Implementing Secure Bridges: Learnings from the Secure Asset Transfer Protocol

Kjell-Erik Marstein 1,1, Alexandru Chiriac 2, Luke Riley 2, Thomas Hardjono 2, and Gilbert Verdian 2

1Quant Network
2Affiliation not available

October 31, 2023

Abstract

Securely transferring tokenised assets between largely non-interoperable blockchain networks is a great challenge to take on. Implementations must coordinate transactions on multiple blockchain networks which have inherently different characteristics and functionality. Moreover, implementations must coordinate transactions in strict order, in an atomic, synchronised way so that owners do not risk their assets becoming lost, while also mitigating the age-old problem of “avoiding double spend”. Several protocols for cross-network asset transfer, i.e. the bridging of assets, exist, and several production bridge implementations exist. However, frequent high-profile security breaches have arguably given bridges a bad reputation. Therefore, there is a clear need for standardised bridge protocols to re-establish trust. One initiative for standardising cross-network bridges is the Secure Asset Transfer Protocol (SATP) developed by the Internet Engineering Task Force (IETF). This protocol establishes four distinctive phases for the asset transfer process, driven by a standardised messaging cycle between specialised bridge orchestration applications, known as gateways. In this paper, we demonstrate a reference framework with functions needed to implement SATP. We describe our implementation of this reference framework, why SATP is unique and why SATP is likely to be an important contribution to powering future bridges.
Implementing Secure Bridges: Learnings from the Secure Asset Transfer Protocol

Kjell-Erik Marstein
Quant Network
London, UK
kjell-erik.marstein@quant.network

Alexandru Chiriac
Quant Network
London, UK
alexandru.chiriac@quant.network

Luke Riley
Quant Network
London, UK
luke.riley@quant.network

Thomas Hardjono
Massachusetts Institute of Technology (MIT)
Massachusetts, USA
hardjono@mit.edu

Gilbert Verdian
Quant Network
London, UK
gilbert.verdian@quant.network

Abstract—Securely transferring tokenised assets between largely non-interoperable blockchain networks is a great challenge to take on. Implementations must coordinate transactions on multiple blockchain networks which have inherently different characteristics and functionality. Moreover, implementations must coordinate transactions in strict order, in an atomic, synchronised way so that owners do not risk their assets becoming lost, while also mitigating the age-old problem of “avoiding double spend”. Several protocols for cross-network asset transfer, i.e. the bridging of assets, exist, and several production bridge implementations exist. However, frequent high-profile security breaches have arguably given bridges a bad reputation. Therefore, there is a clear need for standardised bridge protocols to re-establish trust. One initiative for standardising cross-network bridges is the Secure Asset Transfer Protocol (SATP) developed by the Internet Engineering Task Force (IETF). This protocol establishes four distinctive phases for the asset transfer process, driven by a standardised messaging cycle between specialised bridge orchestration applications, known as gateways. In this paper, we demonstrate a reference framework with functions needed to implement SATP. We describe our implementation of this reference framework, why SATP is unique and why SATP is likely to be an important contribution to powering future bridges.

Index Terms—blockchain, bridge, interoperability, tokenisation

I. INTRODUCTION

The current blockchain and distributed ledger technology (DLT) landscape consists largely of non-interoperable, siloed networks, where a single technology has not taken precedence. The networks are based on different platforms which rely on different mechanisms for block production and consensus, resulting in varying transaction throughput and different considerations for block finality. This gives each network inherently different characteristics. The current landscape is likely to exist for the rest of this decade, thus interoperability between such “walled gardens” is one of the biggest remaining challenges for widespread blockchain adoption [1].

With the expanding popularity of blockchains, many traditional assets are being tokenised [2]. This includes both liquid assets, e.g., funds, commodities and currencies, and illiquid assets such as real estate and land [3]–[6].

A major challenge for tokenisation is inefficient and fragmented markets and systems due to a lack of cross-network functionality. An asset with inherent value on one network is worthless when forced to transact on or with a different network. Interconnecting these different networks and platforms opens access to multiple use cases where the asset is used on the destination network, e.g. to gain newfound asset liquidity in financial markets.

Bridges exist to interconnect these disparate DLT and blockchain ecosystems so that assets can move in one or both directions. In this paper, we take a closer look at the Secure Asset Transfer Protocol (SATP) [7], [8]. The goal of the protocol is to standardise the behaviour of cross-network bridges for the purpose of DLT interoperability. The SATP effort is being conducted in a new technical working group under the auspices of the Internet Engineering Task Force (IETF), which has standardised several of the current Web2 protocols (e.g. TCP/IP, routing and TLS security).

Besides the interoperability of blockchain networks – as measured through ease of asset movements by their holders – a key motivation for the standardisation of a secure cross-network asset transfer protocol is to address the poor security quality of the many proprietary cross-network bridge protocols that have resulted in the loss of over a billion dollars in the past couple of years [10].

Many of these proprietary protocols lack clear written specifications, which makes formal security protocol analysis [11] very difficult to perform. In turn, this lack of verified standard technical specifications discourages financial institutions from investing in the new computing infrastructures that integrate blockchain technology into their existing financial IT infrastructure. SATP aims to provide security through confidentiality-protected, source-authenticated and non-repudiable messages, which allow the asset transfer

1Interested contributors are invited to join. To do so, see [9].
process to satisfy ACID (atomicity, consistency, isolation, and durability) and no-double spend properties. Additionally SATP allows for liabilities to be placed on organisations involved in the running of the bridge, providing a legal safety net for bridge users.

The rest of the paper is structured as follows. Section 2 introduces high level categorisations of bridges. Section 3 describes SATP. Section 4 details our reference implementation. Section 5 presents our learnings from building the reference implementation and Section 6 lists future work and concludes.

II. WHAT ARE BRIDGES?

Bridges are a common name for software systems that perform asset transfer interoperability functionality between DLT networks. Usually, a bridge transfers assets from a single origin DLT network to a single destination DLT network.

The entities who run the bridge software can be referred to as bridge operators. Contrary to the usual belief in blockchain communities, all cross-network interoperability solutions require a trusted third party for the cross-network action to be performed correctly [12]. In terms of an individual bridge, the operators work together to form this trusted third party.

There is a significant amount of bridges in use today. Therefore, it helps to understand them through categorisations. As there are several ways to categorise bridges, in the following subsections we introduce selected categories, which are used to place SATP implementing bridges within the domain.

A. Categorising Bridges by their Operators

The following categories are based on how bridge operators are selected [13]:

- **Single Organisation**: One operator has the authority to update user balances in the destination DLT network. An example is Circle's CCTP bridge [14], which mints USDC tokens on the destination DLT network after they have been burned on the origin DLT network.
- **Multi-Organisation**: A set of operators (K of N) have the authority to update user balances in the destination DLT network. An example is the wBTC bridge [15], which operates via multi-signature.
- **Crypto-Economic (Open)**: A dynamic set of operators can update user balances in the destination DLT network. Due to its open nature, this bridge type usually has a challenge period for other bridge operators to confirm or reject a suggested transaction, such as the Polygon Plasma bridge (linking Polygon and Ethereum mainnets), which has a 7 day challenge period [16]. Additionally in open crypto-economic bridges, the bridge operators may be weighted by stake, such as the Polygon PoS bridge (also linking the Polygon and Ethereum mainnets), which requires the bridge operators with 2/3 + 1 percentage of the stake to approve a transfer before it occurs [17].

B. Categorising Bridges by their Asset Transfer Flow

The following categories are based on how bridges allow for the movement of the assets [18]:

- **Push-based Asset Transfers**: In this scenario, the first transaction related to the cross-network asset transfer occurs on the origin DLT network. Once this transaction is confirmed and accepted by the DLT network, the bridge operators will trigger a corresponding transaction in the destination DLT network. This is the most common form of asset transfer bridging and is a requirement when bridging between one layer 1 network\(^2\) to another. For example, the Avalanche bridge [20], which moves assets from the Ethereum mainnet to the Avalanche mainnet or vice-versa, is push-based.
- **Pull-based Asset Transfers**: In this scenario, the first transaction related to the cross-network asset transfer occurs on the destination DLT network. Once this transaction is confirmed and accepted by the DLT network, the bridge will create/unlock the requested asset on the destination network and any corresponding asset on the origin network is deleted. This type of bridge can only work when a layer 1 network removes an asset from a layer 2 network\(^3\), usually through the layer 2 network's inbuilt dispute resolution process. For example, tokens can be withdrawn from ZK-rollups (a type of scaling solution based on Zero Knowledge Proofs and data compression) on the Ethereum Mainnet via an ownership proof sent to the rollup smart contract on layer 1 [13].

C. Categorising Bridges by their Operator View

The following categories, introduced in this paper, are based on the level of access the bridge operators have on the networks:

- **Full View**: In a full-view bridge, every one of the bridge operators can independently verify cross-chain asset transfers because they all have access to all of the necessary sections of both the origin and destination DLT networks. This is by far the most common setup for bridges. Moreover, given that permissionless (open) networks are publicly viewable, bridges that connect permissionless networks fall into this category.
- **Partial View**: In a partial-view bridge, at least some of the operators cannot independently verify cross-chain asset transfers because at least some operators cannot access all of the necessary sections of both the origin and destination DLT networks. An example of a partial view bridge, is a bridge established using SATP to connect two permissioned DLT networks, where none of the SATP bridge operators can view both distributed ledgers (SATP is further explained in the next section).

III. IETF SECURE ASSET TRANSFER PROTOCOL

The Secure Asset Transfer Protocol (SATP) is unique in that it is a method to establish temporary multi-organisational,\(^2\) A layer 1 network is a standalone DLT network [19], i.e. the consensus between the nodes of this network are not influenced by the state of another DLT network.

\(^3\) A layer 2 network 'tethered' to a layer 1 network [19], i.e. the consensus between the nodes of layer 2 network are influenced by the state of a layer 1 DLT network.
A key design principle underlying SATP is the assumption that some (many) blockchain systems in the future will be private permissioned (closed) in the sense that the ledger is accessible (read/write) to authorised entities only. The notion of a closed blockchain system reflects not only the current closed IT infrastructures found in most financial institutions, but it also reflects the basic human social behaviour of trading within communities. An example of this design principle can be found in the Project UBIN [21] from the Monetary Authority of Singapore (MAS).

A. SATP Gateways

The draft version of SATP states that these temporary bridges established through SATP must contain two gateway applications: the origin gateway and the destination gateway. These gateways are connected to one network each, the origin and destination DLT network respectively. Each gateway orchestrates the bridge on its side of the operation. A different organisation can run each gateway but due to the partial view constraint, they must trust each other.

To establish and maintain trust, gateways identify each other using public keys. The gateways sign all of the SATP standardised messages sent between them, in order to prove their identity and the validity of their messages.

Once the temporary SATP bridge with two gateways has formed, these partner organisations act as a single trusted third party, which the asset owner must rely on for the asset transfer from the origin to the destination DLT network to take place.

The current version of SATP assumes that gateway owners are truthful. This is a known limitation of SATP and may be removed in future work. For now, malicious gateway operators need to be dealt with outside of SATP, for instance by revoking the access of a suspected malicious operator and recovering funds through legal means.

Lastly, each gateway can implement the entirety of SATP, meaning that even if one organisation’s gateway is used as the origin gateway in an initial SATP bridge, it can be used as the destination gateway in a future SATP bridge.

B. SATP Phases

SATP describes four phases of bridging:
0) Identity and Asset Verification Flow
1) Transfer Initiation Flow
2) Lock-Evidence Verification Flow
3) Commitment Establishment Flow

Each phase has its own set of standardised messages, which are signed, exchanged and logged by each gateway. This logging is essential for the system’s auditability.

Phase 0 is concerned with establishing trust between the two gateways. The SATP protocol does not consider how gateways choose to do this, and this phase is left out of the current specification. Essentially though, each gateway must provide proof of who they are to the other gateway.

In phase 1, the origin gateway proposes a bridging profile (specification) to the destination gateway. The profile includes information about the session that is to occur, such as the type of asset to be bridged. The destination gateway must now accept or reject the request.

In phase 2, the origin gateway provides confirmation to the destination gateway that the previously specified transfer should go ahead. To do so, the origin gateway must provide evidence that the assets have been locked on the origin DLT network. If the destination gateway finds the evidence sufficient, it then accepts the transfer and communicates this acceptance back to the origin gateway. We discuss the definition of sufficient evidence in the next sub-section.

Phase 3 of SATP starts with the origin gateway stating its commitment to the transfer, by sending a commitment preparation message to the destination gateway. The destination gateway then mints and temporarily locks the assets on the destination DLT network. Subsequently, the origin gateway burns the assets on the origin network and sends a commitment final message to the destination gateway. In response to this, the destination gateway unlocks the assets on the destination network. The phase ends with the origin gateway confirming it is happy with the bridge by delivering a transfer complete message to the destination gateway.

C. SATP Lock Evidence

The interoperability framework underlying the design of SATP intentionally leaves the form of the lock evidence to be defined separately under distinct technical specifications. This is driven by the fact that, currently, there are numerous blockchains and DLT systems, and each may have its own unique scheme for its lock evidence. Thus, as part of the initial negotiations in phase 0 between the two gateways, the origin gateway must communicate to the destination gateway the type of lock evidence that the origin DLT network supports. In the case that the destination gateway is unable to parse and evaluate the lock evidence in a meaningful way, the destination gateway can simply decline the transfer request.

IV. IMPLEMENTING SECURE ASSET TRANSFER

We present learnings from implementing bridges based on the current version of SATP. Our implementation supports bridging fungible and non-fungible assets between permissioned and permissionless DLT networks using the Ethereum Virtual Machine (EVM)\(^4\), and permissioned Hyperledger Fabric networks, in the following configurations:

- EVM network to EVM network
- EVM network to Hyperledger Fabric network
- Hyperledger Fabric network to EVM network
- Hyperledger Fabric network to Hyperledger Fabric network

\(^4\)Multiple permissionless DLT networks use the Ethereum Virtual Machine, such as Ethereum Mainnet, Polygon, Avalanche C-Chain and the XDC Network. Additionally many permissioned DLT networks use the EVM.
the ID from the origin network. As a prerequisite for the asset destination network is configurable and does not have to equal burn assets must be configured in the asset smart contracts. of assets on the destination network. Permissions to mint and assets on the origin network and to mint the same number functionality of the bridge contracts is thus to lock and burn with the origin and destination asset smart contracts. The main 1. These bridge smart contracts are responsible for interactions from scratch to be SATP compliant.

A. Implemented SATP Gateways

Each SATP gateway application needs to interact with its assigned DLT network. This will normally require the use of DLT-specific software libraries and custom functionality depending on which network the gateway connects to. To avoid having to create multiple implementations to support each gateway to DLT network connection, we use Quant’s Overledger [22]. Overledger provides a set of standardised endpoints that allow us to interact with multiple DLT networks through the same interface. We can therefore use the same gateway application for all networks supported by Overledger, including the EVM and Hyperledger Fabric networks used in our initial implementation.

The gateway application implements a messaging interface compliant with SATP. This interface is described in the SATP specification draft.

B. Bridge Smart Contracts

Details that are unique to the underlying network, such as the design of scripts and smart contracts to move the assets, are not covered by the current version of SATP, and the smart contract implementation we discuss here is thus designed by us from scratch to be SATP compliant.

In our SATP implementation, each gateway is the operator of a bridge smart contract in their network, as illustrated in Fig. 1. These bridge smart contracts are responsible for interactions with the origin and destination asset smart contracts. The main functionality of the bridge contracts is thus to lock and burn assets on the origin network and to mint the same number of assets on the destination network. Permissions to mint and burn assets must be configured in the asset smart contracts.

If the asset is non-fungible, the minted asset’s ID on the destination network is configurable and does not have to equal the ID from the origin network. As a prerequisite for the asset transfer, the asset owner must have approved the bridge smart contract to take the assets.

The contract can also be configured to not burn the assets, effectively keeping the bridged assets infinitely locked on the origin network. However, by burning the assets, we avoid building liquidity in the bridge account. This eliminates the risk of the bridged asset becoming worthless in the case where a hacker releases locked assets from the origin network’s bridge smart contract. A hacker could be able to do this by discovering a vulnerability in the origin network’s bridge contract code or in the gateway’s off-chain infrastructure. For the remainder of this paper, we will only discuss the model where the bridge burns the assets on the origin network.

The bridge smart contract has a set of configurable properties:

- **OperatorAccounts**: The DLT accounts that can invoke the asset transfer functions of the bridge smart contract. Multiple operator accounts are allowed for resiliency reasons. Yet to align with the initial SATP version, it is assumed that all of these accounts are controlled by one organisation; the organisation controlling the bridge gateway of this DLT network.
- **SupportedAssets**: The unique identifiers of the asset types that this bridge supports (e.g. by smart contract address).
- **TimeOutInSeconds**: Defines the default length of time from when a request to transfer assets is sent to when the request expires. If the gateways do not complete the request by this expiry time, the asset owner can cancel the request and reclaim the assets.
- **Owner**: The bridge contract owner’s DLT account. This account has special privileges to add or remove OperatorAccounts and SupportedAssets, and to change the TimeOutInSeconds and bridge contract owner. If these properties are to be static, and thus assigning an owner is not desirable, the role can be assigned to the zero address (Ethereum) or be omitted from the implementation.

All of these configurable properties must be initially specified in the constructor when the contract is deployed. As well as the constructor, our implementation uses the 13 functions described in the following list (where the function parameters are displayed in sub-lists), which we propose as an SATP-compliant smart contract interface. The interface is also illustrated in Fig. 2, which shows an origin and destination bridge smart contract, their respective functions and the events emitted by each contract (further explained in the next section).

In the following list, functions 1 to 6 are listed in the order of which they are executed, and are part of the successful flow. Only the bridge contract OperatorAccounts can call functions 1 to 6. Functions 7 to 13 are supplementary functions, where function 7 is called by either the asset owner or a gateway if the asset transfer cannot be completed (requirements for cancelling a transfer are described in the next section). Functions 8 to 13 can only be called by the bridge contract Owner.

1) **initiateTransfer**: Invoked on the origin network bridge contract to lock assets. Creates an assetTransferId that
### Functions:
1) initiateTransfer
2) acceptTransfer
3) prepareTransfer
4) cancelTransfer
5) finaliseTransfer
6) completeTransfer
7) addOperatorAccount
8) removeOperatorAccount
9) addSupportedAsset
10) removeSupportedAsset
11) changeTransferTimeOut
12) changeContractOwner

### Events:
- Transfer Initiated
- Transfer Committed
- Transfer Complete
- Transfer Finalised
- Transfer Cancelled

---

**Origin Bridge Smart Contract**

<table>
<thead>
<tr>
<th>Functions:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) initiateTransfer</td>
</tr>
<tr>
<td>2) cancelTransfer</td>
</tr>
<tr>
<td>3) completeTransfer</td>
</tr>
<tr>
<td>4) addOperatorAccount</td>
</tr>
<tr>
<td>5) removeOperatorAccount</td>
</tr>
<tr>
<td>6) addSupportedAsset</td>
</tr>
<tr>
<td>7) removeSupportedAsset</td>
</tr>
</tbody>
</table>

**Destination Bridge Smart Contract**

<table>
<thead>
<tr>
<th>Functions:</th>
</tr>
</thead>
<tbody>
<tr>
<td>2) acceptTransfer</td>
</tr>
<tr>
<td>3) prepareTransfer</td>
</tr>
<tr>
<td>4) cancelTransfer</td>
</tr>
<tr>
<td>5) finaliseTransfer</td>
</tr>
<tr>
<td>6) completeTransfer</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Events:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transfer Accepted</td>
</tr>
<tr>
<td>Transfer Prepared</td>
</tr>
<tr>
<td>Transfer Finalised</td>
</tr>
<tr>
<td>Transfer Cancelled</td>
</tr>
</tbody>
</table>

---

Fig. 2. Our reference SATP functions framework for the respective origin and destination bridge smart contracts.

is unique to the particular asset transfer request.

- **originId**: The ID of the account sending the asset from the origin network.
- **originAssetTypeId**: The ID identifying the type of asset being transferred from the origin network (e.g. the asset’s smart contract address).
- **originAssetId (optional)**: The ID of the non-fungible asset to be transferred from the origin network (not defined if the asset is fungible).
- **destinationNetworkId**: The ID of the destination network.
- **destinationBridgeContractId**: The ID of the destination bridge smart contract.
- **destinationAssetTypeId**: The ID identifying the type of asset being created in the destination network (e.g. the asset’s smart contract address).
- **destinationAssetId (optional)**: The ID of the non-fungible asset to be minted on the destination network (not defined if the asset is fungible).
- **destinationId**: The ID of the account on the destination network to send the transferred assets to.
- **amount**: The number of assets to transfer (1 if the asset is non-fungible).
- **expiresAt**: The time at which the asset transfer expires. This value should match the timestamp calculated by the `initiateTransfer` function (which equals the current timestamp plus the number of seconds specified in the `TimeOutInSeconds` property).

3) **prepareTransfer**: Invoked on the destination network to mint the new assets. The assets are temporarily locked in the destination bridge smart contract account.

- **assetTransferId**: The ID of the asset transfer.

4) **commitTransfer**: Invoked on the origin network to burn the assets locked on the origin network.

- **assetTransferId**: The ID of the asset transfer that identifies which assets to be burned.

5) **finaliseTransfer**: Invoked on the destination network to unlock the minted assets on the destination network, i.e. transfer the minted assets to the destination account.

- **assetTransferId**: The ID of the asset transfer.

6) **completeTransfer**: Invoked on the origin network to mark the asset transfer as completed.

- **assetTransferId**: The ID of the asset transfer that has completed.

The `initiateTransfer` and `acceptTransfer` functions are invoked during SATP’s Lock-Evidence Verification Flow, where the origin gateway must present proof to the destination gateway that the origin assets have been locked. In our implementation, the `initiateTransfer` function emits an event on the origin network which holds information about the bridge and the locked assets. The origin gateway uses this event as evidence. The `acceptTransfer` function is invoked by the destination gateway to register the event on the destination network and confirm that it has accepted the evidence.

The `commitTransfer`, `finaliseTransfer` and `completeTransfer` functions are invoked as part of SATP’s Commitment Establishment Flow, i.e. the final phase of the bridge where the original assets are burned on the origin network and new assets are minted on the destination network.

We also add the following supporting functions to the interface:

7) **cancelTransfer**: The asset owner or a bridge OperatorAccount can cancel the asset transfer. The requirements for cancelling are described in the next section.

- **assetTransferId**: The ID of the asset transfer to cancel.
8) **addOperatorAccount**: Adds an account to the list of approved operator accounts.
   - `operatorAccountId`: The operator account ID to add.

9) **removeOperatorAccount**: Removes an account from the list of approved operator accounts.
   - `operatorId`: The operator account ID to remove.

10) **addSupportedAsset**: Creates a relationship between an origin and destination asset, and adds the relationship to the list of assets that are supported by this bridge contract. If the origin and destination assets specified in the `initiateTransfer` and `acceptTransfer` functions do not match a listed relationship, the transfer will be rejected.
   - `originNetworkId`: The ID of the network where the asset is deployed (could be the same network as the bridge contract is deployed on).
   - `originAssetTypeId`: The ID identifying the type of asset being transferred from the origin network.
   - `destinationNetworkId`: The ID of the destination network (could be the same network as the bridge contract is deployed on).
   - `destinationAssetTypeId`: The ID identifying the type of asset being created in the destination network.

11) **removeSupportedAsset**: Removes an asset relationship from the list of supported assets.
    - `originNetworkId`: The ID of the network where the asset is deployed.
    - `originAssetTypeId`: The ID identifying the type of asset being transferred from the origin network.
    - `destinationNetworkId`: The ID of the destination network.
    - `destinationAssetTypeId`: The ID identifying the type of asset being created in the destination network.

12) **changeTransferTimeOut**: Changes the default value used to calculate the time an asset transfer expires, i.e. the time when the asset owner can reclaim the assets on the origin network (e.g. in case the destination network or destination gateway is not responding). Changing the default value does not affect existing requests.
    - `seconds`: The number of seconds before an asset transfer can be cancelled.

13) **changeContractOwner**: Changes the owner of the bridge smart contract.
    - `ownerId`: The ID of the new bridge contract owner.

C. Asset Transfer Events

When the functions in the bridge smart contracts are invoked successfully, events are emitted on the network, to which the gateway or a client application can listen to. The events hold the following details:

- **Asset Transfer ID**: The ID of the asset transfer.
- **Origin Network ID**: The ID of the origin network.
- **Origin Bridge Contract ID**: The ID of the origin bridge smart contract.
- **Origin Asset Type ID**: The ID of the origin asset type.
- **Origin Asset ID (optional)**: The ID of the non-fungible asset that is being transferred from the origin network (not defined if the asset is fungible).
- **Origin Account ID**: The ID of the account in the origin network that the assets are being taken form.
- **Destination Network ID**: The ID of the destination network.
- **Destination Bridge Contract ID**: The ID of the destination bridge smart contract.
- **Destination Asset Type ID**: The ID of the destination asset type.
- **Destination Asset ID (optional)**: The ID of the non-fungible asset that is being minted on the destination network (not defined if the asset is fungible).
- **Destination Account ID**: The account ID on the destination network that the assets are being transferred to.
- **Token Amount**: The number of assets that are being transferred (1 if the asset is non-fungible).
- **Event Timestamp**: The time when the event was emitted.
- **Event Type**: The type of event (explained below).
- **Expiry Timestamp**: The time when the asset owner can reclaim the assets on the origin network. This equals the event timestamp plus the number of seconds specified in the bridge’s `TimeOutInSeconds` property.

These details serve as references to the bridge asset transfer request. The gateway listening to the network picks up the events and includes them in the signed messages exchanged between the gateways (e.g. as part of the lock evidence).

The types of events being emitted on the networks are:

- **Transfer Initiated**: Emitted by the `initiateTransfer` function on the origin network, confirming that the asset owner has initiated the asset transfer.
- **Transfer Accepted**: Emitted by the `acceptTransfer` function on the destination network, confirming that the destination gateway has accepted the asset transfer request.
- **Transfer Prepared**: Emitted by the `prepareTransfer` function on the destination network, confirming that the assets have been minted on the destination network.
- **Transfer Committed**: Emitted by the `commitTransfer` function on the destination network, confirming that the assets have been destroyed on the origin network.
- **Transfer Finalised**: Emitted by the `finaliseTransfer` function on the destination network, confirming that the minted assets have been transferred to the destination account.
- **Transfer Complete**: Emitted by the `completeTransfer` function on the origin network, confirming that the asset transfer has been completed.
- **Transfer Cancelled**: Emitted by the `cancelTransfer` function on the origin network, confirming that the asset transfer has been cancelled.

Each asset transfer is associated with an expiry timestamp. If a **Transfer Committed** event is not emitted on the origin network before the expiry, the asset owner or an **OperatorAccount** of the origin bridge smart contract can invoke the
cancelTransfer function to reclaim the locked assets on the origin network.

Similarly, if a Transfer Finalised event is not emitted in time on the destination network, an OperatorAccount of the destination bridge smart contract can cancel the asset transfer on the destination network, destroying any minted assets. In scenarios where the origin bridge smart contract is allowed to mint assets in the origin network, we can allow for destroyed assets to be re-created in the origin network if the destination gateway never finalises the transfer.

D. A Successful Asset Transfer Summarised

A successful asset transfer must complete all four phases specified in SATP (where each step in a phase is either accepted or rejected by the counterparty). The asset transfer is started by the client application, which instructs the origin gateway to initiate the transfer. The gateways then execute phase 0 to establish the necessary trust, followed by phase 1:

1) The origin gateway sends a transfer initiate message to the destination gateway.

Next, the gateways enter phase 2:

2) The origin gateway sends a transfer commence message to the destination gateway.
3) The origin gateway invokes the initiateTransfer function in the origin bridge smart contract to lock the assets on the origin network.
4) The origin bridge smart contract emits a Transfer Initiated event on the origin network.
5) The origin gateway sends a lock evidence message, using the event emitted in step 4 as evidence, to the destination gateway.
6) The destination gateway invokes the acceptTransfer function in the destination bridge smart contract to indicate to the destination network that the requested asset transfer has been accepted.
7) The destination bridge smart contract emits a Transfer Accepted event on the destination network.

Finally, in phase 3, the asset(s) are created on the destination network and then destroyed on the origin network:

8) The origin gateway sends a commit preparation message to the destination gateway.
9) The destination gateway invokes the prepareTransfer function in the destination bridge smart contract to mint the assets on the destination network.
10) The destination bridge smart contract emits a Transfer Prepared event on the destination network.
11) The origin gateway invokes the commitTransfer function in the origin bridge smart contract to burn the assets on the origin network.
12) The origin bridge smart contract emits a Transfer Committed event on the origin network.
13) The origin gateway sends a commit final message to the destination gateway.
14) The destination gateway invokes the finaliseTransfer function in the destination bridge smart contract to unlock the minted assets on the destination network.
15) The destination bridge smart contract emits a Transfer Finalised event on the destination network.
16) The origin gateway invokes the completeTransfer function in the origin bridge smart contract to mark the asset transfer as completed.
17) The origin bridge contract emits a Transfer Complete event on the origin network.
18) The origin gateway sends a transfer complete message to the destination gateway.

After successfully completing all four phases, the asset is confirmed to have been transferred to the new network, and no further messages are exchanged between the gateways.

V. IMPLEMENTATION LEARNING POINTS

The learning points from our initial SATP bridge implementation cover three main topics discussed below.

A. SATP Network Applicability

SATP specifies the messaging format for all off-chain bridge components, i.e. the gateways that make up the bridge and any application interacting with it. It is designed in a DLT-neutral manner, making it adaptable for bridges from and to both permissioned and permissionless DLT networks.

Therefore, we consider the impact of SATP-based bridges in these different secure asset transfer connection scenarios:

- **Permissioned DLT network to another permissioned DLT network**: We assume that SATP bridges will have the largest impact in this scenario. This is due to the restricted number of entities that can run nodes in permissioned networks. To explain further, consider two permissioned DLT networks, where the first has nodes run by the set of entities $A$, and the second has nodes run by the set of entities $B$. In this scenario, the only possible operators of a full-view bridge are $\Gamma = A \cap B$. Now, for a lot of permissioned DLT network integrations, the following will hold: $\Gamma = \emptyset$. But even if $\Gamma \neq \emptyset$, will any of the entities in $\Gamma$ be willing to run a full-view bridge, as well as being technically and legally set up to do so? Instead, SATP bridges provide more flexibility by allowing one of the entities in $A$ to run the origin gateway and one of the entities in $B$ to run the destination gateway. Trust is required between the gateways, which can usually be established via an off-chain legal agreement due to a large portion of permissioned DLT networks being run by identifiable consortia (for different domains such as finance, supply chain and health care).

- **Permissionless DLT network to another permissionless DLT network**: In this scenario, full-view bridges can easily be built due to everyone having access to both networks if required. Yet there may still be space for SATP bridges. This is because the SATP bridge operators require identification and verification throughout the SATP phases. Depending on who provides the operators with their identification certificates, this could be linked with regulation. Such regulation could allow the recovery of funds in the case of an attack. Should these regulated...
bridges operating over permissionless networks come about, it will be up to the market to decide how popular they are when compared to crypto-economic bridges.

- **Permissioned DLT network to/from a permissionless DLT network**: This is a mix of the scenarios above. Now we have a set of entities $A$ running nodes on the permissioned DLT network, all of which could operate a full-view bridge between the two networks. But an SATP bridge may still be preferred for a few reasons such as: (a) none of the entities in $A$ want to be directly exposed to the permissionless DLT network because they may not want to handle cryptocurrency to pay for the related transaction fees, or they may not have the technical expertise to code smart contracts for that permissionless DLT network, or they may not have the technical expertise to run and manage the additional DLT nodes; (b) it will be technically more scalable to connect to gateways of the permissionless DLT network that are already running (and these are most likely run by an entity $x \notin A$).

### B. SATP Suitability

Firstly, as the SATP specification is still being actively developed, it is likely that the protocol will be further enhanced and introduce more specific recommendations in the future.

However, we discovered that even with this early version of the protocol, secure production-ready bridges can be implemented. This discovery occurred through our focus on three main criteria when creating our SATP implementation:

1) SATP bridges must be generic, meaning they must work with different DLTs and DLT networks.
2) SATP bridge operations must be correct, which means the asset transfer process must complete successfully by avoiding double-spending, or revert.
3) SATP bridges must be auditable, which means a third party must be able to verify when, where and by whom the asset transfer is initiated, as well as what phase is any asset transfer currently in.

We argue that these criteria successfully apply to our implementation, directly because of SATP’s properties. For instance: (1) an SATP bridge is generic to the DLT because SATP provides a standardised gateway and messaging interface; (2) an SATP bridge operates correctly, as it provides a structure for sharing network-specific evidence; and (3) an SATP bridge is auditable, as SATP requires all messages to be logged.

With our implementation, we have taken the generic interoperability aspect even further, by combining the SATP approach with using standardised APIs (Overledger) that provide standardised integration to popular blockchain networks. This allows a gateway application that is agnostic towards the underlying DLT to be built.

A final suitability learning point that we discovered regarding SATP is that the protocol currently states that assets must be burned or permanently locked on the origin DLT network to complete the transfer. This currently limits SATP to only transfer assets that are either: (a) assumed to be permanently transferred from the origin DLT network onto the destination DLT network; (b) or can be re-minted on the origin DLT network if required. However, there are some assets that have a fixed current supply and no re-minting functionality on DLT Networks. We believe (a) and (b) are unnecessary restrictions for SATP, which should be focused on developing a clear definition of the first multi-organisational partial-view bridge.

### C. SATP Compliant Smart Contracts

SATP, as a protocol, does not specify implementation details. The definitions used are broad and do not impose restrictions that would be troublesome when considering implementations targeting vastly different DLTs. SATP, for instance, does not specify details about scripting or smart contract implementations. These are details left for the developers implementing the gateway to network connections.

That said, it was much easier for us to build an SATP bridge with a standardised SATP-compliant smart contract interface, compared to bespoke smart contract implementations for each DLT. Given that our gateway is built on top of an API platform (Overledger) that provides a standardised interface and data model for multiple DLTs, with standardised SATP-compliant smart contracts, we could use nearly the exact same flow for our gateway to DLT network interaction, irrespective of the DLT being integrated (the only change being network identification information). Therefore we propose that an SATP-compliant smart contract interface should be supplementary material for SATP.

### VI. Future Work and Conclusions

We have presented our findings resulting from the implementation of the Secure Asset Transfer Protocol and we have outlined key design decisions regarding aspects that the protocol leaves open to different implementations. In summary, we have built SATP-compliant smart contracts for Solidity (applicable for EVM networks) and chaincode (applicable for Hyperledger Fabric networks). These contracts allow clients to lock their assets securely while the SATP Gateways establish the bridge between the two networks.

With regard to the future, we are aiming to continue tailoring the bridge implementation to the evolution of the SATP protocol. The charter of the SATP working group outlines three deliverable documents at the time of this writing: an architecture document, the protocol itself and a use cases document [23]. After the initial release of these documents, three directions have been primarily discussed within the SATP working group for extending the protocol: cross-network digital asset swaps, viewing data cross-network, and the possibility of extending the SATP protocol to allow for an arbitrary number of gateways to be involved in the asset transfer while maintaining the ACID properties [24].

### Acknowledgements

We thank the SATP working group for their excellent work on the SATP protocol. We also thank everyone involved in bridge-related projects at Quant, for their invaluable expertise and contributions towards realising secure cross-chain bridging and blockchain interoperability.
REFERENCES


