sigBridge: A Cross-chain Bridge for Permissioned Blockchains *and its application to decentralized access control*

Mahmudun Nabi
University of Calgary, Alberta, CA
mahmudun.nabi1@ucalgary.ca

Sepideh Avizheh
University of Calgary, Alberta, CA
sepideh.avizheh1@ucalgary.ca

Preston Haffey
University of Calgary, Alberta, CA
pjjhaffe@ucalgary.ca

Reihaneh Safavi-Naini
University of Calgary, Alberta, CA
rei@ucalgary.ca

Marc Kneppers
Telus Communications Inc., CA
marc.kneppers@telus.com

Abstract

With the rise of decentralized systems and applications that run over multiple blockchains, there is a growing need for architectures and bridges that ensure the trusted transfer of data and assets between chains. zkBridge (ACM CCS'22) is a cross-chain bridge protocol that was proposed for non-permissioned blockchains and uses a network of relays, each working (for example as a light node) on a blockchain. A relay communicates with a corresponding smart contract on another chain to transfer data from the first chain to the second, and the smart contract on the second chain is used to verify the correctness of the received data from the first chain. zkBridge designs and implements optimized zero-knowledge proofs, that minimize the work of the smart contract on the second chain to verify the correctness of the received data. In this paper, we consider applications that work across two or more permissioned blockchains. We propose sigBridge that uses the framework of zkBridge but replaces the costly zero-knowledge proof computation with a pair of algorithms based on the consensus algorithm of the first chain. The algorithms will be run by a relay node on the first chain and a smart contract on the second chain, and provide verifiability of data and asset transfer from the first chain to the second. The pair of algorithms are significantly more efficient compared to generating zero-knowledge proofs, verifying them, or running the full consensus algorithm of the first blockchain. We then show how a decentralized user-centric resource-sharing application will work over this architecture. We give a proof-of-concept implementation of an attribute-based access control system for a resource-sharing application that runs over two private Ethereum blockchains, and report the computation costs of the protocol.

Keywords: Cross-chain Bridge, Permissioned blockchain, Smart contracts, Decentralized access control, Attribute-based resource sharing

1. Introduction

Blockchains provide a powerful framework for the design and implementation of decentralized applications that are run transparently and can use the native coin of the chain for automated paid exchange of goods and services. There is a wide range of blockchain consensus algorithms where some rely on Proof-of-Work (PoW), for example Bitcoin [Nakamoto (2008)] and Ethereum Classic [Buterin (2016)], some use Proof-Of-Stake (PoS) such as Cosmos [Kwon and Buchman (2019)] and Polkadot [Wood (2016a)], and some use other algorithms. Future blockchain ecosystems will include multiple co-existing networks that use cross-chain protocols to transfer assets and data between chains. Systems like Cosmos and Polkadot are examples of such ecosystems. Enabling applications to access resources of a multi-chain ecosystem will open a plethora of new possibilities including access to wider and richer assets and services, and/or having wider user reach. The main challenge of these applications however is to have access to secure cross-chain bridges that enable assets on one chain to be securely offered, exchanged, and paid for on other chains. This can be achieved by a system of bridges where a bridge between blockchains BC_1 and BC_2 proves to an application on BC_2 that a certain event has happened on BC_1, and vice versa. There are numerous cross-chain architectures and protocols for data/asset transfer [Sekniqi et al. (2020), A. Team (2021), P. Team (2020), Xie et al. (2022), and Zarick et al. (2021)] proposed for specific blockchains such as

Cosmos, Polkadot.

zkBridge [Xie et al. (2022)] is a modular bridge infrastructure system that facilitates the communication between the chains, and ensures the correct transfer of assets and messages between them. In zkBridge framework, there is a network of *relays* that enable the communication between the chains, and secure communication is guaranteed if at least one relay works correctly. For communication of blockchain BC_1 to BC_2, a relay will run a light client[1] LN_1 on BC_1 that will communicate with a smart contract SC_2 on BC_2, that keeps track of block headers and events of BC_1 and verifies its block headers. A similar arrangement can be made in PoS chains. Verification of block headers is a costly operation [Xie et al. (2022)] and zkBridge's innovation is to use zk-SNARK (Succinct Non-interactive Argument of Knowledge) [Groth (2016)] to enable LN_1 to efficiently convince SC_2 that the claimed state transition on BC_1 has taken place.

Our Work. We focus on permissioned blockchains and applications that run on multiple permissioned chains. We propose *sigBridge* (signature bridge), a cross-chain bridge architecture that employs the less costly execution of smart contracts on permissioned blockchains, to replace the ZK (zero-knowledge) proof with a transformation that uses signatures of consensus nodes and allows efficient computation for SC_2 and LN_1. We apply our approach to two permissioned blockchain systems BC_1 and BC_2 that communicate with each other and analyze the resulting sigBridge. We also consider a resource-sharing application that uses smart contracts to enable users to define and enforce their access control policies (attribute-based) for their shareable objects, and show how the application can be employed over two blockchains using sigBridge. We implement a proof-of-concept system and evaluate the execution cost of SC_2.

sigBridge for permissioned blockchains. We adopt the zkBridge framework that uses a network of relay nodes, each implementing a light node in one blockchain (i.e., the counter blockchain), and communicating with a smart contract in their home blockchain. For concreteness, we consider transmission from BC_1 to BC_2, where the light node LN_1 is run on BC_1 and the smart contract SC_2 is deployed on BC_2. SC_2 will be initialized with the latest block header [$data$, sig_of_con] of BC_1, where $data$ is the block header data (in BC_1), and sig_of_con is the signature set of the consensus algorithm of BC_1. SC_2 also has the public keys that are required for the verification of signatures. This initialization step is trusted. For a new block header in BC_1, LN_1 will provide sufficient information to SC_2 to verifiably update its state and store the correct new block header [$data'$, sig_of_con']. sigBridge uses a pair of transformations (T, V) that are designed for the consensus algorithm of BC_1, and allow SC_2 to verify the correctness of the new block header. Our main observation is that in permissioned chains, smart contracts can directly use the verification algorithm of digital signature schemes that are used in the consensus algorithm. We can reduce the computation of SC_2, by using a probabilistic verification algorithm V, and design T to replace the full consensus algorithm of BC_1 with a more efficient algorithm. In section 4, we show how the error probability of the probabilistic verification varies with the choice of the system parameters. In section 6, we implement our proposed approach for a Cosmos-like consensus algorithm, and show the trade-off between efficiency and correctness probability (effectiveness).

Application to decentralized user-centric access control. We consider a four-layer architecture for communication between permissioned blockchains, consisting of (i) message passing, (ii) consensus, (iii) cross-chain synchronization, and (iv) application [Jin et al. (2018)]. *sigBridge* implements layer (iii) and provides the required infrastructure for layer (iv). We consider a resource-sharing application based on [Muni et al. (2020)], that uses smart contracts to publish information about users' shareable resources (digital objects) and the conditions of access. The application ensures that access requests are granted in accordance with the stated policies. We use attribute-based access control that describes policies in terms of attributes of objects and users and the required relation between them. Policy evaluation is done by matching attributes and satisfying the specified relations. We provide a proof-of-concept implementation using private Ethereum blockchains. We measure the *gas* cost for on-chain signature verification in sigBridge where 2/3 signatures of consensus nodes are required to validate a block header, and compare this to the on-chain cost of zkBridge[2].

Organization. Section 2 discusses related work, the preliminaries are presented in section 3, and section 4 describes the sigBridge protocol construction. Section 5 gives the resource-sharing application details, section 6 describes our implementation, and section 7 concludes the paper.

[1]A light node only records the block headers.

[2]For permissioned blockchain, we estimate the number of hash operations, and convert it to *gas*.

2. Related work

Bridges. Existing approaches either rely on honest majority committees [Ronin (2021) and A. Team (2021)], or employ light client protocols to synchronize the views of the two chains [Kwon and Buchman (2022), Wood (2016a), and Xie et al. (2022)], or use an incentive-based approach [N. R. Bridge (2022) and N. Bridge (2022)].

Cross-chain communication protocols such as cross chain message passing [XCMP (2015)], inter blockchain communication [IBC (2021)], and optimistic interchain communication [OPTICS (2022)] employ *light client* and *relayer* mechanisms to provide interoperability among blockchains. XCMP is tailored for connecting homogeneous chains within Polkadot [Wood (2016b)], while IBC bridges heterogeneous chains. Optics has similarities with IBC and is used in Celo blockchain [Celo (2019)] and other EVM-compatible chains.

The above bridge protocols are all designed for non-permissioned blockchains.

Smart contract-based access control has been considered for systems that are used in healthcare [Salonikias et al. (2022)] and IoT [Ouaddah et al. (2016), Xu et al. (2018), and Zhang et al. (2018)], supply chain [Li et al. (2023)], finance and banking [Liao et al. (2022) and Sumathi and Sangeetha (2020)], resource sharing [Avizheh et al. (2021) and Muni et al. (2020)] and many more. The schemes in [Avizheh et al. (2021) and Muni et al. (2020)] propose decentralized access control systems for resource sharing in a smart neighborhood using a single permissioned blockchain. Our work is based on [Muni et al. (2020)] and extends it to two chains using sigBridge.

3. Preliminaries

Blockchain. A blockchain (BC) is a set of mutually distrusting nodes that use a *consensus* algorithm to agree on the events in the system that are recorded as a sequence of blocks in a ledger. A block consists of a header $blkHdr$ and a list of transactions $[tx_1, ..., tx_n]$. The header contains the hash of the previous block and a Merkle root ($Mroot$) of the block's transactions. A secure blockchain ensures *consistency* and *liveness*, that guarantee that the views of honest nodes are consistent, and transactions are eventually included in the ledger. Nodes in BC can be a *full node* (FN) or a *light node* (LN) [Burr et al. (2018)]. *Full nodes* store the entire ledger, validate transactions, and participate in the consensus protocol to add them to the ledger. *Light nodes* store only block headers, do not store transaction data, and do not participate in consensus. They use a "*light client (LC) protocol*" to retrieve data from full nodes and verify transactions in the ledger. In permissioned blockchains, access to the network and participation in consensus are restricted to authorized parties and is determined by the governance structure of the system.

Light client (LC) protocol describes how light nodes securely update their list of block headers and verify inclusion proofs for transactions. Given a light node's block history $blkHisdtry$ with block $blkH_{i-1}$ at its head, a new block header $blkH_i$ is validated by running: $VerifyHeaderUpdate(hdrHistory[i-1], blkH_{i-1}, blkH_i) \to \{0,1\}$. It also verifies if tx is in $blkH$ by retrieving a Merkle proof π_{tx} from full nodes, then checking them against the Merkle root (in $blkH$) using the function $VerifyTx(tx, \pi_{tx}, Mroot) \to \{0,1\}$. The security properties of LC protocol are: *Consistency* (i.e. the stored headers are consistent with the ledger of honest full nodes), *Liveness* (a tx received by an honest FN will be included in a block and the related block header will eventually be stored by LN).

Attribute-based access control (ABAC) uses attributes of the access requester, the object being accessed, and the environment of both to grant/deny access. Attributes define the properties of each entity and are assigned by an authority [Hu et al. (2013)]. They are represented with key-value pairs (e.g., "age:15"). Boolean expressions over attributes define the access policy for an object. For example, the policy *"Older than 12 and from Europe or Asia"* is represented by, (Age $>$12) $\wedge$ (Region = Europe) $\vee$ (Region =Asia). *User-centric attribute-based access control* allows users to specify their desired policies.

Merkle tree and Merkle proof. A *Merkle tree* is a binary tree constructed using a collision-resistant hash function to represent the hash values of data blocks, with the root node representing the hash value of all the data blocks. A *Merkle proof* is a list of hash values from the leaf node containing the data block up to the root node, used to verify a specific data block's presence in the tree.

4. sigBridge framework and protocol

We consider a four-layer architecture for interoperability of permissioned blockchains, in-line with [Jin et al. (2018)], and focus on two chains, BC_1 and BC_2 to design sigBridge.

Message passing layer is responsible to relay messages between two blockchains and is divided into two sub-layers: (i) *Communication and Relay* has multiple components; *relay client* to receive, verify, translate and sign messages, *relay connection* that establishes connections with full nodes, *translation* module for message formatting and translation rules, and *security* module for cryptographic operations. (ii) *Relay manager* handles the management of relayers, including their registration with both blockchains and incentives for their honest participation.

Consensus layer guarantees that all honest nodes can achieve consensus on the state and order of blocks in the ledger, even in the presence of malicious nodes. While this layer includes various functionalities, our primary focus is on the consensus protocol.

Cross-chain synchronization layer ensures that the received information originates from the remote ledger. It consists of the components, *updater* module that is responsible for receiving and validating block headers from remote blockchains, *cross-chain consensus rules* module, that provides consensus rules for verifying block headers, and a *verification manager* module which interacts with the application layer and updater module to verify that the received transactions exist in the remote ledger.

Application layer where applications are deployed and executed.

4.1. *sigBridge*: Signature of consensus-based bridge in a permissioned blockchain

A cross-chain bridge enables communication between BC_1 and BC_2, such that an application on BC_1 can prove to an application on BC_2 that a certain event has happened on BC_1 (and vice versa). *sigBridge* uses the signatures of BC_1's consensus protocol as the core part of the proof. *sigBridge* adapts the zkBridge [Xie et al. (2022)] approach to the permissioned setting.

Our main observation is that, compared to permissionless setting, on-chain computation in permissioned blockchains is not costly. The consensus logic is simple and relies on the signature of the consensus nodes who will sign the block headers using a multi-signature scheme. Also, in most cases, the required number of signing nodes is less than the number of consensus nodes. Therefore, the consensus signatures can be directly sent to the smart contract on BC_2 to verify the received block headers from BC_1 by validating signatures according to the consensus rules of BC_1. This also removes the need for coordination of relayers as needed in zkBridge (to distribute the proof generation). In the following, we describe *sigBridge* in more detail for sending block headers from BC_1 to BC_2. The communication is symmetric and the bridge from BC_2 to BC_1 works in the same way.

sigBridge. We consider two application smart contracts $\mathcal{AC}_1$ running on BC_1 and $\mathcal{AC}_2$ running on BC_2. The bridge $sigBridge[\mathcal{AC}_1 \rightarrow \mathcal{AC}_2]$ passes the information from $\mathcal{AC}_1$ to $\mathcal{AC}_2$. For security and system models, we follow zkBridge.

sigBridge protocol entities. We define a pair of transformations (T, V) that are designed according to the consensus mechanism of BC_1. We consider the following system entities:

(a) *Relay network:* It is a set of (volunteer/ designated) nodes registered in both BC_1 and BC_2, and authorized to run light nodes on both blockchains. These nodes connect to full nodes in each blockchain and can interact with their smart contracts. The relay network receives a set of signed block headers from BC_1, $[data,\ sig\_of\_con]$. It applies a transformation T on the signatures and sends $[data,\ T(sig\_of\_con)]$ to BC_2 who applies the verification V, and if successful, adds the block header to its block header history. The transformation T in zkBridge is a zk-SNARK which is efficient for a permissionless blockchain. For a permissioned setting, with a small number of signatures, T can be an identity function: the relayers send all the signatures and BC_2 may verify all of them depending on V. For a larger number of signatures, we consider an implementation of T that takes a set of signatures for a block header, applies the consensus threshold n for BC_1, picks a valid subset of n signatures, and sends them, along with the block header, to BC_2. For example, if a block header has 20 signatures in total, and the consensus rule states that having 14 signatures (e.g., 2/3 of signatures) is sufficient to validate a block, T chooses 14 valid signatures from the set of 20 signatures to be sent to BC_2.

Additionally, in zkBridge, the verification of the transaction tx from BC_1 to BC_2 requires a Merkle proof. This proof is generated by a client in BC_1 and verified by the `updater` contract in BC_2 with respect to a Merkle root in the block header stored in it. However, in zkBridge, this proof is transmitted from BC_1 to BC_2 via the users/developers of the application contract $\mathcal{AC}_2$ or some third party (not the relayers), which is not possible by the resource-sharing application that we are considering since the user (resource requester) belongs to BC_2 only and has no direct connection with any client in BC_1. Therefore, in sigBridge, each relayer is tasked with the proof collection from BC_1's full node using it's light node LN_1 and passing it to `updater` contract in BC_2. So, in contrast to zkBridge, sigBridge utilizes the relay network for two main purposes: (i) transferring block headers from BC_1 to the `updater` contract in BC_2, with relayers selecting and relaying a set of valid signatures for each block based on BC_1's consensus rule, and (ii) fetching the tx information and its corresponding Merkle proof from full nodes in BC_1, and forwarding them to BC_2. Algorithm 1 summarizes relayer network functionalities.

(b) *`updater` contract:* We consider an `updater` contract that validates and stores the block headers in a block header history $hdrHistory$ according to the transformation V. `updater` contract only checks a threshold t number of signatures for each block. For example, if $t = 11$, then the `updater` contract on BC_2 only verifies 11 randomly chosen signatures among the 14 signatures received. If $t = n$ then `updater` contract verifies all received signatures. Additionally, `updater` contract stores the consensus rules and parameters of the BC_1 and uses it for verifying the received block headers. It also interacts with the applications that are built on top of the bridge to verify that the messages (transactions) they have received are correct. This verification retrieves the block header history from $hdrHsitory$ and checks the transaction inclusion. `updater` contract implements the *`updater`*, *verification manager*, and *cross chain consensus rules* in the synchronization layer of the bridge. The functionality of this contract has been shown in Algorithm 2.

(c) *Application contract:* The application contract $\mathcal{AC}_1$ is responsible for determining the information to be passed through the bridge to $\mathcal{AC}_2$. It uses the $RelayTransaction$ functionality of the relayer network shown in Algorithm 1.

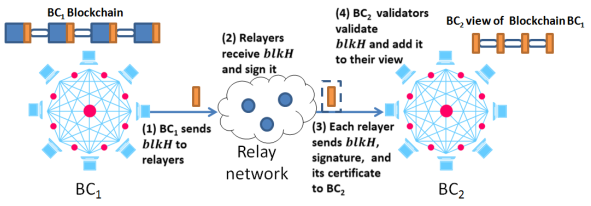
Algorithm 1 Relayer network protocol in *sigBridge*

```
1: procedure RelayNextHeader($blkH_{r-1}$)
2:     Contact $k$ full nodes to get block header following $blkH_{r-1}$ , denoted $blkH'_r = [data, sig\_of\_con]$
3:     Apply $blkH_r = [data, T(sig\_of\_con)]$ where $T$ chooses $n$ valid signature for $blkH_r$.
4:     Sign $blkH_r$ using private key $sk$, denoted $\sigma$
5:     Call HeaderUpdate($blkH_r, blkH_r,\ \sigma,\ Cert_{pk}$) on the updater Contract
6: procedure RelayTransaction($tx\_info$)
7:     Contact $k$ full nodes to get transaction $tx$ and its inclusion Merkle proof $\pi$.
8:     Sign $tx, \pi$ using public key $pk$, denoted by $\sigma$
9:     Call VerifyProof($tx,\ i,\ \pi,\ \sigma,\ Cert_{pk}$) on the updater Contract
```

Security. We consider a set of relay nodes that can be fully malicious except for one of them. They can arbitrarily deviate from the protocol and modify, forge, or drop messages, or try to participate in message-passing without registration. We consider both permissioned blockchains to satisfy consistency and liveness. *sigBridge* must satisfy the following properties:

– *Correctness*: $\mathcal{AC}_2$ accepts the wrong transaction with probability ϵ which can become negligibly small.

– *Liveness*: If $\mathcal{AC}_2$ needs to verify that a transaction occurred on $\mathcal{AC}_1$, the bridge will provide the necessary information eventually.



BC₁ Blockchain
(2) Relayers receive blkH and sign it
(4) BC₂ validators validate blkH and add it to their view
BC₂ view of Blockchain BC₁
(1) BC₁ sends blkH to relayers
Relay network
(3) Each relayer sends blkH, signature, and its certificate to BC₂
BC₁
BC₂

Figure 1. sigBridge protocol between BC_1 and BC_2

***sigBridge* set up**. To initialize the bridge between BC_1 to BC_2, the system administrator of BC_2 initializes the `updater` contract with the required information about BC_1 that will be used for block header and transaction verification. The relay nodes are registered as members of the relay network by an authority that is trusted by both chains. They generate a public/private key pair $(sk,\ pk)$ and receive a public key certificate $Cert_{pk}$ from the trusted authority. When sending any message to the `updater` contract, the nodes need to present this certificate.

Block header synchronization via `updater` contract. Each registered relayer in the system receives a (valid) block header, $blkH$, from BC_1 through *RelayNextHeader* procedure of the Algorithm 1, chooses n signatures that validate the block header, signs it with its private key, and sends it along with its certificate to the `updater` contract on BC_2. The `updater` contract verifies the relayer's certificate and the block header information based on the $hdrHistory$, threshold t number of signatures, and the stored consensus information of BC_1. If the $blkH$ is invalid (even if one signature out of t is invalid), it is dropped. Otherwise, it is stored in $hdrHistory$. Figure 1 shows the synchronization protocol.

Security arguments. The following theorem states the security of the block header synchronization:

Theorem 1. Our scheme for `updater` contract synchronization in a permissioned blockchain setting achieves ϵ-consistency with $\epsilon = \frac{\binom{n-\alpha}{t}}{\binom{n}{t}}$, and liveness assuming the signature scheme used for consensus and by relayers are unforgeable (negligible probability), at least one relayer is honest, and the sender chain ensures consistency and liveness. (For each block, n is the

Algorithm 2 `Updater` Contract in *sigBridge*

```
1:  $hdrHistory := \emptyset$                          ▷ header history
2:  $xccRules$          ▷ consensus rules and parameters of remote chain
3:  procedure HEADERUPDATE($blkH_r$, $\sigma$, $Cert_{pk}$)
4:      if $Cert_{pk}$ is not valid or $\sigma$ on $blkH_r$ then
5:          return $False$
6:      else
7:          retrieve $blkH_{r-1}$ from $hdrHistory$
8:          if VerifyHeaderUpdate($blkH_{r-1}, blkH_r$,
9:  $xccRules, t$)= $True$ then
10:             Insert $blkH_r$ to $hdrHistory$          ▷ $t$ is the threshold value for verifying the signatures in the block header
11: procedure VERIFYPROOF($tx$, $i$, $\pi$, $\sigma$, $Cert_{pk}$)
12:     if $Cert_{pk}$ is invalid || $\sigma$ is invalid signature on $tx, i, \pi$ then
13:         return $False$
14:     $blkH_i = hdrHistory[i]$
15:     if VerifyTx($tx, \pi, blkH_i$) = $True$ then
16:         pass $tx$ info to its receiver
```

number of signatures received by `updater` contract, t is the threshold number of signatures verified by the `updater` contract, and α is the number of invalid signatures generated by the malicious relayer.)

Proof (sketch). We assume that the signature scheme is unforgeable, only messages from registered relayers will be used for block header and transaction validation.
– *Consistency*: A permissioned blockchain employing proof of authority achieves immediate finality. This means that only one block is generated per consensus round and is instantly finalized when the threshold number of validators sign it. As a result, a relayer connected to a sufficient number of validators (full nodes) in BC_1 (with at least one of them being honest) will receive the block header and transmit it, along with n valid signatures, to BC_2. Consequently, the block headers from BC_1 are transferred to BC_2 with a minimal delay attributed to message-passing, and validation is performed by BC_2. Assume a malicious relayer takes a block that has only $n-\alpha$ valid signatures and includes α forged signatures to form the n expected signatures on the block. The `updater` contract selects t signatures randomly and verifies them. The number of possible choices of t signatures out of n signatures is $\binom{n}{t}$. The forged signatures (that do not pass the verification algorithm) will not be detected if the t chosen signatures are from the $n-\alpha$ valid signatures. Thus, the probability of the attack not being detected

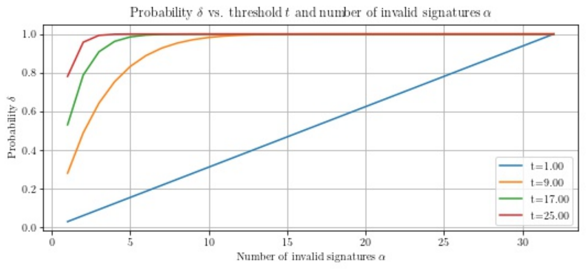


Figure 2. Probability δ with respect to changes in threshold value t and the number of invalid signatures α for an invalid block and assuming $n = 32$

will be, $\epsilon = \frac{\binom{n-\alpha}{t}}{\binom{n}{t}}$, which can be made arbitrarily small by increasing t. As long as BC_1 satisfies consistency for block headers, BC_2 will satisfy it with probability $\delta = 1 - \epsilon$. Figure 2 shows changes of probability $\delta = 1 - \epsilon$ with respect to changes in t and α, where $n = 32$. Note that, by increasing $t = n$ the probability δ becomes 1. Also, t does not need to be fixed in all rounds, it can be chosen randomly while keeping it close to n.
– *Liveness*: Assuming at least one honest relayer exists, the block headers will still be transmitted to BC_2 even if all the malicious relayers drop them.

Message passing in application contract. An application contract $\mathcal{AC}_1$ sends message m to $\mathcal{AC}_2$ through the relay network using the *RelayTransaction* procedure, denoted as $sigBridge[\mathcal{AC}_1 \rightarrow \mathcal{AC}_2](m)$. The relay network retrieves Merkle proof π from full nodes to confirm that an event (e.g., transaction tx), related to m has occurred, and uses the $verifyProof$ function of `updater` contract to verify the proof. Upon successful verification, the message is sent to $\mathcal{AC}_2$. Algorithm 1 shows this process. Theorem 2 states the security of sigBridge where the proof is straightforward:

Theorem 2. *$sigBridge[\mathcal{AC}_1 \rightarrow \mathcal{AC}_2]$ ensures ϵ-correctness and liveness assuming the `updater` contract satisfies liveness and ϵ-consistency for block header synchronization, and the Merkle proof satisfies correctness and soundness.*

5. Cross-chain user-centric access control scheme

We present our resource-sharing application in this section that is built on top of the *sigBridge* protocol and uses smart contracts to share resource information and evaluate access policies.
Cross-chain resource sharing: In a resource-sharing system, users can be resource owners, resource

requesters, or both. Each user chooses a private/public key pair and registers their public key with their home blockchain identity manager. They also get a public key certificate. *Resource owners* obtain certificates for their resources' attributes from a trusted certificate issuer and then store their resources in a database. The database provider verifies the resource certificate and stores it. *Resource requesters* can only access resources using a valid token (stored in either BC_1 or BC_2), which contains information such as the resource, requester's public key, owner's public key, the result of access control evaluation, and auxiliary information (e.g., validity period, request hash, etc). The database uses a light client protocol to check the token's validity. Resource requesters also receive certificates for their attributes from a trusted certificate authority.

Application smart contracts. We consider the following smart contracts (similar to [Muni et al. (2020)]): (i) `obj-dir` contract: It is deployed by the network administrators and stores the resource information (description and access policy information) that can be accessed by the requesters. (ii) SC-ϕ_i contract: It is deployed by owner i and implements the ABAC policy ϕ_i to evaluate the requester's attributes and either grant or deny access.

Security assumptions and threat model. We consider that both blockchains utilize the same cryptographic specifications and request/response formats. We assume all users to be potentially malicious and trying to gain unauthorized access to resources.

Security definition. We require the following properties for cross-chain resource-sharing application:

– *Correctness*: Every requester in BC_2 with an attribute that matches the defined access policy in BC_1 can access the resource.

– *Security*: No unauthorized requester in BC_2 can access the resources in BC_1.

5.1. Cross chain resource sharing protocol

The resource-sharing protocol comprises three stages: (1) Sharing a resource, (2) Browsing resources, and (3) Accessing resource requests. Stage 1 requires registering the resource through `obj-dir` contract and deploying SC-ϕ_i contract by the resource owner in its local blockchain. Below, we explain two types of requests related to stages 2 and 3, respectively.

Browsing request. To browse the shared resources in BC_1 from BC_2, we propose storing two tables in the `obj-dir` contract: 1) a local table for the local blockchain resources (e.g. BC_2) and 2) a cross-chain table for the remote blockchain resources (e.g. BC_1). The cross-chain table can be updated in BC_2 as soon as the `obj-dir` table in BC_1 is updated using the bridge protocol which is shown in Algorithm 3. The presence of the cross-chain `obj-dir` table in BC_2 enables requesters in BC_2 to browse the cross-chain `obj-dir` table in their home blockchain without utilizing the bridge protocol. The flow for browsing the cross-chain `obj-dir` table is similar to browsing the local `obj-dir` table (see Algorithm 4).

Algorithm 3 Updating the view of cross-chain `obj-dir` table in BC_2

```
1: $owner \rightarrow$ obj-dir$_1$ in $BC_1$: update $objInfo$    $\triangleright$ $objInfo = \{objId,\ Pk_o,\ Desc,\ Object\ attributes,$
2: $Access\ Policy$, SC-$\phi_i$address$\}$
3: obj-dir$_1$:$RelayTransaction(objInfo)$    $\triangleright$ $sigBridge$[obj-dir$_1 \rightarrow$ obj-dir$_2$] $(objInfo)$
4: obj-dir$_2$: update $objInfo$ in cross-chain obj-dir$_2$ table
```

Algorithm 4 Browse `obj-dir`

```
1: $requester \rightarrow$ cross-chain obj-dir$_2$: $View\_Req$    $\triangleright$ $View\_Req = (Fields = [obj-dir\ table\ view],\ sign_{pk_r}(Fields),\ Pk_{cert})$
2: obj-dir$_2$ $\rightarrow$ requester: cross-chain obj-dir table
```

Accessing a resource request. The requester in BC_2 sends a resource access request to the SC-ϕ_i contract in BC_1, along with its attributes $\{a_1,\ a_2,\ ..., a_n\}$, which is evaluated against the access policy. The request is transmitted to BC_1 via the relay network, where it is signed by the relayers and accompanied by their signatures, certificate, and Merkle proof (demonstrating the request's presence in a BC_2 block) to the SC-ϕ_i contract. The request is executed in BC_1 and a token is returned to the requester through the relay network. The relay network signs the access token and sends it together with their signature, the certificate, and the Merkle proof (showing that the token is in a block of BC_1). Upon receiving the token, the requester checks the token. If the token does not match the expected token, then it drops it (see Algorithm 5 for details).

5.2. Security analysis

Below we present the security theorem and arguments for our cross-chain resource-sharing scheme.

Theorem 3. Our cross-chain resource-sharing scheme achieves $(1-3\epsilon)$*-correctness and* $(1-2\epsilon)$*-security if the sigBridge protocol ensures liveness and* ϵ*-correctness and assuming both communicating blockchains ensure trusted execution of the smart contracts.*

Algorithm 5 Accessing a resource through $\texttt{SC-}\phi_i$.

```
1: $requester \rightarrow \texttt{obj-dir}_2 : Objreq$    $\triangleright$ $Objreq = (Fields$[“ObjId”, “user attributes”, certs],
2: $sig_{pk_B}(Fields), cert_{pk_B})$
3: $\texttt{obj-dir}_2$: $RelayTransaction(Objreq)$    $\triangleright$ $sigBridge[\texttt{obj-dir}_2 \rightarrow \texttt{SC-}\phi_i](Objreq)$
4: $\texttt{SC-}\phi_i$ evaluates the policy on $Objreq$ and outputs Token
5: $\texttt{SC-}\phi_i : RelayTransaction(Token)$    $\triangleright$ $sigBridge[\texttt{SC-}\phi_i \rightarrow \texttt{obj-dir}_2](Token)$
6: $\texttt{obj-dir}_2 \rightarrow requester : Token$
```

Proof (sketch). To show our protocol ensures correctness,we have to show that: 1) the view of the cross-chain `obj-dir` in BC_2 is consistent and live with respect to `obj-dir` in BC_1, 2) the request information is passed to BC_1 without modification, 3) the access policy evaluation is done correctly, and 4) the token has not been modified or re-sent from an earlier request. Note that the correctness fails if any of the above cases fail. For (1) we rely on the functionality of sigBridge protocol which ensures ϵ-correctness and liveness. So, it fails with probability ϵ. For (2) and (4), we assume the transaction inclusion proof ensures correctness and soundness and will be verified for request and token by the `updater` contract. Since the view of `updater` contract ensures ϵ-consistency and liveness, it fails to detect any old request/token or modified ones with probability ϵ. For (3), we rely on the assumption that the chain that executes the policy evaluation is trusted (according to its consensus trust assumptions). Therefore, correctness fails in the three cases that equal 3ϵ.

For security, we have to show that: (i) the requester is a registered user in the system, (ii) the attributes of the requester are valid, and (iii) the request cannot be re-sent (replay attack from relayers) after the user is removed from the system. Failure of any of the above cases would result in security being breached. For (i), we emphasize that users get public key certificates upon registration. When a user makes a request, its home blockchain will verify the certificate and then add it to the blockchain. Assuming the sigBridge protocol ensures ϵ-consistency and liveness, then BC_1 fails to verify whether the request has been included in a block in BC_2 with probability ϵ. For (ii), note that each user gets an attribute certificate from a trusted authority and the $\texttt{SC-}\phi_i$ contract first verifies the certificate. For (iii), we rely on the `updater` contract in BC_1 which cannot detect if a request transaction is fresh (sent recently) or not with probability ϵ. Therefore, security fails with probability 2ϵ.

6. Implementation

We implement a proof-of-concept cross-chain resource-sharing application in a permissioned environment. Our goal is to *enable users to access resource information from a remote blockchain network while ensuring accurate access policy evaluation and maintaining a consistent view of available resources*. Our resource-sharing platform requires: (i) *permissioned (private) blockchains* that communicate with each other for the resource-sharing purpose, (ii) *smart contracts* to enable cross-chain communication and store the advertised resource information, as well as enforce attribute-based access control (ABAC), (iii) *relays* for data (e.g., block header, transaction) transfer between two private chains. We evaluate the application's performance by measuring the on-chain function execution cost in gas and converting it into a unit equivalent to a deterministic function execution (e.g., hashing). The simulation environment and evaluation results are provided below.

Simulation environment. We create two private Ethereum networks (BC_1 and BC_2) using the Geth[3]. Puppeth[4] is used to generate the genesis files that define the consensus nodes (C-nodes), and Proof-of-Authority (PoA) as the consensus mechanism. Smart contracts are developed using the Solidity language and compiled and deployed on the private chains using Truffle[5].

Smart contracts. The system consists of two types of contracts: (a) application contracts and (2) cross-chain contracts. For the resource-sharing application, we utilize the object directory (`obj-dir`) contract (deployed by the network administrator) and access control evaluator ($\texttt{SC-}\phi_i$) contract (deployed by owner). The `obj-dir` contract extends the functionality of the object directory contract of [Muni et al. (2020)] and includes an additional table for remote chain resources, allowing users to view remote resources locally. The `updater` contract is a cross-chain smart contract deployed in each chain by network administrators. Its *headerUpdate()* function implements the “*light client protocol*” for validating block headers by verifying C-node signatures based on the consensus rules of the remote blockchain. It also includes functions such as *verifyTx()* and *getHeader()* for verifying transaction inclusion in a specific block using a given Merkle proof and retrieving block header hashes, respectively.

Evaluation results. In our evaluation, we consider a scenario where the resource owner is in BC_1 and the

[3]https://geth.ethereum.org/
[4]https://geth.ethereum.org/docs/tools/puppeth
[5]https://trufflesuite.com/docs/truffle

Table 1. Smart contract execution costs in gas and keccak-256 hash equivalent. A single keccak-256 call on a bytes32 data is $K = 323$ gas.

Contract	Function	*Gas* cost	# of Hashing (gas cost/K)
`obj-dir`	*registerResource*	435830	≈ 1349
	updateResource	49552	≈ 153
	deleteResource	99004	≈ 189
	generatePolicy	1268843	≈ 3928
SC-ϕ_i	*requestResourceAccess*	199751	≈ 618

Table 2. Comparison of sigBridge and zkBridge for block header verification. The data size column shows the total size of C-node signatures in sigBridge and proof size in zkBridge, for N signatures. On-chain cost for sigBridge is for signature verification and for zkBridge, is ZK proof verification.

N (# of sigs.)	Data size (bytes)		On-chain cost (gas)		# of Machine required	
	sigBridge	zkBridge (w/ RV)	sigBridge	zkBridge (w/ RV)	sigBridge	zkBridge
8	1064	131	396K	227K	1	8
32	4256	131	5M	227K	1	32

requester is in BC_2, interacting with smart contracts based on the cross-chain resource sharing protocol described in Section 5.1. We use proof-of-authority consensus in both permissioned networks, with a threshold set at 2/3 of C-nodes (e.g., 8 out of 10) to sign a block header. Performance is measured using the cost of a single execution of Ethereum's hash function, *keccak-256*, on a *bytes32* value. Performance is evaluated on a Windows 11 system with a 1.80 GHz AMD Ryzen 7 5700U CPU and 16GB RAM.

Table 1 shows the function execution costs of resource-sharing application contracts and Table 2 shows a comparison of on-chain verification costs between sigBridge and zkBridge [Xie et al. (2022)]. The *gasleft()* function of Solidity is used to measure the gas values.

Efficiency-probability trade-off. Figure 3 shows the trade-off between the threshold t number of signatures that are verified by BC_2, the probability that an invalid signature is being detected, δ, and the verification cost in terms of gas. We consider $\alpha = \{4, 10, 18\}$ for the number of invalid signatures when $n = 32$. It has been shown that increasing the threshold t leads to an increase in the probability of detecting an invalid signature and the verification cost which reduces the efficiency.

7. Concluding remarks

We considered a four-layer architecture and proposed a bridge, called sigBridge, to enable applications to securely communicate across two blockchains. sigBridge follows the approach of zkBridge but avoids

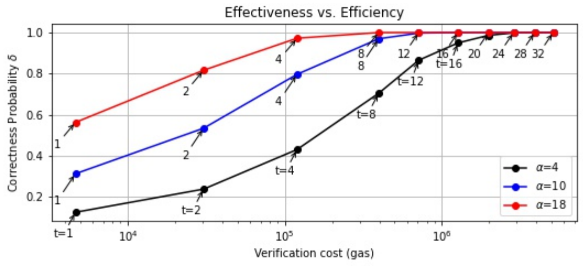


Figure 3. Effectiveness vs. efficiency with respect to changes in t and α, where $n = 32$.

complex zero-knowledge proofs by using the two blockchains' consensus algorithms. sigBridge checks a subset of consensus signatures randomly and provides a trade-off between correctness and efficiency. We used sigBridge to extend a decentralized resource-sharing application over a single permissioned blockchain to work across two blockchains. We implemented and evaluated the computational efficiency of the system using two private Ethereum networks. An interesting open question is to design a sigBridge where the `updater` contract remains oblivious to the consensus algorithm of the other blockchain and uses the "proof" presented by the relayers to verify the state update of the other chain. Compared to zkBridge, the proof system will be able to support more complex consensus algorithms using more general zero-knowledge proof systems. Furthermore, exploring a privacy-preserving resource-sharing scheme within the context of a heterogeneous multi-chain environment presents an intriguing avenue for future research.

References

Avizheh, S., Nabi, M., Rahman, S., Sharifian, S., & Safavi-Naini, R. (2021). Privacy-preserving resource sharing using permissioned blockchains: (the case of smart neighbourhood). *Financial Cryptography and Data Security Workshops: WTSC, Virtual Event, March 5, 2021*, 482–504.

Bridge, N. R. (2022). https://near.org/bridge/

Bridge, N. (2022). https://docs.nomad.xyz/the-nomad-protocol/overview

Burr, W. E., Dodson, D. F., Foote, M., Polk, T., Skowyra, R., & Wieners, C. (2018). *Blockchain technology overview* (tech. rep. No. NISTIR 8202). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.IR.8202

Buterin, V. (2016). Ethereum: Platform review. *Opportunities and challenges for private and consortium blockchains*, *45*.

Celo. (2019). https://celo.org/

Groth, J. (2016). On the size of pairing-based non-interactive arguments. *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*, 305–326.

Hu, V. C., Ferraiolo, D., Kuhn, R., Friedman, A. R., Lang, A. J., Cogdell, M. M., Schnitzer, A., Sandlin, K., Miller, R., Scarfone, K., et al. (2013). Guide to attribute based access control (abac) definition and considerations (draft). *NIST special publication*, *800*(162), 1–54.

IBC. (2021). *Inter-blockchain communication*. https:// ibcprotocol.org/documentation/

Jin, H., Dai, X., & Xiao, J. (2018). Towards a novel architecture for enabling interoperability amongst multiple blockchains. *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, 1203–1211.

Kwon, J., & Buchman, E. (2019). Cosmos whitepaper. *A Netw. Distrib. Ledgers*, 27.

Kwon, J., & Buchman, E. (2022). *Cosmos whitepaper*. https://cosmos.network/

Li, D., Han, D., Crespi, N., Minerva, R., & Li, K.-C. (2023). A blockchain-based secure storage and access control scheme for supply chain finance. *The Journal of Supercomputing*, *79*(1), 109–138.

Liao, C.-H., Guan, X.-Q., Cheng, J.-H., & Yuan, S.-M. (2022). Blockchain-based identity management and access control framework for open banking ecosystem. *Future Generation Computer Systems*, *135*, 450–466.

Muni, K., Avizheh, S., & Safavi-Naini, R. (2020). A blockchain based approach to resource sharing in smart neighbourhoods. *Financial Cryptography and Data Security Workshops: WTSC*, 550–567.

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. *Decentralized business review*, 21260.

OPTICS. (2022). *Optimistic interchain communication*. https://docs.celo.org/protocol/bridge/optics

Ouaddah, A., Abou Elkalam, A., & Ait Ouahman, A. (2016). Fairaccess: A new blockchain-based access control framework for the internet of things. *Security and communication networks*, *9*(18), 5943–5964.

Ronin. (2021). *Ronin whitepaper*. https : / / docs . roninchain.com/docs/intro/whitepaper

Salonikias, S., Khair, M., Mastoras, T., & Mavridis, I. (2022). Blockchain-based access control in a globalized healthcare provisioning ecosystem. *Electronics*, *11*(17), 2652.

Sekniqi, K., Laine, D., Buttolph, S., & Sirer, E. G. (2020). *Avalanche platform*. https : / / www . avalabs.org/whitepapers

Sumathi, M., & Sangeetha, S. (2020). Blockchain based sensitive attribute storage and access monitoring in banking system. *International Journal of Cloud Applications and Computing (IJCAC)*, *10*(2), 77–92.

Team, A. (2021). *Axelar network: Connecting applications with blockchain ecosystems.* https://axelar.network/axelar_whitepaper.pdf

Team, P. (2020). Polynetwork: An interoperability protocol for heterogeneous blockchains.

Wood, G. (2016a). Polkadot: Vision for a heterogeneous multi-chain framework. *White paper*, *21*(2327), 4662.

Wood, G. (2016b). Polkadot: Vision for a heterogeneous multi-chain framework. *White paper*, *21*(2327), 4662.

XCMP. (2015). *Cross consensus message passing*. https://wiki.polkadot.network/docs/learn-xcm-transport # xcmp - cross - consensus - message - passing-design-summary

Xie, T., Zhang, J., Cheng, Z., Zhang, F., Zhang, Y., Jia, Y., Boneh, D., & Song, D. (2022). Zkbridge: Trustless cross-chain bridges made practical. *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 3003–3017.

Xu, R., Chen, Y., Blasch, E., & Chen, G. (2018). Blendcac: A blockchain-enabled decentralized capability-based access control for iots. *2018 IEEE International conference on Internet of Things (iThings) and IEEE green computing and communications (GreenCom) and IEEE cyber, physical and social computing (CPSCom) and IEEE Smart Data (SmartData)*, 1027–1034.

Zarick, R., Pellegrino, B., & Banister, C. (2021). Layerzero: Trustless omnichain interoperability protocol. *arXiv preprint:2110.13871*.

Zhang, Y., Kasahara, S., Shen, Y., Jiang, X., & Wan, J. (2018). Smart contract-based access control for the internet of things. *IEEE Internet of Things Journal*, *6*(2), 1594–1605.