THORChain TSS: Leader-less Threshold Signature Protocol

thorchain.org

Abstract

THORChain TSS is a practical distributed threshold signature protocol which is based on the the scheme proposed by Genarro-Goldfeder in 2018. THORChain TSS is a robust, secured and leader-less signature scheme which allows a number of signers to be involved in signing on a given message. End-to-End encryption ensures the security of the communication. Reliable broadcast reduces the risk of single point of failure. The blame scheme is introduced to punish the nodes who do not comply with the protocol. The attacker has a low motivation to attack the network as they face a high risk of losing their bond.

July 10, 2020

Introduction

The Threshold Signature Scheme (TSS) is the Genarro-Goldfeder 2018 Gennaro & Goldfeder (2018) process, which allows efficient signing with no trusted dealer. There are two multi-party computation routines; key-generation and key-signing. The key-generation ceremony allows the nominated committees to construct the parameters for a new vault and the output is a public key from which the vault addresses for each chain is derived. Both secp256k1 *Secp256k1-Bitcoin Wiki* (n.d.) and ed25519 *edd25519 Wiki* (n.d.) chains are supported, and the vault address derivation is dependent on the chain. When the network delegates an outgoing transaction to be signed from a certain vault public key, relevant signers recognise their participation, prepare a copy of the message to be signed from their local key-value storage, and enter a signing session. The key-signing ceremony begins when the required number of participants are present and allows a signature for an outgoing transaction to be generated. Once a valid signature is generated, all signers attempt to broadcast it to the relevant external network and one will be accepted. Since external networks handle sequence numbers, nonces or UTXOs differently, a chain-specific module is used to translate generic outgoing transactions from the network into compatible transaction messages for each chain. Batch-signing is supported for networks that can handle it in order to increase signing efficiency.

The key-generation ceremony involves a communication round where each participant checks the validity of all other expected members, as prescribed by the system. Additionally to prevent spoofing, a commit-reveal scheme is used to ensure secret shares for each participant cannot be changed after the fact. During key-signing, all signers prepare and sign a locally-generated message only, so they also cannot be spoofed. If any of signers abort a key-generation or key-signing process which results in an attributable failure, all participants can make a blame transaction. If consensus is reached on who to blame, the blamed node is penalised and cycled to be churned out.

Thorchain Tss Structure

2.1 THORChain TSS Architecture

Thorchain Tss accepts the requests from the caller, organises the parties for the keygen/keysign process, manages the Tss messages and establishes the connection with the peers through the P2P network. To achieve the above functionality, we divide Thorchain Tss into five layers shown in fig.1.

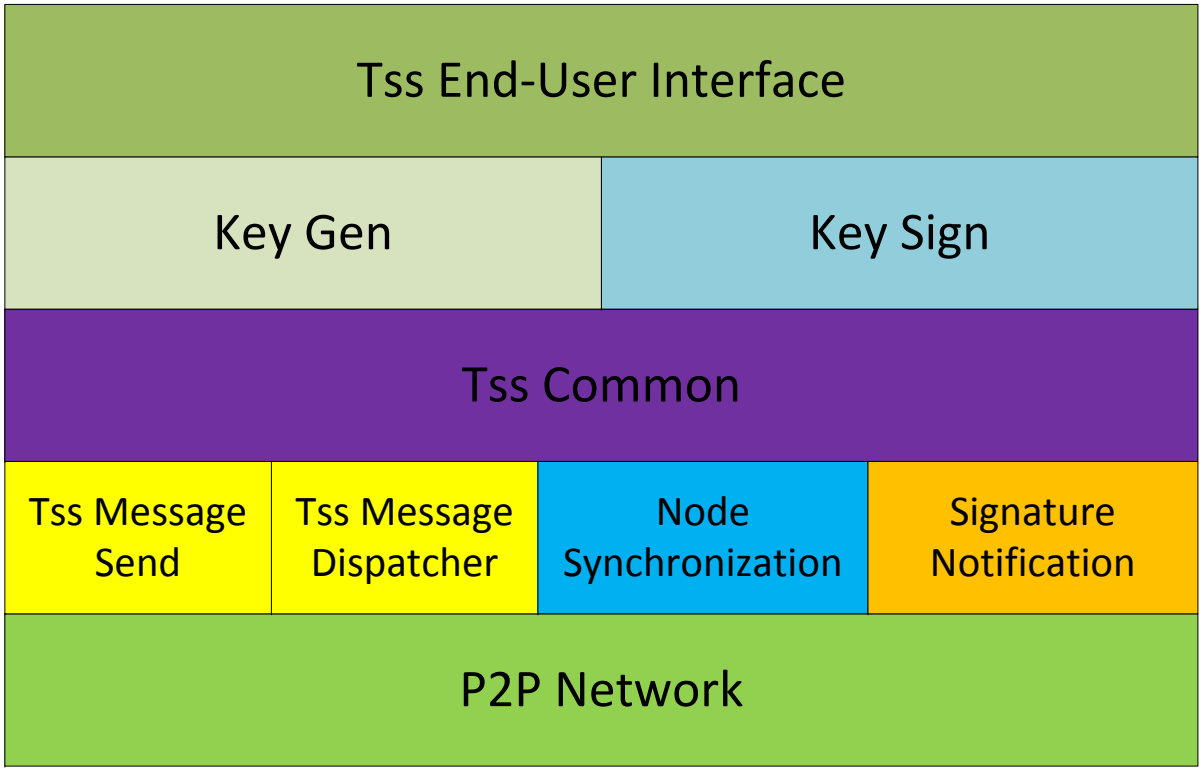

Figure 1: *Thorchain Tss Architecture*

2.1.1 Tss End-User Interface

This layer handles the request from the Tss caller and returns the result back to the caller. It hides the details of how the Tss is working.

For the keygen request, the input is the public keys of the peers that involved in the keygen. The output of the keygen request is the generated Tss public keys and the BECH32 address derived from this public key.

For the keysign request, the input is the 1.) Tss public key. 2.) the public keys of the nodes that involved in the keysign 3.) The hash of the message that needed to be signed. The output of the keysign request is the signature to the given message.

2.1.2 keygen/keysign Modules

The function of these two modules is to instantiate the local and remote parties for the keygen/keysign with the given public keys. When the essential data structures are ready and all the nodes admit starting the keygen/keysign, the local party starts the keygen/keysign process.

It is also in charge of broadcasting the Tss signatures who are not involved in the Tss keysign round to allow all the nodes to be synchronized.

2.1.3 Common Functionality Layer

This layer processes the shared Tss operation in keygen and keysign. Basically, this layer handles the shares generated in each TSS processing round by 1.) packs the shares generated by the keygen/keysign. 2.) sends to the peers. 3.) does the data integrity check of the received shares. 4.) applies the shares to the local party.

2.1.4 Communication Layer

Communication layer focuses on communications between THORChain TSS and the underlying P2P network. THORChain TSS message sender takes care of the messages that send to the network and the message dispatcher receives the messages from the network and pushes the message to the subscribers' channel with a given topic. We separate the node synchronisation and signature notification from the THORChain TSS model as independent services which dedicate for notifying the peers of the generated Tss signatures and synchronise the nodes when they join the THORChain TSS for keygen/keysign.

2.1.5 Network Infrastructure Layer

The network infrastructure layer is in charge of establishing the P2P connections with other peers, discovering other peers in the P2P network and sending/receiving packages from the P2P network.

2.2 THORChain TSS Data Flow

The THORChain TSS data flow of keygen and keysign are similar, to make the discussion concise, we take keysign as an example described in the fig. 2 to demonstrate how THORChain TSS works.

When a keysign request is sent to THORChain TSS, local node firstly checks whether it has enough peers to run the keysign. If this node is an active participant, it runs the join party scheme and waits for other peers to show up within a given time. If any party fail to join the party within a given time, others blames that party and quits the keysign process. If all peers are shown up in a given time, the keysign process is kicked off. It digests the shares received from other peers within a given time otherwise, blame the party who fail to offer the share. Finally, the keysign process is done, and it waits for the notifications sent from the peers which indicating all the peers finish the keysign. Before it quits the keysign process, it broadcasts the generated signature to all the parties who are involved in the same keygen process.

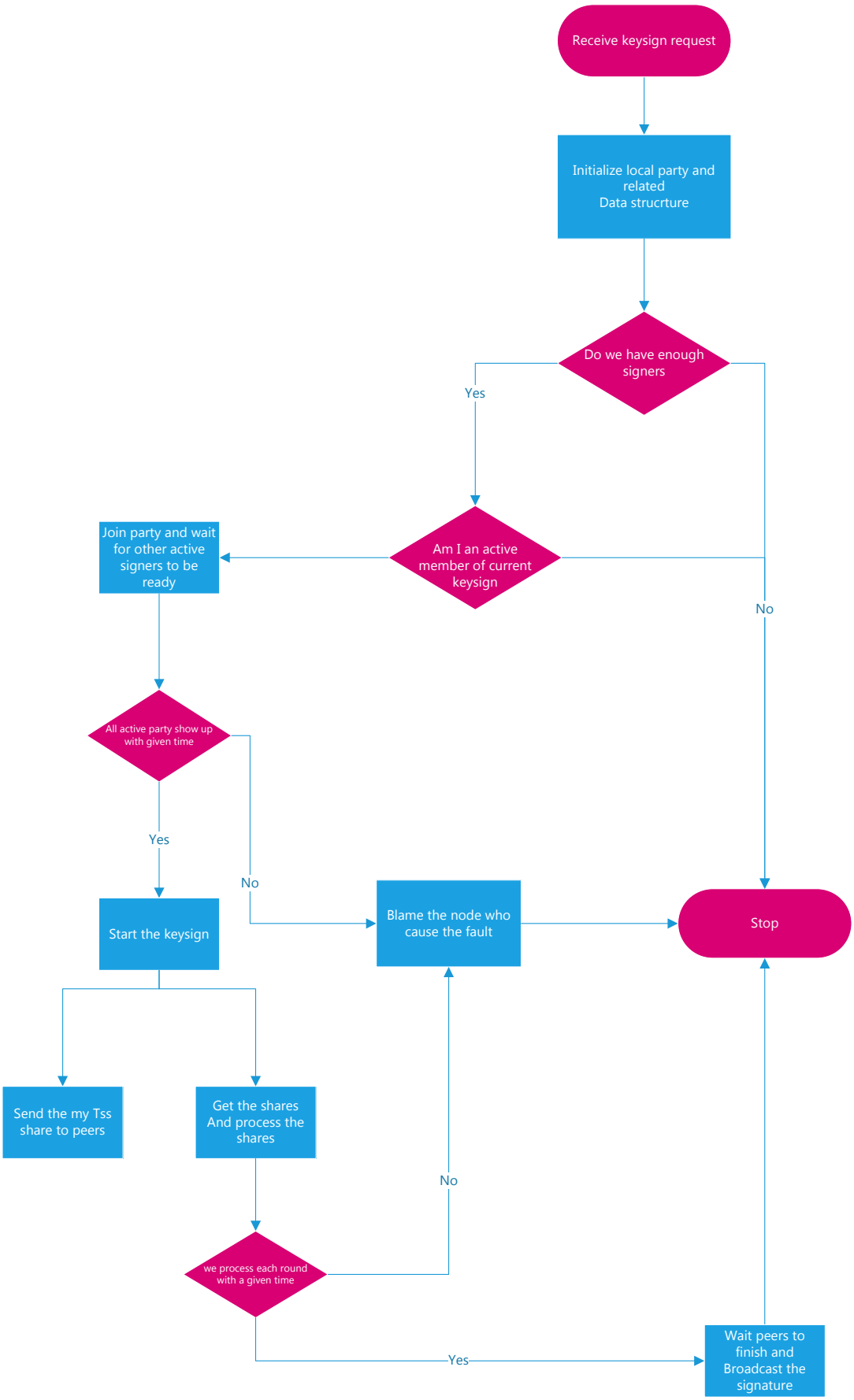

Figure 2: *THORChain TSS data flow chat*

Thorchain TSS Operations and Features

In this section, we firstly introduce the operations that THORChain TSS supports. Then, we discuss the features of the THORChain TSS.

3.1 Thorchain TSS Operations

In general, THORChain TSS supports two operations which are generating of the ECDSA key-pair and signing on a given message.

3.1.1 TSS Key Generation

Before THORChain TSS signs on any messages, all the parties in the THORChain TSS should collaborate together to generate the TSS key-pairs. In THORChain TSS, we require all the parties that participant in the key generation operation to have their clock loosely synchronised which allows them to join the key generation at almost the same time. From the view of each node, they know who are the peers of the key generation process and only communicate with these peers to generate the key.

Once the TSS key-pair is generated, the local party stores the share of the TSS secret key locally and the public key is known and agreed among all the parties. If any parties do not cooperate in the key generation, others blame them.

Since the local party only has the cryptographic share of the TSS secret key, no party can recover the secret key that matches with the public key that generated among all the participants, as a result, no parties can sign the transactions on behalf of the rest of the participants.

3.1.2 TSS Key Sign

THORChain TSS operation allows the a number of participants to sign on the messages that can be verified by the THORChain TSS public key. Since THORChain TSS is a kind of threshold signature scheme, THORChain TSS allows some of the signers to be absent while others can process the message signing requests.

Before the THORChain TSS starts, all the online parties should agree on the participants who suppose to be involved in the signature generation process. Nodes that involved in the keysign process should participant in the signature generation within a given time frame. If any parties do not cooperate and fail the message signing process other parties will blame them.

Once a signature is generated, the participants broadcast the signature to all the parties who are not involved in the signing process.

3.2 THORChain TSS Features

In this section, we discuss the features that supported by THORChain TSS to achieve high transaction throughput.

3.2.1 Parallel TSS Processing

THORChain TSS supports the same signers working on different signing tasks in parallel. This design enables the signers to work on different messages processing threads. **'session ID'** is applied to allow the underlying communication management service to dispatch the TSS messages to the corresponding TSS process thread. The failure of one TSS process will not affect others which makes the side channel attack impossible. From the view of each TSS

process, it occupies all the computing and network resources and has no aware of the existing of other TSS processes.

3.2.2 Batch Signing

Batch Signing is another feature that is supported by THORChain TSS. For the batch signing, TSS signing participants can send more than one messages to the TSS to sign in a single TSS signing request. Batch signing reduces the network communication between different nodes dramatically. As a result, it increases the throughput of the TSS message signing.

To enable the batch signing, THORChain TSS stores the intermediate results of each message signing round for different TSS messages and packs them together as a bulk message that pushes to the peer nodes. On receiving the bulk P2P message, receiver verifies the correctness of the bulk message and process these TSS messages one by one.

With the help of batch signing, THORChain TSS can process all transactions inside a THORChain block as one batch signing request.

THORChain TSS Security and Reliability Design

Security is the highest priority for THORChain TSS. To ensure all the parties in TSS comply with the protocol, we have the following two key components in the THORChain TSS.

4.1 Secured Communication Channel

The TSS shares in the key generation and message signing process are sensitive. To avoid any leakage of the shares to the third party, we employed the libp2p library to ensure our network infrastructures is secured and private.

The key technologies that applied in the THORChain TSS are discussed as follows:

- Identity verification before the establishment of the P2P network. When a node adds a new peer by the P2P address to its network, it needs to verify the identity of the node by checking whether the peer can prove it holds the private key to that P2P address.
- End-to-End encryption. For all the communication, we require the sender and receiver to apply the public key encryption to encrypt the message, thus, only the private key holder can decrypt the message.
- Point to Point communication without P2P relay. Direct point-to-point communication is critical which eliminates the possibility of man-in-the-middle-attack.

4.2 Reliable Broadcast

The nature of signature generation scheme makes it vulnerable under the single point of failure. In THORChain TSS we have two approaches to increase the reliability of TSS by 1.) Reject the nodes attending future TSS operations with peers if they misbehave. 2.) If a node sends incorrect share to a certain number of nodes, the system can correct the error by itself without interrupting the TSS process.

Reliable broadcast tries to eliminate the problem that the system cannot get the consistent view of whom to blame when the THORChain TSS fails to finish the task.

4.2.1 Key Technologies in Reliable Broadcast

1. **Undeniable Broadcast:**
    - Since the share owner signs on the share he broadcasts, he cannot deny that this share is fabricated by others. If he sends the wrong share to the majority and nodes apply it in TSS library, it result in an error, and peers blame him with his signature.
    - A node can never stop cooperating with a node by saying that node gives it the wrong share. If it complains about the received incorrect share, it should provide the signature matches with that wrong share.
    - When a node receives a forwarded share, it can check whether the message is from the claimed share owner against the signature attached to the share (avoid-man-in-the-middle attack).
2. **Share Forwarding:**

When a node fails to receive the correct share (we define the correct share as 2/3 nodes agree on), it can ask this share from the peers who claim to have this share. In this way, TSS process moves forward.

3. **Judge on Majority:** Given the assumption that 2/3 parties are honest, a node judges whether it receives the correct share by checking whether the share it holds is the majority (here, the correct means the share received by the majority, it is irrelevant to the correctness of the data of the share). For a node, if it finds that its share is different from the majority, it will request for the correct share forwarding. In this way, it eliminates the attack like a). share owner sends a wrong share to a node to mislead others to blame this victim. b). one or some malicious nodes send the wrong hash of a share thus trying to halt the victim.

4. **Finish Notification:** For a given TSS process, we need to wait for all the parties finish their process with a time out. During the waiting, if a node requests a share, the peer can send this share and help this node to finish TSS process. This avoids attacks where the malicious owner refuses to send the share in the last phase while others quit the protocol as they finish their task.

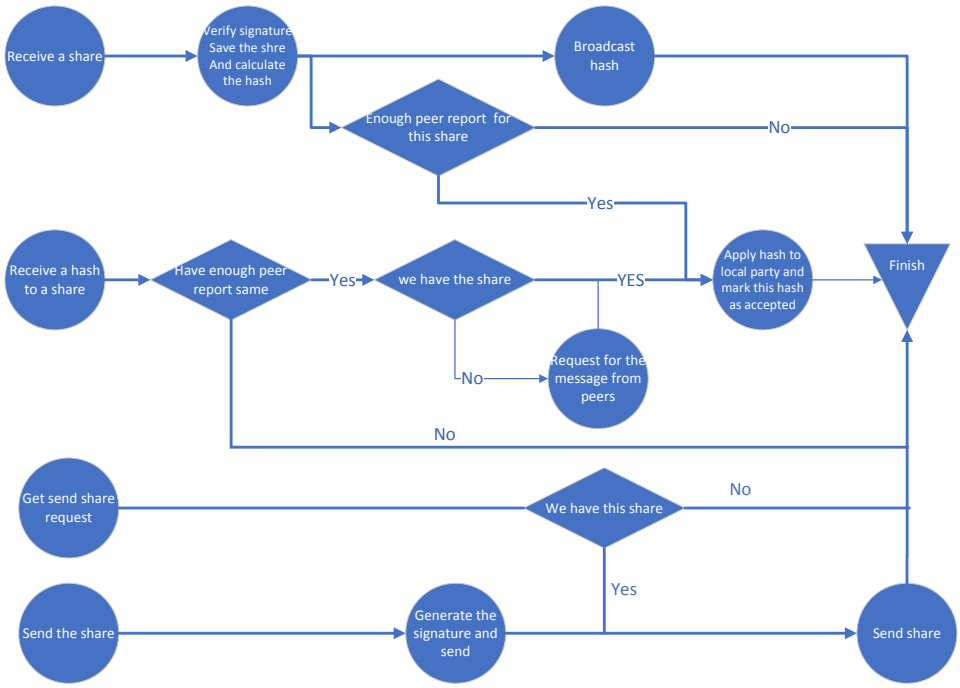

Figure 3: *THORChain TSS broadcast flow*

4.3 THORChain TSS Broadcast Flow

The flow chat of THORChain TSS reliable broadcast is shown in Fig. 3.

Three types of the messages are handled by the THORChain TSS. We explain the processing of the three types of the messages as follows:

- **Receiving a share:** When a receiver receives this share, it verifies the signature of the data to ensure it is from the data owner as it is claimed. The node then stores this share locally and calculate the hash of this message and broadcast to the peers. Lastly, this node check whether more than 2/3 peers report the hashes that match with this share which indicating more than 2/3 nodes in the network receive the same share. if this share

is the one received by the majority, this node accepts this share and applies to its local TSS.

- **Receiving a hash to a share:** When receiving a hash (with a tag which indicates the message owner and for the given TSS round), the node checks whether more than 2/3 nodes report the same hash, if not, it saves this hash and finish processing. Otherwise, it searches its' local storage to find whether it has this share to apply in its local TSS. If it does not have this share, it sends the request to peers to request for this share then applies it in its local TSS.
- **Forward the share request:** When a node receive the request for a given share, it checks its local storage to see whether it can forward the requested share.
- **Send TSS share:** As the share owner, it signs on the share and broadcasts it to all the peers.

THORChain TSS Attack Model

5.1 THORChain TSS Security Assumptions

For our THORChain TSS, we assume the following assumption holds.

- we assume that the third party libraries are secure and have no known security defects.
- We assume that the honest node collaborate with each other, and when TSS process fails, they report the culprits to the best of their knowledge.
- In a TSS keygen/keysign process, 2/3 participants are honest.
- The adversaries can collude with each other, while the honest node have no information exchange privately.

5.1.1 THORChain TSS Attack Discussion

To make the discussion concise, we take keysign as an example to show how THORChain TSS can work properly under the following attacks.

- **Request replay attack:** THORChain TSS employs the **session ID** to avoid the request replay attack. Once a request has been processed, this session ID becomes invalid.
- **THORChain TSS cache overflow attack:** In THORChain TSS, we have the following way to avoid the attack towards the storage and data cache. 1.) We only accept the messages from the parties that involved in a given TSS session. 2.) We only store the latest message from a peer. If a malicious peer keep sending the messages to fill the cache, THORChain TSS will overwrite the previous message it sent. 3.) Subscribe pattern ensures that we only accept the messages with the topic we subscribed and irrelevant message will be dropped.
- **Data integrity attack:** For this attack, the adversary send different shares to different peers, and try to compromise the peer(s). In THORChain TSS we stop this from happening by asking all the peers exchange the hash of the message they receive. For a node, it only accepts the shares whose hash have been confirmed by 2/3 of the total peers.
- **Single point of failure:** In this attack, the adversary tries to stuck the peers by sending incorrect share or reject to communicate with the peers. In THORChain TSS, we introduce the peer forwarding scheme which allows one peer to get the message from another peer. The original share owner need to sign on the share to avid any parties that tamper the share before forwarding.
- **Blame abuse attack:** Since we punish the node who do not cooperate in TSS process by disabling this node from attending the TSS in future. The profit driven nodes will comply with the THORChain TSS protocol. For this attack, the adversary try to fault the THORChain TSS process while free from the blame. As a result, the adversary can hijack the system to blame the nodes they dislike.

  THORChain TSS has the following protections to make this attack impossible.

  - Signing on the share enable the victim to blame the share owner by proving that the culprit indeed signed on the wrong share.

- Share forwarding scheme avoids the malicious node blaming the honest share owner by declaring that I fail the TSS because I have not received the share.
- Take the share that confirmed by 2/3 nodes avoid the malicious share owner partition the network as some nodes accept one share while other accepting the other. Since we assume 2/3 nodes are honest, if the share accepted by the majority is incorrect, the underlying TSS library will throw the error and the share owner itself is blamed.

Conclusion

THORChain TSS is a robust, secured and leader-less signature scheme which allows a number of signers to be involved in signing on a given message. End-to-End encryption ensures the security of the communication. Reliable broadcast reduces the risk of single point of failure. The blame scheme is introduced to punish the nodes who do not comply with the protocol. The attacker has a low motivation to attack the network as they face a high risk of losing their bond.

References

edd25519 Wiki (n.d.). accessed on May 8.
URL: *https://en.wikipedia.org/wiki/Curve25519*

Gennaro, R. & Goldfeder, S. (2018), Fast multiparty threshold ecdsa with fast trustless setup, *in* 'Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security', pp. 1179–1194.

Secp256k1-Bitcoin Wiki (n.d.). accessed on May 8.
URL: *https://en.bitcoin.it/wiki/Secp256k1*