WSTS Audit

Aril 2024

By CoinFabrik

v202311

Executive Summary

CoinFabrik was asked to audit the WSTS project. WSTS stands for Weighted Schnorr Threshold Signatures, a library for threshold signatures supporting weights with an application to Stacks block signing.

During this audit we found 2 critical-severity issues, 3 high-severity issues and several medium and minor-severity issues. Also, several enhancements were proposed.

Scope

The audited files are from the git repository located at `https://github.com/Trust-Machines/wsts`. The audit is based on commit `e98c6dd9432f871851a4b2fb25e568ee285f0428` (tag v8.0.0).

The scope for this audit includes and is limited to the following files:

- `src/common.rs`: various common constructions (polynomial commitments, nonces, shares, and Chaum-Pedersen proofs for Diffie-Hellman tuples)
- `src/compute.rs`: various protocol related computations (binding factors, challenge, lagrangian interpolation coefficients, polynomial evaluation, intermediate values for parties and aggregators, tagged hashing and signature tweaking)
- `src/errors.rs`: error enumeration definitions
- `src/lib.rs`: module declarations
- `src/main.rs`: example usage of base level library functions
- `src/net.rs`: state machine messages and communication interface
- `src/schnorr.rs`: Schnorr ID, challenge and verification
- `src/state_machine/coordinator/fire.rs`: FIRE metaprotocol coordinator state machine
- `src/state_machine/coordinator/frost.rs`: single FROST round coordinator state machine
- `src/state_machine/coordinator/mod.rs`: general state machine functionalities for coordinators
- `src/state_machine/mod.rs`: general state machine functionalities
- `src/state_machine/signer/mod.rs`: signer state machine
- `src/taproot.rs`: BIP-340 compliant Schnorr signature and proof
- `src/traits.rs`: Signer and Aggregator traits
- `src/util.rs`: Diffie-Hellman key exchange, encryption and decryption utilities
- `src/v1.rs`: FROST protocol implementation
- `src/v2.rs`: WSTS protocol implementation
- `src/vss.rs`: construction of a random polynomial with coefficients in Zq

No other files in this repository were audited. Its dependencies are assumed to work according to their documentation.

Methodology

CoinFabrik was provided with the source code, including automated tests, and a technical paper describing the protocol. Our auditors spent seven weeks auditing the source code provided, which includes understanding the cryptographic protocols and context of use, analyzing the boundaries of the expected behavior of the library, understanding the implementation by the development team (including dependencies beyond the scope to be audited) and identifying possible situations in which the code allows the caller to reach a state that exposes some vulnerability. Without being limited to them, the audit process included the following analyses.

- Protocol errors
- Cryptographic errors
- Arithmetic errors.
- Race conditions.
- Denial of service attacks.
- Missing or misused function qualifiers.
- Needlessly complex code and interactions.
- Poor or nonexistent error handling.
- Insufficient validation of the input parameters.
- Incorrect handling of cryptographic signatures.

Findings

In the following table we summarize the security issues we found in this audit. The severity classification criteria and the status meaning are explained below. This table does not include the enhancements we suggest to implement, which are described in a specific section after the security issues.

ID	Title	Severity	Status
CR-01	Missing Inbound Messages Authentication	Critical	Unresolved
CR-02	Authorization Logic Missing in Critical Functions	Critical	Unresolved
HI-01	Neglected Deletions Exposing Signers Unnecessarily	High	Unresolved

ID	Title	Severity	Status
HI-02	Improper Initialization Leading to Insecure Uses	High	Unresolved
HI-03	Insufficient Timeout Handling Could Lead to Denial of Service	High	Unresolved
ME-01	Untrusted Address Book Could Lead to Unavailability	Medium	Unresolved
ME-02	Duplicated Method Calls Could Lead to Denial of Service	Medium	Unresolved
MI-01	Sensitive Data Could Be Exposed	Minor	Unresolved
MI-02	Protocol Deviations Could Lead To Attacks	Minor	Unresolved
MI-03	Use of Message Expansion When Computing Challenge and Binding Values	Minor	Unresolved
MI-04	Undocumented Cryptographic Key Derivation	Minor	Unresolved
MI-05	Unavailability Through Faulty Error Handling	Minor	Unresolved
MI-06	Inconsistent Configuration	Minor	Unresolved
MI-07	Some States Do Not Timeout in FIRE	Minor	Unresolved
MI-08	Bad Randomness in Nonce Generation	Minor	Unresolved

Severity Classification

Security risks are classified as follows:

- **Critical:** These are issues that we manage to exploit. They compromise the system seriously. Blocking bugs are also included in this category. They must be fixed **immediately**.
- **High:** These refer to a vulnerability that, if exploited, could have a substantial impact, but requires a more extensive setup or effort compared to critical issues. These pose a significant risk and **demand immediate attention**.
- **Medium:** These are potentially exploitable issues. Even though we did not manage to exploit them or their impact is not clear, they might represent a security risk in the near future. We suggest fixing them **as soon as possible**.
- **Minor:** These issues represent problems that are relatively small or difficult to take advantage of, but might be exploited in combination with other issues. These kinds of issues do not block deployments in production environments. They should be taken into account and be fixed **when possible**.

Issues Status

An issue detected by this audit has one of the following statuses:

- **Unresolved**: The issue has not been resolved.
- **Acknowledged**: The issue remains in the code, but is a result of an intentional decision. The reported risk is accepted by the development team.
- **Resolved**: Adjusted program implementation to eliminate the risk.
- **Partially resolved**: Adjusted program implementation to eliminate part of the risk. The other part remains in the code, but is a result of an intentional decision.
- **Mitigated**: Implemented actions to minimize the impact or likelihood of the risk.

Critical Severity Issues

CR-01 Missing Inbound Messages Authentication

Classification:

- CWE-306: Missing Authentication for Critical Function

The methods `process_inbound_messages()` used by signer and coordinator do not authenticate the data received. So while signers and the coordinator sign the messages they send, the signature is not checked (neither by signers nor coordinator) within the library and the responsibility is left for the implementers. Note that the packet method `validate()` does not provide all the necessary guarantees for authentication as it will only

ensure that the packet is coming from another signer or coordinator depending on the message type (see in conjunction with CR-02 below).

Recommendation
Each method receiving inbound messages should include logic to decide whether the message is valid according to the message's signer. Alternatively, if the responsibility is left for the implementers then this should be documented so that implementers make informed choices over their code.

Status
Unresolved.

CR-02 Authorization Logic Missing in Critical Functions

Classification:

- CWE-285: Improper Authorization.

The coordinator and signer methods `process_inbound_mesages()` will receive messages from signers or coordinator, and pass down the messages to other methods depending on its current state and the message's type. However, this information does not suffice for proper authorization.

For example, in `frost.rs` at the `gather_public_shares()` method the coordinator inserts shares into `self.dkg_public_shares` for a `signer_id` without actually validating these shares came (i.e., were signed) by a signer underlying this `signer_id`. Hence the coordinator will add these shares for the `signer_id` and remove the `signer_id` from the `ids_to_wait` vector even if this signer never sent his shares[1].

Something analogous happens in other methods including but not limited to the following.

- `signer/mod.rs method dkg_public_share()` in L653 that receives a `dkg_public_shares` object and inserts them in the hashmap for `dkg_public_shares.signer_id` when the sender of this packet could be anyone..
- In `frost.rs` the method `coordinator.gather_private_shares()` L279 similarly accepts private shares without authenticating the sender.
- In `frost.rs` anyone can send a `DkgBegin` when the coordinator is Idle and set the round at any number.

Recommendation
Each method processing inbound messages must decide whether an authenticated message is to be authorized with logic that includes the message signer's identity.

[1] As a side note, proof of knowledge checks are not done by the coordinator, so that it could be the case that the public shares are invalid, ignored by signers, but have been inserted in the vector.

Status
Unresolved.

High Severity Issues

HI-01 Missing Deletions Exposing Signers Unnecessarily

Classification:

- CWE-213: Exposure of Sensitive Information Due to Incompatible Policies

The FROST protocol mandates that certain values are deleted after use to prevent replay attacks. This includes the proof of knowledge *R_i* and *mu_i* (included in the ID class in this implementation), the secret shares `f(j)` for all `j`, and the private and public nonces `e,d, E, D`. None of the values are deleted and therefore signers are unnecessarily exposed to protocol attacks.

Recommendation
Make sure to document what each method is doing and when a protocol security precaution is not taken. In this case, we suggest implementing deletions.

When talking with the development team, developers suggested that some use cases for this library may require not to delete these parameters. In that case, our suggestion is to provide an implementation which does all deletions as mandated, and also to allow library implementers to specifically configure any alternatives. For those cases, documentation describing the risk of these actions is a must have.

Status
Unresolved.

HI-02 Improper Initialization Leading to Insecure Uses

Classification:

- CWE-453: Insecure Default Variable Initialization

The implementation is liable to replay attacks because encryption/validation keys are permanent and round values repeat. For example, whenever a coordinator is created (e.g., `c=Coordinator.new(config)`) it will be instantiated with `c.dkg_id=0` and this value cannot be changed by configuration. This means that every time a new coordinator is instantiated, the `dkg_id` values repeat and hence, encrypted packets used in previous rounds can be reused (e.g., a round with `dkg_id=0` will happen many times, same with `dkg_id=1`, etc). This allows malicious users to reuse broadcasted/public values of earlier rounds at least causing unavailability and allows them to manipulate the protocol logic.

Recommendation

Make sure `dkg_id`, round number and session number values do not repeat, and that all packets include these values so malicious users cannot implement replay attacks simply by reusing older packets.

Status

Unresolved.

HI-03 Insufficient Timeout Handling Could Lead to Denial of Service

Classification:

- CWE-691: Insufficient Control Flow Management.

The FIRE (meta)protocol makes provisions to remove malicious actors and to restart rounds (or sessions) in certain instances, including for example, when time passess without protocol state changes. However, checks are insufficient and bad actors could stall the protocol ignoring these precautions.

For example, the `process_inbound_messages()` function in the FIRE coordinator seems to be susceptible to a DOS. The function includes a first block of code where input messages are processed, and a second block of code that processes results from the previous codeblock results and takes care of timeouts. If for some reason the stalling were to occur in the first block, the logic in this method could never exit and the stalling condition would persist.

One way to exploit this consists in a signer sending a huge amount of shares/nonces for the correct `dkg_id`. All of these messages will be processed and this may take more time than what the timeout allows, yet the time check will happen much later in the code causing fast-forwarding or failure of a DKG or signing round.

Recommendation

Make sure that each possible execution path is protected against timeouts. Also, include security tests that exercise these border cases and allow us to be sure that these attacks do not work. Ideally, staying inside the timeout windows required by protocol should be prioritized over message processing, so as to avoid the possibility of a coordinator's state being manipulated by signers.

Status

Unresolved.

Medium Severity Issues

ME-01 Untrusted Address Book Could Lead to Unavailability

Classification:

- CWE-345: Insufficient Verification of Data Authenticity

The library receives as part of its configuration sensitive information including maps linking `key_id values` and `key_ids` to public keys that are not validated. This is a very sensitive point, as an attacker controlling this address book could gain complete control over the signing process. Moreover, the config object is missing the coordinator' s public key, so it appears there is no way to specify this at signer creation (e.g., you will need to manually set the variable).

Recommendation

Include mechanisms allowing signers and the coordinator to validate the public keys address book and other sensitive information obtained from external sources.

Status

Unresolved.

ME-02 Duplicated Method Calls Could Lead to Denial of Service

Classification:

- CWE-400: Uncontrolled Resource Consumption.

Coordinator methods accepting signers shares and nonces, as well as signer methods accepting shares and nonces do not check if said values have already been received correctly. A malicious signer, for instance, could repeatedly send the same public shares before the coordinator is able to process the public shares from other signers and in this way put a high burden on the coordinator who could not possibly process all messages in time and therefore stall without moving to the next stage. Moreover, older values are overwritten and this could lead to signers losing track of what the valid values are.

Recommendation

When receiving a protocol value, make sure this value has not already been received.

Status

Unresolved.

Minor Severity Issues

MI-01 Sensitive Data Could Be Exposed Via Public Parameters

Classification:

- CWE-922: Insecure Storage of Sensitive Information

Signer and Coordinator classes use a structure where all the parameters are public and its access is not limited. In combination with another bug, this could lead to the exposure of sensitive information.

Recommendation

Limit the access to sensitive parameters so that they can only be accessed via the methods that require its use. The use of secret-handling crates for sensitive information that needs to be permanently stored, like private keys, is further recommended.

Status

Unresolved.

MI-02 Protocol Deviations Could Lead To Attacks

Classification:

- CWE-327: Use of a Broken or Risky Cryptographic Algorithm

In the FROST paper there are a few instances in which strings are concatenated and then hashed to be used as commitments. Sometimes, this implementation changes the order in which strings are concatenated. While this may probably have no security impact, we suggest following the paper for soundness and to prevent library collaborators from misusing these.

The order of R and aG is reversed with respect to the FROST paper in the calculation of c=Hash(ID...) and it is missing the common reference string. Again, while replay attacks may be prevented with the use of `dkg_id`, the impact of this seemingly small protocol deviation grows in view of Issue HI-02.

Again the order of Hash(R Y m) in this library is different from the FROST paper.

While this is most probably not an issue, it presents the implementers and others with questions that could be avoided.

Recommendation

As a general rule, either follow the paper's specification or provide a description to the users explaining the security impact of the change when deviation from protocol.

Status
Unresolved.

MI-03 Use of Message Expansion When Computing Challenge and Binding Values

Classification:

- CWE-327: Use of a Broken or Risky Cryptographic Algorithm.

The computation of the binding and challenge in the standard (RFC9380) are done using a specific message expansion to circumvent cryptographic vulnerabilities. Check https://www.rfc-editor.org/rfc/rfc9380.html#name-expand_message_xmd

Recommendation
Use the message expansion scheme from the standard.

Status
Unresolved.

MI-04 Undocumented Cryptographic Key Derivation

Classification:

- CWE-327: Use of a Broken or Risky Cryptographic Algorithm.

During the private shares step, all signers will compute a private share representing the evaluation of their polynomial `f(X)` at a scalar i and then encrypt it for another signer. This is done by first computing a shared secret using elliptic curve Diffie-Hellman and then using a custom key derivation scheme to derive a key to be used by the symmetric encryption scheme. However, this protocol is not documented and appears to be a custom creation. Since this key-exchange, key derivation and encryption protocol has not been analyzed, and other protocols using very similar elements have been analyzed and provide security guarantees, there is an unnecessary risk being taken that could be eliminated.

Recommendation
Pick a validated protocol and implement as little as possible, reusing libraries and other from well validated alternatives.

Status
Unresolved.

MI-05 Unavailability Through Faulty Error Handling

When a signer runs the method `dkg_private_shares()` upon receiving private share, the execution will run the line

```
    self.dkg_private_shares
        .insert(src_signer_id, dkg_private_shares.clone());
```

before attempting decryption so that if decryption fails, the hashmap will still contain the value.

Recommendation
Handle method errors so that changes to the maps for public shares, private shares, nonces and the ids_to_wait vectors are reverted in the presence of errors.

Status
Unresolved.

MI-06 Inconsistent Configuration

The Config class in coordinator/mod.rs method `new()` includes the following line

```
signer_public_keys: HashMap<u32, Point>,
```

However, later the in `net.rs` the object public keys is different:

```
pub struct PublicKeys {
   /// signer_id -> public key
   pub signers: HashMap<u32, ecdsa::PublicKey>,
   /// key_id -> public key
   pub key_ids: HashMap<u32, ecdsa::PublicKey>,
}
```

This inconsistency is not caught in tests (for example, of the Packet method `validate()` which would catch this error) but this could lead to problems for typical use cases.

Recommendation
Fix the inconsistency.

Status
Unresolved.

MI-07 Some States Do Not Timeout in FIRE

The FIRE implementation includes some states that do not include timeout precautions (`Idle, DkgPublicDistribute, DkgPrivateDistribute, DkgEndDistribute,`

`NonceRequest, SigShareRequest`). Signers could stall when the coordinator is in these provoking a protocol denial of service.

Recommendation
Consider documenting a careful timeout policy and making sure this is how the FIRE coordinator is implemented.

Status
Unresolved.

MI-08 Bad Randomness in Nonce Generation

There is an IETF standard for FROST which makes several relevant security suggestions. In particular to nonces, it hashes the random bytes to include an extra layer of security (here). However, in this library the nonce (e, d) is computed from two random bytestrings extracted from the EC wrapper (`Scalar::random(rng)` where rng represents OsRng) .

Recommendation
Follow the standard. Check EN-09 as well.

Status
Unresolved.

Enhancements

These items do not represent a security risk. They are best practices that we suggest implementing.

ID	Title	Status
EN-01	About Documentation	Not implemented
EN-02	Unavailable Mechanism for Changes	Not implemented
EN-03	Unwanted Comments	Not implemented
EN-04	Unnecessary Computations	Not implemented
EN-05	All Signers Unnecessary in Nonce Generation	Not implemented
EN-06	Variable Naming	Not implemented

ID	Title	Status
EN-07	Duplicated Operation	Not implemented
EN-08	Bad Randomness Source	Not implemented
EN-09	Testing	Not implemented

EN-01 About Documentation

In auditing this project we also went through some alternative FROST implementations. We found the ZF Frost Book that accompanies the Z-Cash implementation to be helpful and suggest writing an analogous text. References to the FROST and WSTS papers, or the IETF standard are also welcome.

Documentation for relevant use cases is missing. In particular, there is no clear indication into what the interfaces that the library exposes to users since all functions are public. The most relevant files and functions should include more detailed documentation.

We also find that examples would complement very well with the code and provide a concise guide to library users (developers).

Recommendation

Most important functions should be documented. Relevant use cases should be covered by the documentation. Also, the code makes use of very heavy notation that should be very well documented.

Status

Not implemented.

EN-02 Unavailable Mechanism for Changes

Signers and the coordinator do not include mechanisms for modifying parameters. For example, if at some point the coordinator where to change, a signer needs a mechanism to make this change effective. Asking for these changes to be done manually, thought the modification and reset of multiple parameters may lead to unexpected faults.

The same happens with the other parties in the address book, but also with parameters such as the threshold or number of keys, which could change during the executions of the protocol.

Recommendation
Include mechanisms for modifications that are likely to happen during the library's use, e.g., in the node use case.

Status
Not implemented.

EN-03 Unwanted Comments

Todo messages and "developer's comments" can be seen throughout the code. Please make sure that these are removed only leaving comments that provide documentation.

- // TODO. state_machine/signer/mod.rs L259, L294
- `// TODO: consider move private shares checks here`
- `// TODO: Once Frost V5 is released, this off by one adjustment will no longer be required` net.rs L451
- `// TODO: see if we have sufficient non-malicious signers to continue` fire.rs L657
- `frost.rs L515-516 fire.rs L944`

```
fn compute_aggregate_nonce(&self) -> Point {
    // XXX this needs to be key_ids for v1 and signer_ids for v2
```

- In src/compute.rs L74 the first comment

```
// Is this the best way to return these values?
#[allow(non_snake_case)]
/// Compute the intermediate values used in both the parties and the aggregator
pub fn intermediate(msg: &[u8], party_ids: &[u32], nonces: &[PublicNonce]) -> (Vec<Point>, Point) {
```

Recommendation
Remove these comments.

Status
Not implemented.

EN-04 Unnecessary Computations

Note that when `get_shares()` is called with the knowledge of what `key_id` values are participating in the round, it is only necessary to evaluate the polynomial (for the private shares) on those key_id values and no other. Moreover, there is no need to encrypt or

transmit these shares. Nor is there any use in following up with these IDs in the nonce generation/gathering and signing steps.

Recommendation
Make the computation, encryption and message sending of private shares only for signers participating in the protocol.

Status
Not implemented.

EN-05 All Signers Unnecessary In Nonce Generation

Similar to the above issue, the implementation of frost.rs method `request_nonces()` initializes ids_to_wait to all the available signers. Hence, later `gather_nonces()` requires (see L388) all parties to send nonces and will not advance until the ids_to_await vector is empty. This is unnecessary, and could be finished earlier. The set ALPHA (from the FROST paper) of all round signers should be defined, and only those signers should be expected to participate.

Recommendation
Allow for signers to drop off a signing round. Consider closing the nonce gathering stage on a coarser condition.

Status
Not implemented.

EN-06 Variable Naming

Some variable names, method names and states could mislead the developer. Consider replicating those names used in the FROST and WSTS papers and adding documentation. For example, calling files and schemes v1 and v2 on one side and FROST and WSTS on the other, Similarly, the names round and session on one side and current_sign_id, current_sign_iter_id (for fire.rs, they are named differently in singer/mod.rs)

Recommendation
Follow the notation from the papers when possible.

Status
Not implemented.

EN-07 Duplicated Operation

It appears that in signer/mod.rs dkg_private_shares it is doing the same multiplication twice, once in L 682 and then inside the run of L 683 make_shared_secret(). Instead use make_shared_secret_from_key(&shared_key) in L 683 saving the double mult

Recommendation
Reuse shared_key so that the multiplication happens only once.

Status
Not implemented.

EN-08 Suboptimal Randomness Source

The library uses the OsRng crate which makes use of the operating system randomness for all sources of randomness. This choice unnecessarily exposes users; for example, it happens to be insecure when other applications run in parallel with the library.

Recommendation
We recommend the use of `rand::thread_rng()` which is thread safe.

Status
Unresolved.

EN-09 Testing

Testing coverage is limited. In particular, there are no end-to-end tests which would both serve to test the library and as documentation. We found that it was difficult to construct tests for border cases, like signers spending too much time to send shares/nonces or malicious messages and this implies that these situations are difficult to test.

Recommendation
Add tests reflecting common use cases.

Status
Unresolved.

Changelog

- 2024-04-12 – Initial report based on commit `e98c6dd9432f871851a4b2fb25e568ee285f0428`.

WSTS Project Audit (extras)
April 2024

Observations

Find here some additional comments which may be of interest to the development team but may not be of interest to the more general public.

- We found the optimization of WSTS over FROST interesting and encourage the author to publish and share this with the FROST authors.
- The message broadcasting and reception for the library relies on the undocumented application layer. For example, it is assumed that the coordinator will not get ahead of signers so that, e.g., signers will be asked to send private shares before they receive the public shares. Other implementations using the library may not guarantee this timing property and messages could be lost. Synchronization requirements should be documented.
- The method `compute::intermediate()`, assumes that nonces are ordered according to singer_id increasing, but are actually stored in this vector in the same order they are received, which might not be according to `signer_id`. When the order changes, the computation of rho[i] will be incorrect and the code will produce bad signatures. We suggest storing and using the object as (party_id, (D,E)) following the FROST paper.
- Config checks. Improve checks at the configuration level. For example, having threshold = 0 is allowed and will exit with overflow.
- Tests in signer subfolder
    - 1. All tests involved use a configuration of a single signer, which conceptually defeats the purpose of the application.
    - 2. dkg_publish_share exposes that the Signer entity is not holding responsability for checking the instantiated signers id falls within the range of valid ids for the session. More so reading the implementation we find that no checks are done when receiving a DkgPublicShares object, thus allowing to populate its `dkg_public_shares` map with arbitrary values.
- The wrapper for elliptic curve crypto has not been audited. The extra code is heavy. Please consider having this code audited.
- Coding error in
```
let dkg_begin = DkgEndBegin {
    dkg_id: self.current_dkg_id,
    key_ids: (0..self.config.num_keys).collect(),
    signer_ids: (0..self.config.num_signers).collect(),
```
Inside start_dkg_end() for frost.rs It should be key_ids=1..num_keys+1.. However, this is not used later.

- There are mismatches in how Coordinator and Signer are initialized. For example, fire.rs L1033:L1041

```
impl<Aggregator: AggregatorTrait> CoordinatorTrait for
Coordinator<Aggregator> {
   /// Create a new coordinator
   fn new(config: Config) -> Self {
       Self {
           aggregator: Aggregator::new(config.num_keys,
config.threshold),
           config,
           current_dkg_id: 0,
           current_sign_id: 0,
           current_sign_iter_id: 0,
```

And signer.rs L137:L140

```
       Signer {
           dkg_id: 0,
           sign_id: 1,
           sign_iter_id: 1,
```

consider reviewing them so that they are consistent.