KUDELSKI SECURITY

Audit of Timelock Encryption

Protocol Labs

28 March 2023

Version: 1.1

Presented by:
Kudelski Security Research Team
Kudelski Security - Nagravision Sàrl

Corporate Headquarters
Route de Genève, 22-24
1033 Cheseaux-sur-Lausanne
Switzerland

Public

TABLE OF CONTENTS

1 EXECUTIVE SUMMARY

Kudelski Security ("Kudelski", "we"), the cybersecurity division of the Kudelski Group, was engaged by Protocol Labs ("the Client") to conduct an external security assessment in the form of a code audit of the cryptographic library Timelock Encryption ("the Product"). The assessment was conducted remotely by the Kudelski Security Team and coordinated by Sylvain Pelissier, Cryptography Expert, Antonio De La Piedra, Senior Cybersecurity Engineer and Nathan Hamiel, Senior Director of Research. The audit took place from November 12, 2022 to November 30, 2022 and involved 15 person-days of work. The audit focused on the following objectives:

- To provide a professional opinion on the maturity, adequacy, and efficiency of the software solution in exam.
- To check compliance with existing standards.
- To identify potential security or interoperability issues and include improvement recommendations based on the result of our analysis.

This report summarizes the analysis performed and findings. It also contains detailed descriptions of the discovered vulnerabilities and recommendations for remediation.

1.1 Engagement Scope

The scope of the audit was a code audit of the Product written in Go and Typescript, with a particular attention to safe implementation of hashing, randomness generation, protocol verification, and potential for misuse and leakage of secrets. The target of the audit was the cryptographic code located in the following repositories:

- https://github.com/drand/tlock/: Implementation in Go of the CLI tool to perform timelock encryption. The audit was on the code up to commit number: `80219e458290ff663dd4adc4976019495f09aa56`.
- https://github.com/drand/tlock-js/: Implementation in TypeScript of timelock encryption and age encryption. The audit was on the code up to commit number: `80ad76434ae9959d5feba7e3b5913296f5aca8df`.
- https://github.com/drand/timevault/: Time vault for encryption of vulnerability reports. The audit was on the code up to commit number: `1ed1c5db1f74b34b683d23cc0a12979178e9aeed`.

The project included the audit of some parts of the dependencies utilized by Timelock

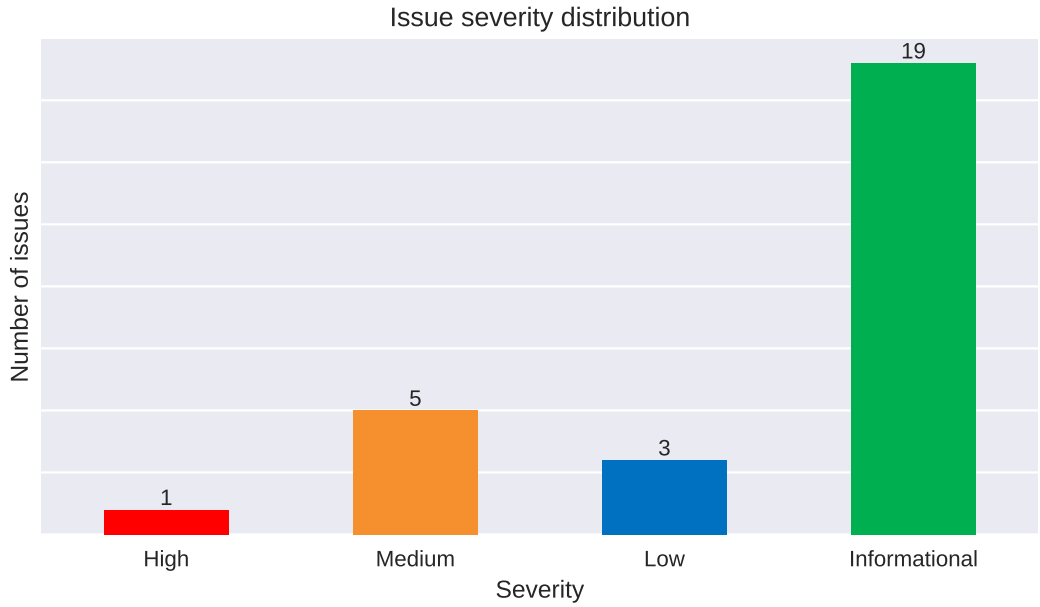
Encryption:

- stablelib/chacha20poly1305 commit `a89a438fcbf855de6b2e9faa2630f03c3f3b3a54`
- stablelib/chacha commit `a89a438fcbf855de6b2e9faa2630f03c3f3b3a54`
- noble/bls12-381 commit `61d36e0f134fc1f8b20db939f14584b3edcadb3a`
- kilic/bls12-381 commit `ca162e8a70f456f4cf733097edfd60d0e9deca2c`
- github.com/FiloSottile/age commit `bf8d2a3911c4305f83118ce7a71c9350949c9939`
  - age.go
  - internal/stream/stream.go
  - parse.go
  - primitives.go
- github.com/drand/kyber commit `dd9751718b1e7d963fdac4d647efc37523a0101c`

1.2 Engagement Analysis

The engagement consisted of a ramp-up phase where the necessary documentation about the technological standards and design of the solution in exam was acquired, followed by a manual inspection of the code provided by the Client and the drafting of this project.

As a result of our work, we have identified **1 High**, **5 Medium**, **3 Low** and **19 Informational** findings.



1.3 Issue Summary List

The following security issues were found:

ID	Severity	Finding	Status
KS-SBCF-F-01	**High**	tlock: Ciphertexts can be decrypted before the chosen date when using time sub-units	**Remediated**
KS-SBCF-F-02	**Medium**	tlock: Encryption in the future wrap to round 1	**Remediated**
KS-SBCF-F-03	**Medium**	tlock-js: Stream cipher encryption nonce reuse	**Remediated**
KS-SBCF-F-04	**Medium**	timevault: Missing web site security header	**Remediated**
KS-SBCF-F-05	**Medium**	kyber: Design flaws in hash to field function	**Remediated**
KS-SBCF-F-06	**Medium**	tlock-js: Design flaws in hash to field function	**Remediated**
KS-SBCF-F-07	**Low**	tlock: Timelock encryption decrypt to files with permissive permission	**Remediated**
KS-SBCF-F-08	**Low**	tlock: No point sanitization in TimeLock and TimeUnlock functions	**Remediated**
KS-SBCF-F-09	**Low**	tlock: tlock dependency tree contains 3 dependencies with vulnerabilities	**Remediated**

The following are observations related to general design and improvements:

ID	Severity	Finding
KS-SBCF-O-01	**Informational**	Missing security policy
KS-SBCF-O-02	**Informational**	tlock: envconfig.Process output is not processed
KS-SBCF-O-03	**Informational**	tlock: Multiple defer calls with Close()
KS-SBCF-O-04	**Informational**	Error in drand documentation
KS-SBCF-O-05	**Informational**	tlock: Unused return value
KS-SBCF-O-06	**Informational**	tlock: Incompatible flags not detected
KS-SBCF-O-07	**Informational**	tlock-js: Special number values NaN and Infinity are not checked
KS-SBCF-O-08	**Informational**	tlock-js: Missing check for malformed recipients

ID	Severity	Finding
KS-SBCF-O-09	**Informational**	tlock-js: Missing check for trailing space after armor footer
KS-SBCF-O-10	**Informational**	tlock-js: Missing check for ciphertext length in Decrypt
KS-SBCF-O-DEP-01	**Informational**	noble-bls12-381: Modular exponentiation is not constant-time
KS-SBCF-O-DEP-02	**Informational**	noble-bls12-381: Field exponentiation is not constant-time
KS-SBCF-O-DEP-03	**Informational**	noble-bls12-381: Speed-up in point arithmetic
KS-SBCF-O-DEP-04	**Informational**	noble-bls12-381: Speed-up in scalar multiplication
KS-SBCF-O-DEP-05	**Informational**	noble-bls12-381: hash-to-curve reference implementation is not updated to last version (16)
KS-SBCF-O-DEP-06	**Informational**	kyber-bls12381: Speed-up on subgroup membership
KS-SBCF-O-DEP-07	**Informational**	kilic/bls12-381: Arithmetic methods for groups never check that the input parameters are on the curve and in the right subgroup
KS-SBCF-O-DEP-08	**Informational**	kilic/bls12-381: The field elements and scalar field elements comparison are not constant-time
KS-SBCF-O-DEP-09	**Informational**	kilic/bls12-381: Exponentiation functions and inversion are not constant-time

2 TECHNICAL DETAILS OF SECURITY FINDINGS

This section contains the technical details of our findings as well as recommendations for mitigation.

2.1 KS-SBCF-F-01: tlock: Ciphertexts can be decrypted before the chosen date when using time sub-units

Severity: High

Status: Remediated

Location: cmd/tle/encrypt.go:57

Description

When using different combination of units for decryption time in the `tle` command line tool, some units are ignored. This means that plaintexts can be accessed before the user knows. The problem is that the `parseDuration` function in encrypt.go:57 ignores certain sub-units without warning the user.

The `tle` command line tool allows the user to enter the interval of time that will make the decryption of the ciphertext effective. This is done via `tle`, using the `-D` or `--duration` flag, which supports (via `--help`):

```
DURATION has a default value of 120d. When it is specified, it
 ↪  expects a number
followed by one of these units: "ns", "us" (or "µs"), "ms", "s",
 ↪  "m", "h", "d", "M", "y").
```

The Client uses `time.ParseDuration` to process a small amount of the values above via:

```
func parseDuration(t time.Time, duration string) (time.Duration,
 ↪  error) {
    d, err := time.ParseDuration(duration)
    if err == nil {
        return d, nil
    }
```

For instance:

```
./tle -D 1s20ms2ns100us -n="http://pl-us.testnet.drand.sh/"
↪  -c="7672797f548f3f4748ac4bf3352fc6c6b6468c9ad40ad456a397545c6
↪  e2df5bf" -o=ciphertext.txt
↪  plaintext.txt
```

However values like days, months and years are processed below, manually e.g. for days and months:

```
    now := time.Now()

    pieces := strings.Split(duration, "d")
    if len(pieces) == 2 {
        days, err := strconv.Atoi(pieces[0])
        if err != nil {
            return time.Second, fmt.Errorf("parse day duration:
            ↪  %w", err)
        }
        diff := now.AddDate(0, 0, days).Sub(now)
        return diff, nil
    }

    pieces = strings.Split(duration, "M")
    if len(pieces) == 2 {
        months, err := strconv.Atoi(pieces[0])
        if err != nil {
            return time.Second, fmt.Errorf("parse month duration:
            ↪  %w", err)
        }
        diff := now.AddDate(0, months, 0).Sub(now)
        return diff, nil
    }
```

This means, that if the user decides to add a complex combination of units involving either days, months of years, and then, add sub-units, some of these units will be ignored.

For instance, the user decides to publish the decryption of a plaintext after one year and 15 hours via:

```
./tle -D 1y15h -n="http://pl-us.testnet.drand.sh/" -c="7672797f54
↪ 8f3f4748ac4bf3352fc6c6b6468c9ad40ad456a397545c6e2df5bf"
↪ -o=ciphertext.txt plaintext.txt
```

If we print the return value `diff`, from:

```
diff := now.AddDate(years, 0, 0).Sub(now)
```

the output is 8760h0m0s. However, for a value of 1 year, the function returns the same value, ignoring 15 hours:

```
tle -D 1y -n="http://pl-us.testnet.drand.sh/" -c="7672797f548f3f4
↪ 748ac4bf3352fc6c6b6468c9ad40ad456a397545c6e2df5bf"
↪ -o=ciphertext.txt plaintext.txt
```

with output 8760h0m0s. This problem can be seen via:

```
    pieces = strings.Split(duration, "y")
    if len(pieces) == 2 {
        years, err := strconv.Atoi(pieces[0])
        if err != nil {
            return time.Second, fmt.Errorf("parse year duration:
            ↪ %w", err)
        }
        diff := now.AddDate(years, 0, 0).Sub(now)
        fmt.Println("amount added y: ", diff)

        return diff, nil
    }
```

in `encrypt.go`.

Further, mixing days with hours and minutes have the same problem. For instance, for obtaining a plaintext after 1 day, 15 hours and 1 minute via:

```
$ ./tle -D 1d15h1m -n="http://pl-us.testnet.drand.sh/" -c="767279
↪  7f548f3f4748ac4bf3352fc6c6b6468c9ad40ad456a397545c6e2df5bf"
↪  -o=ciphertext.txt plaintext.txt
```

The chosen time by `tle` is always 24 hours (diff value: 24h0m0s) instead of 1 day 15 hours and 1 minute.

Recommendation

Timelock users could need the decryption of certain sensitive information with precision. For this reason, we recommend the Client to correctly process sub-units via `parseDuration`.

Status details

The client addressed this issue in the commit `96b5251ca25e105d241e46bcca30837fc4dcf150`. By the time of the review, the client is working on the simplification of the duration parsing logic in the pull request #68.

2.2 KS-SBCF-F-02: tlock: Encryption in the future wrap to round 1

Severity: Medium

Status: Remediated

Location:

Description

Giving a year too far in the future as an argument leads to encryption for round 1 and thus is instantly accessible. The problem comes from `Date` in the Go language. The `Date` function of `time` package has an integer overflow. The year number is used to compute the number of days since epoch and then multiply be the number of seconds in a day which leads to an erroneous negative result. Then the function `RoundNumber` from tlock will return 1 for such negative results:

```
$ ./tle -D 1000000000000y -o encrypted_file data_to_encrypt
$ cat encrypted_file
```

```
age-encryption.org/v1
-> tlock 1
7672797f548f3f4748ac4bf3352fc6c6b6468c9ad40ad456a397545c6e2df5bf

hH/Rge2Um1qQVldiRByfg8MftReTkvr36gOlYDNj4jqdMJu3xuUdPsJ+ZDEnFRC8
+814SBSK+1frE6eoPzoATpClIy1jRwlsdStgFW7yHYU
--- ML9Z9pxb8gGuc3Cu8ng3wyZtFENsWA41TrfQhEY3vK0
```

This problem has been reported to the Go language team in issue #56909 https://github.com/golang/go/issues/56909.

Recommendation

The function `Date` may be patched in the future with the following patch: https://go-review.googlesource.com/c/go/+/453475. Meanwhile, tlock should check the year to be in the range 0..292277024627.

Status details

Issue has been fixed by PR #36. The duration is now checked if it was overflowed and a test has been added.

2.3 KS-SBCF-F-03: tlock-js: Stream cipher encryption nonce reuse

Severity: Medium

Status: Remediated

Location: tlock-js/src/age/stream-cipher.ts:89

Description

The nonce used in functions `encryptChunk` and `decryptChunk` wraps to zero after 2^{32} encryptions of 64Kb of data which is 256TB. This comes from the function `setUint32` which updates only 32 bits of the nonce in the function `incrementCounter`:

```
// Increments Big Endian Uint8Array-based counter.
    // [0, 0, 0] => [0, 0, 1] ... => [0, 0, 255] => [0, 1, 0]
```

```
    incrementCounter() {
        this.counter += 1
        this.nonceView.setUint32(7, this.counter, false)
    }
```

After that the stream will start to repeat and the difference of the plaintexts may be revealed.

Recommendation

Either forbid data size bigger than 256TB or increase counter size value. The specifications of age considers the first eleven bytes of the nonce as a counter (See https://github.com/C2SP/C2SP/blob/main/age.md#payload).

Status details

Issue has been fixed by PR #11. If the counter reaches the value $2^{32} - 1$ an error is reported.

2.4 KS-SBCF-F-04: timevault: Missing web site security header

Severity: Medium

Status: Remediated

Location: timevault

Description

The website does not implement any security header (See https://securityheaders.com/?q=https%3A%2F%2Ftimevault.drand.love%2F). Two of them are important:

- Strict Transport Security to enforce the use of HTTPS. This prevents man-in-the-middle attack.
- Content Security Policy to protect your site from XSS attacks.

Other less important policies may be enabled:

- `X-Frame-Options` to prevent the site to be framed.

- `X-Content-Type-Options` to stop a browser from trying to MIME-sniff the content type.
- `Permissions-Policy` to allow a site to control which features and APIs can be used in the browser.

Recommendation

The first two header should be set on the server. For the last three, even though the risk is limited for this website, it may be a good practice to set them.

Status details

All the headers have been enabled with PR #42.

2.5 KS-SBCF-F-05: kyber: Design flaws in hash to field function

Severity: Medium

Status: Remediated

Location: kyber/group/mod/int.go:437

Description

The function Hash is not properly designed and suffers several problems. The first is that the length of the message to be hashed is not taken into account. Thus hashing a string with a common prefix of the size of the modulus will result in the same digest. The following test exhibits such behavior.

```
func TestScalarHashCollision(t *testing.T) {
    msg1 := []byte("Take a walk on the wild sideTake")
    msg2 := []byte("Take a walk on the wild sideTake a")
    modulo := big.NewInt(0)
    modulo.SetString("73eda753299d7d483339d80809a1d80553bda402fff
↪   e5bfeffffffff00000001",
↪   16)
    i := new(Int).Init64(0, modulo)
    hashed1, err := i.Hash(new(hh), bytes.NewReader(msg1))
    require.NoError(t, err)
```

```
    i = new(Int).Init64(0, modulo)
    hashed2, err := i.Hash(new(hh), bytes.NewReader(msg2))
    require.NoError(t, err)
    require.Equal(t, hashed2, hashed1)
}
```

In addition, for messages with the same length it is still possible to have second preimage attacks and thus, collisions. The message will be hashed iteratively until the result size fits the scalar size. Thus, two messages with one being the hash of the other and with the first byte shifted by the proper number of bits will result in a collision. The following test demonstrate the collision:

```
func TestScalarHashBLS(t *testing.T) {
    msg1 := []byte("Take a walk on the wild side")
    msg2 := []byte{0x74, 0x58, 0x8c, 0x0d, 0xb3, 0xa7, 0x8a,
 ↪  0x81, 0xdc, 0xaf, 0xe5, 0xf8, 0x63, 0x77, 0x82, 0x77, 0x9a,
 ↪  0xaa, 0x74, 0x9c, 0xdc, 0xac, 0x47, 0xc2, 0x60, 0xc1, 0xe8,
 ↪  0x87, 0xaf, 0x99, 0x55, 0x31}
    modulo := big.NewInt(0)
    modulo.SetString("73eda753299d7d483339d80809a1d80553bda402fff
 ↪  e5bfeffffffff00000001",
 ↪  16)
    i := new(Int).Init64(0, modulo)
    hashed1, err := i.Hash(new(hh), bytes.NewReader(msg1))
    require.NoError(t, err)

    i = new(Int).Init64(0, modulo)
    hashed2, err := i.Hash(new(hh), bytes.NewReader(msg2))
    require.NoError(t, err)
    require.Equal(t, hashed2, hashed1)
}
```

This issue impacts tlock implementation which uses the h3 function from Kyber library. However, h3 apply only the Hash function on a SHA256 hash thus this seems not possible to practically exploit this issue.

Another issue, is that length of the modulus is not check to be the same as the length of the hash digest. For example, using a modulus of 48 bytes will result of a scalar with a length of 32 bytes which is not uniformly distributed among the range.

Recommendation

It seems a function similar to `hash_to_field` from https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-hash-to-curve-06#section-5.2 should be more appropriate here.

Status details

The issue has be corrected wit PR#48 in Kyber library. An iterative hash function is still implemented but the iteration number is prefix to each value hashed to avoid collision.

2.6 KS-SBCF-F-06: tlock-js: Design flaws in hash to field function

Severity: Medium

Status: Remediated

Location: tlock-js/src/crypto/ibe.ts

Description

Similarly than the previous issue, the implementation of the h3 function in `ibe.ts` rejection sampling is used at the end, via the `tofield` function:

```
// we are hashing the data until we get a value smaller than the
 ↪  curve order
export function toField(h3ret: Uint8Array) {
    let data = h3ret
    // assuming Big Endianness
    let n: bigint = bytesToNumberBE(data)
    do {
        data = sha256(data)
        // assuming Big Endianness
        data[0] = data[0] >> BitsToMaskForBLS12381
        n = bytesToNumberBE(data)
    } while (n <= 0 || n > bls.CURVE.r)
```

```
    return n
}
```

Recommendation

It seems a function similar to `hash_to_field` from https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-hash-to-curve-16#section-5.2 should be more appropriate here.

Status details

The problem was corrected wit PR #26 similarly to the previous issue.

2.7 KS-SBCF-F-07: tlock: Timelock encryption decrypt to files with permissive permission

Severity: Low

Status: Remediated

Location: src/tlock/cmd/tle/tle.go: 44 and 54

Description

According to `OpenFile` comments:

```
// OpenFile is the generalized open call; most users will use Open
// or Create instead. It opens the named file with specified flag
// (O_RDONLY etc.). If the file does not exist, and the O_CREATE
 ↪  flag
// is passed, it is created with mode perm (before umask). If
 ↪  successful,
// methods on the returned File can be used for I/O.
// If there is an error, it will be of type *PathError.
```

If a file does not exist, `tle` will create it with a public read permission. Thus, during a decryption if the output file does not exist it is created with read permission for all users of the system (644):

```
$ ./tle -d -o data encrypted_file
$ ls -al data
-rw-r--r-- 1 sylvain sylvain 20 Nov  8 10:03 data
```

It may be a problem if the decrypted data contains sensitive information.

Recommendation

Created file should have 0600 permission when written.

Status details

Issue has been fixed by PR #59.

2.8 KS-SBCF-F-08: tlock: No point sanitization in TimeLock and TimeUnlock functions

Severity: Low

Status: Remediated

Location: src/tlock/tlock.go: 99

Description

It is possible to use the point-at-infinity to encrypt using the `TimeLock` function (see tlock.go:99), even if those public keys have trivial secret keys. *e.g.* via:

```
    cipherText, err := tlock.TimeLock(network.PublicKey().Null(),
↪   futureRound, data)
```

If by any means the drand nodes involved in the network choose or generate a collective public key with a trivial associated private key, every ciphertext generated by the `TimeLock` function can be easily decrypted.

This behavior is not consistent with `tlock-js`, which detects the point-at-infinity and aborts any attempt of using a pairing with it.

For instance, via `npm test`:

```
 1) timelock
      encryption
        should pass for a valid roundNumber:
    Error: No pairings at point of Infinity
     at Object.pairing
↪  (node_modules/@noble/bls12-381/lib/index.js:559:15)
     at Object.encrypt (src/crypto/ibe.ts:20:21)
```

Finally, there is no subgroup membership check on the incoming public keys. This type of check is enforced by https://www.ietf.org/id/draft-irtf-cfrg-bls-signature-05.html. Particularly, Section 5.2 of such document (Validating public keys) notes:

```
KeyValidate requires all public keys to represent valid,
↪  non-identity points in the correct subgroup. A valid point and
↪  subgroup membership are required to ensure that the pairing
↪  operation is defined (Section 5.3).
```

```
A non-identity point is required because the identity public key
↪  has the property that the corresponding secret key is equal to
↪  zero, which means that the identity point is the unique valid
↪  signature for every message under this key. A malicious signer
↪  could take advantage of this fact to equivocate about which
↪  message he signed. While non-equivocation is not a required
↪  property for a signature scheme, equivocation is infeasible
↪  for BLS signatures under any nonzero secret key because it
↪  would require finding colliding inputs to the hash_to_point
↪  function, which is assumed to be collision resistant.
```

```
Prohibiting SK == 0 eliminates the exceptional case, which may
↪  help to prevent equivocation-related security issues in
↪  protocols that use BLS signatures.
```

Recommendation

We recommend the Client to warn the user about possible weak keys used by the network as well as to provide consistency between the different implementations of

Timelock Encryption.

Status details

Based on the last commit reviewed, `2e5177054b1fbdbafa09265b5985d9831e9dd225`, public keys that are the point-at-infinity are now detected via:

```
func TimeLock(publicKey kyber.Point, roundNumber uint64, data
↪  []byte) (*ibe.Ciphertext, error) {
    if publicKey.Equal(publicKey.Null()) {
        return nil, ErrInvalidPublicKey
    }
```

The Client communicated to us that they are using the Killic `FromCompressed` method in the signature parsing functionality (`UnmarshalBinary`). This is utilized in the `TimeUnlock` function, which performs a subgroup membership check. Moreover, the network layer also relies on the `UnmarshalBinary` function.

2.9 KS-SBCF-F-09: tlock: tlock dependency tree contains 3 dependencies with vulnerabilities

Severity: Low

Status: Remediated

Location: General

Description

The `tlock` dependency tree contains dependencies with vulnerabilities:

- `pkg:golang/github.com/aws/aws-sdk-go@v1.32.11`:
  - CVE-2020-8911: Use of a Broken or Risky Cryptographic Algorithm
  - CVE-2020-8912: Use of a Broken or Risky Cryptographic
- `pkg:golang/github.com/lucas-clemente/quic-go@v0.24.0`:
  - CVE-2022-30591: Uncontrolled Resource Consumption (‘Resource Exhaustion’)
- `pkg:golang/golang.org/x/text@v0.3.7`:
  - CVE-2022-32149: Missing Release of Resource after Effective

Recommendation

We recommend the client to update those dependencies.

Status details

Based on the last commit reviewed, `2e5177054b1fbdbafa09265b5985d9831e9dd225`, the `tlock` dependency tree only contains a problematic package (quic-go) which has been categorized as disputed by the vendor. We recommend the client to follow the current discussion and updates on this issue if it is not possible to update it soon:

```
** DISPUTED ** quic-go through 0.27.0 allows remote attackers to
 ↪  cause a denial of service (CPU consumption) via a Slowloris
 ↪  variant in which incomplete QUIC or HTTP/3 requests are sent.
 ↪  This occurs because mtu_discoverer.go misparses the MTU
 ↪  Discovery service and consequently overflows the probe timer.
 ↪  NOTE: the vendor's position is that this behavior should not
 ↪  be listed as a vulnerability on the CVE List.
```

See for instance https://nvd.nist.gov/vuln/detail/CVE-2022-30591.

3 OTHER OBSERVATIONS

This section contains additional observations that are not directly related to the security of the code, and as such have no severity rating or remediation status summary. These observations are either minor remarks regarding good practice or design choices or related to implementation and performance. These items do not need to be remediated for what concerns security, but where applicable we include recommendations.

3.1 KS-SBCF-O-01: Missing security policy

Location: tlock, tlock-js, timevault

Description

Currently there is no instructions for how to report a security vulnerability regarding the repositories and the website nor security contacts.

Recommendation

Create a `SECURITY.md` file in the root directory with all the necessary information. See for example: https://docs.github.com/en/code-security/getting-started/adding-a-security-policy-to-your-repository. Timevault website may also contain a `security.txt` with such information. Such file may be generated from https://securitytxt.org/.

Status details

A file `SECURITY.md` was created in PR #64 for tlock and PR #45 for timevault.

3.2 KS-SBCF-O-02: tlock: envconfig.Process output is not processed

Location: tlock/cmd/tle/commands/commands.go:83

Description

According to the reference description of Process, the `Process` call returns an error.

However, this error is not validated at: `cmd/tle/commands/commands.go:83`:

```
envconfig.Process("tle", &f)
```

Recommendation

We recommend the Client to check the result of the function.

Status details

The problem has been solved by PR #56.

3.3 KS-SBCF-O-03: tlock: Multiple defer calls with Close()

Location: tlock/cmd/tle/tle.go:48 and cmd/tle/tle.go:58

Description

In Golang, the `Close()` function can return an error value. The client defers to `Close()` calls in the following parts of tlock:

```
cmd/tle/tle.go:48:  defer f.Close()
cmd/tle/tle.go:58:  defer f.Close()
```

Recommendation

We recommend the Client to check if there are errors after the Close() call. This is especially important when writing files in Go. For more information, the Client can check https://www.joeshaw.org/dont-defer-close-on-writable-files/

3.4 KS-SBCF-O-04: Error in drand documentation

Location: https://drand.love/docs/cryptography/#randomness-generation

Description

In the drand reference documentation, in the Beacon phase, Signature Verification Section, the check should be $e(H(m), S) == e(\sigma, G_2)$ instead of $e(H(m), S_i) == e(\sigma_i, G_2)$.

This check describes the verification of a collective signature instead of a partial signature.

3.5 KS-SBCF-O-05: tlock: Unused return value

Location: cmd/tle/commands/commands.go: 84

Description

The value returned by the function `parseCmdline` is not used in the `Parse` function.

Recommendation

The function `parseCmdline` may not return a value.

Status details

Issue has been fixed by PR #56.

3.6 KS-SBCF-O-06: tlock: Incompatible flags not detected

Location: tlock/cmd/tle/commands/commands.go: 132 and 143

Description

Some combination of incompatible flags are not detected properly when the duration is set to the default duration of 120 days. For example it is possible to specify a round number and a duration:

```
$ ./tle -D 120d -r 10000000 data_to_encrypt
age-encryption.org/v1
-> tlock 10000000
 ↪  7672797f548f3f4748ac4bf3352fc6c6b6468c9ad40ad456a397545c6e2df5bf
pNYa1k6gtspRASycP9k6rCbgRZw/IKp2Ld6z97yrrw3y1Igq/OKKjsDL2cwYf/lq
XL6pM1mbXIqt+yWlztgciIV2Hox6rNl7xrO0+9Kk5Aw
--- s0nn2yUgrrm4eIdoKsYxMGqzVc5/UfGhk+VqK+Gq5Qs
```

Or it is possible to specify decryption with a duration:

```
$ ./tle -d -D 120d -r 100 encrypted_file
too early to decrypt
```

Recommendation

These edge cases should be detected and rejected. Some failing tests could be integrated in the test corpus to ensure non regression.

Status details

Issue has been fixed by PR #17 and tests have been added.

3.7 KS-SBCF-O-07: tlock-js: Special number values NaN and Infinity are not checked

Location: tlock-js/src/drand/drand-client.ts: 132 and 143

Description

The values `NaN` and `Infinity` are not checked by the `timeForRound` and `roundForTime` functions:

```
> tlock.timeForRound(1e305, info)
Infinity
> tlock.timeForRound(Infinity, info)
Infinity
> tlock.roundForTime(NaN, info)
NaN
```

Recommendation

Those special value may be checked and filtered.

3.8 KS-SBCF-O-08: tlock-js: Missing check for malformed recipients

Location: tlock-js/src/age//age-reader-writer.ts:109

Description

In the official implementation of age (i.e. at https://github.com/FiloSottile/age/), the functions described at `format.go` check if every recipient is well-formed based on the string value and the allowed type of characters via `isValidString`:

```
        } else if bytes.HasPrefix(line, recipientPrefix) {
            if r != nil {
                return nil, nil, errorf("malformed body line %q:
                 ↪  new stanza started without previous stanza
                 ↪  being closed\nNote: this might be a file
                 ↪  encrypted with an old beta version of rage.
                 ↪  Use rage to decrypt it.", line)
            }
            r = &Stanza{}
            prefix, args := splitArgs(line)
            if prefix != string(recipientPrefix) || len(args) < 1
             ↪  {
                return nil, nil, errorf("malformed recipient:
                 ↪  %q", line)
            }
            for _, a := range args {
                if !isValidString(a) {
                    return nil, nil, errorf("malformed recipient:
                     ↪  %q", line)
                }
            }
```

In this case, `isValidString` checks:

```
func isValidString(s string) bool {
    if len(s) == 0 {
        return false
    }
    for _, c := range s {
        if c < 33 || c > 126 {
            return false
        }
```

```
    }
    return true
}
```

However, this check is not performed in `parseRecipients` (age-reader-writer.ts):

```
        const body = parseRecipientBody(lines)
        if (!body) {
            throw Error(`expected stanza ' to have a body, but it
             ↪  didn't`)
        }

        recipients.push({type, args, body: Buffer.from(body,
↪   "base64")})
    }

    if (recipients.length === 0) {
        throw Error("Expected at least one stanza! (beginning
         ↪  with -->)")
    }
```

3.9 KS-SBCF-O-09: tlock-js: Missing check for trailing space after armor footer

Location: tlock-js/src/age/armor.ts

Description

The function `decodeArmor` in `armor.ts` validates the length of each line of the payload between the armor header and footer:

```
    const lines = base64Payload.split("\n")
    if (lines.some(line => line.length > chunkSize)) {
        throw Error(`Armor to decode cannot have lines longer
↪   than  (configurable) in order to stop padding attacks`)
    }
```

```
    if (lines[lines.length - 1].length >= chunkSize) {
        throw Error(`The last line of an armored payload must be
↪   less than  (configurable) to stop padding attacks`)
    }
```

However, it doesn't validate the existence of trailing data and and trailing white-space after the footer. In the original implementation of age this check is implemented in `armor.go` via the `drainTrailing` function:

```
    const maxWhitespace = 1024
    drainTrailing := func() error {
        buf, err := io.ReadAll(io.LimitReader(r.r, maxWhitespace))
        if err != nil {
            return err
        }
        if len(bytes.TrimSpace(buf)) != 0 {
            return errors.New("trailing data after armored file")
        }
        if len(buf) == maxWhitespace {
            return errors.New("too much trailing whitespace")
        }
        return io.EOF
    }
```

The user receives an error and the reader aborts, if there is trailing data after the footer as well as if there is too much trailing white-space. This check is not implemented in tlock-js, where white-spaces are always ignored, which is ensured by the following test in test/armor.test.ts:57:

```
        it("should ignore whitespace at the beginning and end
        ↪   when decoding", () => {
            const somePlaintext = "wow that's a lot of armor"
            expect(decodeArmor("    \n " +
            ↪   encodeArmor(somePlaintext) + "\t \n
            ↪   ")).to.equal(somePlaintext)
        })
```

Status details

The problem has been solved by PR #17 in tlock-js.

3.10 KS-SBCF-O-10: tlock-js: Missing check for ciphertext length in Decrypt

Location: tlock-js/src/crypto/ibe.ts:47

Description

The function `decrypt`, described in `ibe.ts`, doesn't validate the length of the ciphertext provided, more precisely, of the `c.W` component, which cannot be longer than the output of the hash function utilized. This check is performed in the `Decrypt` implementation of the kyber library, in `ibe.go`:

```
func Decrypt(s pairing.Suite, private kyber.Point, c *Ciphertext)
 ↪  ([]byte, error) {
    if len(c.W) > s.Hash().Size() {
        return nil, errors.New("ciphertext too long for the hash
         ↪  function provided")
    }

    // 1. Compute sigma = V XOR H2(e(rP,private))
    rGid := s.Pair(c.U, private)
    hrGid, err := gtToHash(s, rGid, len(c.W), H2Tag())
    if err != nil {
        return nil, err
    }
    if len(hrGid) != len(c.V) {
```

However, this check is not performed in `ibe.ts`:

```
export async function decrypt(p: PointG2, c: Ciphertext):
 ↪  Promise<Uint8Array> {
    // 1. Compute sigma = V XOR H2(e(rP,private))
```

```
    const gidt = bls.pairing(c.U, p)
    const hgidt = gtToHash(gidt, c.W.length)

    if (hgidt.length != c.V.length) {
        throw new Error("XorSigma is of invalid length")
    }
    const sigma = xor(hgidt, c.V)
```

4 OBSERVATIONS ON DEPENDENCIES

4.1 KS-SBCF-O-DEP-01: noble-bls12-381: Modular exponentiation is not constant-time

Location: https://github.com/paulmillr/noble-bls12-381/blob/main/math.ts at line 90.

Description

The modular exponentiation function provided by noble-bls12-381 is not constant-time:

```
/**
 * Efficiently exponentiate num to power and do modular division.
 * @example
 * powMod(2n, 6n, 11n) // 64n % 11n == 9n
 */
export function powMod(num: bigint, power: bigint, modulo: bigint)
 ↪  {
  if (modulo <= 0n || power < 0n) throw new Error('Expected
   ↪  power/modulo > 0');
  if (modulo === 1n) return 0n;
  let res = 1n;
  while (power > 0n) {
    if (power & 1n) res = (res * num) % modulo;
    num = (num * num) % modulo;
    power >>= 1n;
  }
```

```
  return res;
}
```

4.2 KS-SBCF-O-DEP-02: noble-bls12-381: Field exponentiation is not constant-time

Location: https://github.com/paulmillr/noble-bls12-381/blob/main/math.ts at line 388.

Description

The field exponentiation function provided by noble-bls12-381 is not constant-time:

```
function powMod_FQP(fqp: any, fqpOne: any, n: bigint) {
  const elm = fqp;
  if (n === 0n) return fqpOne;
  if (n === 1n) return elm;
  let p = fqpOne;
  let d = elm;
  while (n > 0n) {
    if (n & 1n) p = p.multiply(d);
    n >>= 1n;
    d = d.square();
  }
  return p;
}
```

4.3 KS-SBCF-O-DEP-03: noble-bls12-381: Speed-up in point arithmetic

Location: General

Description

noble-bls12-381 uses projective coordinates and the following formulas for point arithmetic:

- http://hyperelliptic.org/EFD/g1p/auto-shortw-projective.html#doubling-dbl-1998-cmo-2 for point doubling (6M + 5S).
- http://hyperelliptic.org/EFD/g1p/auto-shortw-projective.html#addition-add-1998-cmo-2 for point addition (12M + 2S)

However, it is possible to greatly reduce the number of multiplications for point doubling (and only if the Client prefers performance vs. the utilization of complete formulas e.g. via Renes et al., https://eprint.iacr.org/2015/1060) by relying on the following approach:

- https://hyperelliptic.org/EFD/g1p/auto-shortw-jacobian-3.html#doubling-dbl-2007-bl (1M + 8S)
- https://hyperelliptic.org/EFD/g1p/auto-shortw-jacobian-3.html#addition-add-2007-bl (11M + 5S)

4.4 KS-SBCF-O-DEP-04: noble-bls12-381: Speed-up in scalar multiplication

Location: https://github.com/paulmillr/noble-bls12-381/blob/main/math.ts from line 1039.

Description

noble-bls12-381 provides double and add and constant-time scalar multiplication in math.ts (from 1039):

```
// Non-constant-time multiplication. Uses double-and-add
 ↪  algorithm.
 // It's faster, but should only be used when you don't care
  ↪  about
 // an exposed private key e.g. sig verification.
 multiplyUnsafe(scalar: bigint): this {
   let n = this.validateScalar(scalar);
   let point = this.getZero();
   let d: this = this;
   while (n > 0n) {
     if (n & 1n) point = point.add(d);
     d = d.double();
     n >>= 1n;
```

```
    }
    return point;
  }

  // Constant-time multiplication
  multiply(scalar: bigint): this {
    let n = this.validateScalar(scalar);
    let point = this.getZero();
    let fake = this.getZero();
    let d: this = this;
    let bits = Fp.ORDER;
    while (bits > 0n) {
      if (n & 1n) {
        point = point.add(d);
      } else {
        fake = fake.add(d);
      }
      d = d.double();
      n >>= 1n;
      bits >>= 1n;
    }
    return point;
  }
```

The performance of the scalar multiplication operation could be improved by relying on the GLV endomorphism (https://www.iacr.org/archive/crypto2001/21390189.pdf).

4.5 KS-SBCF-O-DEP-05: noble-bls12-381: hash-to-curve reference implementation is not updated to last version (16)

Location: General

Description

noble-bls12-381 relies on the hash-to-curve draft version 11. However, the last version at the time of writing this document is Version 16. See https://datatracker.ietf.org/doc

/draft-irtf-cfrg-hash-to-curve/.

4.6 KS-SBCF-O-DEP-06: kyber-bls12381: Speed-up on subgroup membership

Location: kyber_g2.go:149 and g2.go:758 (kilic/bls12-381)

Description

The drand kyber library utilizes the strategy proposed by Bowe https://eprint.iacr.org/2019/814.pdf to check the membership of points to the right subgroup in $\mathbb{G}_2$. There is a faster technique proposed by Scott in 2021 that the client could consider at https://eprint.iacr.org/2021/1130.pdf, Section 4.

4.7 KS-SBCF-O-DEP-07: kilic/bls12-381: Arithmetic methods for groups never check that the input parameters are on the curve and in the right subgroup

Location: g1.go and g2.go

Description

The library only sanitize external points (`PointG1`, `PointG2` types) in the `FromCompressed` and `FromUncompressed` functions via the `InCorrectSubgroup` and `IsOnCurve` methods.

4.8 KS-SBCF-O-DEP-08: kilic/bls12-381: The field elements and scalar field elements comparison are not constant-time

Location: field_element.go and fr.go.

Description

The comparison of field elements is not constant-time:

```
func (fe *fe) cmp(fe2 *fe) int {
    for i := fpNumberOfLimbs - 1; i >= 0; i-- {
```

```
        if fe[i] > fe2[i] {
            return 1
        } else if fe[i] < fe2[i] {
            return -1
        }
    }
    return 0
}
```

as well as the comparison of elements in $\mathbb{F}_r$:

```
func (e *Fr) Cmp(e1 *Fr) int {
    for i := frNumberOfLimbs - 1; i >= 0; i-- {
        if e[i] > e1[i] {
            return 1
        } else if e[i] < e1[i] {
            return -1
        }
    }
    return 0
```

4.9 KS-SBCF-O-DEP-09: kilic/bls12-381: Exponentiation functions and inversion are not constant-time

Location: fp.go and fp2.go.

Description

For $\mathbb{F}_p$ elements, the exponentiation is performed via the square-and-multiply algorithm:

```
func exp(c, a *fe, e *big.Int) {
    z := new(fe).set(r1)
    for i := e.BitLen(); i >= 0; i-- {
        mul(z, z, z)
        if e.Bit(i) == 1 {
```

```
            mul(z, z, a)
        }
    }
    c.set(z)
}
```

as well as in the case of $\mathbb{F}_p^2$:

```
func (e *fp2) exp(c, a *fe2, s *big.Int) {
    z := e.one()
    for i := s.BitLen() - 1; i >= 0; i-- {
        e.square(z, z)
        if s.Bit(i) == 1 {
            e.mul(z, z, a)
        }
    }
    c.set(z)
}
```

Finally, the inversion function in $\mathbb{F}_p$ is not constant-time:

```
func inverse(inv, e *fe) {
    if e.isZero() {
        inv.zero()
        return
    }
    u := new(fe).set(&modulus)
    v := new(fe).set(e)
    s := &fe{1}
    r := &fe{0}
    var k int
    var z uint64
    var found = false
    // Phase 1
    for i := 0; i < sixWordBitSize*2; i++ {
        if v.isZero() {
            found = true
```

```
            break
        }
        if u.isEven() {
            u.div2(0)
            s.mul2()
        } else if v.isEven() {
            v.div2(0)
            z += r.mul2()
        } else if u.cmp(v) == 1 {
            lsubAssign(u, v)
            u.div2(0)
            laddAssign(r, s)
            s.mul2()
        } else {
            lsubAssign(v, u)
            v.div2(0)
            laddAssign(s, r)
            z += r.mul2()
        }
        k += 1
    }

    if !found {
        inv.zero()
        return
    }

    if k < fpBitSize || k > fpBitSize+sixWordBitSize {
        inv.zero()
        return
    }

    if r.cmp(&modulus) != -1 || z > 0 {
        lsubAssign(r, &modulus)
    }
    u.set(&modulus)
```

```
    lsubAssign(u, r)

    // Phase 2
    for i := k; i < 2*sixWordBitSize; i++ {
        double(u, u)
    }
    inv.set(u)
}
```

5 APPENDIX A: ABOUT KUDELSKI SECURITY

Kudelski Security is an innovative, independent Swiss provider of tailored cyber and media security solutions to enterprises and public sector institutions. Our team of security experts delivers end-to-end consulting, technology, managed services, and threat intelligence to help organizations build and run successful security programs. Our global reach and cyber solutions focus is reinforced by key international partnerships.

Kudelski Security is a division of Kudelski Group. For more information, please visit https://www.kudelskisecurity.com.

Kudelski Security
Route de Genève, 22-24
1033 Cheseaux-sur-Lausanne
Switzerland

Kudelski Security
5090 North 40th Street
Suite 450
Phoenix, Arizona 85018

6 APPENDIX B: METHODOLOGY

For this engagement, Kudelski used a methodology that is described at high-level in this section. This is broken up into the following phases.

Figure 1: Methodology flow

6.1 Kickoff

The project was kicked off when all of the sales activities had been concluded. We set up a kickoff meeting where project stakeholders were gathered to discuss the project as well as the responsibilities of participants. During this meeting we verified the scope of the engagement and discussed the project activities. It was an opportunity for both sides to ask questions and get to know each other. By the end of the kickoff there was an understanding of the following:

- Designated points of contact
- Communication methods and frequency
- Shared documentation
- Code and/or any other artifacts necessary for project success
- Follow-up meeting schedule, such as a technical walkthrough
- Understanding of timeline and duration

6.2 Ramp-up

Ramp-up consisted of the activities necessary to gain proficiency on the particular project. This included the steps needed for gaining familiarity with the codebase and technological innovations utilized, such as:

- Reviewing previous work in the area including academic papers
- Reviewing programming language constructs for the languages used in the code
- Researching common flaws and recent technological advancements

6.3 Review

The review phase is where a majority of the work on the engagement was performed. In this phase we analyzed the project for flaws and issues that could impact the security posture. This included an analysis of the architecture, a review of the code, and a specification matching to match the architecture to the implemented code.

In this code audit, we performed the following tasks:

1. Security analysis and architecture review of the original protocol
2. Review of the code written for the project
3. Assessment of the cryptographic primitives used
4. Compliance of the code with the provided technical documentation

The review for this project was performed using manual methods and utilizing the experience of the reviewer. No dynamic testing was performed, only the use of custom-built scripts and tools were used to assist the reviewer during the testing. We discuss our methodology in more detail in the following subsections.

Code Safety

We analyzed the provided code, checking for issues related to the following categories:

- General code safety and susceptibility to known issues
- Poor coding practices and unsafe behavior
- Leakage of secrets or other sensitive data through memory mismanagement
- Susceptibility to misuse and system errors
- Error management and logging

This is a general and not comprehensive list, meant only to give an understanding of the issues we have been looking for.

Cryptography

We analyzed the cryptographic primitives and components as well as their implementation. We checked in particular:

- Matching of the proper cryptographic primitives to the desired cryptographic functionality needed
- Security level of cryptographic primitives and their respective parameters (key lengths, etc.)

- Safety of the randomness generation in general as well as in the case of failure
- Safety of key management
- Assessment of proper security definitions and compliance to use cases
- Checking for known vulnerabilities in the primitives used

Technical Specification Matching

We analyzed the provided documentation and checked that the code matches the specification. We checked for things such as:

- Proper implementation of the documented protocol phases
- Proper error handling
- Adherence to the protocol logical description

6.4 Reporting

Kudelski delivered to the Client a preliminary report in PDF format that contained an executive summary, technical details, and observations about the project, which is also the general structure of the current final report.

The executive summary contains an overview of the engagement, including the number of findings as well as a statement about our general risk assessment of the project as a whole.

In the report we not only point out security issues identified but also informational findings for improvement categorized into several buckets:

- High
- Medium
- Low
- Informational

The technical details are aimed more at developers, describing the issues, the severity ranking and recommendations for mitigation.

As we performed the audit, we also identified issues that are not security related, but are general best practices and steps, that can be taken to lower the attack surface of the project.

As an optional step, we can agree on the creation of a public report that can be shared and distributed with a larger audience.

6.5 Verify

After the preliminary findings have been delivered, we verified the fixes applied by the Client. After these fixes were verified, we updated the status of the finding in the report.

The output of this phase was the current, final report with any mitigated findings noted.

6.6 Additional Note

It is important to notice that, although we did our best in our analysis, no code audit assessment is per se guarantee of absence of vulnerabilities. Our effort was constrained by resource and time limits, along with the scope of the agreement.

In assessing the severity of some of the findings we identified, we kept in mind both the ease of exploitability and the potential damage caused by an exploit. Since this is a library, we ranked some of these vulnerabilities potentially higher than usual, as we expect the code to be reused across different applications with different input sanitization and parameters.

Correct memory management is left to Go and was therefore not in scope. Zeroization of secret values from memory is also not enforceable at a low level in a language such as Go.

While assessment the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. This is a solid baseline for severity determination. Information about the severity ratings can be found in **Appendix C** of this document.

7 APPENDIX C: SEVERITY RATING DEFINITIONS

Kudelski Security uses a custom approach when determining criticality of identified issues. This is meant to be simple and fast, providing customers with a quick at a glance view of the risk an issue poses to the system. As with anything risk related, these findings are situational. We consider multiple factors when assigning a severity level to an identified vulnerability. A few of these include:

- Impact of exploitation
- Ease of exploitation
- Likelihood of attack
- Exposure of attack surface
- Number of instances of identified vulnerability
- Availability of tools and exploits

Severity	Definition
High	The identified issue may be directly exploitable causing an immediate negative impact on the users, data, and availability of the system for multiple users.
Medium	The identified issue is not directly exploitable but combined with other vulnerabilities may allow for exploitation of the system or exploitation may affect singular users. These findings may also increase in severity in the future as techniques evolve.
Low	The identified issue is not directly exploitable but raises the attack surface of the system. This may be through leaking information that an attacker can use to increase the accuracy of their attacks.
Informational	Informational findings are best practice steps that can be used to harden the application and improve processes.