

Secure Asset Transfer Protocol
Internet-Draft
Intended status: Informational
Expires: 17 September 2026

M. Hargreaves
Quant Network
T. Hardjono
MIT
R. Belchior
Tecnico Lisboa, Blockdaemon
V. Ramakrishna
IBM
A. Chiriac
Quant Network
16 March 2026

Secure Asset Transfer Protocol (SATP) Core
draft-ietf-satp-core-13

Abstract

This memo describes the Secure Asset Transfer Protocol (SATP) for digital assets. SATP is a protocol operating between two gateways that conducts the transfer of a digital asset from one gateway to another, each representing their corresponding digital asset networks. The protocol establishes a secure channel between the endpoints and implements a 2-phase commit (2PC) to ensure the properties of transfer atomicity, consistency, isolation and durability.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at <https://ietf-satp.github.io/draft-ietf-satp-core/draft-ietf-satp-core.html>. Status information for this document may be found at <https://datatracker.ietf.org/doc/draft-ietf-satp-core/>.

Discussion of this document takes place on the Secure Asset Transfer Protocol Working Group mailing list (<mailto:sat@ietf.org>), which is archived at <https://mailarchive.ietf.org/arch/browse/sat/>. Subscribe at <https://www.ietf.org/mailman/listinfo/sat/>.

Source for this draft and an issue tracker can be found at <https://github.com/ietf-satp/draft-ietf-satp-core>.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 17 September 2026.

Copyright Notice

Copyright (c) 2026 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	4
2. Conventions used in this document	5
3. Terminology	5
4. The Secure Asset Transfer Protocol	6
4.1. Overview	6
4.2. SATP Model	7
4.3. Stages of the Protocol	7
4.4. Gateway Cryptographic Keys	7
5. SATP Message Format, identifiers and Descriptors	8
5.1. Overview	8
5.2. SATP Message Digital Signatures and Key Types	8
5.3. SATP Message Format and Payloads	9
5.3.1. Protocol version	9
5.3.2. Message Type	9
5.3.3. Digital Asset Identifier	10
5.3.4. Asset Profile Identifier	11
5.3.5. Gateway Network ID (NetworkID)	11
5.3.6. Transfer-Context ID:	11
5.3.7. Session ID:	12
5.3.8. Client Credential Types Supported by Gateways	12

5.3.9.	Gateway Supported TLS Schemes	12
5.3.10.	Client Offers Other Supported TLS Schemes	13
5.3.11.	Gateway Identifier	13
5.3.12.	Payload Hash	13
5.3.13.	Hash Algorithm Supported	14
5.3.14.	Signature Algorithms Supported	14
5.3.15.	Asset Lock Mechanism within a Network	14
5.3.16.	Lock assertion Claim Format	15
5.3.17.	Lock assertion Claim	15
5.4.	Negotiation of Security Protocols and Parameters	15
5.4.1.	TLS Secure Channel Establishment	15
5.4.2.	Client asserts or proves identity	15
5.4.3.	Messages can now be exchanged	15
6.	Overview of Message Flows	15
7.	Identity and Asset Verification Stage (Stage 0)	17
8.	Transfer Initiation Stage (Stage 1)	18
8.1.	Transfer Initialization Claim	18
8.2.	Conveyance of Gateway and Network Capabilities	20
8.3.	Transfer Proposal Message	21
8.4.	Transfer Proposal Receipt Message	23
8.5.	Reject Message	24
8.6.	Transfer Commence Message	25
8.7.	Commence Response Message (ACK-Commence)	25
9.	Lock Assertion Stage (Stage 2)	26
9.1.	Lock Assertion Message	27
9.2.	Lock Assertion Receipt Message	28
10.	Commitment Preparation and Finalization (Stage 3)	28
10.1.	Commit Preparation Message (Commit-Prepare)	29
10.2.	Commit Ready Message (Commit-Ready)	30
10.3.	Commit Final Assertion Message (Commit-Final)	31
10.4.	Commit-Final Acknowledgement Receipt Message (ACK-Final-Receipt)	32
10.5.	Transfer Complete Message	33
10.6.	Error Message	34
10.7.	Session abort message	34
10.8.	SATP Session Resumption	34
11.	Error Messages	35
11.1.	Session Termination Notification	35
11.2.	Connection Errors	35
11.3.	SATP Protocol Errors	36
11.4.	Effectiveness of Session Aborts	36
12.	Security Consideration	37
13.	IANA Consideration	38
13.1.	SATP Error Codes Registry	38
13.2.	URN Registration	45
13.3.	SATP Message Types Registry	45
13.4.	Initial Registry Contents	46
13.4.1.	Parameter name: transfer-proposal-msg	46

13.4.2.	Parameter name: proposal-receipt-msg	46
13.4.3.	Parameter name: reject-msg	46
13.4.4.	Parameter name: transfer-commence-msg	46
13.4.5.	Parameter name: ack-commence-msg	46
13.4.6.	Parameter name: lock-assert-msg	46
13.4.7.	Parameter name: assertion-receipt-msg	47
13.4.8.	Parameter name: commit-prepare-msg	47
13.4.9.	Parameter name: commit-ready-msg	47
13.4.10.	Parameter name: commit-final-msg	47
13.4.11.	Parameter name: ack-commit-final-msg	47
13.4.12.	Parameter name: commit-transfer-complete-msg	47
13.4.13.	Parameter name: error-msg	48
13.4.14.	Parameter name: session-abort-msg	48
14.	Error Types and Codes	48
14.1.	Protocol Error Codes	48
15.	Acknowledgements	49
16.	References	49
16.1.	Normative References	49
16.2.	Informative References	50
	Authors' Addresses	51

1. Introduction

This memo proposes a secure asset transfer protocol (SATP) that is intended to be deployed between two gateway endpoints to transfer a digital asset from an origin asset network to a destination asset network. Readers are directed first to [ARCH] for a description of the architecture underlying the current protocol.

Both the origin and destination asset networks are assumed to be opaque in the sense that the interior construct of a given network is not read/write accessible to unauthorized entities.

The protocol utilizes the asset burn-and-mint paradigm whereby the asset to be transferred is permanently disabled or destroyed (burned) at the origin asset network and is re-generated (minted) at the destination asset network. This is achieved through the coordinated actions of the peer gateways handling the unidirectional transfer at the respective networks.

A gateway is assumed to be trusted to perform the tasks involved in the asset transfer.

The overall aim of the protocol is to ensure that the state of assets in the origin and destination networks remain consistent, and that asset movements into (out of) networks via gateways can be accounted for.

There are several desirable technical properties of the protocol. The protocol must ensure that the properties of atomicity, consistency, isolation, and durability (ACID) are satisfied.

The requirement of consistency implies that the asset transfer protocol always leaves both asset networks in a consistent state (that the asset is located in one system/network only at any time).

Atomicity means that the protocol must guarantee that either the transfer commits (completes) or entirely fails, where failure is taken to mean there is no change to the state of the asset in the origin (sender) asset network.

The property of isolation means that while a transfer is occurring to a digital asset from an origin network, no other state changes can occur to the asset.

The property of durability means that once the transfer has been committed by both gateways, that this commitment must hold regardless of subsequent unavailability (e.g. crash) of the gateways implementing the SATP protocol.

All messages exchanged between gateways are assumed to run over TLS1.3. HTTPS/S must be used instead of plain HTTP.

The endpoints at the respective gateways should provide access to credentials (or other identification mechanisms) to prove the legal owner (or operator) of the gateway. An example of credentials include X509 certificates.

2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [REQ-LEVEL].

In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying significance described in RFC 2119.

3. Terminology

The following are some terminology used in the current document, some borrowed from [NIST]:

- * Digital asset: digital representation of a value or of a right that is able to be transferred and stored electronically using distributed ledger technology or similar technology [MICA].

4.2. SATP Model

The model for SATP is shown in Figure 1 [ARCH]. The Client (application) interacts with its local gateway (G1) over an interface (API1) in order to provide instructions to the gateway with regards to actions to assets and related resources located in the local system or network (NW1).

Gateways interact with each other over a gateway interface (API2). A given gateway may be required to access resources that are not located in network NW1 or network NW2. Access to these types of resources are performed over an off-network interface (API3).

4.3. Stages of the Protocol

The SATP protocol defines three (3) stages for a unidirectional asset transfer:

- * Transfer Initiation stage (Stage-1): These flows deal with commencing a transfer from one gateway to another. In this stage the sender gateway delivers a proposal containing the parameters agreed upon in Stage-0.
- * Lock-Assertion stage (Stage-2): These flows deal with the conveyance of signed assertions from the sender gateway to the receiver gateway regarding the locked status of an asset at the origin network.
- * Commitment Preparation and Finalization stage (Stage-3): These flows deal with the asset transfer and commitment establishment between two gateways.

In order to clarify discussion, the interactions between the peer gateways prior to the transfer initiation stage is referred to as the setup stage (Stage-0), which is outside the scope of the current specification.

The Stage-1, Stage-2 and Stage-3 flows will be discussed below.

4.4. Gateway Cryptographic Keys

SATP recognizes the following cryptographic keys which are intended for distinct purposes within the different stages of the protocol.

- * Gateway signature public key-pair: This is the key-pair utilized by a gateway to digitally sign assertions and receipts.
- * Gateway secure channel establishment public key-pair: This is the key-pair utilized by peer gateways to establish a secure channel (e.g. TLS1.3) for a transfer session.
- * Gateway identity public key pair: This is the key-pair that uniquely identifies a gateway.
- * Gateway-owner identity public key pair: This is the key-pair that identifies the owner (e.g. legal entity) who is the legal owner of a gateway.

When peer gateways deliver public-keys, these are expressed in JSON Web Key (JWK) format [RFC7517].

This document assumes that the relevant X.509 certificates are associated with these keys. However, the mechanisms to obtain X.509 certificates is outside the scope of this specification.

5. SATP Message Format, identifiers and Descriptors

5.1. Overview

This section describes the SATP message-types, the format of the messages exchanged between two gateways, the format for resource descriptors and other related parameters.

The mandatory fields are determined by the message type exchanged between the two gateways (see Section 7).

5.2. SATP Message Digital Signatures and Key Types

All SATP messages exchanged between gateways must be signed [ECDSA], using the JSON Web Signatures mechanism [RFC7515].

Signature algorithms used by gateways for SATP messages SHOULD be selected from those defined in the JSON Web Algorithms (JWA) specification [RFC7518], with key types defined in JSON Web Key (JWK) specification [RFC7517].

The choice of signature algorithm and key-type must be agreed upon between the gateways prior to the commencement of the SATP protocol session. The agreed values are then included within the Transfer Initialization Claim body in Transfer Proposal Message.

All SATP implementations MUST implement at minimal the ECDSA signature algorithm with the P-256 curve and the SHA-256 hash function.

Additional signature algorithms and keying parameters may be negotiated by peer gateways. However, the negotiation protocol is outside the scope of this specification.

5.3. SATP Message Format and Payloads

SATP messages are exchanged between peer gateways, where depending on the message type one gateway may act as a client of the other (and vice versa).

All SATP messages exchanged between gateways are in JSON format [RFC8259].

5.3.1. Protocol version

This refers to SATP protocol Version, encoded as "major.minor" (separated by a period symbol).

The current version is "1.0" defined in this specification. Implementations not understanding a future option value should return an appropriate error response and cease the negotiation.

5.3.2. Message Type

This refers to the type of request or response to be conveyed in the message.

The possible values are defined in the IANA SATP Message Types Registry Section 13.3, Paragraph 1:

- * transfer-proposal-msg: This is the transfer proposal message from the sender gateway carrying the set of proposed parameters for the transfer.
- * proposal-receipt-msg: This is the signed receipt message indicating acceptance of the proposal by the receiver gateway.
- * reject-msg: This is a reject message from a gateway to the peer gateway in the session, indication the reason and the resulting action.
- * transfer-commence-msg: Request to begin the commencement of the asset transfer.

- * `ack-commence-msg`: Response to accept the commencement of the asset transfer.
- * `lock-assert-msg`: Sender gateway has performed the lock of the asset in the origin network.
- * `assertion-receipt-msg`: Receiver gateway acknowledges receiving the signed `lock-assert-msg`.
- * `commit-prepare-msg`: Sender gateway requests the start of the commitment stage.
- * `ack-prepare-msg`: Receiver gateway acknowledges receiving the previous `commit-prepare-msg` and agrees to start the commitment stage.
- * `commit-final-msg`: Sender gateway has performed the extinguishment (burn) of the asset in the origin network.
- * `ack-commit-final-msg`: Receiver gateway acknowledges receiving the signed `commit-final-msg` and has performed the asset creation and assignment in the destination network.
- * `commit-transfer-complete-msg`: Sender gateway indicates closure of the current transfer session.
- * `error-msg`: This message is used to indicate that an error has occurred at the SATP layer. It can be transmitted by either gateways.
- * `session-abort-msg`: This message is used by a gateway to abort the current session.

5.3.3. Digital Asset Identifier

This is the identifier that uniquely identifies the digital asset in the origin network which is to be transferred to the destination network.

The digital asset identifier is a value that is derived by the applications utilized by the originator and the beneficiary prior to starting the asset transfer.

The mechanism used to derive the digital asset identifier is outside the scope of the current document.

The digital asset identifier is a JSON object, which may be encoded as a string in base64 [RFC4648].

5.3.4. Asset Profile Identifier

This is the unique identifier of the asset schema or asset profile which defines the class or type of asset in question. The asset profile is relevant from a regulatory perspective.

In some cases the profile identifier may be needed by the receiver gateway at the destination network in order to evaluate whether the asset is permitted to enter the destination network.

The default format of the asset profile identifier is JSON, with base64 encoding.

The formal specification of asset profiles and their identification is outside the scope of this document.

5.3.5. Gateway Network ID (NetworkID)

The network identifier (NetworkID) is the unique alphanumeric string representing the asset network behind a gateway. A gateway may simultaneously stand in front of multiple asset networks. As such, for a specific asset transfer instance both the sender gateway and recipient gateway must indicate which asset networks are the origin network and destination network respectively.

The network identifier values of the origin network (senderGatewayNetworkId) and destination network (recipientGatewayNetworkId) must be communicated and agreed upon prior to the commencement of the asset transfer. This selection is confirmed by peer gateways in the Transfer Initialization Claim that is transmitted within Transfer Proposal Message.

The mechanism to allocate globally unique network identifier is outside the scope of the current specification.

5.3.6. Transfer-Context ID:

This is the unique immutable identifier representing the application layer context of a single unidirectional transfer. The method to generate the transfer-context ID is outside the scope of the current document.

The transfer-context may be a complex data structure that contains all information related to a SATP execution instance. Examples of information contained in a transfer-context may include identifiers of sessions, gateways, networks or assets related to the specific SATP execution instance. The sender gateway provides this value to the receiver gateway.

The default format of the transfer context identifier is JSON, with base64 encoding.

The Transfer Context ID (`transferContextId`) value is established by the sender application (possibly with the assistance of the sender gateway) in the origin network. The value is then communicated to the receiving application in the destination network prior to the commencement of the SATP protocol. Both the sender gateway and receiver gateway must understand how to process the `transferContextId` value. The value is used in the Transfer Proposal Message (with message type `satp:msgtype:transfer-proposal-msg`) between the two gateways.

The mechanism to derive the Transfer Context ID value and to communicate it between the applications is outside the scope of the current specification.

5.3.7. Session ID:

This is the unique identifier representing a session between two gateways handling a single unidirectional transfer. This may be derived from the Transfer-Context ID at the application level. There may be several session IDs related to a SATP execution instance. Only one Session ID may be active for a given SATP execution instance. Session IDs may be stored in the transfer-context for audit trail purposes.

The sender gateway provides this value to the receiver gateway.

5.3.8. Client Credential Types Supported by Gateways

SATP Gateways must support JSON Web Tokens (JWT) [RFC7519] with OAuth2.0 [RFC6749] as the minimal credential type for authenticating incoming API calls from Client Applications (see Figure 1).

A gateway may support additional credential mechanisms, which may be advertised by the gateway through different mechanisms (e.g. config file at a well-known endpoint). However, these mechanisms are out of scope for the current specification.

5.3.9. Gateway Supported TLS Schemes

Gateways must support TLS1.3 [RFC8446].

The TLS scheme is used by peer gateways to establish the TLS session prior to the commencement of an asset transfer. Gateways must at minimal support the AES-128 in GCM mode with SHA-256 (`TLS_AES_128_GCM_SHA256`).

If the client (sender gateway) transmits a list of supported credential schemes, the server (recipient gateway) selects one acceptable credential scheme from the offered schemes.

If no acceptable credential scheme was offered, a "unsupported gatewayTlsScheme" (err_1.1.34) reject message is returned by the server Section 8.5, Paragraph 1.

5.3.10. Client Offers Other Supported TLS Schemes

If a client (sender gateway) wishes to use TLS schemes other than the basic scheme (AES-128 in GCM mode with SHA-256), then the client may choose to send a JSON object listing the client's supported TLS schemes.

These must be selected from those defined in TLS1.3 [RFC8446].

5.3.11. Gateway Identifier

This is the unique identifier of the gateway service. The gateway identifier MUST be uniquely bound to its SATP endpoint (e.g. via X.509 certificates).

This gateway identifier is distinct from the gateway operator business identifier (e.g., legal entity identifier (LEI) number). A gateway operator may operate multiple gateways. Each of the gateways within an asset network MUST be identified by a unique gateway identifier.

The mechanisms to establish the gateway identifier or the operator identifier is outside the scope of this specification.

5.3.12. Payload Hash

This is the hash of the current message payload. This is utilized in some crucial messages within the SATP message flows to detect errors or attacks, where the payload-hash of the previously received message (from a sender gateway) is included in the response message to that sender gateway.

For example, the hash of the Transfer Proposal message from the sender gateway is included in the Transfer Commence message. Similarly, the hash of the lock-assertion message (from the gateway at the origin network) is included in the Lock Assertion Receipt Message (sent in response by the gateway at the destination network).

5.3.13. Hash Algorithm Supported

All cryptographic hash operations in the current specification follow that used for JSON structures, which includes the canonicalization (normalization) and serialization of the data, prior to the application of the hash algorithm.

The default hash algorithm that all SATP implementations MUST support is the SHA-256 algorithm [RFC7515].

5.3.14. Signature Algorithms Supported

This is a JSON list of digital signature algorithms supported by a gateway. Each entry in the list should be either an Algorithm Name value registered in the IANA "JSON Web Signature and Encryption Algorithms" registry established by [RFC7518] or be a value that contains a Collision-Resistant Name.

See Section (Section 5.2, Paragraph 1).

5.3.15. Asset Lock Mechanism within a Network

SATP gateways may be providing service to multiple types of asset networks, each of which may utilize different local mechanisms to immobile (lock) a given asset as way to provide exclusion in the case of multiple attempts to change the state of the asset.

The origin network and the destination network may in fact utilize distinct asset locking mechanisms, and the type of mechanisms to immobile (lock) a given asset may have different convergence (finalization) speeds. Peer gateways must exchange information about the asset locking information in their respective network to enable both gateways to compute an approximate time of convergence (assetLockExpirationTime) and set timers for the transfer of asset. A timer that expires too soon may result in the SATP protocol terminating too early before reaching the final commitment stage.

Currently, the most common type of mechanisms (NetworkLockType) to temporarily lock an asset in a network are (i) TIME_LOCK, (ii) HASH_LOCK, (iii) HASH_TIME_LOCK.

Some examples of the use of lock mechanisms are follows. Bitcoin [BTC] utilizes a time-lock for delays in some of its operations (e.g., nLockTime). Ethereum [ETH] uses a Hash-Lock mechanism for atomic swaps. Ethereum and Ripple [XRP] uses Hashed Time-lock in their contracts (HTLC).

The exact definition of these asset locking mechanisms are network-dependent, and such are out of the scope of the current work.

5.3.16. Lock assertion Claim Format

This is the format of the claim regarding the state of the asset in the origin network. The default format is JSON, with parts being base64 encoded as needed.

If the sender gateway offers multiple choices of other formats to the receiver gateway, the selection must occur prior to the establishment of the session.

5.3.17. Lock assertion Claim

The actual encoded JSON string representation of the claim using the format as specified by the corresponding Lock Assertion Claim Format value.

5.4. Negotiation of Security Protocols and Parameters

The peer gateways in SATP must establish a TLS session between them prior to starting the transfer initiation stage (Stage-0). The TLS session continues until the transfer is completed at the end of the commitment establishment stage (Stage-3).

In the following steps, the sender gateway is referred to as the client while the receiver gateway as the server.

5.4.1. TLS Secure Channel Establishment

TLS 1.3 MUST be implemented to protect gateway communications.

5.4.2. Client asserts or proves identity

The details of the assertion/verification step are specific to the chosen credential scheme and are outside the scope of this document.

5.4.3. Messages can now be exchanged

Handshaking is complete at this point, and the client and server can begin exchanging SATP messages.

6. Overview of Message Flows

The SATP message flows are logically divided into three (3) stages [ARCH], with the preparatory stage denoted as Stage-0. How the tasks are achieved in Stage-0 is out of the scope of this specification.

The Stage-1 flows pertain to the initialization of the transfer between the two gateways.

After both gateways agree to commence the transfer at the start of Stage-2, the sender gateway G1 must deliver a signed assertion that it has correctly performed the relevant action on the asset within the origin network (NW1). Examples of actions by G1 include performing a temporary lock on the asset, or performing a permanent disablement (burn) of the asset in NW1.

If that signed assertion is accepted by gateway G2, it must in return transmit a signed receipt to gateway G1 that it has correctly performed the relevant corresponding action on destination network (NW2). Examples of actions by G2 include creating (minting) a temporary asset under its control in NW2.

The Stage-3 flows commit gateways G1 and G2 to the burn and mint in Stage-2. The sender gateway G1 must make the lock on the asset in the origin network NW1 to be permanent (burn). The receiver gateway G2 must assign (mint) the asset in the destination network NW2 to the correct beneficiary.

The reader is directed to [ARCH] for further discussion of this model.

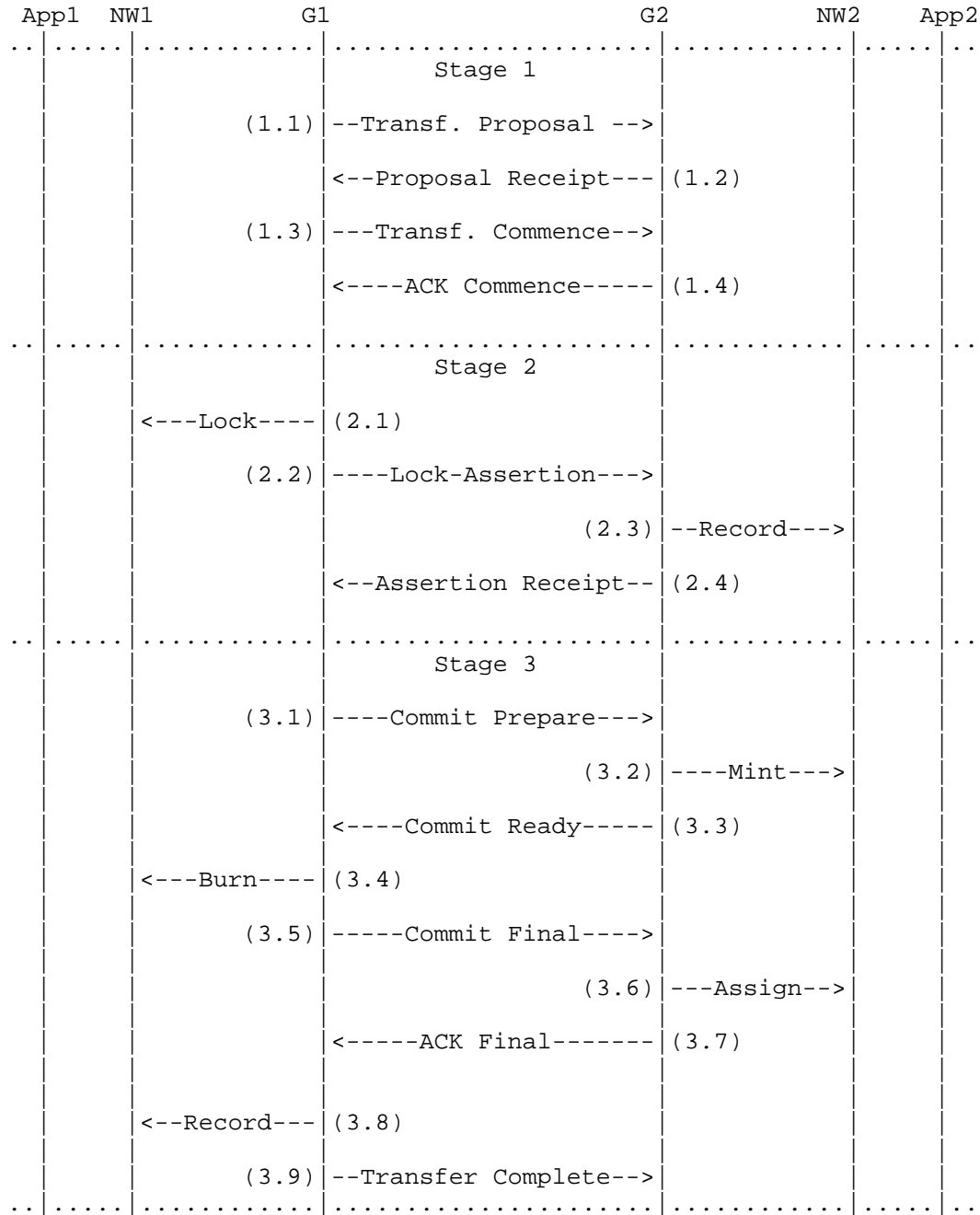


Figure 2

7. Identity and Asset Verification Stage (Stage 0)

Prior to commencing the asset transfer from the sender gateway (client) to the recipient gateway (server), both gateways must perform a number of verification steps. The types of information required by both the sender and recipient are use-case dependent and asset-type dependent.

The verifications include, but not limited to, the following:

- * Verification of the gateway signature public key: The sender gateway and receiver gateway must validate their respective signature public keys that will later be used to sign assertions and claims. This may include validating the X509 certificates of

these keys.

- * Gateway owner verification: This is the verification of the identity (e.g. LEI) of the owners of the gateways.
- * Gateway device and state validation: This is the device attestation evidence [RFC9334] that a gateway must collect and convey to each other, where a verifier is assumed to be available to decode, parse and appraise the evidence.

- * Originator and beneficiary identity verification: This is the identity and public-key of the entity (originator) in the origin network seeking to transfer the asset to another entity (beneficiary) in the destination network.

These are considered out of scope in the current specification, and are assumed to have been successfully completed prior to the commencement of the transfer initiation flow. The reader is directed to [ARCH] for further discussion regarding Stage-0.

8. Transfer Initiation Stage (Stage 1)

This section describes the transfer initiation stage, where the sender gateway and the receiver gateway prepare for the start of the asset transfer.

The sender gateway proposes the set of transfer parameters and asset-related artifacts for the transfer to the receiver gateway. These are contained in the Transfer Initiation Claim.

If the receiver gateway accepts the proposal, it returns a signed receipt message for the proposal indicating it agrees to proceed to the next stage. If the receiver gateway rejects any parameters or artifacts in the proposal, it can provide a counteroffer to the sender gateway by responding with a proposal reject message carrying alternative parameters.

Gateways MUST support the use of the HTTP GET and POST methods for the endpoint [RFC9110].

Clients (sender gateway) MAY use the HTTP GET or POST methods to send messages in this stage to the server (recipient gateway). If using the HTTP GET method, the request parameters may be serialized using URI Query String Serialization.

8.1. Transfer Initialization Claim

This is set of artifacts pertaining to the asset that must be agreed upon between the client (sender gateway) and the server (recipient gateway).

The format of the identity fields in this message, unless otherwise stated, is a JSON text.

The Transfer Initialization Claim consists of the following:

- * digitalAssetId REQUIRED: This is the globally unique identifier for the digital asset located in the origin network.

- * `assetProfileId` REQUIRED: This is the globally unique identifier for the asset-profile definition (document) on which the digital asset was issued.
- * `networkLockType` REQUIRED: The default locking mechanism used for an asset. These can be (i) `TIME_LOCK`, (ii) `HASH_LOCK`, (iii) `HASH_TIME_LOCK`.
- * `assetLockExpirationTime` OPTIONAL: The duration of time (in seconds) for an asset lock to expire in the network, if it is a `HASH_TIME_LOCK` or a `TIME_LOCK`.
- * `verifiedOriginatorEntityId` REQUIRED: This is the identity data of the originator entity (person or organization) in the origin network. This information must be verified by the sender gateway.
- * `verifiedBeneficiaryEntityId` REQUIRED: This is the identity data of the beneficiary entity (person or organization) in the destination network. This information must be verified by the receiver gateway.
- * `originatorPublicKey` REQUIRED: This is the public key of the asset owner (originator) in the origin network or system.
- * `beneficiaryPublicKey` REQUIRED: This is the public key of the beneficiary in the destination network.
- * `senderGatewaySignaturePublicKey` REQUIRED: This is the public key of the key-pair used by the sender gateway to sign assertions and receipts.
- * `receiverGatewaySignaturePublicKey` REQUIRED: This is the public key of the key-pair used by the receiver gateway to sign assertions and receipts.
- * `senderGatewayId` REQUIRED: This is the identifier of the sender gateway.
- * `recipientGatewayId` REQUIRED: This is the identifier of the receiver gateway.
- * `senderGatewayNetworkId` REQUIRED: This is the identifier of the origin network or system behind the client.
- * `recipientGatewayNetworkId` REQUIRED: This is the identifier of the destination network or system behind the server.

- * senderGatewayDeviceIdentityPublicKey OPTIONAL: The device public key of the sender gateway (client).
- * receiverGatewayDeviceIdentityPublicKey OPTIONAL: The device public key of the receiver gateway
- * senderGatewayOwnerId OPTIONAL: This is the identity information of the owner or operator of the sender gateway.
- * receiverGatewayOwnerId OPTIONAL: This is the identity information of the owner or operator of the recipient gateway.

Here is an example representation in JSON format (with the public keys in JWK being replaced with hexadecimal for brevity):

```
{ "digitalAssetId": "2c949e3c-5edb-4a2c-9ef4-20de64b9960d", \
  "assetProfileId": "38561", \ "verifiedOriginatorEntityId": "CN=Alice,
  OU=Example Org Unit, O=Example, L=New York, C=US", \
  "verifiedBeneficiaryEntityId": "CN=Bob, OU=Case Org Unit, O=Case,
  L=San Francisco, C=US", \ "originatorPublicKey":
  "0304b9f34d3898b27f85b3d88fa069a879abel14db5060dde466dd1e4a31ff75e44", \
  "beneficiaryPublicKey":
  "02a7bc058elc6f3a79601d046069c9b6d0cb8ea5afc99e6074a5997284756fc9ae", \
  "senderGatewaySignaturePublicKey":
  "02a7bc058elc6f3a79601d046069c9b6d0cb8ea5afc99e6074a5997284756fc9ae", \
  "receiverGatewaySignaturePublicKey":
  "0243b12ada6515ada3bf99a7da32e84f00383b5765fd7701528e660449ba5ef260", \
  "senderGatewayId": "GW1", \ "recipientGatewayId": "GW2", \
  "senderGatewayNetworkId": "1", \ "recipientGatewayNetworkId":
  "43114", \ "senderGatewayDeviceIdentityPublicKey":
  "0245785e34b4a7b457dd4683a297ea3d78bab35f8b2583df55d9df8c69604d0e73", \
  "receiverGatewayDeviceIdentityPublicKey":
  "03763f0bc48ff154cff45ea533a9d8a94349d65a45573e4de6ad6495b6e834312b", \
  "senderGatewayOwnerId": "CN=GatewayOps, OU=GatewayOps Systems,
  O=GatewayOps LTD, L=Austin, C=US", \ "receiverGatewayOwnerId":
  "CN=BridgeSolutions, OU=BridgeSolutions Engineering,
  O=BridgeSolutions LTD, L=Austin, C=US" \ }
```

8.2. Conveyance of Gateway and Network Capabilities

This is the set of parameters pertaining to the origin network and the destination network, and the technical capabilities supported by the peer gateways.

Some network-specific parameters regarding the origin network may be relevant for a receiver gateway to evaluate its ability to process the proposed transfer. For example, if the duration of the lock-time (networkLockExpirationTime) in the origin network is too short, a receiver gateway at the destination network may decline to proceed.

The gateway capabilities list is as follows:

- * gatewayDefaultSignatureAlgorithm REQUIRED: The default digital signature algorithm (algorithm-id) from the IANA "JSON Web Signature and Encryption Algorithms" registry used by a gateway to sign claims.
- * gatewaySupportedSignatureAlgorithms OPTIONAL: The list of other digital signature algorithms (algorithm-id) from the IANA "JSON Web Signature and Encryption Algorithms" registry supported by a gateway to sign claims
- * networkLockType REQUIRED: The default locking mechanism used by a network. The values allowed are "TIME_LOCK", "HASH_LOCK", "HASH_TIME_LOCK". Future updates to this specification may define new values and implementations not supporting a value or not understanding a value for this field must return an appropriate error and cease the negotiation.
- * networkLockExpirationTime REQUIRED: The duration of time (in integer seconds) for a lock to expire in the network.
- * gatewayTlsScheme REQUIRED: Specify the TLS1.3 scheme.

Here is an example representation in JSON format:

```
json { "gatewayDefaultSignatureAlgorithm": "ES256",  
  "gatewaySupportedSignatureAlgorithms": ["ES256", "RSA"],  
  "networkLockType": "HASH_TIME_LOCK", "networkLockExpirationTime":  
  120, "gatewayTlsScheme": "TLS_AES_128_GCM_SHA256" }
```

8.3. Transfer Proposal Message

The purpose of this message is for the sender gateway as the client to initiate an asset transfer session with the receiver gateway as the server.

The client transmits a proposal message that carries the claim related to the asset to be transferred. This message must be signed by the client.

This message is sent from the client to the Transfer Initialization Endpoint at the server.

The parameters of this message consist of the following:

- * version REQUIRED: SATP protocol Version (see Section 5.3.1, Paragraph 1) as a string "major.minor".
- * messageType REQUIRED: urn:ietf:satp:msgtype:transfer-proposal-msg.
- * sessionId REQUIRED: A unique identifier chosen by the client to identify the current session.
- * transferContextId REQUIRED: A unique identifier used to identify the current transfer session at the application layer.
- * transferInitClaimFormat REQUIRED: The default format is JSON, with parts being base64 encoded as needed. The default format is denoted as "TRANSFER_INIT_CLAIM_FORMAT_1".
- * transferInitClaim REQUIRED: The set of artifacts and parameters as the basis for the current transfer.
- * gatewayAndNetworkCapabilities REQUIRED: The set of origin gateway and network parameters reported by the client to the server.

Here is an example of the message request body (with the public keys in JWK being replaced with hexadecimal for brevity):

```
json { "version": "1.0", "messageType":
"urn:ietf:satp:msgtype:transfer-proposal-msg", "sessionId":
"d66a567c-11f2-4729-a0e9-17celfaf47c1", "transferContextId":
"89e04e71-bba2-4363-933c-262f42ec07a0", "transferInitClaimFormat":
"TRANSFER_INIT_CLAIM_FORMAT_1", "transferInitClaim": {
"digitalAssetId": "2c949e3c-5edb-4a2c-9ef4-20de64b9960d",
"assetProfileId": "38561", "networkLockType": "HASH_TIME_LOCK",
"assetLockExpirationTime": 120, "verifiedOriginatorEntityId":
"CN=Alice, OU=Example Org Unit, O=Example, L=New York, C=US",
"verifiedBeneficiaryEntityId": "CN=Bob, OU=Case Org Unit, O=Case,
L=San Francisco, C=US", "originatorPublicKey":
"0304b9f34d3898b27f85b3d88fa069a879abel4db5060dde466dd1e4a31ff75e44",
"beneficiaryPublicKey":
"02a7bc058elc6f3a79601d046069c9b6d0cb8ea5afc99e6074a5997284756fc9ae",
"senderGatewaySignaturePublicKey":
"02a7bc058elc6f3a79601d046069c9b6d0cb8ea5afc99e6074a5997284756fc9ae",
"receiverGatewaySignaturePublicKey":
"0243b12ada6515ada3bf99a7da32e84f00383b5765fd7701528e660449ba5ef260",
"senderGatewayId": "GW1", "recipientGatewayId": "GW2",
```



```
"senderGatewayNetworkId": "1", "recipientGatewayNetworkId": "43114",
"senderGatewayDeviceIdentityPublicKey":
"0245785e34b4a7b457dd4683a297ea3d78bab35f8b2583df55d9df8c69604d0e73",
"receiverGatewayDeviceIdentityPublicKey":
"03763f0bc48ff154cff45ea533a9d8a94349d65a45573e4de6ad6495b6e834312b",
"senderGatewayOwnerId": "CN=GatewayOps, OU=GatewayOps Systems,
O=GatewayOps LTD, L=Austin, C=US", "receiverGatewayOwnerId":
"CN=BridgeSolutions, OU=BridgeSolutions Engineering,
O=BridgeSolutions LTD, L=Austin, C=US" },
"gatewayAndNetworkCapabilities": {
"gatewayDefaultSignatureAlgorithm": "ES256",
"gatewaySupportedSignatureAlgorithms": ["ES256", "RSA"],
"networkLockType": "HASH_TIME_LOCK", "networkLockExpirationTime":
120, "gatewayTlsScheme": "TLS_AES_128_GCM_SHA256" } }
```

8.4. Transfer Proposal Receipt Message

The purpose of this message is for the server to indicate explicit acceptance of the parameters in the claim part of the transfer proposal message.

The message must be signed by the server.

The message is sent from the server to the Transfer Proposal Endpoint at the client.

The parameters of this message consist of the following:

- * version REQUIRED: SATP protocol Version see {satp-protocol-version}} as a string "major.minor".
- * messageType REQUIRED: urn:ietf:satp:msgtype:proposal-receipt-msg.
- * sessionId REQUIRED: A unique identifier chosen by the client to identify the current session.
- * transferContextId REQUIRED: A unique identifier used to identify the current transfer session at the application layer.
- * hashTransferInitClaim REQUIRED: Hash of the Transfer Initialization Claim received in the Transfer Proposal Message.
- * timestamp REQUIRED: timestamp referring to when the Initialization Request Message was received.

Here is an example of the message request body:

```
{\ "version": "1.0",\ "messageType": "urn:ietf:satp:msgtype:proposal-  
receipt-msg",\ "sessionId": "d66a567c-11f2-4729-a0e9-17celfaf47c1",\  
"transferContextId": "89e04e71-bba2-4363-933c-262f42ec07a0",\  
"hashTransferInitClaim":  
"154dfaf0406038641e7e59509febf41d9d5d80f367db96198690151f4758ca6e",\  
"timestamp": "2024-10-03T12:02+00Z",\ }\}
```

8.5. Reject Message

The purpose of this message is for the server to indicate explicit rejection of the previous message received from the client. This message can be sent at any time in the session. The server **MUST** include an error code (see Section 14, Paragraph 1) in this message. A reject message is taken to mean an immediate termination of the session.

The message must be signed by the server.

The parameters of this message consist of the following:

- * version REQUIRED: SATP protocol Version see {satp-protocol-version}} as a string "major.minor".
- * messageType REQUIRED: urn:ietf:satp:msgtype:reject-msg
- * sessionId REQUIRED: A unique identifier chosen by the client to identify the current session.
- * transferContextId REQUIRED: A unique identifier used to identify the current transfer session at the application layer.
- * hashPrevMessage REQUIRED: The cryptographic hash of the last message that caused the rejection to occur. The default hash algorithm is SHA256.
- * reasonCode REQUIRED: the error code (see Section 14, Paragraph 1) causing the rejection.
- * timestamp REQUIRED: timestamp of this message.

Here is an example of the message request body:

```
{\ "version": "1.0",\ "messageType": "urn:ietf:satp:msgtype:reject-  
msg",\ "sessionId": "d66a567c-11f2-4729-a0e9-17celfaf47c1",\  
"transferContextId": "89e04e71-bba2-4363-933c-262f42ec07a0",\  
"hashPrevMessage":  
"154dfaf0406038641e7e59509febf41d9d5d80f367db96198690151f4758ca6e",\  
"reasonCode": "err_2.1",\ "timestamp": "2024-10-03T12:02+00Z",\ }\}
```

8.6. Transfer Commence Message

The purpose of this message is for the client to signal to the server that the client is ready to start the transfer of the digital asset. This message must be signed by the client.

This message is sent by the client as a response to the Transfer Proposal Receipt Message previously received from the server.

This message is sent by the client to the Transfer Commence Endpoint at the server.

The parameters of this message consist of the following:

- * messageType REQUIRED: MUST be the value `urn:ietf:satp:msgtype:transfer-commence-msg`.
- * sessionId REQUIRED: A unique identifier chosen earlier by the client in the Initialization Request Message.
- * transferContextId REQUIRED: A unique identifier used to identify the current transfer session at the application layer.
- * hashTransferInitClaim REQUIRED: Hash of the Transfer Initialization Claim in the Transfer Proposal message.
- * hashPrevMessage REQUIRED. The cryptographic hash of the last message, in this case the Transfer Proposal Receipt message. The default hash algorithm is SHA256.

For example, the client makes the following HTTP request using TLS:

```
{\ "messageType": "urn:ietf:satp:msgtype:transfer-commence-msg",\
"sessionId": "d66a567c-11f2-4729-a0e9-17ce1faf47c1",\
"transferContextId": "89e04e71-bba2-4363-933c-262f42ec07a0",\
"hashTransferInitClaim":
"154dfaf0406038641e7e59509febf41d9d5d80f367db96198690151f4758ca6e",\
"hashPrevMessage":
"0b0aecc2680e0d8a86bece6b54c454fba67068799484f477cdf2f87e6541db66",\
}\
```

8.7. Commence Response Message (ACK-Commence)

The purpose of this message is for the server to indicate agreement to proceed with the asset transfer, based on the artifacts found in the previous Transfer Proposal Message.

This message is sent by the server to the Transfer Commence Endpoint at the client.

The message must be signed by the server.

The parameters of this message consist of the following:

- * messageType REQUIRED: urn:ietf:satp:msgtype:ack-commence-msg
- * sessionId REQUIRED: A unique identifier chosen earlier by the client in the Initialization Request Message.
- * transferContextId REQUIRED: A unique identifier used to identify the current transfer session at the application layer.
- * hashPrevMessage REQUIRED. The cryptographic hash of the last message, in this case the Transfer Commence Message. The default hash algorithm is SHA256.

An example of a success response could be as follows:

```
{\ "messageType": "urn:ietf:satp:msgtype:ack-commence-msg",\
"sessionId": "d66a567c-11f2-4729-a0e9-17celfaf47c1",\
"transferContextId": "89e04e71-bba2-4363-933c-262f42ec07a0",\
"hashPrevMessage":
"dd5a61a26fc8f5d72e5ca6052c2a1fca1613115e5582d9417d336375c196db89",\
}\
```

9. Lock Assertion Stage (Stage 2)

The messages in this stage pertain to the sender gateway providing the recipient gateway with a signed assertion that the asset in the origin network has been locked or disabled and under the control of the sender gateway.

In the following steps, the sender gateway takes the role of the client while the recipient gateway takes the role of the server.

The flow follows a request-response model. The client makes a request (POST) to the Lock-Assertion Endpoint at the server.

Gateways MUST support the use of the HTTP GET and POST methods for the endpoint.

Clients MAY use the HTTP GET or POST methods to send messages in this stage to the server. If using the HTTP GET method, the request parameters may be serialized using URI Query String Serialization.

(NOTE: Flows occur over TLS. Nonces are not shown).

9.1. Lock Assertion Message

The purpose of this message is for the client (sender gateway) to convey a signed claim to the server (receiver gateway) declaring that the asset in question has been locked or escrowed by the client in the origin network (e.g. to prevent double-spending).

The format of the claim is dependent on the network or system of the client and is outside the scope of this specification.

This message is sent from the client to the Lock Assertion Endpoint at the server.

The server must validate the claim (payload) in this message prior to the next step.

The message must be signed by the client.

The parameters of this message consist of the following:

- * messageType REQUIRED: urn:ietf:satp:msgtype:lock-assert-msg.
- * sessionId REQUIRED: A unique identifier chosen earlier by the client in the Initialization Request Message.
- * transferContextId REQUIRED: A unique identifier used to identify the current transfer session at the application layer.
- * lockAssertionClaimFormat REQUIRED. The default format is JSON, with parts being base64 encoded as needed. The default format is denoted as "LOCK_ASSERTION_CLAIM_FORMAT_1".
- * lockAssertionClaim REQUIRED: The lock assertion claim or statement by the client.
- * lockAssertionExpiration REQUIRED. The expiration date and time [DATETIME] of the lock or escrow upon the asset on the origin network.
- * hashPrevMessage REQUIRED. The cryptographic hash of the last message. The default hash algorithm is SHA256.

Example:

```
{\ "messageType": "urn:ietf:satp:msgtype:lock-assert-msg",\
"sessionId": "d66a567c-11f2-4729-a0e9-17celfaf47c1",\
"transferContextId": "89e04e71-bba2-4363-933c-262f42ec07a0",\
"lockAssertionClaimFormat": "LOCK_ASSERTION_CLAIM_FORMAT_1",\
"lockAssertionClaim": {},\ "lockAssertionExpiration":
"2024-12-23T23:59:59.999Z",\ "hashPrevMessage":
"b2c3e916703c4ee4494f45bcf52414a2c3edfe53643510ff158ff4a406678346",\
}\
```

9.2. Lock Assertion Receipt Message

The purpose of this message is for the server (receiver gateway) to indicate acceptance of the claim in the lock-assertion message delivered by the client (sender gateway) in the previous message.

This message is sent from the server to the Assertion Receipt Endpoint at the client.

The message must be signed by the server.

The parameters of this message consist of the following:

- * messageType REQUIRED: urn:ietf:satp:msgtype:assertion-receipt-msg.
- * sessionId REQUIRED: A unique identifier chosen earlier by the client in the Initialization Request Message.
- * transferContextId REQUIRED: A unique identifier used to identify the current transfer session at the application layer.
- * hashPrevMessage REQUIRED. The cryptographic hash of the last message. The default hash algorithm is SHA256.

Example:

```
{\ "messageType": "urn:ietf:satp:msgtype:assertion-receipt-msg",\
"sessionId": "d66a567c-11f2-4729-a0e9-17celfaf47c1",\
"transferContextId": "89e04e71-bba2-4363-933c-262f42ec07a0",\
"hashPrevMessage":
"16c983122d7506c78f906c15caldcc7142a0fa94552cdea9578fe87419c2c5d0",\
}\
```

10. Commitment Preparation and Finalization (Stage 3)

This section describes the transfer commitment agreement between the client (sender gateway) and the server (receiver gateway).

This stage must be completed within the time specified in the `lockAssertionExpiration` value in the lock-assertion message. This value is the time when the lock or escrow upon the asset will expire on the origin network.

The completion of this stage is denoted by the signed Commit-Final Acknowledgement Receipt Message sent from the receiver gateway (server) to the sender gateway (client). If the `lockAssertionExpiration` timer at the client expires before the Commit-Final Acknowledgement Receipt Message is received by the client, the client may terminate the session.

The flow follows a request-response model. The client makes a request (POST) to the Transfer Commitment endpoint at the server.

Gateways MUST support the use of the HTTP GET and POST methods for the endpoint.

Clients MAY use the HTTP GET or POST methods to send messages in this stage to the server. If using the HTTP GET method, the request parameters may be serialized using URI Query String Serialization.

The client and server may be required to sign certain messages in order to provide standalone proof (for non-repudiation) independent of the secure channel between the client and server. This proof may be required for audit verifications post-event.

(NOTE: Flows occur over TLS. Nonces are not shown).

10.1. Commit Preparation Message (Commit-Prepare)

The purpose of this message is for the client to indicate its readiness to begin the commitment of the transfer.

This message is sent from the client to the Commit Prepare Endpoint at the server.

The message must be signed by the client.

The parameters of this message consist of the following:

- * `messageType` REQUIRED: It MUST be the value `urn:ietf:satp:msgtype:commit-prepare-msg`
- * `sessionId` REQUIRED: A unique identifier chosen earlier by the client in the Initialization Request Message.

- * transferContextId REQUIRED: A unique identifier used to identify the current transfer session at the application layer.
- * hashPrevMessage REQUIRED. The cryptographic hash of the last message. The default hash algorithm is SHA256.

Example:

```
{\ "messageType": "urn:ietf:satp:msgtype:commit-prepare-msg",\
"sessionId": "d66a567c-11f2-4729-a0e9-17celfaf47c1",\
"transferContextId": "89e04e71-bba2-4363-933c-262f42ec07a0",\
"hashPrevMessage":
"399bdadc07fe0bd57c4dfdd6cc176ceeca50a5e744f774154eccbeee8908fbaa",\
}\
```

10.2. Commit Ready Message (Commit-Ready)

The purpose The purpose of this message is for the server to indicate to the client that: (i) the server has created (minted) an equivalent asset in the destination network; (ii) that the newly minted asset has been self-assigned to the server; and (iii) that the server is ready to proceed to the next step.

This message is sent from the server to the Commit Ready Endpoint at the client.

The message must be signed by the server.

The parameters of this message consist of the following:

- * messageType REQUIRED: It MUST be the value urn:ietf:satp:msgtype:commit-ready-msg.
- * sessionId REQUIRED: A unique identifier chosen earlier by client in the Initialization Request Message.
- * transferContextId REQUIRED: A unique identifier used to identify the current transfer session at the application layer.
- * hashPrevMessage REQUIRED. The cryptographic hash of the last message. The default hash algorithm is SHA256.
- * mintAssertionFormat REQUIRED. The default format is JSON, with parts being base64 encoded as needed. The default format is denoted as "MINT_ASSERTION_CLAIM_FORMAT_1".
- * mintAssertionClaim REQUIRED: The mint assertion claim or statement by the server.

Example:

```
{\ "messageType": "urn:ietf:satp:msgtype:commit-ready-msg",\
"sessionId": "d66a567c-11f2-4729-a0e9-17ce1faf47c1",\
"transferContextId": "89e04e71-bba2-4363-933c-262f42ec07a0",\
"hashPrevMessage":
"8dcc8dc4e6c2c979474b42d24d3747ce4607a92637d1a7b294857ff7288b8e46",\
"mintAssertionClaimFormat": "MINT_ASSERTION_CLAIM_FORMAT_1",\
"mintAssertionClaim": {},\ }\}
```

10.3. Commit Final Assertion Message (Commit-Final)

The purpose of this message is for the client to indicate to the server that the client (sender gateway) has completed the extinguishment (burn) of the asset in the origin network.

The message must contain a standalone claim related to the extinguishment of the asset by the client. The standalone claim must be signed by the client.

This message is sent from the client to the Commit Final Assertion Endpoint at the server.

The message must be signed by the server.

The parameters of this message consist of the following:

- * messageType REQUIRED: It MUST be the value urn:ietf:satp:msgtype:commit-final-msg.
- * sessionId REQUIRED: A unique identifier chosen earlier by the client in the Initialization Request Message.
- * transferContextId REQUIRED: A unique identifier used to identify the current transfer session at the application layer.
- * hashPrevMessage REQUIRED. The cryptographic hash of the last message. The default hash algorithm is SHA256.
- * burnAssertionClaimFormat REQUIRED. The default format is JSON, with parts being base64 encoded as needed. The default format is denoted as "BURN_ASSERTION_CLAIM_FORMAT_1".
- * burnAssertionClaim REQUIRED: The burn assertion signed claim or statement by the client.

Example:

```
{\ "messageType": "urn:ietf:satp:msgtype:commit-final-msg",\
"sessionId": "d66a567c-11f2-4729-a0e9-17celfaf47c1",\
"transferContextId": "89e04e71-bba2-4363-933c-262f42ec07a0",\
"hashPrevMessage":
"b92f13007216c58f2b51a8621599c3aef6527b02c8284e90c6a54a181d898e02",\
"burnAssertionClaimFormat": "BURN_ASSERTION_CLAIM_FORMAT_1",\
"burnAssertionClaim": {},\ }\}
```

10.4. Commit-Final Acknowledgement Receipt Message (ACK-Final-Receipt)

The purpose of this message is to indicate to the client that the server has completed the assignment of the newly minted asset to the intended beneficiary at the destination network.

This message is sent from the server to the Commit Final Receipt Endpoint at the client.

The message must be signed by the server.

The parameters of this message consist of the following:

- * messageType REQUIRED: It MUST be the value urn:ietf:satp:msgtype:ack-commit-final-msg.
- * sessionId REQUIRED: A unique identifier chosen earlier by client in the Initialization Request Message.
- * transferContextId REQUIRED: A unique identifier used to identify the current transfer session at the application layer.
- * hashPrevMessage REQUIRED. The cryptographic hash of the last message. The default hash algorithm is SHA256.
- * assignmentAssertionClaimFormat REQUIRED. The default format is JSON, with parts being base64 encoded as needed. The default format is denoted as "ASSIGNMENT_ASSERTION_CLAIM_FORMAT_1".
- * assignmentAssertionClaim REQUIRED: The claim or statement by the server that the asset has been assigned by the server to the intended beneficiary.

Example:

```
{\ "messageType": "urn:ietf:satp:msgtype:ack-commit-final-msg",\
"sessionId": "d66a567c-11f2-4729-a0e9-17celfaf47c1",\
"transferContextId": "89e04e71-bba2-4363-933c-262f42ec07a0",\
"hashPrevMessage":
"9c8f07c22ccf6888fc0306fee0799325efb87dfd536d90bb47d97392f020e998",\
```

```
"assignmentAssertionClaimFormat":  
"ASSIGNMENT_ASSERTION_CLAIM_FORMAT_1",\ "assignmentAssertionClaim":  
{},\ }\
```

10.5. Transfer Complete Message

The purpose of this message is for the client to indicate to the server that the asset transfer session (identified by `sessionId`) has been completed and no further messages are to be expected from the client in regards to this transfer instance.

The message closes the first message of Stage 2 (Transfer Commence Message).

This message is sent from the client to the Transfer Complete Endpoint at the server.

The message must be signed by the client.

The parameters of this message consist of the following:

- * `messageType` REQUIRED: It MUST be the value `urn:ietf:satp:msgtype:commit-transfer-complete-msg`.
- * `sessionId` REQUIRED: A unique identifier chosen earlier by the client in the Initialization Request Message.
- * `transferContextId` REQUIRED: A unique identifier used to identify the current transfer session at the application layer.
- * `hashPrevMessage` REQUIRED. The cryptographic hash of the last message. The default hash algorithm is SHA256.
- * `hashTransferCommence` REQUIRED: The hash of the Transfer Commence message at the start of Stage 2.

Example:

```
{\ "messageType": "urn:ietf:satp:msgtype:commit-transfer-complete-  
msg",\ "sessionId": "d66a567c-11f2-4729-a0e9-17celfaf47c1",\  
"transferContextId": "89e04e71-bba2-4363-933c-262f42ec07a0",\  
"hashPrevMessage":  
"9c8f07c22ccf6888fc0306fee0799325efb87dfd536d90bb47d97392f020e998",\  
"hashTransferCommence":  
"4ba76c69265f4215b4e2d2f24fe56e708512fdb49e27f50d2ac0095928e1531b",\  
}\
```

10.6. Error Message

The purpose of this message is for either the sender or the receiver gateways to indicate to its peer that an error has occurred within the transfer protocol flow.

This message must contain the error type (see the appendix) and the course of action indicated by the severity level. Typically, the action taken will be the immediate termination of the session.

- * messageType REQUIRED: It MUST be the value urn:ietf:satp:msgtype:error-msg.
- * sessionId REQUIRED: This is the current session in which the error pertains.
- * errorMsgType: The previous msg-type that was erroneous.
- * errorType REQUIRED: This is the error code being reported (Section 14, Paragraph 1).
- * errorSeverity REQUIRED: This is the severity level of the error, leading to the action.

Futher discussion on protocol errors can be found below (Section 14, Paragraph 1).

10.7. Session abort message

The purpose of this message is to indicate that one of the peer gateways has decided not to proceed with the session. No further messages will be delivered after the abort message.

- * messageType REQUIRED: It MUST be the value urn:ietf:satp:msgtype:session-abort-msg.
- * sessionId REQUIRED: This is the current session in which the abort occurs.

The effect of session aborts on the state of the asset is discussed below.

10.8. SATP Session Resumption

Session recovery and resumption is not supported in the current version of the SATP protocol. These may be addressed in a future version of SATP, or be defined in a separate specification.

The reader interested in this topic is directed to [BELC] for further discussion.

11. Error Messages

SATP distinguishes between session termination initiated by the user at the application layer from session termination caused by errors at the SATP protocol layer.

A gateway can transmit an error message at any point in the SATP protocol flow to its peer gateway.

The default action to be taken by the transitting gateway is to terminate the session immediately.

Error messages at the SATP protocol layer is distinct from time-outs due to gateway crashes.

11.1. Session Termination Notification

Session closure initiated at the application layer is not considered to be an error at the SATP protocol layer.

The message type used for application-initiated session termination: session-abort-msg.

The message type used to indicate protocols errors: error-msg.

A gateway can transmit the session abort message at any point in the SATP protocol flow. No further messages will be sent by the gateway.

Any data received after the session termination message MUST be ignored.

11.2. Connection Errors

Errors may occur at the connection layer, independent of the flows at the SATP layer and errors there.

(a) `connectionError`: There is an error in the TLS session establishment (TLS error codes should be reported-up to the gateway level)

(b) `badCertificate`: The gateway TLS certificate was corrupt, contained signatures, that did not verify correctly, etc. (Some common TLS level errors: `unsupported_certificate`, `certificate_revoked`, `certificate_expired`, `certificate_unknown`, `unknown_ca`).

Connection errors resulting in the time-out of the session MUST result in the termination of the transfer session. In the case of a transfer session termination, gateways SHOULD release its local computing resources and release asset-locks in their respective networks.

11.3. SATP Protocol Errors

The errors at the SATP level pertain to protocol flow and the information carried within each message. These are enumerated in the appendix.

11.4. Effectiveness of Session Aborts

The effectiveness of a session-abort message on the state of the asset depends on where the abort message occurs in the SATP protocol flow in Figure 2.

Note that a session-abort message maybe lost and never be received by the peer gateway. Gateways can crash prior to receiving an abort message.

If gateway G2 transmits a session-abort message after gateway G1 performs a lock (msgtype:lock-assert-msg) on the asset in network NW1, the gateway G1 can always unlock the asset and restore its state.

If either gateway G1 or gateway G2 transmits a session-abort message after gateway G1 sends a lock-assert message (msgtype:lock-assert-msg) but before G2 sends the commit ready message (msgtype:commit-ready-msg), the gateway G1 can always unlock the asset and restore its state in network NW1.

Similarly, if either gateway G1 or gateway G2 transmits a session-abort message immediately after gateway G1 sends a commit-prepare message (msgtype:commit-prepare-msg) but before G2 sends the commit ready message (msgtype:commit-ready-msg), the gateway G2 can always reverse the changes made by G2 to NW2 (i.e. reverse the assignment-to-self of the minted asset).

However, an abort message (occurring in either direction) after gateway G1 transmits the commit final message (msgtype:commit-final-msg) will not be effective. This is because G1 has already burned the asset in NW1 and G2 has already minted the asset in NW2 and has legally agreed to assign the asset to the appropriate beneficiary in NW2.

In general, the termination of sessions or aborts occurring before the sender gateway G1 disables (burns) the asset in NW1 (in flow 3.4 in Figure 2) will incur a minimal cost in terms of computing resources or fees on the part of both gateways G1 and G2.

12. Security Consideration

Gateways may be of interest to attackers because they enable the transferal of digital assets across networks and therefore are an important function in the digital economy.

- * Disruptions in transfers and denial of service: Disruptions to a transfer session may cause not only resource waste (e.g. CPU usage), but in some cases may result in financial loss on the part of the gateway operator (e.g. fees charged by network). Denial-of-service attacks by third parties to a run of the protocol may result in the termination of the current run (e.g. time-outs at the SATP layer), and for new attempts to be conducted.
- * Dishonest gateways: The SATP protocol requires gateways to sign messages related to the transfer layer, not only to provide message source authentication and integrity but also to maintain honesty on the part of the gateways. Gateway-operators may take-on legal and financial liabilities in certain jurisdictions by digitally signing messages. Dishonest gateways may intentionally delay the delivery of certain messages or intentionally fail (abort) the protocol run at certain crucial points [ARCH]. Two such crucial points in the message flows are the following: (i) the commit-final-msg, where the sender G1 asserts it has extinguished (burned) the asset in the origin network, and (ii) the ack-prepare-msg where the receiver gateway G2 asserts it is ready to proceed with the final commitment. If gateway G1 intentionally drops the commit-final-msg (commit-final) such that gateway G2 times-out, then G2 may suffer financial loss due to roll-back costs in network NW2. Similarly, if G2 intentionally drops the ack-prepare-msg to signal that it is ready to proceed with the commitment (commit-ready), then gateway G1 may time-out and terminate the protocol run, causing resource waste at G1. Operators of gateways should utilize relevant tools to detect possible dishonest behavior of certain gateways, and select to have their gateways peer with other reliable gateways.

- * Protection of gateway keys: It is crucial to protect the cryptographic keys utilized by gateways. This includes keys for secure session establishment (TLS1.3) and keys utilized for signing SATP messages. Loss of gateway keys may incur financial loss on the part of the gateway-operator. Implementation of gateways should consider utilizing tamper-resistant hardware to store and manage the relevant keys for gateways operational functions.
- * Gateway identification: Mechanisms must be utilized to provide unique identifiers to gateway implementations to ensure global uniqueness and reachability. Existing identification mechanisms such as X509 certificates [RFC5280] and Verifiable Credentials [W3CVC] may be applied for gateway identification.
- * Identification of networks: There needs to be mechanism for gateways to declare or disclose the asset networks it current serves. Combined with strong gateway identification, this allows remote gateways to quickly locate suitable gateways to peer with for the purposes of asset transfers.

13. IANA Consideration

The following request is being made to IANA.

13.1. SATP Error Codes Registry

This registry defines the error codes used in SATP protocol messages.

Many of the errors due to invalid identifiers (e.g., invalid transferContextId, invalid digitalAssetId) may arise within the execution of the SATP protocol because these identifiers depart from those agreed-upon in Transfer Initialization Claim in the transfer proposal message. The validity of these identifiers must be verified by the gateways during set-up stage (Stage-0), which is beyond the scope of the current specification. See Section 7 on the Identity and Asset Verification Stage.

In the following table, each entry consists of:

- * *Code*: The enumeration string (e.g., err_3.3.1)
- * *Category*: The protocol stage or message type (e.g., Commit Ready errors)
- * *Type*: The error type (e.g., badly formed message)

* *Description*: A brief description (e.g., mismatch transferContextId)

Code	Category	Type	Description
err_1.1.1	Transfer Proposal/Receipt errors	badly formed message	invalid transferContextId
err_1.1.2	Transfer Proposal/Receipt errors	badly formed message	invalid sessionId
err_1.1.3	Transfer Proposal/Receipt errors	badly formed message	incorrect transferInitClaimFormat
err_1.1.4	Transfer Proposal/Receipt errors	badly formed message	bad signature
err_1.1.11	Transfer Proposal/Receipt errors	badly formed claim	invalid digitalAssetId
err_1.1.12	Transfer Proposal/Receipt errors	badly formed claim	invalid assetProfileId
err_1.1.13	Transfer Proposal/Receipt errors	badly formed claim	invalid verifiedOriginatorEntityId
err_1.1.14	Transfer Proposal/Receipt errors	badly formed claim	invalid verifiedBeneficiaryEntityId
err_1.1.15	Transfer Proposal/Receipt errors	badly formed	invalid originatorPublicKey

	Receipt errors	claim	
err_1.1.16	Transfer Proposal/Receipt errors	badly formed claim	invalid beneficiaryPublicKey
err_1.1.17	Transfer Proposal/Receipt errors	badly formed claim	invalid senderGatewaySignaturePublicKey
err_1.1.18	Transfer Proposal/Receipt errors	badly formed claim	invalid receiverGatewaySignaturePublicKey
err_1.1.19	Transfer Proposal/Receipt errors	badly formed claim	invalid senderGatewayId
err_1.1.20	Transfer Proposal/Receipt errors	badly formed claim	invalid recipientGatewayId
err_1.1.31	Transfer Proposal/Receipt errors	badly formed parameter	unsupported gatewayDefaultSignatureAlgorithm
err_1.1.32	Transfer Proposal/Receipt errors	badly formed parameter	unsupported networkLockType
err_1.1.33	Transfer Proposal/Receipt errors	badly formed parameter	unsupported networkLockExpirationTime
err_1.1.34	Transfer Proposal/Receipt errors	badly formed parameter	unsupported gatewayTlsScheme

err_1.1.35	Transfer Proposal/ Receipt errors	badly formed parameter	unsupported gatewayLoggingProfile
err_1.1.36	Transfer Proposal/ Receipt errors	badly formed parameter	unsupported gatewayAccessControlProfile
err_1.2.1	Transfer Proposal/ Receipt errors	badly formed message	mismatch transferContextId
err_1.2.2	Transfer Proposal/ Receipt errors	badly formed message	mismatch sessionId
err_1.2.3	Transfer Proposal/ Receipt errors	badly formed message	mismatch hashTransferInitClaim
err_1.2.4	Transfer Proposal/ Receipt errors	badly formed message	bad signature
err_1.3.1	Transfer Commence errors	badly formed message	mismatch transferContextId
err_1.3.2	Transfer Commence errors	badly formed message	mismatch sessionId
err_1.3.3	Transfer Commence errors	badly formed message	mismatch hashTransferInitClaim
err_1.3.4	Transfer Commence errors	badly formed message	mismatch hashPrevMessage
err_1.3.5	Transfer Commence	badly formed	bad signature

	errors	message	
err_1.4.1	ACK Commence errors	badly formed message	mismatch transferContextId
err_1.4.2	ACK Commence errors	badly formed message	mismatch sessionId
err_1.4.3	ACK Commence errors	badly formed message	mismatch hashPrevMessage
err_1.4.4	ACK Commence errors	badly formed message	bad signature
err_2.2.1	Lock Assertion errors	badly formed message	mismatch transferContextId
err_2.2.2	Lock Assertion errors	badly formed message	mismatch sessionId
err_2.2.3	Lock Assertion errors	badly formed message	unsupported lockAssertionClaimFormat
err_2.2.4	Lock Assertion errors	badly formed message	unsupported lockAssertionExpiration
err_2.2.5	Lock Assertion errors	badly formed message	mismatch hashPrevMessage
err_2.2.6	Lock Assertion errors	badly formed message	bad signature
err_2.4.1	Lock Assertion Receipt errors	badly formed message	mismatch transferContextId
err_2.4.2	Lock	badly	mismatch sessionId

	Assertion Receipt errors	formed message	
err_2.4.3	Lock Assertion Receipt errors	badly formed message	mismatch hashPrevMessage
err_2.4.4	Lock Assertion Receipt errors	badly formed message	bad signature
err_3.1.1	Commit Preparation errors	badly formed message	mismatch transferContextId
err_3.1.2	Commit Preparation errors	badly formed message	mismatch sessionId
err_3.1.3	Commit Preparation errors	badly formed message	mismatch hashPrevMessage
err_3.1.4	Commit Preparation errors	badly formed message	bad signature
err_3.3.1	Commit Ready errors	badly formed message	mismatch transferContextId
err_3.3.2	Commit Ready errors	badly formed message	mismatch sessionId
err_3.3.3	Commit Ready errors	badly formed message	mismatch hashPrevMessage
err_3.3.4	Commit Ready errors	badly formed message	unsupported mintAssertionFormat
err_3.3.5	Commit Ready	badly formed	bad signature

	errors	message	
err_3.5.1	Commit Final Assertion errors	badly formed message	mismatch transferContextId
err_3.5.2	Commit Final Assertion errors	badly formed message	mismatch sessionId
err_3.5.3	Commit Final Assertion errors	badly formed message	mismatch hashPrevMessage
err_3.5.4	Commit Final Assertion errors	badly formed message	unsupported burnAssertionClaimFormat
err_3.5.5	Commit Final Assertion errors	badly formed message	bad signature
err_3.7.1	Commit Final Ack Receipt errors	badly formed message	mismatch transferContextId
err_3.7.2	Commit Final Ack Receipt errors	badly formed message	mismatch sessionId
err_3.7.3	Commit Final Ack Receipt errors	badly formed message	mismatch hashPrevMessage
err_3.7.4	Commit Final Ack Receipt errors	badly formed message	unsupported assignmentAssertionClaimFormat
err_3.7.5	Commit	badly	bad signature

	Final Ack Receipt errors	formed message	
err_3.9.1	Transfer Complete errors	badly formed message	mismatch transferContextId
err_3.9.2	Transfer Complete errors	badly formed message	mismatch sessionId
err_3.9.3	Transfer Complete errors	badly formed message	mismatch hashPrevMessage
err_3.9.4	Transfer Complete errors	badly formed message	mismatch hashTransferCommence
err_3.9.5	Transfer Complete errors	badly formed message	bad signature

Table 1

13.2. URN Registration

URN: Request to be assigned by IANA.

Common Name: urn:ietf:satp

Registrant Contact: IESG

Description: The secure asset transfer protocol (SATP) requires message types, endpoints and parameters to be defined within a unique namespace to prevent collision.

13.3. SATP Message Types Registry

This specification establishes the SATP Message Types registry. The purpose of this registry is to define the various message types utilized in the secure asset transfer protocol (SATP).

13.4. Initial Registry Contents

The SATP Message Types registry's initial contents are as follows:

13.4.1. Parameter name: transfer-proposal-msg

- * Parameter usage location: Transfer Proposal
- * Change controller: IETF
- * Specification document(s): Section 8.3 of draft-ietf-satp-core.

13.4.2. Parameter name: proposal-receipt-msg

- * Parameter usage location: Transfer Proposal Receipt Message
- * Change controller: IETF
- * Specification document(s): Section 8.4 of draft-ietf-satp-core.

13.4.3. Parameter name: reject-msg

- * Parameter usage location: Transfer Reject
- * Change controller: IETF
- * Specification document(s): Section 8.5 of draft-ietf-satp-core.

13.4.4. Parameter name: transfer-commence-msg

- * Parameter usage location: Transfer Commence
- * Change controller: IETF
- * Specification document(s): Section 8.6 of draft-ietf-satp-core.

13.4.5. Parameter name: ack-commence-msg

- * Parameter usage location: Transfer Commence Response
- * Change controller: IETF
- * Specification document(s): Section 8.7 of draft-ietf-satp-core.

13.4.6. Parameter name: lock-assert-msg

- * Parameter usage location: Lock Assertion

- * Change controller: IETF
- * Specification document(s): Section 9.1 of draft-ietf-satp-core.

13.4.7. Parameter name: assertion-receipt-msg

- * Parameter usage location: Lock Assertion Receipt
- * Change controller: IETF
- * Specification document(s): Section 9.2 of draft-ietf-satp-core.

13.4.8. Parameter name: commit-prepare-msg

- * Parameter usage location: Commit Preparation
- * Change controller: IETF
- * Specification document(s): Section 10.1 of draft-ietf-satp-core.

13.4.9. Parameter name: commit-ready-msg

- * Parameter usage location: Commit Ready
- * Change controller: IETF
- * Specification document(s): Section 10.2 of draft-ietf-satp-core.

13.4.10. Parameter name: commit-final-msg

- * Parameter usage location: Commit Final Assertion
- * Change controller: IETF
- * Specification document(s): Section 10.3 of draft-ietf-satp-core.

13.4.11. Parameter name: ack-commit-final-msg

- * Parameter usage location: Commit-Final Acknowledgement Receipt
- * Change controller: IETF
- * Specification document(s): Section 10.4 of draft-ietf-satp-core.

13.4.12. Parameter name: commit-transfer-complete-msg

- * Parameter usage location: Transfer Complete

- * Change controller: IETF
- * Specification document(s): Section 10.5 of draft-ietf-satp-core.

13.4.13. Parameter name: error-msg

- * Parameter usage location: Error message
- * Change controller: IETF
- * Specification document(s): Section 10.6 of draft-ietf-satp-core.

13.4.14. Parameter name: session-abort-msg

- * Parameter usage location: Session Abort
- * Change controller: IETF
- * Specification document(s): Section 10.7 of draft-ietf-satp-core.

14. Error Types and Codes

This appendix defines the error codes that may be returned in SATP protocol messages.

14.1. Protocol Error Codes

The following error codes are defined for SATP protocol errors:

- * err_1.1: Invalid message type
- * err_1.2: Invalid session ID
- * err_1.3: Invalid transfer context ID
- * err_1.4: Invalid signature
- * err_1.5: Invalid hash value
- * err_2.1: Asset not found
- * err_2.2: Asset already locked
- * err_2.3: Asset lock expired
- * err_2.4: Insufficient permissions
- * err_3.1: Network connection failure

- * err_3.2: Gateway unavailable
- * err_3.3: Timeout exceeded
- * err_4.1: Unsupported credential scheme
- * err_4.2: Invalid credential format
- * err_4.3: Credential verification failed

15. Acknowledgements

The authors would like to thank the following people for their input and support:

Andre Augusto, Denis Avrilionis, Rafael Belchior, Alexandru Chiriac, Anthony Culligan, Claire Facer, Martin Gfeller, Wes Hardaker, David Millman, Krishnasuri Narayanam, Anais Ofranc, Luke Riley, John Robotham, Orie Steele, Yaron Scheffer, Peter Somogyvari, Weijia Zhang.

16. References

16.1. Normative References

- [DATETIME] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <<https://www.rfc-editor.org/rfc/rfc3339>>.
- [REQ-LEVEL] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/rfc/rfc2119>>.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<https://www.rfc-editor.org/rfc/rfc4648>>.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <<https://www.rfc-editor.org/rfc/rfc5280>>.
- [RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <<https://www.rfc-editor.org/rfc/rfc6749>>.

- [RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <<https://www.rfc-editor.org/rfc/rfc7515>>.
- [RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <<https://www.rfc-editor.org/rfc/rfc7517>>.
- [RFC7518] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, DOI 10.17487/RFC7518, May 2015, <<https://www.rfc-editor.org/rfc/rfc7518>>.
- [RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <<https://www.rfc-editor.org/rfc/rfc7519>>.
- [RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <<https://www.rfc-editor.org/rfc/rfc8259>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/rfc/rfc8446>>.
- [RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <<https://www.rfc-editor.org/rfc/rfc9110>>.

16.2. Informative References

- [ARCH] Hardjono, T., Hargreaves, M., Smith, N., and V. Ramakrishna, "Secure Asset Transfer (SAT) Interoperability Architecture", June 2024, <<https://datatracker.ietf.org/doc/draft-ietf-satp-architecture/>>.
- [BELC] Belchior, R., Vasconcelos, A., Correia, M., and T. Hardjono, "Hermes a Fault-tolerant middleware for blockchain interoperability", April 2024, <<https://doi.org/10.1016/j.future.2021.11.004>>.
- [BTC] Nakamoto, S., "Bitcoin A Peer-to-Peer Electronic Cash System", 2008, <<https://bitcoin.org/bitcoin.pdf>>.
- [ECDSA] "Digital Signature Standard (FIPS 186-5)", February 2023, <<https://doi.org/10.6028/NIST.FIPS.186-5>>.

- [ETH] Buterin, V., "Ethereum A next-generation smart contract and decentralized application platform", 2018, <<https://ethereum.org>>.
- [MICA] European Commission, "EU Directive on Markets in Crypto-Assets Regulation (MiCA)", June 2023, <<https://www.esma.europa.eu/esmas-activities/digital-finance-and-innovation/markets-crypto-assets-regulation-mica>>.
- [NIST] Yaga, D., Mell, P., Roby, N., and K. Scarfone, "NIST Blockchain Technology Overview (NISTR-8202)", October 2018, <<https://doi.org/10.6028/NIST.IR.8202>>.
- [RFC9334] Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote ATtestation procedureS (RATS) Architecture", RFC 9334, DOI 10.17487/RFC9334, January 2023, <<https://www.rfc-editor.org/rfc/rfc9334>>.
- [W3CVC] W3C, "Verifiable Credentials Overview", September 2025, <<https://www.w3.org/TR/vc-overview/>>.
- [XRP] Schwartz, D., Youngs, N., and A. Britto, "The Ripple Protocol Consensus Algorithm", 2014, <<https://ripple.com/files/ripple-consensus-whitepaper.pdf>>.

Authors' Addresses

Martin Hargreaves
Quant Network
Email: martin.hargreaves@quant.network

Thomas Hardjono
MIT
Email: hardjono@mit.edu

Rafael Belchior
INESC-ID, Tcnico Lisboa, Blockdaemon
Email: rafael.belchior@tecnico.ulisboa.pt

Venkatraman Ramakrishna
IBM
Email: vramakr2@in.ibm.com

Alex Chiriac
Quant Network
Email: alexandru.chiriac@quant.network