonymous and pseudonymous) share a birthplace, however they differ in their applications and use cases. The first of the user-related anonymous constructions is BlindCons. It relies on having an authority, and uses blind signatures. The user authenticates themself to the authority, and if the user is part of the corresponding authorized set, then the authority blind signs their operation. And the data must be deemed valid by the miners if it is signed by the authority. The second construction is GroupCons, and it also relies on an authority but uses group signatures. The authority plays the role of the group manager, with the set of authorized users forming the signing group. The operation is group-signed by its issuer. Group-signed data must be deemed valid by miners. In this scenario, the authority can revoke the anonymity of the issuer. The third construction, RingCons does not rely on an authority, and uses ring signatures. The operation is ring-signed by its issuer, forming a ring with as many of the authorized users as the issuer wants. The ring must not include any user not member of the authorized set. Ring-signed operations in this manner must be deemed valid by miners. As for the two pseudonymous constructions for users, LinkGroupCons and LinkRingCons, they require respectively linkable group signatures and linkable ring signatures. Similarly, endorsers benefit of EndGroupCons and EndRingCons, while orderers have OrdGroupCons and OrdRingCons. Furthermore, the underlying blockchain, thanks to the PBFT structure, guarantees safety and liveness with whatever overlaid privacy construction. The different privacy constructions have different efficiencies, although all can be implemented without much added overhead computation (compared to the underlying blockchain). For a recap of the blockchain's properties refer to Tables 1 and 2. Note that these constructions could very easily be applied on top of any blockchain without inducing much latency into the system, though we do it here for a Hyperledger-like blockchain for which we prove the privacy properties

	Revoke	Authority		Privacy
	User	No	Inactive	Level
BlindCons	✓			Ano.
GroupCons	✓		✓	Ano.
RingCons		✓	✓	Ano.
LinkGroupCons	✓		✓	Pseu.
LinkRingCons		✓	✓	Pseu.

Table 1: Users' Privacy Preserving Protocols. Ano. : Anonymous; Pseu. : Pseudonymous.

	Revoke Endorser	Authority Not Needed	Privacy Level
(Ord) EndGroupCons	✓		Pseu.
(Ord) EndRingCons		✓	Pseu.

Table 2: Orderers' and Endorsers' Protocols.

Related Work. Most known privacy works on blockchains aim at anonymizing cryptocurrencies. Monero (built on Cryptonote [33]) relies on ring signatures [4] and ring confidential transactions [26, 32]. Zerocoin [24] introduced zero knowledge proofs (ZKPs) of set membership (which are quite onerous). Zcash [20] (resembling Zerocoin) relies on zk-SNARKs [3] which are much more effficient.

Blockchain privacy has been addressed quite extensively, but rarely has the identity confidentiality of private or permissioned distributed ledgers been addressed. One way to solve the issue is by using identity providers [18]: one permissions issuer and one or more permissions verifiers. The permissions issuer is in charge of verifying a user's identity and issuing them keying material; while the permissions verifier is in charge of verifying that an entity has valid keying material and issuing them a special key allowing them to transact. For the latter verification, the verifier permissions need not know the user's identity, but only verify the keying material.

Another way to ensure privacy on a private network uses puzzle-solving mechanism [12] and is dedicated to IoT networks. They propose to have two blockchains: a public one and a private one. To be allowed to transact on the private chain, users must solve a puzzle on the public chain inside a specific time-lapse. Legitimate users should be able to solve the puzzle within the time-lapse while non-legitimate users should not be able to do it. Other papers consider blockchain privacy from a network level perspective [19]. Some propose privacy-preserving private cryptocurrencies [21]. Even a dedicated survey of privacy-preserving solutions for blockchains [5] barely mentions privacy on private or permissioned blockchains. It mentions a blockchain architecture relying on blind signature that achieves trustless privacy-preserving reputation system [30].

Outline. In Section 2, we start by going over Fabric, PBFT and the cryptographic tools. In Section 3 we showcase our blockchain SignCons, before outlining the many privacy preserving constructions in Section 4. Subsequently we discuss the security and privacy properties in Section 5, the complexities in Section 6 and we conclude in Section 7.

2 BACKGROUND

SignCons uses an instantiation of Fabric and a PBFT consensus.

- *Practical Byzantine Fault Tolerance (PBFT)* [10] which is a Replicated State Machine protocol. For f faulty (byzantine) nodes, it permits a network of $3f+1$ nodes to reach consensus.
- *Hyperledger Fabric* [2, 15] is a blockchain framework. Is is a highly tweakable permissioned blockchain framework.

2.1 Practical Byzantine Fault Tolerance (PBFT)

Practical Byzantine Fault Tolerance (PBFT) [10] is designed for systems of at least $3f+1$ nodes where f is the number of faulty (byzantine) nodes. The protocol allows for a very powerful adversary that can coordinate faulty nodes, delay communication, or delay correct nodes. However, it is assumed that the adversary cannot delay correct nodes indefinitely. The adversary is also assumed to be a Probabilistic Polynomial Time (PPT) algorithm. The nodes are called *replicas* and form the set $\mathcal{R}$. Each replica is identified using an integer in $\{0,1,...,|\mathcal{R}|-1\}$. One replica is the *primary*, and the rest are *backups*. A *view* is a configuration of replicas that determines the primary. For a view number v, the primary is $p=v \mod |\mathcal{R}|$. The algorithm comprises three stages: *pre-prepare*, *prepare* and *commit*. When a client c wishes to do an operation on the State Machine Replication [22], they send $m=(REQUEST,o,t,c)_{\sigma_c}$ to who the client c believes is the primary. In that message m, o is the operation and t is the timestamp.

If the client does not receive replies soon enough, it broadcasts the request to all replicas: this takes care of the client's possible erroneous view and the primary's possible faultiness. Upon receiving a request, the primary p multicast a pre-prepare message to all replicas. That pre-prepare message is $((PRE-PREPARE,v,n,d)_{\sigma_p},m)$ where v is the current view, n is a sequence number assigned by p and d is m's digest. When a backup receives the pre-prepare message, it checks if the signatures in the request and the pre-prepare message are correct and if d is m's digest; it checks if it is in v; it checks if it has not accepted a pre-prepare message for view v with the same sequence number n and a different digest d; and it checks if the sequence number $h < n < H$ with h,H low and high watermarks. If the pre-prepare message passes all thoses tests, the backup i accepts it and multicast $(PREPARE,v,n,d,i)_{\sigma_i}$. Otherwise, it does nothing. Upon receiving a prepare message, a replica accepts it and appends it to its log if the signatures check out and if d corresponds to its view and if $h < n < H$. When replica i has accepted $2f$ nonconflicting prepare messages, it then multicast a commit message $(COMMIT,v,n,D(m),i)_{\sigma_i}$ where $D(m)$ is the digest of m. A replica accepts and appends a commit message to its log if the signature, the view number check out and if $h<n<H$. As soon as a replica i accepted $f+1$ nonconflicting commit messages, it executes the operation o on its state machine replication and sends a reply message to the client $(REPLY,v,t,c,i,r)_{\sigma_i}$ where r is the result of executing o.

View Change. Each replica has a timer that counts down to zero. It runs when the replica received a valid request and has not executed it, and pauses otherwise. When the countdown reaches zero, the replica stops accepting new messages and initiates a view change. This is done by multicasting a view change message. When the primary of view $v+1$ has received more than $2f$, it multicasts a new view message, and undertakes its role as primary.

Properties. Some details have been purposely omitted. We only want to encapsulate the gist of PBFT, since we use a modified version of it for consensus. PBFT relies on the assumption that for $|\mathcal{R}|$ replicas, at most $\left\lfloor \frac{|\mathcal{R}|-1}{3} \right\rfloor$ are faulty and the rather weak syncrony assumption that basically the delay time of communication of honest nodes is less than a given upper bound. With this, it is shown in [9] that PBFT achieves safety and liveness.

2.2 Hyperledger Fabric

Hyperledger Fabric [2, 15] is a permissioned blockchain framework. It divides miners into two sets: on the one hand they have *endorsers* who check transactions, and on the other they have *orderers* who order transactions into blocks. When a client wishes to transact on the blockchain, the client sends the transaction to a set of endorsers who are accredited to endorse it. If deemed valid, the endorsers sign it as a sign of endorsement and send it back to the client. When the client collects enough endorsements, the client sends them to the orderers who then work on incorporating said transaction in a block. The structure of Fabric is modulable, in the sense that it is possible to define different endorsement policies for different types of transactions, and it is possible to choose whatever consensus mechanism for the orderers. For example, for a given type of transaction, there must be unanimous endorsement of a given transaction for it to be considered valid; and the orderers compete doing a Proof-of-Work for creating the next block. Fabric does not follow the regular *order-execute*, but rather the *execute-order-validate*. Order-execute blockchains have many limitations, such as sequential execution on all peers, non-deterministic code, and confidentiality of execution. Fabric does provide some privacy, for instance through the use of different channels. Indeed, users only have access to their subset of channels. Another facet of privacy is through the execute-order-validate architecture and the endorsement policies. Only a subset of endorsers have to execute the transaction, and only the state after execution is ultimately written into the blockchain, so those who have access to the channel can see the resulting state but cannot necessarily know what the operation was. However, endorsers have full knowledge of users and their operations.

2.3 Cryptographic Tools

Signatures schemes. Here we present digital signatures which are at the foundation of our constructions. A signature scheme guaranties integrity, authentication and non-repudiation of digital transmissions. A Signature schemes S is a tuple of algorithms composed of a key generation algorithm $\mathsf{KeyGen}(1^{\mathfrak{K}}):$] returning a key pair (pk,sk). The latest is used to sign through algorithm $\mathsf{Sign}(\mathsf{sk},m):$ producing a signature σ on a message m. This signature being verified by $\mathsf{Verif}(\mathsf{pk},m,\sigma):$ returning a bit b. It must achieve *EUF-CMA* (Existential Unforgeability under Chosen Message Attack) [27].

Our transformations are based on three extensions of digital signatures: *blind signatures* [27] $\mathsf{BS} = (\mathsf{KeyGen}_{\mathsf{blind}},\mathsf{BlindSign}\langle\cdot,\cdot\rangle, \mathsf{Verify}_{\mathsf{BS}})$, producing signatures to someone else on messages unknown by the signer. *Group signatures* [11] $\mathsf{GS} = (\mathsf{KeyGen}_{\mathsf{group}}, \mathsf{GroupSign}, \mathsf{Verify}_{\mathsf{group}}, \mathsf{OpenGroupSign})$ where users can output signatures, where the said signature is not linkable to them but to the group. This works under supervision of a group authority. *Ring signatures* [29] $\mathsf{RS} = (\mathsf{KeyGen}_{\mathsf{ring}},\mathsf{RingSign},\mathsf{Verify}_{\mathsf{ring}})$ work similarly, but this time on a decentralized base. Refinements of group and ring signatures called *linkable group signatures* and *linkable ring signatures* allow achieving pseudonymity instead of anonymity. The formers rely on an additional Link algorithm to provide a link between the signatures. Here give their security properties:

Blind Signature. EUF-CMA and blindness
Ring Signature. EUF-CMA and anonymity
Group Signature. EUF-CMA, anonymity and traceability.

Authentication Channel. These processes mainly rely on digital signature schemes. It can also be achieved through other means among which we can cite message authentication codes. Authenticated channel allows authenticated communications with a third party. Public key infrastructure or certificates are among the most used solutions. They do not directly address confidentiality of the transmitted information, but most protocols guaranteeing authentications also address this issue through a key exchange mechanism at the beginning of the communication. This is not a hypothesis we need to make here in order to guarantee the security of our blockchains. In our protocols, we assume that the administrator of the blockchain (when it exists) has knowledge of all the entities' identities that are allowed to transact on or mine the blockchain. And when a key is used to interact through our protocol, it should have been registered before by the authority after potential verifications. The authority's public key pk_{auth} is considered as a global parameter and is not stated into the inputs of the algorithms.

While using blind signatures any peer has to authenticate themselves before the authority blind signs their message. In the upcoming algorithms we put together both the authentication process and the blind signature protocol and call it BlindSign. This aggregation is denoted by $\mathsf{BlindSign}\langle\mathcal{U}(M,\mathsf{pk}_{\mathcal{S}},\mathsf{sk}_{\mathcal{U}}),\mathcal{S}(\mathsf{sk}_{\mathcal{S}})\rangle$ for two polynomial time algorithms $\mathcal{U}$ and $\mathcal{S}$.

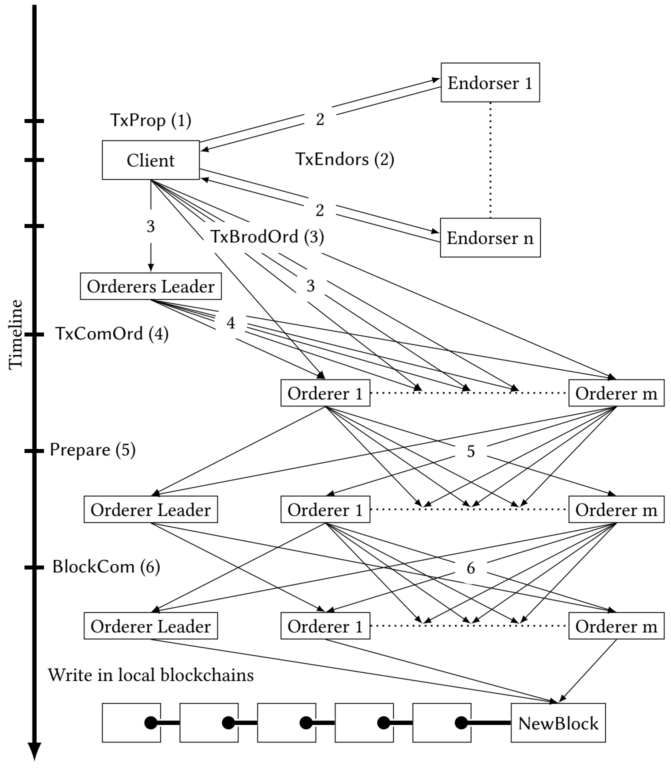

Figure 1: Transaction Flow of our BFT Consensus.

Algorithm 1 TxProp(o_{bc},Payload,$\mathsf{sk}_{c_{bc}}$)

```
1: $(\mathsf{verDep},\mathsf{stateUpdate}) \leftarrow EXEC(o_{bc},\mathsf{Payload})$
2: $\mathsf{trans_{prop}} \leftarrow \mathsf{pk}_{c_{bc}} || \mathsf{Payload} || \mathsf{verDep} || \mathsf{stateUpdate}$
3: $\sigma \leftarrow \mathsf{Sign}(\mathsf{trans_{prop}},\mathsf{sk}_{c_{bc}})$
4: return $(\mathsf{trans_{prop}},\sigma)$
```

Algorithm 2 TxEndors($\mathsf{trans_{prop}}$,σ,SecPolicies,$\mathsf{pk}_{c_{bc}}$,sk_{ep})

```
1: if $\mathsf{Verify}_{\mathsf{pk}_{c_{bc}}}(\mathsf{trans_{prop}},\sigma)=0$ or $\mathsf{pk}_{c_{bc}} \notin \mathsf{Clients}$:
2:     return $\perp$
3: $(\mathsf{verDep}_{ver},\mathsf{stateUpdate}_{ver})$
       $\leftarrow EXEC(\mathsf{trans_{prop}}.o_{bc},\mathsf{trans_{prop}}.\mathsf{Payload})$
4: if $\mathsf{trans_{prop}}.\mathsf{verDep} \neq \mathsf{verDep}_{ver}$ or $\mathsf{SecPolicies}=invalid$ or
   $\mathsf{trans_{prop}}.\mathsf{stateUpdate} \neq \mathsf{stateUpdate}_{ver}$:
5:     return $\perp$
6: $\sigma_{ver} \leftarrow \mathsf{Sign}_{\mathsf{sk}_{ep}}(\mathsf{trans_{prop}})$
7: return $\sigma_{ver}$
```

Algorithm 3 TxBrodOrd($\mathsf{trans_{prop}}$,$\{\sigma_{ver,i},\mathsf{pk}_i\}_{1\leq i\leq l}$)

```
1: $endorsements \leftarrow \perp$
2: for all $1\leq i\leq k$ do
3:     if $\mathsf{Verify}_{\mathsf{pk}_i}(\mathsf{trans_{prop}},\sigma_{ver,i})=1$ and $\mathsf{pk}_\sigma \in \mathsf{Endor}$:
4:         $endorsements \leftarrow endorsements \cup \{\sigma_{ver,i}\}$
5:         if $|endorsements| \geq \lfloor \frac{\mathsf{TotEnd}}{2} \rfloor + 1$:
6:             $blob \leftarrow (\mathsf{trans_{prop}},endorsements)$
7:             return blob
8: return $\perp$
```

3 OUR BLOCKCHAIN SignCons

Our generic SignCons blockchain is highly inspired by Hyperledger Fabric's modular blockchain framework [2, 15]. It comprises: (1) a finite set Clients of authorized entities, (2) a finite set Miners of authorized miners divided into two categories: (2a) a set Endor of entities who endorse transactions of cardinal TotEnd and (2b) a set Order of entities who achieve consensus of cardinality TotOrd. The process is outlined in Figure 1 and goes as follows:

(1) **Transaction Proposal.** The user signs their operation and sends it to the endorsers.
(2) **Transaction Endorsement.** Each endorser peer verifies the transaction, and if valid, signs it (as a sign of endorsement) and sends it back to the user.
(3) **Broadcasting to Consensus.** The client collects the endorsements and when enough are received, sends them to the orderers.
(4) **Block Proposal.** The orderer leader, upon receiving enough endorsed transactions, creates a block and proposes it to the other orderers.
(5) **Block Preparation.** The other orderers, upon receiving the block from the leader, check it, and validate it by signing and broadcasting it to the other orderers.
(6) **Appending Block to Blockchain.** Finally, when each orderer receives more than $\frac{2\mathsf{TotOrd}}{3}$ block validations, then each orderer appends the new block to their local version of the blockchain, and broadcasts the new block to users.

Each step is detailed into a corresponding algorithm. Steps 1, 2 and 3 are about gathering enough endorsements, and steps 3, 4, 5 and 6 are about incorporating the transaction in a block and reaching consensus: this is done in a very similar way to how PBFT works. We go thoroughly through each step of our protocol:

Transaction Proposal. User client c_{bc} of public key $\mathsf{pk}_{bc} \in \mathsf{Clients}$ and secret key sk_{bc} applies Algorithm 1 where o_{bc} denotes the operation with payload Payload. EXEC takes in the operation and its payload, simulates the execution, outputs verDep and stateUpdate that refer respectively to the set of variables invoked by the operation, and to the result of the simulation (these are later used to prevent respectively double spending and non-deterministic execution). At the end of Algorithm 1, c_{bc} finds themself with $\mathsf{trans_{prop}}$ and its signature σ, which they send to the endorsement peers.

Transaction Endorsement. Each endorsement peer ep_{bc} of public key $\mathsf{pk}_{ep} \in \mathsf{Endor}$ and of secret key sk_{ep}, upon receiving the transaction proposal $\mathsf{trans_{prop}}$ and its signature σ, initiates Algorithm 2. The endorsement peer simulates the execution of the operation. If it yields different outputs as the one sent over (by checking verDep and stateUpdate), the algorithm outputs $\perp$. SecPolicies is an algorithm returning *valid* or *invalid* based on the blockchain endorsement policy and the current state of knowledge of the entity. If all checks out, the algorithm outputs a signature of the operation. Known optimization are possible using aggregation [7] or threshold [14] signatures to fasten the endorsement.

Broadcasting to Consensus. As user client c_{bc} collects a new endorsement of their operation, it applies Algorithm 3 until it has enough endorsements to actually send them to the orderers as a message $\mathsf{blob}=(\mathsf{trans_{prop}},\{\sigma_{ver,i}\}_{1\leq i\leq l})$.

Block Proposal. The orderer leader, using Algorithm 4, verifies the endorsements and checks SecPolicies: here this algorithm behaves

Algorithm 4 TxComOrd(NewBlock, blob, SecPolicies, $\{\mathsf{pk}_{ep,i}\}_{1\le i\le \mathsf{TotEnd}}$,$\mathsf{sk}_{ord_{lead}}$)

```
1: $\sigma \leftarrow blob.\mathsf{trans}_{\mathsf{prop}}.\sigma$
2: counter = 0
3: for all $\sigma \in$ blob.endorsements and $\mathsf{pk}_{ep,i}$ associated and not already used do
4:     if $\mathsf{Verify}_{\mathsf{pk}_\sigma}(\mathsf{trans}_{\mathsf{prop}},\sigma) = 1$ and $\mathsf{pk}_{ep,i} \in \mathsf{Endor}$:
5:         counter = counter + 1
6: if counter $< \lfloor \frac{\mathsf{TotEnd}}{2} \rfloor + 1$ or SecPolicies = invalid:
7:     return $\perp$
8: else if NewBlock.length = $\mathsf{Block}_{\mathsf{size}}$:
9:     $\sigma_{ord_{lead}} \leftarrow \mathsf{Sign}_{\mathsf{sk}_{ord_{lead}}}(\mathsf{NewBlock}||\mathsf{SecPolicies})$
10:    return NewBlock$||\sigma_{ord_{lead}}$
11: else
12:    return NewBlock $\leftarrow$ NewBlock||blob
```

Algorithm 5 Prepare($\mathsf{Block}_{\mathsf{size}}$, NewBlock, $\sigma_{ord_{lead}}$, SecPolicies, $\{\mathsf{pk}_{ep,i}\}_{1\le i\le \mathsf{TotEnd}}$,$\mathsf{sk}_{ord}$)

```
1: if $\mathsf{Verify}_{\mathsf{pk}_{leader}}(\mathsf{NewBlock}||\mathsf{SecPolicies},\sigma_{ord_{lead}}) \neq 1$:
2:     return $\perp$
3: Set counter = 0 and parse NewBlock = $\{\mathsf{blob}_i\}_{1\le i\le \mathsf{Block}_{\mathsf{size}}}$
4: for all $1 \le i \le \mathsf{Block}_{\mathsf{size}}$ do
5:     for all $\sigma \in \mathsf{blob}_i$.endorsements and $\mathsf{pk}_{ep,i}$ associated and not already used do
6:         if $\mathsf{Verify}_{\mathsf{pk}_\sigma}(\mathsf{blob}_i.\mathsf{trans}_{\mathsf{prop}},\sigma) = 1$ and $\mathsf{pk}_{ep,i} \in \mathsf{Endor}$:
7:             counter = counter + 1
8:         if (SecPolicies = invalid):
9:             return $\perp$
10:    if counter $< \lfloor \frac{\mathsf{TotEnd}}{2} \rfloor + 1$:
11:        return $\perp$
12: $\sigma_{ord} \leftarrow \mathsf{Sign}_{\mathsf{sk}_{ord}}(\mathsf{NewBlock}||\mathsf{SecPolicies})$
13: return (NewBlock,$\sigma_{ord}$)
```

such that it verifies that the verDep of $blob.\mathsf{trans}_{\mathsf{prop}}$ does not collide with the verDep of a transaction already added to the current block (this prevents any potential conflict among transactions). If all the verifications check out, then either the leader creates a block or waits for the next endorsed transaction.

Preparing Block. Upon receiving a block proposal, the orderer verifies the validity of the block proposed by the leader, see Algorithm 5. Sanity checks are conducted: verifying the authenticity of the newly proposed block, checking the validity of the transactions' approvals (*i.e.,* the signatures of the endorsers) and if the absolute majority of endorsement is reached. If none of the blockchain policies have been violated, the block is approved by broadcasting (NewBlock,σ_{ord}).

Add Block to Blockchain. When enough valid prepare messages (NewBlock,σ_{ord}) have been received by an orderer from its peers, it commits the changes to its local version of the blockchain. It sends a message approval to the group as detailed in Algorithm 6.

The orderers' consensus is highly inspired from PBFT [10], to which we have added endorsers and modified the messages' contents. The orderers mimic the protocol described in [10] for orderer leader's update in order to ensure safety and liveness. Note that what we call leader, they call *primary.* To quickly summarize the view change: if the leader is inactive or misbehaving, then another orderer can initiate a view change; it stops confirming new blocks, and proposes to the set of orderers to change leaders (as per a predetermined schedule). When enough of orderers reply positively, the view change happens.

Algorithm 6 BlockCom(NewBlock, $\{\mathsf{pk}_{ord_i}, \sigma_{ord_i}\}_{1\le i\le l}$, SecPolicies,$\mathsf{sk}_{ord}$)

```
1: if NewBlock.length $\neq \mathsf{Block}_{\mathsf{size}}$ or $l \le \lfloor \frac{\mathsf{TotOrd}}{3} \rfloor$:
2:     return $\perp$
3: for all $1 \le i \le l$ do
4:     if $\mathsf{Verify}_{\mathsf{pk}_{ord_i}}(\mathsf{NewBlock}||\mathsf{SecPolicies},\sigma_{ord_i}) \neq 1$:
5:         return $\perp$
6: $\sigma_{BlockCom} \leftarrow \mathsf{Sign}_{\mathsf{sk}_{ord}}(NewBlock)$
7: return $NewBlock||\sigma_{BlockCom}$
```

Network Model. We assume an asynchronous distributed system where nodes are connected by a network who may fail to deliver messages, delay them, duplicate them or deliver them out of order. We allow for the adversary to coordinate faulty nodes, delay communication, or delay correct nodes but not indefinitely.

4 PRIVACY PRESERVING BLOCKCHAINS

In Section 3 we introduced a blockchain based on PBFT and requiring a signature scheme. Our construction can be extended in order to allow multiple privacy preserving settings. We can bring anonymity for the issuers and pseudonymity for issuers, endorsers and/or orderers through the use of privacy preserving signatures. In particular, we use blind signatures, group signatures and ring signatures. All these privacy preserving settings can be achieved independently for any of the defined roles. In Section 5, we show that composing any of these settings gives a secure blockchain.

4.1 User's Anonymity

Based on Blind Signatures. Considering the permissioned BFT-based consensus protocol introduced in Section 3, users sign their transactions with own key. This causes a strong linkability issue between the users and the transactions, affecting the privacy level of the blockchain network. In order to overcome this issue, we propose to hide the users' identities by using blind signatures with a trusted entity. Thus, the protocol follows the following steps:

(1) The client c_{bc} authenticates themselves with their registered keys ($\mathsf{pk}_{c_{bc}}$,$\mathsf{sk}_{c_{bc}}$) to one of the membership authorities $\mathbf{A}$,
(2) Once the authentication succeeds (*i.e.,* $\mathsf{pk}_{c_{bc}} \in$ Clients), the client initiates a blind signature process with $\mathbf{A}$,
(3) The client derives a signature σ for its transaction request,
(4) We apply the consensus protocol introduced in Section 2.1 on the client's blindly signed transaction. The blind signature authorize the client to transact on the blockchain.

Aiming to keep the same structure as the original construction, we replace the user's ID with a random value crand. Now, to address the issue related to the digital signature, linking the client to a transaction, we replace it with a blind signature scheme. TxProp defined in Algorithm 7 is the modified version of the original TxProp of Algorithm 1. To maintain consistency and liveness, we keep the rest of the transactional flow unchanged. However, some steps are modified to accept the blind signature scheme to authenticate the clients and the peers. This variant of SignCons is called BlindCons.

Based on Group Signatures. In the previously presented transaction mechanism, every transaction must first go through the authority to be blind signed before anything else can be done with it. Using

Algorithm 7 TxProp(o_{bc},Payload,$\mathsf{sk}_{c_{bc}}$)

```
1: (verDep,stateUpdate) $\leftarrow EXEC(o_{bc}$,Payload)
2: $\mathsf{crand}_{bc} \xleftarrow{\$} \mathbb{N}$
3: $\mathsf{trans}_{\mathsf{prop}} \leftarrow \mathsf{crand}_{bc}||o_{bc}||$Payload||verDep||stateUpdate
4: $\sigma \leftarrow \mathsf{BlindSign}\langle \mathcal{U}(\mathsf{trans}_{\mathsf{prop}},\mathsf{pk}_{auth},\mathsf{sk}_{c_{bc}}),\mathbf{A}(\mathsf{sk}_{auth})\rangle$
5: return ($\mathsf{trans}_{\mathsf{prop}}$,$\sigma$)
```

a group signature there is a way to obtain a decentralized transaction proposal mechanism. Let GS = ($\mathsf{KeyGen}_{\mathsf{group}}$, GroupSign, $\mathsf{Verify}_{\mathsf{group}}$, OpenGroupSign) be a group signature scheme with its usual security requirements (outlined in Section 2.3), this next variant of our scheme enables any registered user to sign hiding amongst the group of authorized users. We assume that authorized users are registered with the authority and that a public record of all of them is available. Hence, all keys are generated through a protocol with the authority and registered in the Clients record.

We call GroupCons the variant of SignCons adopting group signatures. This version is instantiated by replacing the signature in Algorithm 1 (as well as the signature verifications) with a group signature where the group consists of all the people having rights to write in the blockchain. By using group signatures instead of blind signatures, we can make the protocol less relying on the authority while simultaneously giving the authority power to reveal a message's signer if need be. This revealing can be done using the OpenGroupSign algorithm. Compared to the blind signature construction, group signatures also limit the computational load on the authority as it no longer needs to execute its part of the blind signature protocol for each new transaction.

Based on Ring Signatures. This version is instantiated by replacing the signatures in Algorithm 1 (as well as the signature's verifications in Algorithm 2) with a ring signature where the ring consists of (potentially all the) authorized users. We call this variant RingCons. We thus come up with a private blockchain architecture with no authority where transactions are unlinkable to their issuers. Let $U = u_1,\ldots,u_n$ be the set of authorized users. Suppose u_1 wants to make a transaction. In the regular SignCons protocol, u_1 would sign its transaction using a regular signature protocol (Algorithm 1), and then send it for endorsement (Algorithm 2). To anonymize the user's identity we change the type of signature: u_1 signs the transaction using a ring signature in the name of U (or in the name of a subset of U if U is too big). That way the endorsers verifies the ring signature in Algorithm 2, and can thus know that it was indeed someone of S that produced the signature without being able to know which member it was (since it is a ring signature). The rest of the protocol remains unchanged.

4.2 User's Pseudonymity

Let GS_{link} be a *linkable group signature* and let RS_{link} be a *linkable ring signature*. We keep on relying on the same idea and operate these two types of signatures. In this modified version of our blockchain, the signature in the anonymized version of TxProp (Algorithm 1) is replaced by one of these linkable signatures, again the verification in Algorithm 2 is modified adequately. We call the version using linkable group signature LinkRingCons and the other version using linkable ring signature LinkGroupCons. These two primitives retain the properties of being unlinkable to the signature issuer, however they allow for linking the transaction to other transactions signed by the same secret key using the Link algorithm (see Section 2.3). As such, signers remain unknown, but one can track all the transactions created by the same entity. We emphasize in Section 5 why this achieves pseudonymity, it is straight forward to see that it does not achieve anonymity due to the linkability of the signatures. LinkGroupCons being based on a group signature requires an authority to be implemented, and gives it the power to revoke the privacy of the signer. On the other hand, LinkRingCons requires no authority and naturally is not revocable.

4.3 Endorser and Orderer Pseudonymity

In the current scheme, the endorsers and orderers are respectively linked to the transactions they endorse and to the blocks they commit. Enabling endorsers and orderers to use group or ring signature would not be fruitful, since it enables each entity to produce an unlimited number of different signatures without being detected, which is problematic since we need to count the number of approvals. Using linkable group signatures and linkable ring signatures does help us keep the endorsers' and the orderers' privacy, while at the same time restricting just enough of the excessive anonymity that is brought by group and ring signatures.

Thus, taking the same perspective on this as in Section 4.2, we can use linkable group signatures and linkable ring signatures in Algorithm 2 for the endorsers and call it respectively EndGroupCons and EndRingCons. The same modification is possible in Algorithms 5 and 6 for orderers, we call these protocols OrdGroupCons and OrdRingCons. Note that an extra step needs to be added after verifying the signature (with $\mathsf{Verify}_{\mathsf{group}}$ and $\mathsf{Verify}_{\mathsf{ring}}$): the verifier must also check if this signature can be linked to another signature of the same content (transaction or block) before taking it into account. For endorsers' pseudonymity, this extra verification is done in Algorithms 3 (executed by the orderer leader) and 4 (executed by the orderers). As for orderers, this is done in Algorithm 6. Note also that the orderer leader cannot use a pseudonym with these constructions, since the other orderer peers must be able to check that the leader's status and that this allows block proposal.

We claim, and later prove (in Section 5), that it is impossible to link the endorsers (respectively orderers) to their transaction endorsement (respectively block generation), while concurrently not allowing them to produce multiple acceptable signatures for the same transaction endorsement (respectively block generation).

All the presented layers of constructions maintain a Byzantine Fault Tolerant blockchain. They all allow some privacy for entities of various roles. In fact, it is possible to combine any construction for users (from Sections 4.1 and 4.2) with any of the constructions for endorsers and orderers from the current Section.

5 PROTOCOL PROPERTIES

Under two hypotheses, our constructions satisfy *Safety*, *Liveness*, *Unforgeability* of a block and some privacy preserving settings, namely, *Anonymity* or *Pseudonymity*. The latter can be obtained for any of the three existing roles: issuers, endorsers and orderers and in any possible settings. For example, it is proven that composing anonymity of the clients with pseudonymity for the endorsers is still secure. Safety and Liveness are both inherited from PBFT, leading to the first hypothesis being that the adversary cannot delay correct nodes indefinitely. We model and prove the other security properties using game based formalism and reductions. In general, we consider a security experiment where a PPT challenger C interacts with a PPT adversary $\mathcal{A}$. The adversary simulates the behaviour of a malicious entity, while the challenger runs the rest of the system honestly. Based on these, we show that the full security of our protocols also depends on the secure primitives used for instantiating it. Any secure signature could instantiate ours blockchains.

Safety. A protocol is said to be *consistent* if it ensures that a transaction generated by a valid user stays immutable in the blockchain. Our blockchains are based on PBFT which leads a consensus for each of the deciding steps of our blockchain.

Definition 5.1 (Safety). A protocol $\mathcal{BC}$ is T*-safe* if a transaction tx generated by an honest client c_{cb} to execute a valid operation o_{bc}, is confirmed and stays immutable in the blockchain after T-round of new blocks.

Theorem 5.2. *Our new protocols based on PBFT are 1-safe if at most* $\lfloor \frac{n-1}{3} \rfloor$ *out of total n orderer peers are malicious.*

Proof sketch. The described protocols are BFT based consensus. Safety is achieved by agreeing with the validity of the transaction through a byzantine agreement process. Hence, for a transaction tx that has reached a majority of valid endorsement for an operation o_{bc}, the probability of not settling it in a new block and having forks in the chain is neglected if we have at most $\lfloor \frac{n-1}{3} \rfloor$ malicious orderers, out of total n orderers as it has been shown in [8, 10]. It is 1-consistent because we do not have any fork; hence only one block is needed to wait to have a transaction validated. □

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

Definition 5.3 (Liveness). A consensus protocol $\mathcal{BC}$ ensures *liveness for a blockchain C* if $\mathcal{BC}$ ensures that after a period of time t, the new version of the blockchain C' verifies $C' > C$, if a valid client $c_{i_{bc}}$ has broadcasted a valid transaction tx_i during the time t.

Theorem 5.4. *Our protocols satisfy liveness when at most* $\lfloor \frac{\mathsf{TotEnd}-1}{2} \rfloor$ *out of a total of* TotEnd *endorsers and* $\lfloor \frac{\mathsf{TotOrd}-1}{3} \rfloor$ *out of total* TotOrd *orderers are malicious.*

Proof sketch. Our protocols are BFT-based consensus protocol. Thus, liveness is achieved if after the transaction endorsement process, the ordering services propose a new block NewBlock with the transactions broadcast by the clients during a period of time t. Hence, for valid transactions tx_i (*i.e.,* accepted by the endorsers), where $i \in \mathbb{N}$, issued by valid a client 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 orderers. A detailed proof for the type of consensus we are using can be found in [9]. □

Unforgeability. An adversary against the *unforgeability* of a protocol tries to overstep the validation process of a transaction in order to engrave a transaction in the blockchain without obtaining the full transaction acceptance from the endorsers and the orderers. Our blockchain composes a validation procedure conducted by the Issuer with the endorser and a consensus agreement made by the orderers. This property ensures that these compositions retains security *i.e.,* no adversary could possibly overstep the validation procedure nor the consensus to engrave a block in the blockchain. The formalism of this property is postponed to the technical report [1], whereas a discussion on the arguments is conducted below.

Theorem 5.5 (informal). *Consider a blockchain* $\mathcal{BC}$ *defined by one of the above settings,* i.e.,*instantiated with some of the above described signatures. For any security parameters* $\mathfrak{K}$, $\mathcal{BC}$ *is unforgeable.*

The proof of this property, relies on the unforgeability of the signatures used in $\mathcal{BC}$. While the versions using linkable group (*resp.* ring) signature for the endorsers also depend on the traceability (*resp.* linkability) of the signatures. Otherwise, it would be possible for a single node to output multiple signatures. Hence, it is infeasible to append an invalid transaction to the existing blockchain without being given the endorsers and orderers agreement.

Pseudonymity. A entity $\mathcal{E}$ and a witness w are said to be *linked* in a group G's perspective, if it is possible for G to infer that $\mathcal{E}$ produced w based on the available information to G. *Pseudonymity* of an entity holds when $\mathcal{E}$ cannot be linked to the witnesses $w_1,\ldots,w_k$ it has produced, but this property does not prevent from linking the witnesses one to each other. When linking w_i and w_j is hard for all $1 \leq i < j \leq k$, it is considered as a stronger privacy preserving property called *anonymity.* We split the actors of our blockchain into two groups, on one side the users and on the other side the endorsers and orderers to consider their pseudonymity.

Users. Let $\mathcal{A}$ be an attacker against an user *pseudonymity.* Its goal is to link it to a transaction tx it has produced. Nevertheless, we assume that it is only possible to link tx to the client using information from the blockchain. As a consequence, we assume that it would be hard to identify the provenance of a transaction due to the redundancy of the sent communications as upon receiving a message each entity broadcasts it to all its peers (gossip). This is a classical assumption in blockchain. In order fulfil pseudonymity, o_{bc} and Payload should not leak information on the transaction requester. A public transaction always reveals a certain amount of information as this information is publicly enclosed, here we show that no additional information is revealed throughout the protocol.

Theorem 5.8 (informal). *Consider secure* BS *a blind signature,* GS *a (linkable) group signature and* RS *a (linkable) ring signature. Assume that an adversary* $\mathcal{A}$ *is unable to identify a user at the origin of a transaction based on* o_{bc} *and* Payload*. Then the consensus presented in Section 4.2, instantiated with these signatures is pseudonymous.*

Users sign their transaction with a signature mechanism and sends it to the nodes. Once this is executed, they are no longer involved in the process. Hence, what they output should be unlinkable to them. This result relies directly on the anonymity properties of the signatures considered in the article.

Endorsers and Orderers. Endorsers and orderers can be unlinked from transactions and blocks they validated through the use of *linkable ring or group signatures.* The linking algorithm Link allows this feature, hence, enabling detection of nodes producing multiple validations. The signature anonymity requirement prevents from recovering the identity of the executant.

Theorem 5.9. *Base on a secure linkable group* (resp. *ring) signature the EndGroupCons,* (resp.*EndRingCons) protocol is pseudonymous for the endorsers. Under the same conditions, the protocols OrdGroupCons and OrdRingCons are pseudonymous for the orderers (excluding the orderer leader).*

Arguments supporting these properties are the same the ones evoked before for the users. A detailed argumentation of these properties and their proofs is given in the Technical report [1].

Anonymity of the users. As stated before, anonymity is defined by two requirements: (i) there should be no link between the clients and the transactions they produced, (ii) transaction of the same user should not be linkable. We yet know that BlindCons, RingCons, GroupCons securely realize user's pseudonymity, statement (i). In fact, it is also possible to show that (ii) holds, as it appears that the output transaction are in these cases unlinkable one to each other.

Theorem 5.10. *Given that the client proceeds to a secure blind, ring or group signature to authenticate its transaction as defined in Section 4, anonymity of the client holds.*

This theorem relies on the unkinkability of the signatures produced by a user. The statement is proven in the technical report [1].

6 COMPLEXITY

Our approach is generic, hence allows to instantiate the protocol with the most efficient signature schemes in the literature. In order to provide a theoretical evaluation that can benefits from further works on these primitives, we evaluate the number of executions of each algorithm for the various entities in the blockchain. Let S_C be the signature used by the clients, S_E the signature used by the endorsers and S_O used by the orderers. These signatures can refer to any of the signature schemes used in our blockchains as specified through Section 3 and 4 due to modularity of our proposed constructions. In order to obtain a validated transaction, a client needs to execute S_C once and verify $\lfloor \mathsf{TotEnd}/2 \rfloor + 1$ signatures S_E. In the meantime, an endorser needs to verify a signature from S_C once and produce one signature S_E. Now, in order to validate a block, the orderer leader needs to verify $\mathsf{Block}_{\mathsf{size}}$ signatures S_C, $\mathsf{Block}_{\mathsf{size}} \cdot (\lfloor \mathsf{TotEnd}/2 \rfloor + 1)$ signatures S_E and later $2(\lfloor 2\mathsf{TotOrd}/3 \rfloor + 1)$ signatures S_O. It also needs to produce two regular signatures. The orderer verifies these signatures and $\mathsf{Block}_{\mathsf{size}} \cdot (\lfloor \mathsf{TotEnd}/2 \rfloor + 1)$ signatures from the endorsers and $2(\lfloor 2\mathsf{TotOrd}/3 \rfloor + 1)$ signatures from other orderers. An orderer also produces two signatures S_O.

State-of-the-art blind, group or ring signature are known to be less efficient than regular signature schemes. All still achieve constant execution time [6, 16, 23]. Also their longer computation time must be put into perspective with the time needed for the numerous communications require by a blockchain. As the order of magnitude of a signature execution does not generally exceed the order of magnitude of a RTT (Round-Trip Time), the overhead brought by bringing anonymity to our blockchain seems acceptable. Blind signature usually requires 2 (at best) or 3 additional communication yielding as much aditional communications. Again this does not increase much the number of communications of the blockchain, hence has low impact on the performance. On another hand group and ring signatures requires to obtain the keys of the members of a ring. In the case of group signatures, we can assume that they are all provided by the registration authority on demand. This considered, for an equivalent level of security, our protocol is expected to be less efficient than a blockchain without any anonymity, as a counterpart it brings more security for its peers as their identity is not publicly disclosed.

7 CONCLUSION

In this paper we bring forth a blockchain solution to the apparent dilemma of combining private permissioned blockchains with privacy. We divide the miners into endorsers and orderers. And we propose different constructions for different entities privacy (user, endorser or orderer). The different constructions use different building blocks and have different use cases. Some yield anonymity and others pseudonymity, some rely on an authority, some propose privacy revocation rights, some require less computation than others. We sketched security arguments providing the security of our construction. Arguments based on game-based proofs show their security in [1]. As future works, we envision extending our results in the UC model, by considering the ideal functionality presented in [17].

REFERENCES

[1] Frederic A. Hayek, Mirko Koscina, Pascal Lafourcade, and Charles Olivier-Anclin. 2022. Generic Privacy Preserving Private Permissioned Blockchains. https://hal.uca.fr/hal-03906880.
[2] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. 2018. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *EuroSys conference*.
[3] Aritra Banerjee, Michael Clear, and Hitesh Tewari. 2020. Demystifying the Role of zk-SNARKs in Zcash. In *IEEE conference on application, information and network security (AINS)*.
[4] Adam Bender, Jonathan Katz, and Ruggero Morselli. 2009. Ring signatures: Stronger definitions, and constructions without random oracles. *Journal of Cryptology* (2009).
[5] Jorge Bernal Bernabe, Jose Luis Canovas, Jose L Hernandez-Ramos, Rafael Torres Moreno, and Antonio Skarmeta. 2019. Privacy-preserving solutions for blockchain: Review and challenges. *IEEE Access* (2019).
[6] Dan Boneh, Xavier Boyen, and Hovav Shacham. 2004. Short Group Signatures. In *Advances in Cryptology – CRYPTO 2004*, Matt Franklin (Ed.).
[7] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. 2003. Aggregate and verifiably encrypted signatures from bilinear maps. In *EUROCRYPT*.
[8] Gabriel Bracha and Sam Toueg. 1985. Asynchronous consensus and broadcast protocols. *Journal of the ACM (JACM)* (1985).
[9] Miguel Castro and Barbara Liskov. 1999. A correctness proof for a practical Byzantine-fault-tolerant replication algorithm. Tech report MIT/LCS/TM-590.
[10] Miguel Castro, Barbara Liskov, et al. 1999. Practical Byzantine fault tolerance. In *OSDI*.
[11] David Chaum and Eugène van Heyst. 1991. Group signatures. In *EUROCRYPT*. Springer.
[12] Jollen Chen. 2018. Hybrid blockchain and pseudonymous authentication for secure and trusted IoT networks. *ACM SIGBED Review* (2018).
[13] George Danezis and Seda Gürses. 2010. A critical review of 10 years of privacy technology. *Proceedings of surveillance cultures: a global surveillance society* (2010).
[14] Yvo Desmedt and Yair Frankel. 1989. Threshold cryptosystems. In *CRYPTO*.
[15] Linux Foundation. 2019. Hyperledger. https://www.hyperledger.org/.
[16] Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. 2015. Practical Round-Optimal Blind Signatures in the Standard Model. In *CRYPTO*. Springer.
[17] Mike Graf, Daniel Rausch, Viktoria Ronge, Christoph Egger, Ralf Küsters, and Dominique Schröder. 2021. A security framework for distributed ledgers. In *ACM SIGSAC*.
[18] Thomas Hardjono and Alex Pentland. 2019. Verifiable Anonymous Identities and Access Control in Permissioned Blockchains. *CoRR* abs/1903.04584 (2019).
[19] Ryan Henry, Amir Herzberg, and Aniket Kate. 2018. Blockchain access privacy: Challenges and directions. *IEEE Security & Privacy* (2018).
[20] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. 2016. Zcash protocol specification. *GitHub: San Francisco, CA, USA* (2016).
[21] Aram Jivanyan. 2019. Lelantus: Towards Confidentiality and Anonymity of Blockchain Transactions from Standard Assumptions. *IACR ePrint.* (2019).
[22] Leslie Lamport. 2019. Time, clocks, and the ordering of events in a distributed system. In *Concurrency: the Works of Leslie Lamport*.
[23] Giulio Malavolta and Dominique Schröder. 2017. Efficient ring signatures in the standard model. In *CANS*.
[24] Ian Miers, Christina Garman, Matthew Green, and Aviel D Rubin. 2013. Zerocoin: Anonymous distributed e-cash from bitcoin. In *IEEE S & P*. IEEE.
[25] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008).
[26] Shen Noether, Adam Mackenzie, and the Monero Research Lab. 2016. Ring confidential transactions. *Ledger* (2016).
[27] David Pointcheval and Jacques Stern. 2000. Security Arguments for Digital Signatures and Blind Signatures. *J. Cryptol.* (2000).
[28] Fergal Reid and Martin Harrigan. 2013. An analysis of anonymity in the bitcoin system. In *Security and privacy in social networks*. Springer.
[29] Ronald L Rivest, Adi Shamir, and Yael Tauman. 2001. How to leak a secret. In *ASIACRYPT*.
[30] Alexander Schaub, Rémi Bazin, Omar Hasan, and Lionel Brunie. 2016. A trustless privacy-preserving reputation system. In *IFIP SEC*.
[31] Yonatan Sompolinsky and Aviv Zohar. 2015. Secure high-rate transaction processing in bitcoin. In *FC*.
[32] Shi-Feng Sun, Man Ho Au, Joseph K Liu, and Tsz Hon Yuen. 2017. Ringct 2.0: A compact accumulator-based (linkable ring signature) protocol for blockchain cryptocurrency monero. In *ESORICS*.
[33] Nicolas Van Saberhagen. 2013. CryptoNote v 2.0. (2013).