PUBLIC

# Limited Code Review

of the SSV-DKG Tool

Nov 20th, 2024

Produced for

ssv.network

by

CHAINSECURITY

# Contents

1 Executive Summary 3
2 Review Overview 5
3 Limitations and use of report 8
4 Terminology 9
5 Findings 10
6 Resolved Findings 11
7 Informational 15
8 Notes 17

# 1 Executive Summary

Dear SSV Team,

Thank you for trusting us to help SSV Labs with this security review. Our executive summary provides an overview of subjects covered in our review of the latest reviewed code of SSV-DKG according to Scope to support you in forming an opinion on their security risks. The review was executed by 2 engineers over a period of 3 weeks. It's important to note that, due to the extensive scope and codebase, our time-limited review does not capture the full depth of a comprehensive security analysis.

SSV Labs implements a distributed key generation tool to enable the creation of threshold keypairs for Ethereum validators.

The most critical subjects covered in our review are protocol correctness and network security. Security regarding protocol correctness is high. Network security is improvable, see Insecure TLS Default Configuration.

The general subjects covered are behavior in the presence of malicious nodes and denial-of-service vectors. Security regarding all the aforementioned subjects is good. Functionality issues may arise in the presence of malicious nodes, see Crash by Malicious Operator and Ignored DKG Phases. Plausible denial-of-service vectors have been found, see Denial of Service via Spam.

In summary, we find that the codebase provides a good level of security.

It is important to note that security reviews are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

## 1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

| | |
|---|---|
| Critical-Severity Findings | 0 |
| High-Severity Findings | 3 |
| • Code Corrected | 3 |
| Medium-Severity Findings | 3 |
| • Code Corrected | 2 |
| • Risk Accepted | 1 |
| Low-Severity Findings | 2 |
| • Code Corrected | 1 |
| • Specification Changed | 1 |

# 2 Review Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

## 2.1 Scope

This review was not conducted as an exhaustive search, but rather as a best-effort sanity check. It was performed on the source code files inside the SSV-DKG repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

**dkg**

| V | Date | Commit Hash | Note |
|---|---|---|---|
| 1 | 14 Oct 2024 | b24742520a59ad045f535e9049d542e6c8625717 | Initial Version |
| 2 | 18 Nov 2024 | bf4be233a622b9705c6ec45a383624b9f3fc145f | Fixes |
| 3 | 19 Nov 2024 | 3cf2e91e4388e561b17bab7d84ed14565c0abdc9 | Fixes, Final Version |

**dkg-spec**

| V | Date | Commit Hash | Note |
|---|---|---|---|
| 1 | 14 Oct 2024 | 40e39a71b6bd35e332c51dfe714c055a5fbc9050 | Initial Version |
| 2 | 18 Nov 2024 | 2a2ca086d5296d585d858129a8b95a15687e837e | Fixes |
| 3 | 19 Nov 2024 | f956634b520b220d298ae6ddd0d0fe5e2401dab4 | Fixes, Final Version |

For the Go modules, the compiler version `1.23.0` was chosen.

In dkg, the following directories were in scope (excluding test files)

- `cli/`
- `cmd/`
- `pkgs/`

In `dkg-spec`, only the following files were in-scope:

- `result.go`
- `resign.go`
- `rehare.go`
- `proof.go`
- `init.go`
- `crypto/` (all non-test files)
- `eip1271/common.go` (just magic values)

### 2.1.1 Excluded from scope

The DKG implementation and Kyber library are out of scope. The Ethereum validator is out of scope.

## 2.2 System Overview

This system overview describes the initially received version (Version 1) of the tool as defined in the Review Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

SSV Labs provides a tool that implements a distributed key generation (DKG) protocol in order to let multiple cooperating node operators produce threshold keypairs for distributed Ethereum validators.

In the absence of a DKG tool, the owner of the validator locally generates a random polynomial $f(x)$ over a prime field. The shared private key is defined to be $s = f(0)$. They then send shares of the form $(i, y)$, where $i > 0$ is a unique index, and $y = f(i)$ to operators. Using threshold cryptography, operators are then able to cooperate to compute the public key as well as sign Ethereum attestations without ever reconstructing the private key. The degree of the polynomial $d$ determines the threshold of validators $t = d + 1$ that must agree to a signature.

A DKG protocol aims to produce a threshold keypair as the previously described process does, but without ever reconstructing the private key in one location. This requires multiple rounds of communication between operators.

In SSV-DKG, nodes play two distinct roles: operators participate in the DKG protocol with the aim of receiving a share of the final private key, while an *initiator* relays messages between operators as well as from the validator owner to the operators. The owner trusts the initiator to run the protocol honestly. They also trust that at most $t - 1$ of their chosen operators are malicious.

Operators are registered in a central data repository. For DKG purposes, it provides their unique integer ID, Internet HTTPS endpoint, and RSA public key. Operators run the tool as an HTTPS server with a valid certificate and listen for connections. Initiators act as clients connect to the operators.

The owner is represented by their Ethereum address. They can sign using standard ECDSA signatures, or EIP-1271 if the address is a smart contract.

We highlight that operators do not keep state in-between active DKG runs. Instead they produce signed *proofs* attesting to the DKG detail and containing encrypted shares. The initiator is expected to present the proofs to the operators during subsequent operations.

### 2.2.1 Ping

To check if certain given operators are available, an initiator can perform a GET request on the "health check" endpoint. Live and honest operators will reply with their ID, public key, whether they support EIP-1271 signatures, and whether their Ethereum node appears to be working.

### 2.2.2 Init

To generate a distributed key, the initiator first sends an INIT message signed by itself. (not by the owner) It contains the list of operators, the threshold, the beacon chain withdraw credentials, the beacon chain fork id, the address of the owner, a nonce, and the deposit amount in wei. Operators respond with a DKG exchange message signed with their RSA key. It contains an ECIES public key. The initiator collects all exchanges and broadcasts them to every operator. Operators respond with deal messages that contain shares encrypted with the ECIES keys of other operators. The initiator collects all deals and broadcasts them to every operator. Finally, each operator responds with a partial signature of the beacon chain deposit data, a partial signature of the owner and nonce, and an RSA-signed proof, which consists of the full validator public key, the RSA encrypted key share, the public key of the share, and the owner address. Both the initiator and the operators save the proofs to disk.

### 2.2.3 Resign

The resign process allows the owner of the validator to request that operator sign a different beacon chain deposit object and owner-nonce. This is useful if there was a mistake in the first set of deposit parameters. The owner is authenticated by a signature coming from their ethereum address: either recoverable ECDSA if it is an EOA, or EIP-1271 if it is a smart contract. The initiator relays one or more signed request to the operators and collects the partial signatures, It then saves them to disk.

In **Version 2**, the owner can also change their address using by running a resign.

### 2.2.4 Reshare

During the reshare process, the owner requests that the operators share their existing key shares in such a way that a new set of operators can operate the same existing validator. The protocol is broadly similar to the one performed for Init. However, like Resign, multiple instances can be set up batched and perfomed sequentially.

# 3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

# 4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

| Likelihood | Impact | | |
|---|---|---|---|
| | High | Medium | Low |
| High | Critical | High | Medium |
| Medium | High | Medium | Low |
| Low | Medium | Low | Low |

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

# 5 Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

- Security: Related to vulnerabilities that could be exploited by malicious actors
- Design: Architectural shortcomings and design inefficiencies
- Correctness: Mismatches between specification and implementation

Below we provide a numerical overview of the identified findings, split up by their severity.

| | |
|---|---|
| Critical-Severity Findings | 0 |
| High-Severity Findings | 0 |
| Medium-Severity Findings | 1 |

- Use of Discouraged Asymmetric Encryption Primitive Risk Accepted

| | |
|---|---|
| Low-Severity Findings | 0 |

## 5.1 Use of Discouraged Asymmetric Encryption Primitive

Security Medium Version 1 Risk Accepted

*CS-SSVDKG-006*

Secret shares of the validator key are protected using PKCS#1 v1.5 encryption. This scheme is vulnerable to Bleichenbacher's attack, which can allow the attacker to decrypt the shares.

In practice, the attack requires that the node operator expose an interactive padding oracle. ChainSecurity does not believe that such an oracle is exposed in-scope.

# 6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

| | |
|---|---|
| Critical-Severity Findings | 0 |

| | |
|---|---|
| High-Severity Findings | 3 |

- Denial of Service With Malicious Initiator Code Corrected
- Insecure TLS Default Configuration Code Corrected
- Missing Signature Check Code Corrected

| | |
|---|---|
| Medium-Severity Findings | 2 |

- Crash by Malicious Operator Code Corrected
- Incorrect EIP-1271 Magic Value Code Corrected

| | |
|---|---|
| Low-Severity Findings | 2 |

- Ignored DKG Phases Specification Changed
- Reshare Denial-of-Service via Predicable Instance IDs Code Corrected

| | |
|---|---|
| Informational Findings | 2 |

- Arbitrary Path Access Code Corrected
- Use of Potentially Vulnerable Package Code Corrected

## 6.1 Denial of Service With Malicious Initiator

Security High Version 1 Code Corrected

*CS-SSVDKG-015*

In `InitInstance`, any INIT message with a valid signature causes an entry to be added to the `s.Instances` array. If that array contains 1024 entries not older than 5 minutes, no new DKG can be initiated. Anyone with network access can create valid INIT messages, easily denying service to the legitimate initiator.

**Code corrected:**

Client has decreased the time-to-live to 1 minute and increased the amount of entries to 102400.

## 6.2 Insecure TLS Default Configuration

Security High Version 1 Code Corrected

*CS-SSVDKG-001*

In `pkgs/initiator/initiator.go`, the following lines are used to set TLS parameters:

```
if len(certs) > 0 {
        client.SetRootCertsFromFile(certs...)
} else {
        client.SetTLSClientConfig(&tls.Config{InsecureSkipVerify: true})
}
```

This means that if the user passes any certificates on the command-line, the TLS library is configured to use them as only roots of trust, to the exclusion of the default root certificates. If they do not pass any certificates, the library is configured to accept any certificate presented by the server and thus allow a man-in-the-middle attack to occur. Thus the default setting, which users are likely to use, is highly insecure in production.

In System Overview, we assert that the system trusts the "official" certificate authorities, in part because the web application is served over HTTPS. Therefore, it is surprising that the DKG tool doesn't do the same by default.

**Code corrected:**

In Version 2, the `InsecureSkipVerify` flag is no longer set by default, it must be set explicitly.

## 6.3 Missing Signature Check

Correctness High Version 1 Code Corrected

*CS-SSVDKG-002*

At line 282 of `pkgs/initiator/initiator.go`, `verifyMessageSignatures()` is called on the `kyberMsgs` variable. However, those messages have already been verified at line 264. The context suggests that the intent was to check the signatures on `dkgResult`, which are currently never verified.

**Code corrected:**

In Version 2, signatures on both sets of messages are verified once.

## 6.4 Crash by Malicious Operator

Security Medium Version 1 Code Corrected

*CS-SSVDKG-003*

When processing a message of type `ReshareExchangeMessageType` from another operator relayed by the initiator, the operator dereferences the `DKGData.reshare` field to access the list of new operators. These types of messages are only expected during reshare DKG and not during the initial DKG. As a consequence, a malicious operator could intentionally submit an reshare message. This would cause other operators to dereference a nil pointer and crash.

**Code corrected:**

In Version 2, the request terminates with an error instead of crashing.

## 6.5 Incorrect EIP-1271 Magic Value

Correctness Medium Version 1 Code Corrected

*CS-SSVDKG-004*

In `dkg-spec/eip1271/common.go`, the magic return value is defined as `[4]byte{16, 26, 0xba, 0x7e}`. Note that the first two bytes are expressed in decimal notation. In the EIP-1271 standard, the value is `0x1626ba7e`, which means that the first and second bytes are set to decimal 22 and 38 respectively. This will cause EIP-1271 verification to reject valid signatures.

**Code corrected:**

In (Version 2), the constant has been fixed.

## 6.6 Ignored DKG Phases

Design Low Version 1 Specification Changed

*CS-SSVDKG-009*

In the presence of faulty participants, DKG protocols can take additional rounds (complaint, justification) to terminate. The tool does not take these rounds into account and instead expect termination after a fixed number of rounds. If the DKG protocol does not terminate early, the tool might behave unexpectedly.

**Code corrected:**

SSV Labs has clarified that if a node misbehaves, the client should abort and does not need to handle the additional phases. In (Version 2), the initiator outputs more precise errors when it detects that a justification round has been initiated.

## 6.7 Reshare Denial-of-Service via Predicable Instance IDs

Security Low Version 1 Code Corrected

*CS-SSVDKG-010*

A reshare message can initiate multiple DKG instances. The request IDs for those DKG instances are chosen deterministically by hashing the reshare message. Then, the DKG instances are performed by the initiator in sequence. Given that instances become stale after one minute, it is possible to evict later reshare instances from the buffer by reusing their instance ID. This will prevent the legitimate initiator from finishing.

The same applies to the resign message flow.

**Code corrected:**

In (Version 2), the instance ID is chosen by combining the fresh random request ID of the outer reshare message with the hash of the current inner reshare message. This means that knowing the reshare message is no longer sufficient to perform a targeted DoS. In (Version 3), the instance ID is deterministically computed from the initiator public key, the hash of the instance, and the request ID. Thus the attack is no longer possible without knowing the initiator public key.

## 6.8 Arbitrary Path Access

Informational Version 1 Code Corrected

*CS-SSVDKG-011*

In `cli/utils/utils.go` at line 260, the tool defends against path traversal. However, `OutputPath` could be an absolute path, such as `/etc/passwd`, defeating the defense.

Severity is informational since in our model, the configuration values are not controlled by an adversary.

This also applies to `ConfigPath`, `LogFilePath`, `OperatorsInfoPath`, elements of `ClientCACertPath`, `ProofsFilePath`, `KeystorePath`, `ServerTLSCertPath`, `ServerTLSKeyPath`, and `CeremonyDir`.

**Code corrected:**

In Version 2, it is enforced that the paths cannot escape the current directory thanks to the `"path/filepath".IsLocal()` predicate.

## 6.9 Use of Potentially Vulnerable Package

Informational Version 1 Code Corrected

*CS-SSVDKG-014*

In `dkg/drand.go` at line 19, the package `github.com/drand/kyber/sign/bls` is imported. The scheme implemented by this package is known to be vulnerable to rogue key attacks. DEDIS recommend `github.com/drand/kyber/sign/bdn` instead.

As far as we can tell, SSV-DKG does not rely on the part of the scheme that is vulnerable to rogue key attacks. Nevertheless, the `bdn` package is backwards compatible to `bls` except for the problematic part, meaning that it could be substituted in place.

**Code corrected:**

In Version 2, the `bdn` package is used.

# 7 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

## 7.1 DKG Can Produce Full Validator Private Key

**Informational** **Version 1**

*CS-SSVDKG-007*

In secret-sharing schemes based on Shamir Secret Sharing such as Drand's DKG, the secret share at index 0 is conventionally the private key. This means that, when distributing secret shares, one must choose non-zero indices. A share with index 0 is in a sense a "backdoor" share that reveals the full private key.

In Drand Kyber 1.1.18, the backdoor share has index -1. In `computeDKGResult()`, the `uint32` share index is cast to a platform-sized `int`. As a result, on a 32-bit system, an index of `2**32 -1` can access the backdoor share. If every dealer were to use a version of SSV-DKG compiled for 32-bit, they would leave the operator with that index in possession of the full private key.

In SSV-DKG, indices are computed by subtracting one from the operator IDs, which is always 64 bits and assigned sequentially by the SSV smart contract, starting at one. Thus, this issue cannot occur in a realistic setting since more than 4 billion operators would need to be registered.

ChainSecurity has reported this issue to Drand so that the root cause can be addressed.

## 7.2 Unnecessary Use of BLS12-381 Curve

**Informational** **Version 1**

*CS-SSVDKG-013*

In `pkgs/dkg/drand`, ECIES is instantiated using the BLS12-381. While this is perfectly functional and secure, ECIES does not require the pairing functionality carried by BLS and is usually deployed with pairing-unfriendly curves that are more performant in this use case. For instance, Kyber supports ECIES with Curve25519.

SSV Labs noted that:

> Kyber supports only same schemas for VSS and Auth messages: if we use BLS12-381 for VSS and edwards25519/Curve25519 for ECIES, then we get an error at this place https://github.com/drand/kyber/blob/master/share/dkg/dkg.go#L220
>
> Based on that, we can't use different schemes for VSS and ECIES, so we continue to use BLS12-381 despite its slower

## 7.3 Misleading Function Prototype

**Informational** **Version 1** **Acknowledged**

*CS-SSVDKG-012*

In the package `github.com/ssvlabs/ssv/utils/rsaencryption`, the following function is defined:

```
// DecodeKey with secret key (rsa) and hash (base64), return the decrypted key
func DecodeKey(sk *rsa.PrivateKey, hash []byte) ([]byte, error) {
	decryptedKey, err := rsa.DecryptPKCS1v15(rand.Reader, sk, hash)
	if err != nil {
		return nil, errors.Wrap(err, "could not decrypt key")
	}
	return decryptedKey, nil
}
```

Rather than simply a decoding function, it is a decryption function. Its `hash` argument is not a hash but a ciphertext. This is confusing for readers, who have to check the implementation of the function to find out that it is in fact performing PKCS1v1.5 decryption.

SSV Labs has opened an issue to fix this.

# 8 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

## 8.1 Consequences of Picking Malicious Node Operators

Note Version 1

As highlighted in System Overview, when choosing operators with the help of the web application, the initiator must not choose a majority of malicious operators. Ideally, they should assess which operators are reputable/trustworthy in their opinion.

We want to expand on the risks that can materialize if the majority of operators in a given distributed validator are malicious. Without EIP-7002, there is no way to initiate a beacon chain withdrawal without using the validator key. Thus, the malicious operators can hold the initiator's funds hostage indefinitely. They can also intentionally slash themselves, affecting the initiator's fund.

## 8.2 Denial of Service via Spam

Note Version 1

*CS-SSVDKG-008*

Requesting a DKG ceremony with an INIT message is unpermissioned. Anyone who is willing to run the initiator client can coordinate ceremonies without necessarily holding any ETH. While this is by design, it can also be used to spam the operators. The cost to run a honest but useless DKG is not so high compared to the computational load incurred by the operators as a result.

Replay of DKG Reshare could also be used to run useless reshare DKGs.

In Version 2, IP-based rate limits have been added on the `init`, `reshare`, and `resign` routes to mitigate the DoS vector. This does not completely prevent spam, but makes it more difficult. SSV Labs has stated that this is sufficient for their purposes.

More precise metrics would need to be collected to determine how this compares to existing DoS vector and thus whether it could be an issue in practice.

## 8.3 Lack of Forward Security

Note Version 1

After a DKG, operators encrypt their key share with their long-term RSA key and store it in the ceremony proof. This means that, by design, if enough long-term keys are compromised at any point, the master key can be recovered from protocol transcripts.

DKG systems often feature proactive key rotation, whereby a DKG reshare is performed so that key shares are fully rotated and future break-ins do not result in compromise. The current system does not achieve this property due to the presence of encrypted key shares in the transcripts.

In addition, the owner change mechanism introduced in Version 2 does not invalidate the old owner address, which means that if it is later compromised, it can be used to recover the validator key using a reshare.

## 8.4 Replay of DKG Reshare

Note Version 1

*CS-SSVDKG-005*

Reshare messages signed by the owner do not include the initiator public key. As a result, anyone in possession of a valid reshare message can act as an initiator for an existing shared keypair. Since honest operators do not keep state, the DKG parameters can be replayed anytime and repeatedly, with valid and valuable keyshares as input.

When this is done, the initiator can behave arbitrarily. In particular, the assumption that a broadcast channel can be used by DKG participants is no longer valid since the initiator mediates all communications. ChainSecurity was not able to study the behavior of DKG protocols under these conditions. However, it appears unlikely that these conditions lead to key disclosure.

We note that "forget-and-forgive"-style attacks are not applicable due to the operators' statelessness.