Extending the Secure Asset Transfer Protocol for Enhanced Security and Privacy

Tomás Silva, André Augusto, Rafael Belchior
André Vasconcelos, Miguel Correia
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa
Lisbon, Portugal
{tomas.v.silva, andre.augusto, rafael.belchior, andre.vasconcelos, miguel.p.correia}@tecnico.ulisboa.pt

***Abstract*—Blockchain interoperability has gained increasing attention, particularly for cross-chain asset transfers. However, fragmented designs and the lack of common standards have contributed to significant security vulnerabilities. The Internet Engineering Task Force (IETF) has sought to address this gap through the Secure Asset Transfer Protocol (SATP), a gateway-mediated protocol for cross-chain asset transfers. Nevertheless, SATP solely relies on gateways to report critical on-chain operations and does not define a mechanism for independently proving these claims.**

We introduce a zkSNARK-based mechanism that enforces the gateways' commitment to transactions and allows them to prove that reported transactions satisfy predefined constraints while limiting the disclosure of the transactions' underlying data. Our architecture introduces a Proof Computation Unit (PCU), responsible for generating and validating proofs, an attesting authority who oversees the correct operation of the PCU, and an external server for storing proofs and verification keys, thereby reducing SATP's reliance on gateways only for transaction reporting. This added assurance incurs a significant performance cost: in our proof of concept, transfers required approximately 220 seconds, compared with the 15 seconds of the baseline SATP. Under concurrent proof generation, we managed to test the PCU with up to six parallel requests, with latency remaining fairly consistent, and memory constituting the primary resource bottleneck.

***Index Terms*—zkSNARK, Privacy, SATP, Resilience, Trustworthiness**

I. Introduction

In recent years, several efforts have been developed towards addressing the challenges of establishing ways for independent blockchains to interoperate [1] [2], though they still suffer from major security risks, becoming a target for exploits [3]. Fragmented designs, faulty implementations and the absence of common standards have contributed to these failures [4].

The IETF has been working towards filling this gap by proposing the Secure Asset Transfer Protocol (SATP), designed to be executed between two gateways [5].

SATP provides a suitable foundation, as supporting a blockchain does not require substantial modifications to the chain's core logic. Yet, several aspects of the protocol remain open to further improvement. Its heavy reliance on gateways being honest, although a valid approach, leads to gateways becoming a possible single point of failure, which is further reinforced by the fact that these types of components are frequent targets of attacks [3]. While gateways can keep their crucial role, any efforts towards decreasing the protocol's dependability on them render it more robust. SATP authors did not specify a formal mechanism for proving the execution of on-chain transactions. In the current implementation, gateways exchange signed transaction outputs, such as transaction receipts in EVM-compatible blockchains. This reinforces the protocol's dependence on gateways and may hinder the integration of private blockchains, where transaction proofs may need to satisfy confidentiality requirements [6]. Our work addresses these challenges by combining the employment of zero-knowledge Succinct Non-Interactive Arguments of Knowledge (zkSNARKs) with the contribution of two new independent components to the protocol.

Figure 1 illustrates, at a high level, our proposed architecture and interactions. The new suggested components are the Proof Computation Unit (PCU), responsible for generating and verifying zkSNARKs, and the external server, responsible for permanently storing both the generated zkSNARKs and the respective proving keys, allowing their persistence post-use in a protocol session. We also propose an attesting authority, whose role is to supervise security-critical PCU operations. Our goal is for our proposed infrastructure to support the generation of verifiable proofs that a gateway has committed itself to a transaction that has occurred on a blockchain, without having to disclose internal data specific to the blockchain and, simultaneously, not being capable of, alone, hijacking a protocol session by providing forged proofs. By also supplying a mechanism to publish the zkSNARKs and making them always verifiable, this work ensures that these proofs are accessible to external verifiers after their employment in a SATP session, enabling transparency and auditability of past protocol executions.

II. Background and Related Work

a) Secure Asset Transfer Protocol: SATP is a protocol designed for transferring digital assets across different networks, through two intermediate gateways, representing the origin and destination networks [5]. During a transfer, an asset is permanently disabled (burned) at the origin network and re-generated (minted) at the destination network. Gateways are assumed to be trusted to carry out all the stages of the protocol and to sign messages, taking liability for their validity in the process. As previously mentioned, they are also expected

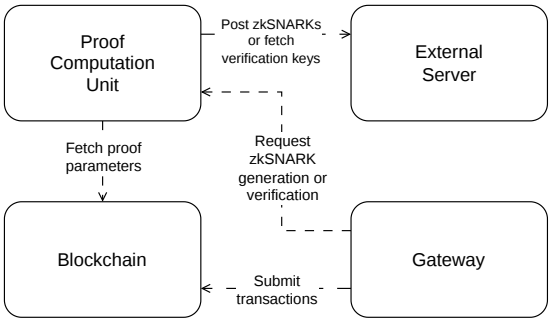


Fig. 1. High-level solution architecture

to convey proofs of correct transaction occurrence on the blockchain they act on behalf of.

As defined in the protocol's documentation, it is composed of three main stages:

- *Transfer Initiation Stage (Stage 1)*: Encompasses the message flows to commence a transfer.
- *Lock-Assertion Stage (Stage 2)*: Encompasses the message flows related to gathering signed assertions from the sender gateway on the locked status of the asset being transferred.
- *Commitment Preparation and Finalization Stage (Stage 3)*: Encompasses the message flows related to the commitment of state changes issued by both gateways to their respective networks to fully achieve the asset transfer (the minting of new assets on the destination chain, the burning of the original assets on the origin chain, and the final assignment of the new assets to their respective owners on the destination chain).

Figure 2 illustrates the protocol's message flow between the peer gateways and between them and their respective networks along its main stages.

As previously established, SATP's reliance on gateways, not requiring extensive changes to blockchains to support them, contributes to the protocol's suitability as a standard for cross-chain transfers. Yet, no formal proving mechanism was specified by the authors, which can constitute a barrier to wide adoption, as permissioned blockchains, for example, have stricter limitations on the internal data that can be shared. In addition, while the protocol foments accountability on the gateways by requiring them to always sign exchanged data, a single malicious gateway is able to compromise an entire protocol session by forging proofs on the execution of a local on-chain transaction.

b) Zero-Knowledge Proofs and Privacy: Zero-knowledge proofs allow the verification of a computation's correctness without revealing the underlying data in the process, or requiring the full computation to be re-executed [7]. *Zero-knowledge Succinct Non-Interactive Arguments of Knowledge* (zkSNARKs) provide a succinct proof size and short verification time, yet they require a predecessor trusted setup phase [8], during which a set of random values

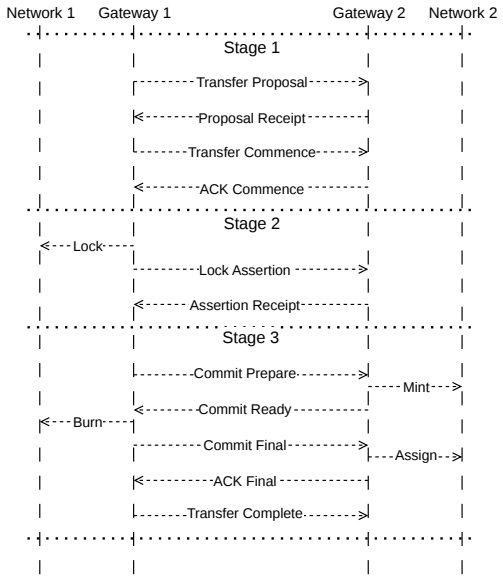


Fig. 2. SATP message flow

are generated and are required to be disposed of, to avoid compromising the entire setup. *Zero-knowledge Scalable Transparent Arguments of Knowledge* (zkSTARKs) eliminate the need for a trusted setup, offer improved scalability and quantum-resistance, at the cost of larger proof sizes [9]. *Bulletproofs* allow proving one or more committed values lie within a certain range without revealing their actual values [10].

Several blockchain platforms have successfully integrated zero-knowledge proofs to enhance privacy or improve transaction validation scalability within a single blockchain, embedding them directly in the underlying protocol's execution model or transaction semantics. *Zcash* [11], for instance, is a fork of Bitcoin that introduces optional anonymity through zkSNARK-based shielded transactions, enabling transaction details to remain hidden, at the cost of changing the transaction format. Similarly, *Monero* provides a fully private cryptocurrency [12], which achieves strong privacy guarantees through mechanisms embedded within its respective chain's core design, such as stealth addresses (specifically generated for each transaction) and ring confidential transactions. The latter are supported by Bulletproofs to conceal transaction amounts while keeping them verifiable, making transactions unlinkable and untraceable.

While these systems effectively address privacy within a single blockchain, they reflect a broader design pattern in which zero-knowledge mechanisms are tightly integrated into the internal logic of specific blockchains, limiting the portability of such techniques to heterogeneous environments or to existing blockchain systems without requiring structural modifications.

Interoperability frameworks, such as SATP, which focus on

enabling cross-chain communication across independent and different ledgers, have their adoptability impacted by requiring internal logic changes in order to provide support for a specific ledger. The *Aztec Network* [13], a Layer-2 protocol, does explore a layered approach by offloading sensitive computations to its users' devices, validating them through zkSNARKs. Yet, this logic is bounded to assets created in Aztec's own execution environment.

In this context, our work explores a complementary design approach in which zero-knowledge proofs are not embedded in a single blockchain's protocol but instead applied as a decoupled verification mechanism for cross-chain interactions. This approach offers multiple blockchains the benefits of zero-knowledge guarantees when cross-communicating, allowing them to keep certain levels of their internal state private to each other, without requiring modifications to their respective protocols and execution models.

Furthermore, by employing zero-knowledge proofs over SATP, our work contributes towards standardizing their usage.

c) Interoperability Platforms Attacks: Zhang et al. [4] clearly categorized the type of attacks conducted against interoperability platforms:

1) Permission-related: Bypassing permission mechanisms through leaked keys, invalid signatures or the exploit of faulty and improperly configured approval mechanisms.
2) Logic-related: Exploiting flaws in the logic of smart contracts, such as faulty initialization processes and incorrect balance checks.
3) Event-related: Fake or incorrect event emissions within an ongoing transaction.

The vulnerabilities that have rendered past attacks possible stem from both implementation errors and an absence of standardized and verifiable mechanisms for cross-chain transfers. Heavily relying on privileged validators concentrates trust assumptions, making incorrect validation exploits a considerable threat. These findings reinforce our initial considerations regarding the benefits to SATP's robustness by reducing its dependence on gateways and their correct behavior.

III. Solution

Our proposed solution aims at reinforcing SATP's current guarantees on the following principles:

- *Counter-Blockchain State Confidentiality*: Considering an asset transfer between blockchains B_A and B_B, represented by gateways G_A and G_B, respectively, G_A does not obtain additional information on the internal state of B_B, nor does G_B obtain information on the internal state of B_A, other than the public values agreed during the initial protocol stages, such as the transfer amount and the counter-gateway address.
- *Decentralization*: Assuming that the involved components in the protocol operate independently, no single component is solely responsible for the conveyance of a proof over a local blockchain transaction. All proposed components are required to cooperate with each other for a valid proof to be generated.
- *Robustness*: A single compromised gateway cannot unilaterally cause the generation of a valid proof over a forged transaction. Proofs shall be generated over transactions that have occurred within a certain time frame and that fulfill a set of constraints enforced by a zkSNARK circuit.
- *Generality*: While a set of mechanisms to complement SATP is proposed, they shall not require extensive changes to the protocol for their support.

Figure 3 illustrates the proposed components their interaction on a single blockchain and gateway deployment of SATP.

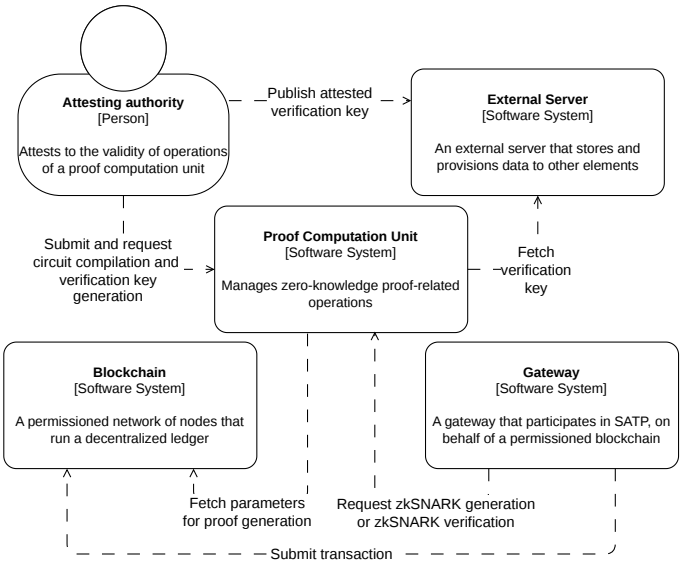


Fig. 3. Proposed integration architecture

Groth16 zkSNARKs are well suited for our architecture. As proof-generation latency is highly dependent on the implementation for both zkSNARKs and zkSTARKs, we decided to prioritize the smaller proof size and efficient verification of zkSNARKs. These are particularly advantageous considering our decision to generate, exchange and verify four independent proofs. Moreover, the enforced transaction constraints remain fixed, allowing the trusted setup to be performed once per defined circuit and therefore introducing no runtime overhead to the protocol execution.

We defined the component responsible for all the tasks related to zkSNARKs' generation and verification to be the Proof Computation Unit (PCU). To accomplish the intended operational independence between the gateway and the PCU, the latter is required to query a blockchain directly for the data it needs to generate the witness of the proof (assigning inputs to the circuit and computationally verifying that they satisfy the circuit's constraints).

Due to the sensitiveness of the trusted setup session, where the proving and verification keys for the zkSNARK are generated, we propose this process be overseen by another independent component, which we refer to as the attesting authority. The attesting authority is expected to oversee and

assure the security and correction of this setup session, such as guaranteeing that the process was executed in a safe environment, and that the reference string used to generate both keys was correctly disposed of, committing itself to the results by signing the resulting verification key.

While the attesting authority may seem to constitute a new point of centralized trust (an aspect of SATP that we intend to mitigate), it is important to clarify that the role of this element is to provide an accountable entity responsible for detecting any misbehavior or faulty operation of the PCU, without having direct control over it. These two elements should work independently, but in cooperation, with the PCU performing its tasks while the attesting authority validates or rejects their outcomes, without exerting any control over the PCU's internal processes. Our primary goal is for these components to preserve operational independence, while simultaneously having the PCU be subjected to the accountable endorsement of the attesting authority. The attesting authority should have access to the execution logs of the PCU, to be capable of deliberating over its operational status.

We propose the existence of yet another component, defined as the external server, contributing to the overall decentralization of our solution. It is expected to store both the verification keys for the zkSNARKs (avoiding having to rely on gateways to exchange them directly) and the zkSNARKs themselves, after a SATP session finishes. This way, the external server allows all generated proofs to always be verifiable.

The responsibility for correctly storing these artifacts is also shared between components. As described above, an attesting authority should sign the verification keys, avoiding any other component to tamper with them directly. To provide a similar guarantee to the zkSNARKs, we propose the maintenance of a Merkle Mountain Range (MMR) tree to keep track of the hashes of generated zkSNARKs, per blockchain. This is an append-only structure, where each new root is directly derived from a combination of the older root with the new elements. This allows the root update to be tracked on-chain. As the PCU holds the proofs it has generated, it first submits its hashes to the external server, which computes the new root. The hashes are then submitted by the PCU, along with the computed root, to the respective blockchain, which only accepts the update if the on-chain computed root matches the provided one. These structures also allow the generation of inclusion proofs, which, if combined with the root stored on-chain, do not allow proofs to be unilaterally tampered with or inserted into the tree without it being detected.

The guarantees provided by the zkSNARK depend on the logic enforced by its circuit. In our design, the PCU receives the transaction hash, signature and part of the signed message from the gateway, creating a limited dependency on the gateway for transaction identification. The PCU independently retrieves the transaction and reconstructs part of the signed data inside the circuit. A valid proof therefore establishes that the gateway has signed data derived from an existing on-chain transaction, preventing arbitrary parameter forgery. It does not guarantee that the selected transaction belongs to the SATP session, as a malicious gateway may select an alternative transaction that satisfies the constraints. This can be mitigated by limiting the interval between transaction inclusion and proof request, at the cost of greater sensitivity to request-delivery latency.

Changes required to SATP's flow and message composition are minimal, mainly focusing on requesting the generation and validation of proofs, which is illustrated in Figure 4, for part of the flow of stage 0 (a setup stage), and the flow regarding a lock during stage 2. The changes illustrated in this Figure apply, without loss of generality, to the mint, burn, and assign steps in stage 3. The credentials from the attesting authorities, used to validate the verification keys, shall be made available through a credential server, with each attesting authority being responsible for its own. The publication of zkSNARKs on the external server is to be executed after a protocol session finishes, without introducing any latency to it.

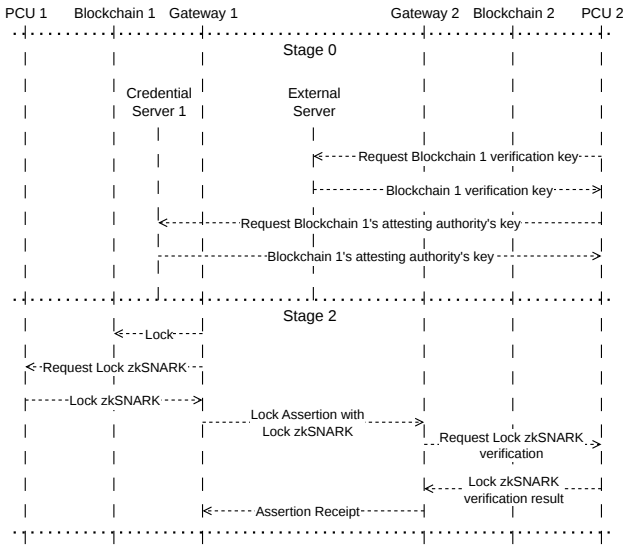


Fig. 4. SATP message flow changes

IV. Security Analysis

Any compromise in the zkSNARKs' trusted setup leads to the resulting proofs becoming unsafe to use. To avoid this, the attesting authority is always required to confirm the validity of this process, denying the capacity for the PCU to hijack the correction of this phase by itself. An attesting authority should never sign or publish any keys that were either generated by an untrustworthy PCU or resulted from a trusted setup whose correctness or integrity is in doubt. By requiring the PCU and the attesting authority to operate independently, the attesting authority has a strong incentive to validate the PCU correctly, as its own credibility depends on the validity of the keys it attests.

During proof generation, the PCU is not capable of forging proof parameters, as it is necessary that part of the used inputs

match the data that was originally signed by the gateway, avoiding full input spoofing. If a PCU tries to provide a fully forged proof, or a proof generated over the wrong circuit, its counterparty in this process should detect it, as the verification key loaded at the beginning of the protocol would not be compatible. Finally, by enforcing a combination of SATP's sessionId with the on-chain action being proven to be a public input, the possibility for proof reuse across different sessions is avoided. The PCU is also responsible for passing the time at which it receives the request for a proof, so that the freshness of the transaction can be guaranteed by the circuit. Yet, unless the PCU is colluding with the gateway to allow the acceptance of an older transaction, forging this value does not allow the PCU to unilaterally change or forge the transaction being proved, as the signature of the gateway over the transaction parameters needs to be correctly validated for the proof to be valid.

By being the sole responsible for storing verification keys, challenges may arise regarding the external server keeping their correction for later use. Yet, due to the way zkSNARKs work, adulterating a verification key renders it unusable with its respective proofs. This also renders any tries towards the usage of old and updated verification keys unsuccessful. By also requiring these keys to be signed or certified by an attesting authority, any capacity for full key forging is removed.

The use of MMR trees also contributes towards countering the trustability of the external server regarding the stored zkSNARKs. This mechanism allows for providing proofs, when necessary, that a zkSNARK has indeed been generated as a part of a SATP session. The external server keeps an MMR tree for each blockchain supported by our solution, with the state of each tree being tracked by its respective blockchain. The hashes of the generated proofs are submitted to the chain by the PCU, along with the new root, computed by the external server. This on-chain traceability requires the external server to keep the MMR state up to date and append only correct information. The only way for the external server to either omit zkSNARKs or tamper with them is by colluding with a PCU on what to add to the MMR.

Gateways vouch for a transaction execution and trigger the zkSNARK generation process. Unlike the current implementation of SATP, our work stops gateways from forging data regarding an on-chain transaction without colluding with a PCU, as the second component is expected to fetch the transaction parameters directly from the blockchain. Unless the two components agree on which parameters to use, there is no way for the gateway to use parameters of a non-existing transaction. Gateways are still required to sign the parameters of the transactions it vouches for, with the validity of the signature being assured by a valid zkSNARK, not allowing gateways to subsequently deny their commitments.

Gateways may be capable of finding alternative transactions whose on-chain parameters fulfill the constraints of the zkSNARK circuit, while not having any relation to the in-progress SATP session. Despite this, our solution employs several mechanisms to mitigate this. The constraints enforced by the circuit logic influence the number of alternative transactions that can be successfully provided. Storing the value of certain expected parameters (such as the wrapper contract address) in the circuit forcibly decreases the type of transactions that may be accepted. The amount, which is already a public parameter defined in the transaction request message to the origin gateway, also allows for transactions with alternative amounts to be automatically discarded.

Using a timestamp from when a proof request was received, and confirming that a certain time has not passed between the transaction occurrence and the respective proof request, is another mitigation mechanism. Any bypasses of this threshold imply a compromised PCU to lie about the proof request arrival timestamp, and a dishonest gateway to have submitted an older transaction.

With the described architecture, each component is, at most, capable of conducting a denial-of-service attack by itself, by refusing to answer any received requests.

The previous security considerations assume that all components operate independently of each other, and have no relation between them, whether on the hardware on which they run, and to who is responsible for them, officially. If the different components are not fully independent, our described assurances may not hold, allowing for at least two components to collude with each other, undermining the security and correction of SATP.

Despite the previous points, this solution steers SATP towards an architecture where gateways are not solely responsible for the correction of the protocol, and are required to cooperate with other components in order for a protocol session to be correctly completed. We manage to strengthen SATP by moving away from an architecture where each gateway is a single point of failure towards an organization where at least two independent components are required to cooperate in order for the protocol to be compromised in any way.

One alternative to our proposed architecture is the use of a multi-party computation (MPC) ceremony for the generation of the zkSNARKs' proving and verification keys. Under this approach, multiple PCUs would collaboratively participate in the generation of a single key pair, distributing the trust associated with the setup session and potentially eliminating the need for an attesting authority to oversee this process and attest to the resulting keys. However, implementing such a mechanism would require additional coordination and communication among PCUs, which would increase the complexity of both the architecture and its implementation. While we recognize this approach as a valid and promising alternative, it was considered beyond the scope of the current prototype.

V. Implementation

A proof of concept (POC) implementation is publicly available at https://github.com/Tomas-Silva2187/cacti/releases/tag/ProofOfConcept, under the tag `ProofOfConcept`, in the `packages/cacti-plugin-zk-provider` package.

Our POC was implemented with our components as web servers, exposing their functionalities through a series of endpoints that can be reached through HTTPS requests. We implemented the attesting authority as a simple server that signs verification keys for zkSNARKs and makes its own public key available upon request.

Circuits were implemented using ZoKrates [14], a toolbox that provides a Domain Specific Language to write programs where constraints, computational logic and public and private inputs can be inter-related and later compiled to an arithmetic circuit to which subsequent zkSNARKs attest. Default ZoKrates zkSNARKs are built over the BN128 elliptic curve, using the Groth16 scheme [15].

ZoKrates developers offer a JavaScript module, zokrates.js, to implement logic regarding the handling of a zkSNARK's life cycle. They also offer a series of Python scripts [16] to execute some cryptographic operations, such as SHA256 hashing and EdDSA signature generation (over the Baby Jubjub curve [17]), in a way that their validity can be verified within a ZoKrates program.

Our proposed program is showcased in Listing 1.

```
const field THRESHOLD = 240;
const u8 VERSION = 49; //1 in u8

def main(u8[55] sessionId, private u32[8] paramsHash,
     private u8[3] ethTimestamp, private u8[3]
     systemTimestamp, private u8 status, private u8[3]
     blockNumber, u8[3] amount, field[2] signR, field
     signS, field[2] signA) -> u8[55] {
    field ethTime = u8_array_to_field(ethTimestamp);
    field sysTime = u8_array_to_field(systemTimestamp);
    field txAmount = u8_array_to_field(amount);

    bool case1 = (sysTime - ethTime) <= THRESHOLD;
    bool case2 = (ethTime - sysTime) <= THRESHOLD;
    assert(case1 || case2);
    assert(status == 49);
    assert(txAmount > 0);

    u8[8] mut fullData = [0; 8];
    for u32 m in 0..3 {
        fullData[m] = amount[m];
    }
    for u32 n in 0..3 {
        fullData[3 + n] = blockNumber[n];
    }
    fullData[6] = VERSION;
    fullData[7] = status;
    u32[8] fullDataHash = sha256(fullData);
    BabyJubJubParams context = babyjubjubparams();
    bool signValidity = check_babyjubjub_sign(signR,
         signS, signA, paramsHash, fullDataHash,
         context);
    assert(signValidity == true);

    return sessionId;
}
```

Listing 1. Proposed ZoKrates program

Due to the testing setup, which deployed new blockchain instances for each run, we did not enforce the transaction's target address as an additional constraint. Under normal conditions, SATP transactions are submitted through a specific wrapper smart contract, whose address remains fixed, therefore making the target address known in advance on each blockchain.

As for the threshold, as we previously stated, the objective is to limit the universe of acceptable transactions and avoid the generation of proofs over older transactions. As so, the best-fitting value is highly dependent on each blockchain's mining frequency.

The version allows proofs to be tied to a certain version of the circuit. Every time the circuit suffers changes, this value should be incremented. This way, gateways can guarantee that their requested proofs are generated over a specific version of the circuit, with its respective constraints.

We decided to keep as public inputs any values that are already exchanged between the gateways during the protocol communications, setting as private inputs all values that are specific metadata of the transaction itself, such as the block number where the transaction was included, and its respective timestamp.

As our POC was built for a transfer between Ethereum and Hyperledger Besu, the parameters for the circuit were fetched from the transactions' receipts, using ethers.js [18].

The MMR-related components were implemented using the HerodotusDev/solidity-mmr [19] open-source library, which provides mechanisms to deal with MMR trees both in JavaScript and Solidity. Listing 2 showcases part of the logic, in a Solidity contract, to track an MMR tree's state.

The root update function takes a single hash (the hash of one or multiple transactions), along with the root that was computed by the external server. The computed new root needs to match between the blockchain and the external server in order for the root update to be accepted and permanently integrated on-chain.

```
contract MMRChecker {
    bytes32 treeRoot_local;
    bytes32[] treePeaks_local;
    uint elementsCount_local;
    function rootUpdate(bytes32 element, bytes32
         newRoot, bytes32[] calldata new_peaks, uint
         new_elements_count) external returns (bool) {

        (uint nextElementsCount, bytes32 nextRootHash,
             bytes32[] memory nextPeaks) = StatelessMmr
             .appendWithPeaksRetrieval(
                element,
                treePeaks_local,
                elementsCount_local,
                treeRoot_local
            );
        require(newRoot == nextRootHash, "Root mismatch
             ");
        require(new_elements_count == nextElementsCount
             , "Elements count mismatch");
        require(keccak256(abi.encodePacked(new_peaks))
             == keccak256(abi.encodePacked(nextPeaks)),
              "Peaks mismatch");
        treePeaks_local = nextPeaks;
        elementsCount_local = nextElementsCount;
        treeRoot_local = nextRootHash;
        return true;
    }
}
```

Listing 2. Proposed MMR tracker smart contract

VI. Evaluation

We compare our solution with the current version of SATP, which provides no formal proving mechanism and enforces no explicit privacy guarantees on the commitments exchanged regarding transaction execution.

A. Experimental Evaluation

The experimental evaluation was conducted on a machine with 12GB of dedicated RAM and a 12-core CPU, with all components sharing these resources simultaneously. All benchmarks were performed on pre-initialized components to mitigate any effects from their startup on this measurement. We benchmarked the latency introduced by adding four zkSNARKs to SATP, the PCU latency for proof generation under increasing levels of concurrency, and the corresponding variation in resource consumption. Concurrency benchmarking was focused on proof generation, as proof verification is comparatively brief and computationally lightweight.

Table I shows the average execution times of both SATP implementations, with our solution substantially increasing the overall protocol execution latency by more than 10 times.

TABLE I
LATENCY STATISTICS FOR THE BASELINE AND MODIFIED PROTOCOL VERSIONS ACROSS 30 EXECUTIONS

Protocol	Mean	Standard Deviation	Minimum	Maximum
Standard SATP	15.157	0.99992	14.228	20.112
SATP with zkSNARKs	223.51	4.56075	215.04	230.89

As zkSNARKs support succinct verification, witness and proof generation are the cryptographic processes that mostly affect the solution's latency. The average duration of these steps is shown in Figures 5 and 6, which reflect their execution stability as concurrency increases. 6-threaded executions do become more unstable since, as shown in Figure , almost all allocated memory is used.

The circuit's signature verification and hash computation increase the proof complexity, contributing to the high generation latency.

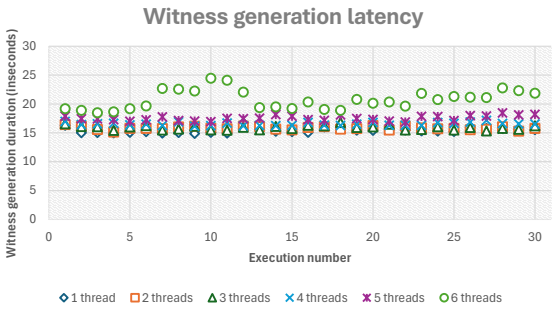


Fig. 5. Latency of witness generation in different concurrency environments

Concurrency does not, by itself, significantly affect the overall latency of our solution, which is directly connected to resource exhaustion from the PCU when generating zkSNARKs.

B. Results Discussion

We proposed a set of objectives for our work, listed at the start of Section III.

The first one regards avoiding gateways learning more about the internal state of the opposite blockchain in the transfer, by limiting the information that is exchanged about transactions, while still requiring gateways to verifiably commit themselves to a transaction. By relying on the PCU to fetch a transaction, and then employing zkSNARKs to verify that part of that transaction's parameters were signed, without disclosing their entire set, it is possible to make gateways accountable for a real transaction on a blockchain.

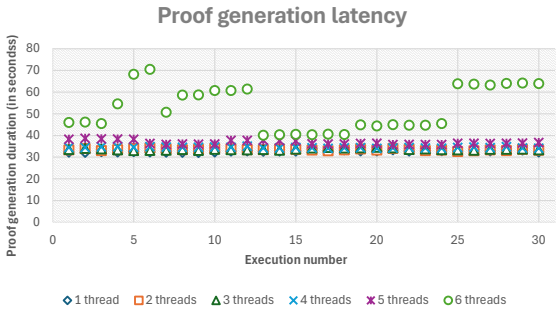


Fig. 6. Latency of proof generation in different concurrency environments

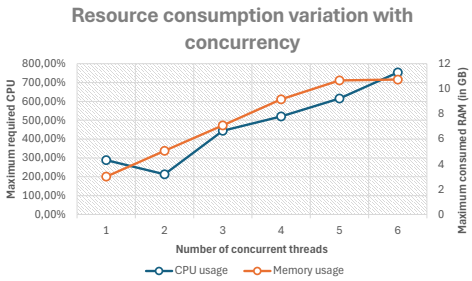


Fig. 7. Maximum resource consumption across increased concurrency levels

Our second objective aimed at distributing the proof generation and verification process, so that gateways are not the sole responsible components for these crucial tasks, requiring them to cooperate with other independent components.

As highlighted at the beginning of Section III, SATP currently relies solely on the gateways, with no verification being applied to what they provided as proof, allowing for full parameter forgery. This is mitigated by enforcing that the gateways sign transaction parameters, requiring the PCUs to fetch the parameters for the same transaction and guarantee that they have been signed.

The possibility of misreporting is decreased but still assumed possible (a gateway requiring a zkSNARK on a transaction not related to SATP, but whose parameters satisfy the constraints enforced by the zkSNARK circuit). By limiting the transaction to have occurred within a certain time frame, as described in Section III, and combining this with the verification of any transaction data parameters that are expected to have a certain pre-defined value, this risk is mitigated.

By proposing new components that execute their role self-sufficiently and on-demand, to simply provide gateways with a cryptographic proof, the generality of SATP is not compromised. The proposed solution was validated only with a test scenario involving Ethereum and Hyperledger Besu, two

EVM-compatible blockchains, and part of the logic enforced by the proofs' circuit depends on the type of information these chains provide. However, these constraints do not fundamentally affect the generality of this solution's approach, as the core mechanisms remain applicable without requiring an overall solution design.

While our solution introduces considerable latency to SATP, this overhead may represent an acceptable trade-off in use cases where the transferred assets carry real-world value or where unnecessary disclosure of transaction data is to be minimized. Ongoing efforts toward real-world asset tokenization [20] [21] create scenarios in which tokens held on a blockchain represent ownership of, or claims over, associated off-chain assets. In such cases, where fraudulent transfers may have consequences beyond the blockchains themselves, stronger security assurances may be preferred. Moreover, particularly with permissioned blockchains, minimizing metadata disclosure is a desirable property that may increase operators' willingness to adopt the protocol, as it assures that only the information required to prove the relevant predicates for cross-chain transfers needs to be shared.

In both cases, the additional introduced latency may therefore be justified by the stronger correctness and privacy guarantees.

VII. Conclusion

Recent efforts on developing cross-blockchain asset transfer mechanisms have highlighted a lack of established standards. SATP, currently being developed within the IETF, addresses this gap through a gateway-based protocol for transferring assets across blockchain networks.

Our first contribution is a privacy-preserving proving mechanism based on zkSNARKs, allowing relevant transaction properties to be proved, in the context of SATP, without disclosing unnecessary internal data of blockchains. The second contribution improves resilience against gateway misreporting. In the current implementation of SATP, claims about on-chain transactions are primarily trusted if signed by the gateway. Our solution reduces this dependency by introducing the PCU, an external server and an attesting authority. The PCU independently retrieves transaction data from the blockchain and verifies that the gateway has committed itself to parameters derived from an existing transaction, preventing arbitrary transaction claims. However, a malicious gateway may still select an unrelated transaction whose parameters satisfy the constraints enforced by the zkSNARK's underlying circuit. This risk is mitigated by the strictness of these constraints, including restrictions on the acceptable transaction time-frame.

Although the proposed solution adds substantial latency, this overhead may be acceptable in scenarios where stronger security and privacy guarantees outweigh performance requirements.

VIII. Acknowledgments

This work was supported by national funds through the project Blockchain.PT - Decentralize Portugal with Blockchain Agenda (Project No. 51), WP7: Interoperability, Call no 02/C05-i01.01/2022. It was funded by the Portuguese Recovery and Resilience Plan (PRR), the Portuguese Republic, the European Union (EU) under the Next Generation EU framework, and supported by Fundação para a Ciência e a Tecnologia, I.P. (FCT), under projects UID/50021/2025, DOI https://doi.org/10.54499/UID/50021/2025 and UID/PRR/50021/2025, DOI https://doi.org/10.54499/UID/PRR/50021/2025, associated with INESC-ID.

References

[1] R. Belchior, A. Vasconcelos, S. Guerreiro, and M. Correia, "A survey on blockchain interoperability: Past, present, and future trends," *Acm Computing Surveys (CSUR)*, vol. 54, no. 8, pp. 1–41, 2021.

[2] R. Belchior, J. Süßenguth, Q. Feng, T. Hardjono, A. Vasconcelos, and M. Correia, "A brief history of blockchain interoperability," *Communications of the ACM*, vol. 67, no. 10, pp. 62–69, 2024.

[3] A. Augusto, R. Belchior, M. Correia, A. Vasconcelos, L. Zhang, and T. Hardjono, "Sok: Security and privacy of blockchain interoperability," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 3840–3865.

[4] M. Zhang, X. Zhang, J. Barbee, Y. Zhang, and Z. Lin, "Sok: Security of cross-chain bridges: Attack surfaces, defenses, and open problems," *arXiv preprint arXiv:2312.12573*, 2023.

[5] M. Hargreaves, T. Hardjono, R. Belchior, V. Ramakrishna, and A. Chiriac, "Secure Asset Transfer Protocol (SATP) Core," Internet Engineering Task Force, Internet-Draft draft-ietf-satp-core-12, Nov. 2025, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-satp-core/12/

[6] E. Abebe, Y. Hu, A. Irvin, D. Karunamoorthy, V. Pandit, V. Ramakrishna, and J. Yu, "Verifiable observation of permissioned ledgers," in *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2021, pp. 1–9.

[7] C. Reitwiessner, "zksnarks in a nutshell," *Ethereum blog*, vol. 6, pp. 1–15, 2016.

[8] M. Bellés-Muñoz, M. Isabel, J. L. Muñoz-Tapia, A. Rubio, and J. Baylina, "Circom: A circuit description language for building zero-knowledge applications," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 6, pp. 4733–4751, 2022.

[9] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Scalable, transparent, and post-quantum secure computational integrity," *Cryptology ePrint Archive*, 2018.

[10] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *2018 IEEE symposium on security and privacy (SP)*. IEEE, 2018, pp. 315–334.

[11] D. Hopwood, S. Bowe, T. Hornby, N. Wilcox *et al.*, "Zcash protocol specification," *GitHub: San Francisco, CA, USA*, vol. 4, no. 220, p. 32, 2016.

[12] N. Van Saberhagen, "Cryptonote v 2.0," 2013.

[13] Z. J. Williamson, "The aztec protocol," *URL: https://github.com/AztecProtocol/AZTEC*, 2018.

[14] ZoKrates Project, "Zokrates documentation," https://zokrates.github.io/, 2026, accessed: 2026-03-15.

[15] J. Groth, "On the size of pairing-based non-interactive arguments," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2016, pp. 305–326.

[16] ZoKrates, "pycrypto: Crypto application code for the zksnarks toolbox zokrates," https://github.com/Zokrates/pycrypto, 2024, accessed: 2026-03-16.

[17] B. WhiteHat, J. Baylina, and M. Bellés, "Baby jubjub elliptic curve," *Ethereum Improvement Proposal, EIP-2494*, vol. 29, p. 27, 2020.

[18] ethers Project, "ethers.js v6 documentation," https://docs.ethers.org/v6/, 2026, accessed March 15, 2026; official documentation for the ethers.js JavaScript library for Ethereum blockchain interaction.

[19] HerodotusDev, "solidity-mmr: Solidity merkle mountain range library," https://github.com/HerodotusDev/solidity-mmr, 2024, gitHub repository.

[20] H. Benedetti and G. Rodríguez-Garnica, "Tokenized assets and securities," 2023.

[21] G. Sazandrishvili, "Asset tokenization in plain english," *Journal of Corporate Accounting & Finance*, vol. 31, no. 2, pp. 68–73, 2020.