VAAs

VAAs (Verified Action Approvals)

VAAs are the core messaging primitive in Wormhole. You can think of them as packets of cross chain data that are emitted any time a cross chain application contract interacts with the Core Contract.

The basic VAA has two components -- a Header and a Body.

Messages emitted by contracts need to be validated by the guardians before they can be sent to the target chain. Once a majority of guardians observe the message and finality has been achieved, the Guardianssign a keccak256 hash of the message body .

The message is wrapped up in a structure called a VAA which combines the message with the guardian signatures to form a proof.

VAAs are uniquely indexed by the (emitter_chain ,emitter_address ,sequence) tuple. A VAA can be obtained by querying the GuardianRPC or theAPI with this information.

These VAA's are ultimately what a smart contract on a receiving chain must process in order to receive a wormhole message.

VAA Format

VAA's contain two sections of data.

Header

The header holds metadata about the current VAA, the guardian set that is currently active, and the list of signatures gathered so far.

```

Copy byte version// VAA Version u32guardian_set_index// Indicates which guardian set is signing u8len_signatures// Number of signatures stored []signature signatures// Collection of guardian signatures

```

where eachsignature is:

```

Copy u8index// The index of this guardian in the guardian set [65]byte signature// The ECDSA signature

```

Body

The body is deterministically derived from an on chain message. Any two guardians must derive the exact same body. This requirement exists so that there is always a one to one relationship between VAA's and messages to avoid double processing of messages.

```

Copy u32timestamp// The timestamp of the block this message was published in u32nonce// u16emitter_chain// The id of the chain that emitted the message [32]byte emitter_address// The contract address (wormhole formatted) that called the core contract u64sequence// The auto incrementing integer that represents the number of messages published by this emitter u8consistency_level// The consistency level (finality) required by this emitter []byte payload// arbitrary bytes containing the data to be acted on

```

The Body is the relevant information for consumers and is handed back from a call likeparseAndVerifyVAA . Because theemitterAddress is included as part of the Body, the developer is able to tell if this VAA originated from a trusted contract.

Because VAAs have no destination, they are effectively multicast. They will be verified as authentic by any Core Contract on any chain in the network. If a VAA has a specific destination, it is entirely the responsibility of relayers to complete that delivery appropriately.

Signatures

Thebody of the VAA is hashedtwice withkeccak256 to produce the digest message that is signed.

```

Copy // hash the bytes of the body twice digest=keccak256(keccak256(body)) // sign the result signature=ecdsa_sign(digest, key)

```

Different implementations of the ECDSA signature validation may apply akeccak256 hash to the message passed, so care must be taken to pass the correct arguments.

For example theSolana secp256k1 program will hash the message passed. In this case, the argument for the message should be asingle hash of the body, not thetwice hashed body.

Payload Types

Different applications that are built on wormhole may specify a format for the payloads attached to a VAA. This payload provides information the target chain and contract so it can take some action (e.g. minting tokens to a receiver address).

Token Transfer

Tokens are transferred from one chain to another using a lockup/mint and burn/unlock mechanism. While many bridges work on this basic premise, this implementation achieves this by relying on the generic message passing protocol provided by Wormhole to support routing the lock and burn events from one chain to another. This makes Wormhole's token bridge completely chain agnostic. As long as a wormhole contract exists on the chain we wish to transfer to, an implementation can be quickly incorporated into the network. Due to the generic message passing nature of Wormhole, programs emitting messages do not need to know anything about the implementation details of any other chain.

In order to transfer tokens from A to B, we must lock the tokens on A and mint them on B. It is important that the tokens on A have been proven to be locked before the minting can occur on B. To facilitate this process, chain A first locks the tokens and emits a message indicating that the locking has been completed. This message has the following structure, and is referred to as a transfer message:

```

Copy u8payload_id=1// Token Transfer u256 amount// Amount of tokens being transferred. u8[32] token_address// Address on the source chain. u16token_chain// Numeric ID for the source chain. u8[32] to// Address on the destination chain. u16to_chain// Numeric ID for the destination chain. u256 fee// Portion of amount paid to a relayer.

```

This structure contains everything needed for the receiving chain to learn about a lockup event. Once Chain B receives this payload it can mint the corresponding asset.

Note that Chain B is agnostic as to how the tokens on the sending side were locked. They could have been burned by a mint or locked in a custody account. The protocol simply relays the event once a sufficient amount of guardians have attested its existence.

Attestation

The Transfer event above is missing an important detail. While the program on Chain B can trust the message to inform it of token lockup events, it has no way to know what the token being locked up actually is. The address alone is a meaningless value to most users. To solve this, the Token Bridge supportstoken attestation .

For a Token Attestation, Chain A emits a message containing metadata about a token which Chain B may use to persist the name, symbol, and decimal precision of a token address.

The message format for this action is as follows:

```

Copy u8payload_id=2// Attestation [32]byte token_address// Address of the originating token contract u16token_chain// Chain ID of the originating token u8decimals// Number of decimals this token should have [32]byte symbol// Short name of asset [32]byte name// Full name of asset

```

Attestations use a fixed-length byte-array to encode UTF8 token name and symbol data.

Because the byte array is fixed length, the data contained may truncate multibyte unicode characters. Whensending an attestation VAA, we recommend sending the longest UTF8 prefix that does NOT truncate a character, and then right-

padding it with 0 bytes.

Whenparsing an attestation VAA, we recommend trimming all trailing 0 bytes, and converting the remainder to UTF8 via any lossy algorithm.

Be mindful that different on-chain systems may have different VAA parsers, that may result in different names/symbols on different chains if the string is long, or contains invalid UTF8. An important detail of the token bridge is that an attestation is required before a token can be transferred. This is because without knowing a tokens decimal precision, it is not possible for Chain B to correctly mint the correct amount of tokens when processing a transfer.

Token + Message

This VAA type is also referred to as apayload3 message or aContract Controlled Transfer . The Token + Message data structure is identical to the Token only data structure with the addition of apayload field that contains arbitrary bytes. This arbitrary byte field is where an app may include additional data in the transfer to inform some application specific behavior.

```

Copy u8payload_id=3// Token Transfer with Message u256 amount// Amount of tokens being transferred. u8[32] token_address// Address on the source chain. u16token_chain// Numeric ID for the source chain. u8[32] to// Address on the destination chain. u16to_chain// Numeric ID for the destination chain. u256 fee// Portion of amount paid to a relayer. []byte payload// Message, arbitrary bytes, app specific

```

Governance

Governance VAAs don't have apayload_id field like the above formats, they're used to trigger some action in the deployed contracts (e.g. upgrade).

Action Structure

Governance messages contain pre-defined actions, which can target the various wormhole modules currently deployed on chain. The structure contains the following fields:

```

Copy u8[32] module// Contains a right-aligned module identifier u8action// Predefined governance action to execute u16chain// Chain the action is targeting, 0 = all chains ...args// Arguments to the action

```

Here is an example message containing a governance action triggering a code upgrade to the solana core contract. The module field here is a right-aligned encoding of the ASCII "Core", represented as a 32 byte hex string.

```

Copy module:0x0000000000000000000000000000000000000000000000000000000000000000436f7265 action:1 chain:1 new_contract:0x3485672937589571623749593761923748845625222819374462348283888283

```

Actions

The meaning of each numeric action is pre-defined and documented in the wormhole design documents. For each application, the relevant definitions can be found via these links:

- Core governance actions are documentedhere
- .
- Token Bridge governance actions are documentedhere
- .
- NFT Bridge governance actions are documentedhere
- .
-

Lifetime of a Message

Note: Anyone can submit the VAA to the target chain. The guardians typically do not perform this step to avoid transaction fees. Instead, applications built on top of wormhole can acquire the VAA via the guardian RPC and do the submission in a separate flow. With the concepts now defined, we can illustrate what a full flow for a message passing between two chains looks like. The following stages demonstrate each stage of processing the wormhole network performs in order to route a

message.

- 1: A message is emitted by a contract running on chain A.
- Messages can be emitted by any contract, and the guardians are programmed to observe all chains for these events. Here, the guardians are represented as a single entity to simplify the graphics but the observation of the message must be performed individually by each of the 19 guardians
- 2: Signatures are aggregated.
- Guardians independently observe and sign the message. Once enough guardians have signed the message, the collection of signatures are combined with the message and metadata to produce a VAA.
- 3: VAA submitted to target chain.
- The VAA acts as proof that the guardians have collectively attested the existence of the message payload, in order to complete the final step, the VAA itself is submitted (orrelayed
-) to the target chain to be processed by a receiving contract.
-

Last updated2 months ago

On this page * VAAs (Verified Action Approvals) * VAA Format * Signatures * Payload Types * Lifetime of a Message

Was this helpful? Edit on GitHub