Report 1907551

Source Code Review Proton Crypto Library

for

Proton Technologies AG

conducted by

SEC Consult

Version: 1.2 | **Date:** 2019-05-14
Responsible: SEC Consult | **Author**: SEC Consult
Confidentiality class: Public

Table of Contents

1 Management Summary

The following chapter summarizes the scope and timetable of the code review, the results of the code review, and outlines the measures recommended by SEC Consult.

1.1 Scope and Timetable

During the initial security assessment for Proton Technologies AG, SEC Consult performed a source code review of the Proton Crypto Library - a set of supplementary cryptography libraries written in Go, which allows encrypting, decrypting, signing, and verifying messages using symmetric and asymmetric cryptography. Objective of the review was to reveal security issues and to offer suggestions for improvement. The focus of the code review was to identify:

- vulnerabilities in cryptographic algorithms,
- routines that could cause user data compromise,
- routines that could be abused for user monitoring.

The initial review was conducted in March 2019 and a total effort of 7 days was dedicated to identifying and documenting security issues in the code base of the Proton Crypto Library.

The following subsystems were in the scope of the code review:

- openpgp
- ed25519
- brainpool

The source code of the library was publicly available and could be obtained from the respective online repository (https://github.com/ProtonMail/crypto/tree/5bcbe4637f9ffa5c89be9a5c1f2bcbdac5b881a9).

In April 2019, Proton Technologies fixed the identified issues and supplied the fixes to SEC Consult for verification (https://github.com/ProtonMail/crypto/tree/efb430e751f2f00d8d9aedb254fc14ef76954880). Goal of the fix verification was to confirm remediation provided by the applied fixes. SEC Consult verified the fixes in May 2019.

1.2 Results

During the initial code review, SEC Consult found **a high-risk vulnerability** in the reviewed source code. An attacker can, given that specific preconditions are met, abuse this vulnerability to:

- spoof arbitrary cleartext messages without invalidating the signatures.

In addition, SEC Consult discovered **a medium-risk vulnerability** in the reviewed source code. An attacker can, given that specific preconditions are met, abuse these vulnerabilities to:

- force victims to generate small-size private keys.

All security issues that were identified in the initial code review were properly fixed by Proton Technologies AG.

1.3 Disclaimer

At the request of Proton Technology AG, this report has been declassified from strictly confidential to public. While the report was shortened for public release, relevant vulnerability information has been maintained.

In this particular project, a timebox approach was used to define the consulting effort. This means that SEC Consult allotted a prearranged amount of time to identify and document vulnerabilities. Because of this, there is no guarantee that the project has discovered all possible vulnerabilities and risks.

Furthermore, the security check is only an immediate evaluation of the situation at the time the check was performed. An evaluation of future security levels or possible future risks or vulnerabilities may not be derived from it.

2 Vulnerability Summary

This chapter contains all identified vulnerabilities in the reviewed source code of the company Proton Technologies AG.

Risk assessment	Initial no. of vulnerability classes	Current no. of vulnerability classes
Low	0	0
Medium	1	0
High	1	0
Critical	0	0
Total	**2**	**0**

2.1 Total Risk Per System

The following table contains a risk assessment for each system which contained security flaws.

System	Field of application	Initial risk	Current risk
Proton Crypto Library	Cryptography Library	High	-
Total	**-**	**High**	**-**

2.2 Risk of Each Vulnerability

The following table contains a risk assessment for the discovered vulnerabilities.

Vulnerability	System	Initial risk	Current risk	Page
Cleartext message spoofing	Proton Crypto Library	High	FIXED	6
Small RSA keys are allowed	Proton Crypto Library	Medium	FIXED	8
Total	**-**	**High**	**-**	**-**

3 Detailed Analysis

This chapter outlines the attacks and found vulnerabilities in detail.

3.1 Proton Crypto Library

3.1.1 General Information

This section describes vulnerabilities found in the Proton Crypto Library. The parts of the library, which were reviewed implement the core OpenPGP functionalities as well as key generation, encrypting, decrypting, signing and signature verification functionalities.

3.1.2 Cleartext message spoofing - FIXED

According to RFC 4880 Chapter 7, the cleartext signed message can contain one or more optional “Hash” Armor Headers. The “Hash” Armor Header specifies the message digest algorithm(s) used for the signature. However, the Proton Crypto library ignores the value of this Header, which allows an attacker to spoof it. Thereby, an attacker can lead a victim to believe the signature was generated using a different message digest algorithm than what was actually used.

Moreover, since the library skips Armor Header parsing in general, an attacker cannot only embed arbitrary Armor Headers, but also prepend arbitrary text to cleartext messages without invalidating the signatures.

CVSS-v3 Base Score: 7.1 (High)

CVSS-v3 Vector String: CVSS:3.0/AV:N/AC:L/PR:N/UI:R/S:U/C:L/I:H/A:N

This issue also affects mainline Go Cryptography Libraries from which Proton Crypto Library was forked. SEC Consult contacted the respective vendor and a patch has been published for this library as well by now.

3.1.2.1 Recheck results

Like in the initial code review, for the recheck the following cleartext messages were used:

- the cleartext message with a valid SHA-1 signature;
- the message with a valid SHA-1 signature was tampered by changing the value of the “Hash” Armor Header from SHA-1 to SHA-512;
- the message with a valid SHA-1 signature was modified by embedding Unicode-encoded “LINE TABULATION” in the “Hash” Armor Header.

The script *sig_spoof.go* performing signature verification is given in section 4.1. The output of the script shows, that the issue was successfully fixed:

```
$ go run sig_spoof.go
Verifying not tampered...
Signature accepted!

Verifying spoofed hash...
failed to check signature: %s openpgp: invalid data: unknown hash algorithm
in cleartext message headers
Verifying spoofed cleartext...
No clearsign text found
```

For protecting against the initial vulnerability, a new, higher-level API *clearsign.Block.VerifySignature*, was added. It is recommended to use this method instead (marked as bold text in *sig_spoof.go* script in section 4.1).

3.1.3 Small RSA keys are allowed - FIXED

Several parameters can be configured externally where the size of the generated RSA key can be specified. However, the key generation routine does not perform checks on the given key size (except if equal to zero). Thus, if an attacker can influence the external configuration, the small-sized key can be generated, which can be used for compromising the legitimate user.

CVSS-v3 Base Score: 5.8 (Medium)

CVSS-v3 Vector String: CVSS:3.0/AV:L/AC:L/PR:H/UI:R/S:U/C:H/I:H/A:N

3.1.3.1 Recheck results

The following code fragment (same as in the initial code review) was used to verify if the issue is fixed:

```
package main

import ("fmt"
        "golang.org/x/crypto/rsa"
        "golang.org/x/crypto/openpgp"
        "golang.org/x/crypto/openpgp/packet")

func main() {
    en, err := openpgp.NewEntity("user", "", "", &packet.Config{RSABits: 64,
Algorithm: packet.PubKeyAlgoRSA})
    if err != nil {
        fmt.Println(err)
    }
    fmt.Println(en.PrivateKey.PrivateKey.(*rsa.PrivateKey).D)
}
```

The output of the script shows, that the issue was successfully fixed:

```
$ go run poc_short_keys.go
crypto/rsa: bits must be >= 1024
panic: runtime error: invalid memory address or nil pointer dereference
[signal SIGSEGV: segmentation violation code=0x1 addr=0x8 pc=0x51bc2f]

goroutine 1 [running]:
main.main()
    /go/src/proton/short_rsa_keys/poc_short_keys.go:14 +0xdf
exit status 2
```

3.1.1 Notes

The following lists peculiarities and minor issues that were identified during the code review.

1. Implementation allows both interpretation and generation of RSA Sign-Only and RSA Encrypt-Only keys. The following code fragment from *openpgp/packet/private_key.go* demonstrates the issue:

```
func NewSignerPrivateKey(currentTime time.Time, signer crypto.Signer)
*PrivateKey {
      pk := new(PrivateKey)
      switch pubkey := signer.Public().(type) {
      case rsa.PublicKey:
            pk.PublicKey = *NewRSAPublicKey(currentTime, &pubkey)
            pk.PubKeyAlgo = PubKeyAlgoRSASignOnly
      case ecdsa.PublicKey:
```

   According to RFC 4880 Chapter 9.1 this is deprecated and should not be used anymore: *"RSA Encrypt-Only (2) and RSA Sign-Only are deprecated and SHOULD NOT be generated but may be interpreted."*

   It is recommended to use key flags instead.

   Fix verification on 2019-04-30: the issue was fixed.

4 Appendix

4.1 sig_spoof.go

```
package main

import ("fmt"
        "golang.org/x/crypto/openpgp/clearsign"
        "golang.org/x/crypto/openpgp"
        "golang.org/x/crypto/openpgp/packet"
        "bytes"
        )

func verify(input []byte) {
    var err error
      b, _:= clearsign.Decode(input)
      if b == nil {
        fmt.Println("No clearsign text found")
        return
      }
      keyring, err :=
openpgp.ReadArmoredKeyRing(bytes.NewBufferString(signingKey))
      if err != nil {
            fmt.Println(err)
        return
      }

    if _, err := openpgp.CheckDetachedSignature(keyring,
bytes.NewBuffer(b.Bytes), b.ArmoredSignature.Body, &packet.Config{RSABits:
64, Algorithm: packet.PubKeyAlgoRSA}); err != nil {
            fmt.Println(err)
        return
      }
      b, _ = clearsign.Decode(input)
    if _, err := b.VerifySignature(keyring, &packet.Config{RSABits: 2048,
Algorithm: packet.PubKeyAlgoRSA}); err != nil {
            fmt.Println("failed to check signature: %s", err)
        return
      }
    fmt.Println("Signature accepted!\n")
}

func main() {
    fmt.Println("Verifying not tampered...")
    verify(no_spoof)
    fmt.Println("Verifying spoofed hash...")
```

```
    verify(hash_spoof)
    fmt.Println("Verifying spoofed cleartext...")
    verify(cleartext_spoof)
}

var no_spoof = []byte(`
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

Message to be signed
-----BEGIN PGP SIGNATURE-----

iQEzBAEBAgAdFiEEAXWUn665cAXgInLZXVs62dBO+u4FAlyeCMMACgkQXVs62dBO
+u6WeQgAvOTZAkwtXCZ2woIbHk+g3fgOiCOF8YtXgZCyDYZgR/JIf1+iCh7lWAjq
9/JcnifNB9lX6hyxy4qoT8loLAHNeoUzSkKiliRMcQFhtfCPInRCRtAnKDfkiA5N
0C9CesJYXoASBRafUgxeI7Q29tVdPNC8WVjJtA72yafu4b63TXKdCcu+TCHtH5lV
l0rqS1JET/+UGycO+gbvegsAoNhmQp8qkFnJTTS6kJgmCs9TJlAmeX1wT8V5f5L+
7pRe45ZBmlA7oi4lylvIp+WG1KJVgrPzeQOkybF2rFRuMxjlvqfO1/4lLrtXgA/7
v8H3ZsqUV9T/HNx5bFPOQJjbOhBVRg==
=Bb6N
-----END PGP SIGNATURE-----
`)

var hash_spoof = []byte(`
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1024

Message to be signed
-----BEGIN PGP SIGNATURE-----

iQEzBAEBAgAdFiEEAXWUn665cAXgInLZXVs62dBO+u4FAlyeCMMACgkQXVs62dBO
+u6WeQgAvOTZAkwtXCZ2woIbHk+g3fgOiCOF8YtXgZCyDYZgR/JIf1+iCh7lWAjq
9/JcnifNB9lX6hyxy4qoT8loLAHNeoUzSkKiliRMcQFhtfCPInRCRtAnKDfkiA5N
0C9CesJYXoASBRafUgxeI7Q29tVdPNC8WVjJtA72yafu4b63TXKdCcu+TCHtH5lV
l0rqS1JET/+UGycO+gbvegsAoNhmQp8qkFnJTTS6kJgmCs9TJlAmeX1wT8V5f5L+
7pRe45ZBmlA7oi4lylvIp+WG1KJVgrPzeQOkybF2rFRuMxjlvqfO1/4lLrtXgA/7
v8H3ZsqUV9T/HNx5bFPOQJjbOhBVRg==
=Bb6N
-----END PGP SIGNATURE-----
`)

var cleartext_spoof = []byte(`
-----BEGIN PGP SIGNED MESSAGE-----` +
"\nHash: SHA512\u000b This data is part of the header\n" +
`
Message to be signed
```

```
-----BEGIN PGP SIGNATURE-----

iQEzBAEBAgAdFiEEAXWUn665cAXgInLZXVs62dBO+u4FAlyeCMMACgkQXVs62dBO
+u6WeQgAvOTZAkwtXCZ2woIbHk+g3fgOiCOF8YtXgZCyDYZgR/JIf1+iCh7lWAjq
9/JcnifNB9lX6hyxy4qoT8loLAHNeoUzSkKiliRMcQFhtfCPInRCRtAnKDfkiA5N
0C9CesJYXoASBRafUgxeI7Q29tVdPNC8WVjJtA72yafu4b63TXKdCcu+TCHtH5lV
l0rqS1JET/+UGycO+gbvegsAoNhmQp8qkFnJTTS6kJgmCs9TJlAmeX1wT8V5f5L+
7pRe45ZBmlA7oi4lylvIp+WG1KJVgrPzeQOkybF2rFRuMxjlvqfO1/4lLrtXgA/7
v8H3ZsqUV9T/HNx5bFPOQJjbOhBVRg==
=Bb6N
-----END PGP SIGNATURE-----
`)

var signingKey = `-----BEGIN PGP PUBLIC KEY BLOCK-----

mQENBFyeB6MBCAC+X0+7sQkrpg4zjQGj9NQSwPvDV5JjWxIXpf1n+mtrZewO8RvR
EO6OnMK/F6mjVKSE3rI9wnpeBoAnNvXgQHY9ckt3qgUq04LTgWoaj89LXi+QjazB
JJPa4cQtoraMtsT2mhIuG88VqPSlgSlvRBGD425kOh+jX7VPIeQIYLJ92G6nVgSV
aIh46n1kme/8PLd8BLFTNmCKr1axUZ118KX/d6y5uB1puPQJ6iZ+YYDk5K+xeQv8
RHyfIcVoGbVL+gR6iwukNxxmNdL6k0DfRwi/qESvOYJ483K1uo+08YvtgULSAKO4
BG/rxKynO4wtcMpe0YSPR+qG0rGF2bZ+trFxABEBAAG0GmNyeXB0byA8Y3J5cHRv
QGNyeXB0by5jb20+iQFUBBMBCAA+FiEEAXWUn665cAXgInLZXVs62dBO+u4FAlye
B6MCGwMFCQPCZwAFCwkIBwIGFQoJCAsCBBYCAwECHgECF4AACgkQXVs62dBO+u6B
rwgArfFSrBiPYQkB9WkaZRyJqJMuYiG9tqbcYYp1Wui9gLPf/IS+iO+6WQGzZ7qp
vdoG45YGajNsxDcd0M7j0VKtq5VYiwF7AWB6aRsJDsdNFmJVgzJYiPTyDfFnx8jr
k1k74TE9ZI2GWpYca0sMT6wCtq8YbmhVB3bty7Zu1L9ahAklhyLpoH4T01NPc2ey
0VhUVQdmKtC0Eqn4tKvWUv4Gx6tGIv4xhZFuDqtoUNbFxvaHZBeURkZJr+jR/mDM
iXE3hpamCzLueBlA8cNJfDKCb0EnK2SngPYBwCSx4MVBNpRPuQveMAtx/o39PCPw
GN9fXHV6mwWFpdoA4RMP59Mqr7kBDQRcngejAQgA7n3wBQewsZYow0DFvGwj+g3m
nCuHqSAEGi1m6zr64dPtDKpR6F4L5nSVoDueki+uQeqz8IwH89+rJIyJZHMHhYD8
MwxdkE+6D9FssY+9kxMZgt50FjXcAFUvlkuDBpJFM8fZRHYyejc4jDw02PC/ssdZ
s5kEcaH00LzecaE+3XM3kQWXMNGePZ0yzwgqNSc3+WjSHvtA71JBJsxYWOUrq0W5
aOz9B/Z4Zcq6KNfRUrI1DVoW6P6qQCGwSrm6zyKxwG/LKQBKJRMYiebQ923iCPDN
9rvOaRfWFn5NH7A2M7PEkky6EWZVZpZTqoDZUJUbgmS9pxcEnPqaWkCvGjHXnQAR
AQABiQE8BBgBCAAmFiEEAXWUn665cAXgInLZXVs62dBO+u4FAlyeB6MCGwwFCQPC
ZwAACgkQXVs62dBO+u665QgAihptwWQFiHPphHxA+LCeRvznBm56/s4nvxyNVKGn
pR1PpN4BVjv0/Tc+4qKJRvAi1kVqmNNCjhpcU8eLQ6enIz7Z2n3VYYbbzG5akvlQ
m8dYWVJWb8FPbBIp9AEG59mFkIz+wXfYJonGWh8+kRDtWAqBLmgvpDsZLPCGQgu0
+HYqht3EiLq7Yv7lw0H2dAYEWzhA/2m1+E43rNBFDxTqflmstux5L02P2JF00COu
oYstzhVvOHJL9nPPdrtbmRHvfm4+QniCAqW9TRzXwOY6P0h2RBf3d9o4Np8Z5JjZ
Rtv1+9ofUzvnkaTr+FXFjvw+baNF1pHMTVcc1f0IoT1Ymg==
=4/3D
-----END PGP PUBLIC KEY BLOCK-----
`
```

5 Version History

Version	Date	Status/Changes	Created by	Responsible
1.0	2019-03-29	Initial report	SEC Consult	SEC Consult
1.1	2019-05-02	Fix verification	SEC Consult	SEC Consult
1.2	2019-05-14	Public report	SEC Consult	SEC Consult