SafeHeron MPC Wallet

Hangzhou Xianbing Technology

07 December 2021

Version: 1.0

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

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

TABLE OF CONTENTS

1 EXECUTIVE SUMMARY

Kudelski Security ("Kudelski", "we"), the cybersecurity division of the Kudelski Group, was engaged by Hangzhou Xianbing Technology ("the Client") to conduct an external security assessment in the form of a code audit of the cryptographic library SafeHeron ("the Product"). The audit was conducted remotely by the Kudelski Security Research Team. The audit took place from October 4th, 2021 to October 15th, 2021 and 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.

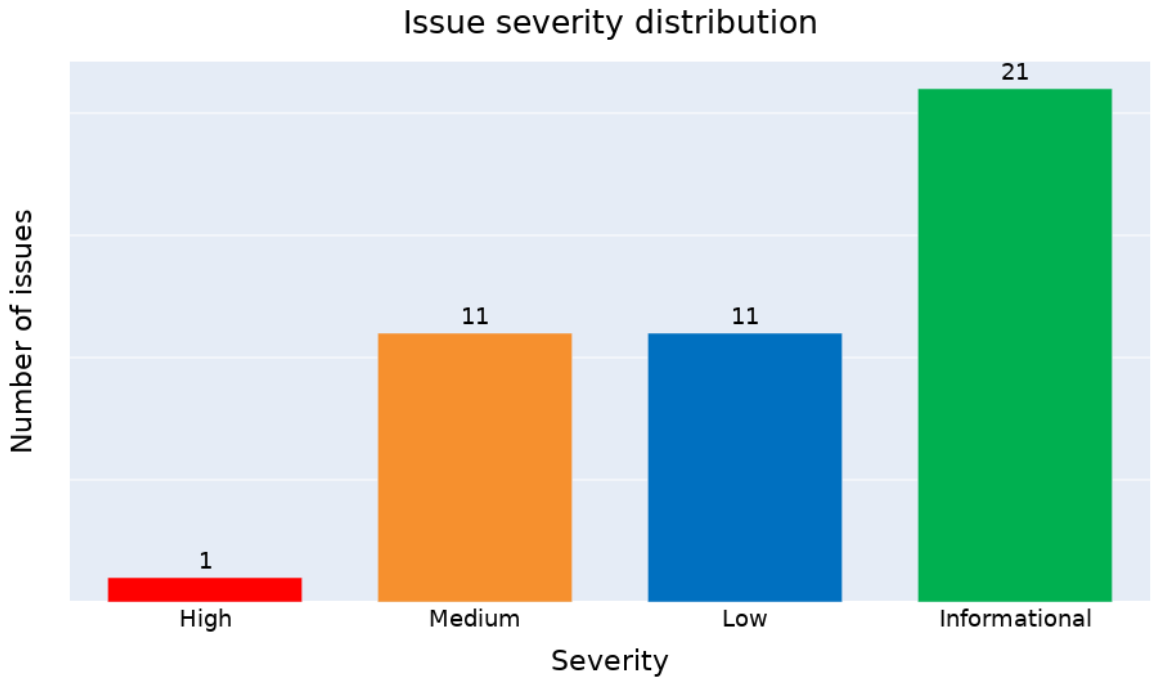
1.1 Engagement Scope

The scope of the audit was a code audit of the Product written in C++ and C, 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 branch `audit` at https://github.com/safeheron/mpc-dsa-lib. We audited the commit number: `b6adc8a0c79c7d03e65b09bd2ff16f1ab2c914f5`. Particular attention was given to side-channel attacks, in particular constant timeness and secure erasure of secret data from memory.

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 report.

As a result of our work, we have identified **1 High**, **11 Medium**, **11 Low** and **21 Informational** findings.



1.3 Issue Summary List

The following security issues were found:

ID	Severity	Finding	Status
KS-SBCF-F-01	**High**	Single-Party Attack On Key Resharing	**Acknowledged**
KS-SBCF-F-02	**Medium**	Paillier Key Generation Uses Unsafe Primes	**Acknowledged**
KS-SBCF-F-03	**Medium**	Not Robust Randomness Generation	**Acknowledged**
KS-SBCF-F-04	**Medium**	Possible OOB Situations	**Acknowledged**
KS-SBCF-F-05	**Medium**	NULL Pointer Dereference Cases	**Acknowledged**
KS-SBCF-F-06	**Medium**	Use Of mini-gmp Library Is Not Constant-Time	**Acknowledged**
KS-SBCF-F-07	**Medium**	No Zeroization Of Sensitive Values	**Acknowledged**
KS-SBCF-F-08	**Medium**	No Curve Point Validation In The VSS Implementation	**Acknowledged**

ID	Severity	Finding	Status
KS-SBCF-F-09	Medium	Unchecked Use Of malloc, calloc And New Operations	Acknowledged
KS-SBCF-F-10	Medium	Paillier Prover Modulus Not Checked	Acknowledged
KS-SBCF-F-11	Medium	Coordinate y is ignored in the computation of the challenge c in both prover and verifier implementation of Schnorr proof	Acknowledged
KS-SBCF-F-12	Medium	Deviation From Standard Paillier Bitsize Of GG18	Acknowledged
KS-SBCF-F-13	Low	Lack Of Hardening Flags Against Binary Explotation	Open
KS-SBCF-F-14	Low	Zero Witness	Acknowledged
KS-SBCF-F-15	Low	Modulo Bias In Mask Generation Of Paillier Proof	Acknowledged
KS-SBCF-F-16	Low	Paillier Input Plaintext Size Not Checked	Acknowledged
KS-SBCF-F-17	Low	Number Of Parties and Threshold Can Be Zero Or Negative	Acknowledged
KS-SBCF-F-18	Low	Small Subgroup Elements And Low Order Points	Acknowledged
KS-SBCF-F-19	Low	mpz_export, mpz_get_str Calls Not Required To Free Result	Acknowledged
KS-SBCF-F-20	Low	Serialization of Blinding Factors Truncate Representation in CreateComWithBlind	Acknowledged
KS-SBCF-F-21	Low	Reduction mod q of the challenge c is not performed	Acknowledged
KS-SBCF-F-22	Low	Enforce curve point validation of pk and g_r in DlogProof::InternalVerify	Acknowledged
KS-SBCF-F-23	Low	Use Of strcpy In Dependency	Acknowledged

The following are observations related to general design and improvements:

ID	Severity	Finding	Status
KS-SBCF-O-01	**Informational**	sign_key_gen, vault_key_gen and sign deviate from the GG18 protocol	**Acknowledged**
KS-SBCF-O-02	**Informational**	The MPC implementation uses an outdated version of protobuf	**Acknowledged**
KS-SBCF-O-03	**Informational**	README.md compilation instructions does not specify which operating system should be used	**Acknowledged**
KS-SBCF-O-04	**Informational**	Dependency installation instructions in README.md are incorrect	**Acknowledged**
KS-SBCF-O-05	**Informational**	Missing dependency makes the compilation process fail	**Acknowledged**
KS-SBCF-O-06	**Informational**	Two tests fail	**Remediated**
KS-SBCF-O-07	**Informational**	Pragma in random number generation implementation of dependency says NOT SUITABLE FOR PRODUCTION USE! during compilation	**Acknowledged**
KS-SBCF-O-08	**Informational**	Comparison between unsigned long and int	**Acknowledged**
KS-SBCF-O-09	**Informational**	Clang is not able to compile the MPC implementation	**Acknowledged**
KS-SBCF-O-10	**Informational**	The Trezor cryptographic library is not updated to last version	**Acknowledged**
KS-SBCF-O-11	**Informational**	The Trezor cryptographic library contains algorithms that are not used by the MPC implementation	**Acknowledged**
KS-SBCF-O-12	**Informational**	It is not possible to easily update the mini-gmp library	**Acknowledged**
KS-SBCF-O-13	**Informational**	Rely on an audited, constant-time, cryptographic library	**Acknowledged**

ID	Severity	Finding	Status
KS-SBCF-O-14	**Informational**	Multiple methods are commented in bip32.h	**Acknowledged**
KS-SBCF-O-15	**Informational**	Empty classes	**Acknowledged**
KS-SBCF-O-16	**Informational**	Make the wallet easy to configure according to the selected curve and/or add clarification on the curve used	**Acknowledged**
KS-SBCF-O-17	**Informational**	Utilize the original name of the rounds for clarification	**Acknowledged**
KS-SBCF-O-18	**Informational**	Comments in the SqrtM implementation	**Acknowledged**
KS-SBCF-O-19	**Informational**	The share conversion protocols (MtA and MtAwc) do not perform the range proofs	**Acknowledged**
KS-SBCF-O-20	**Informational**	Message class attribute is_share_encrypted_ is never used	**Acknowledged**
KS-SBCF-O-21	**Informational**	Missing reference to chosen number of iterations in Paillier proof	**Acknowledged**

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: Single-Party Attack On Key Resharing

Severity: High

Status: Acknowledged

Location: src/mpc_hd_ecdsa/sign_key_refresh

Description

The GG18 original paper (see [1]) does not offer a protocol for key resharing. Despite this, many implementations add their own key resharing functionality by running a new distributed protocol where the Shamir shares of the old key are "re-randomized" with new polynomials involving all new parties, and the success of the procedure is ensured by a VSS, e.g., Feldmann. In https://eprint.iacr.org/2020/1052.pdf it was pointed out that this is not secure, and mitigation strategies are discussed. Unfortunately, Kudelski found out during a previous audit that these mitigations are not sufficient, and can actually make things worse, by allowing a single malicious attacker to "segment" the new committee into two pools both of them not having enough valid shares to reconstruct the secret without the collaboration of the malicious party. This would allow a single malicious party to lock or delete funds, and to blackmail the rest of the committee. The details of the attack can be seen at https://research.kudelskisecurity.com/2021/04/08/audit-of-ings-threshold-ecdsa-library-and-a-dangerous-vulnerability-in-existing-gennaro-goldfeder18-implementations.

Recommendation

The best defense against this type of attack is to implement a robust broadcast channel, e.g. through a trusted node acting as a relay. Alternatively, the protocol must be modified by inserting further checks in a peer-to-peer way.

Notes

Hangzhou Xianbing Technology has proposed a solution, dubbed 'Double ACK', which is claimed to guarantee that the funds remain secure if no more than $t-1$ parties are

corrupt and no less than t parties are honest. Unfortunately we found the proposed solution insecure, and we think that further remediation might be required. More details on the specific attack are given below.

The Double ACK solution works by adding a second final round of acknowledgement. In the first round, as in the (also vulnerable) solution proposed by another third-party library product, every participant sends either an ACK or NON-ACK message to every other peer, according to their own (local) verification of the secret sharing scheme. In the Double ACK scheme, after this phase, a second round is introduced, where every participant sends to each other participant a complete list of the PeerID/ACKstatus pairs received during the first round (including their own), and the protocol is completed successfully if and only if all the lists (views of parties) agree.

First of all, for the sake of the correctness of the protocol and in order to avoid trivial network attackers, we think the following modifications are necessary:

1. Change the success condition from “all the participant’s views must agree” to “all the entries in every participant’s view must be ACK”. This was not explicitly mentioned in the protocol proposal we received, but we think it’s necessary that no single NON-ACK is ever observed for the resharing protocol to be considered successful, not only against malicious participants but also, e.g., against network errors or other software flaws.

2. The ACK tables (participants’ views) should be authenticated in order to avoid manipulation in-transit.

Moreover, we observe that the proposed protocol modification is likely to add considerable latency: In addition to introducing an extra round, it has a communication complexity overhead that scales quadratically with the number of participants. We are not able to assess the performance impact in the Client’s use case, but this might be unavoidable in order to achieve the desired level of security.

But, most importantly, the Double ACK solution fails to achieve the claimed security guarantees. In order to show this, we provide an example of an attack, where a single malicious party during the key reshare phase is able to compromise entirely the security of the solution, being able to lock or delete funds, or to blackmail the honest committee.

Let's suppose the initial deployment of the TSS scheme is in a committee with four parties: Alice, Bob, Charlie, and Dave, in a 3-out-of-4 threshold setting. The assumption is that at least 3 of them are honest, but that doesn't matter for the following attack, they could also be all 4 honest. The committee wants now to add another member to the pool, Eve, in a 3-out-of-5 configuration. Unfortunately, E is malicious.

The parties engage in the resharing protocol, and at the end everyone checks that the protocol was (locally) successful, i.e., their new secret share is consistent. Supposing there is no network errors or software issues, in the first confirmation round every party sends "ACK" to every other party (each party sends 4 "ACK", so 20 entries sent total).

In the second confirmation round, according to the Double ACK solution, each party collects a table (local view) of the ACK messages received by every party, including themselves (1 table = 5 entries partyID/ACKstatus), and each party sends their table to each other party (each party sends 4 tables with 5 entries each, so $4 \cdot 4 \cdot 5 = 80$ entries sent total).

Here comes the attack: Eve sends a table full of "ACK" entries to Alice and Bob, but sends a table with some "non-ACK" entries to Charlie and Dave. Now, Alice and Bob think that the protocol worked well, so they discard their old share and move to the next share, but Charlie and Dave think something went wrong, so they discard the new share and stick to the old one. This way, Eve has managed to "segment" the committee into two pools, each of them not having enough valid shares to reconstruct the secret, so Eve (a single malicious party) can erase or withhold funds or blackmail the committee.

Notice that is not even possible for the honest parties to detect who cheated (Eve) without additional rounds of verification. Notice also that this attack works with any threshold configuration and number of parties, as long as the threshold is at least the majority of the parties (which is anyway a minimum condition to ensure security against a dishonest minority).

After validating the above attack, Hangzhou Xianbing Technology has proposed a new solution, dubbed 'Improved ACK', which aims at solving the vulnerability by bundling together signed Schnorr proofs of share commitment received by each party before broadcasting these bundles together for a double verification. The solution is involved, and we did not validate its safety within the scope of this report.

It is important to stress that finding a practical, fully decentralized solution to secret resharing for Threshold ECDSA protocols derived from [1] is highly nontrivial - and, we think, this is the reason why [1] does not propose such a solution in the first place. The only case where this can happen easily is 2-out-of-2 multiparty ECDSA schemes such as [2], which in fact does propose such a protocol. This works because if there is only 2 parties then there cannot be dishonest minority for the protocol to succeed, so in a certain sense they rely on stronger security guarantees.

2.2 KS-SBCF-F-02: Paillier Key Generation Uses Unsafe Primes

Severity: Medium

Status: Acknowledged

Location: src/pail/pail.cpp:18

Description

There is no parameter validation in the Paillier keypair generation implementation. Namely, the following checks are mandatory:

1. The Paillier cryptosystem relies on the StrongRSA assumption and requires safe primes. Safe primes ($q = 2p + 1$) are not used when generating the Paillier key pair in pail.cpp, line 18. Instead p, q are generated independently via the Rand::RandomPrimeStrict function:

```
2737
2738
2739 BN Rand::RandomPrimeStrict(size_t byteSize) {
2740   BN n;
2741   int time = 0;
2742   do{
2743     n = RandomBNStrict(byteSize);
2744     time ++ ;
2745   }while (!n.IsProbablyPrime());
2746  //std::cout << "times: " << time << std::endl;
2747
```

```
2748   return n;
2749 }
2750
2751 BN Rand::RandomBNStrict(size_t byteSize) {
2752   auto * buf = (unsigned char )calloc(1, byteSize);
2753   while((buf[0] & 0x80) == 0){
2754     RandomBytes(buf, byteSize);
2755   }
2756
2757   BN n = BN::FromBytesBE(buf, byteSize);
2758   free(buf);
2759   return n;
2760 }
2761
```

2. For $N = q \cdot p$ primes, check that $p - q$ is at least 1020 bits to avoid square root attacks.

Recommendation

Implement these checks to avoid running the MPC implementation with incorrect and/or unsafe parameters.

Notes

Hangzhou Xianbing Technology communicated to us that they have addressed this issue according to our recommendations.

2.3 KS-SBCF-F-03: Not Robust Randomness Generation

Severity: Medium

Status: Acknowledged

Location: src/rand/rand.cpp

Description

RandomBytes returns a desired amount of bytes from /dev/urandom. However, two issues are present:

1. If RandomBytes cannot open /dev/urandom it returns false. However, the returned value is never checked across src/rand/rand.cpp.
2. If /dev/urandom opening succeeds, there is no check on the amount of bytes read, i.e., it is not checked that the desired amount of bytes is actually read.

```
11
12 bool Rand::RandomBytes(unsigned char buf, size_t size) {
13   int fd = open("/dev/urandom", O_RDONLY);
14   if(-1 == fd){
15     close(fd);
16     return false;
17   }
18
19   size_t curLen = 0;
20   while(curLen < size){
21     ssize_t len = read(fd, buf + curLen, size - curLen);
22     curLen += len;
23   }
24
25   close(fd);
26   return true;
27 }
28
```

This means that all the generated randomness in the MPC implemented is never validated.

Recommendation

Perform the two checks above or use a different implementation. For example, check the LibreSSL getentropy_urandom function as cited by https://github.com/veo

rq/cryptocoding in https://github.com/veorq/cryptocoding#use-strong-randomness. In C/C++ a suggestion is to depend on one of the following functions: getrandom, BCryptGenRandom, and OpenSSL RAND_bytes.

Notes

Hangzhou Xianbing Technology communicated to us that they have addressed this issue according to our recommendations.

2.4 KS-SBCF-F-04: Possible OOB Situations

Severity: Medium

Status: Acknowledged

Location: ntl/bn.cpp:537, ntl/bn.cpp:551, 3rdparty/base64/base64.c, 3rdparty/base64/base64.h

Description

In the following functions, overflows and OOB write operations might happen if bounds are not checked:

- In the ToBytes32BE and ToBytes32LE the size of buf32 is never checked. Moreover, the function memcpy and memset are used, which do not check bounds. Further, the input parameter buf32 is never validate against NULL, which could create a NULL pointer dereference situation.
- base64_encode and base64_decode functions do not validate the length of the output parameter. Both functions ignore the size of the output parameter out. That means that if in and out input parameter do not have the same length, a memory corruption might happen.

```
72 unsigned int
73 base64_encode(const unsigned char *in, unsigned int inlen, char *out)
74  {
75       int s;
76       unsigned int i;
77       unsigned int j;
```

```
78      unsigned char c;
79      unsigned char l;
80
81      s = 0;
82      l = 0;
83      for (i = j = 0; i < inlen; i++) {
84          c = in[i];
85
86          switch (s) {
87          case 0:
88              s = 1;
89              out[j++] = base64en[(c >> 2) & 0x3F];
90              break;
91          case 1:
92              s = 2;
93              out[j++] = base64en[((l & 0x3) << 4) | ((c >> 4) & 0xF)];
94              break;
95          case 2:
96              s = 0;
97              out[j++] = base64en[((l & 0xF) << 2) | ((c >> 6) & 0x3)];
98              out[j++] = base64en[c & 0x3F];
99              break;
100         }
101         l = c;
102     }
103
104     switch (s) {
105     case 1:
106         out[j++] = base64en[(l & 0x3) << 4];
107         out[j++] = BASE64_PAD;
108         out[j++] = BASE64_PAD;
109         break;
110     case 2:
111         out[j++] = base64en[(l & 0xF) << 2];
```

```
112             out[j++] = BASE64_PAD;
113             break;
114         }
115 
116         out[j] = 0;
117 
118         return j;
119     }
120 
121     unsigned int
122     base64_decode(const char *in, unsigned int inlen, unsigned char *out)
123     {
124         unsigned int i;
125         unsigned int j;
126         unsigned char c;
127 
128         if (inlen & 0x3) {
129             return 0;
130         }
131 
132         for (i = j = 0; i < inlen; i++) {
133             if (in[i] == BASE64_PAD) {
134                 break;
135             }
136             if (in[i] < BASE64DE_FIRST || in[i] > BASE64DE_LAST) {
137                 return 0;
138             }
139 
140             c = base64de[(unsigned char)in[i]];
141             if (c == 255) {
142                 return 0;
143             }
144 
145             switch (i & 0x3) {
```

```
146             case 0:
147                 out[j] = (c << 2) & 0xFF;
148                 break;
149             case 1:
150                 out[j++] |= (c >> 4) & 0x3;
151                 out[j] = (c & 0xF) << 4;
152                 break;
153             case 2:
154                 out[j++] |= (c >> 2) & 0xF;
155                 out[j] = (c & 0x3) << 6;
156                 break;
157             case 3:
158                 out[j++] |= c;
159                 break;
160             }
161         }
162 
163         return j;
164     }
```

Recommendation

Validate the length of both parameters accordingly to avoid buffer overflows. Verify that buf32 is not NULL. Check bounds of the input parameter and/or utilize a safe alternative to memset and memcpy.

Notes

Hangzhou Xianbing Technology communicated to us that they have addressed this issue according to our recommendations.

2.5 KS-SBCF-F-05: NULL Pointer Dereference Cases

Severity: Medium

Status: Acknowledged

Location: 3rdparty/base64/base64.c:73, 3rdparty/base64/base64.c:122,

src/common/utils.c:9, src/common/utils.c:61, curve_point.cpp:293, curve_point.cpp:319, curve_point.cpp:331, curve_point:355, curve_point:366, curve_point:370

Description

There is no validation of the input and output pointers in the following functions:

- In both base64_encode and base64_decode functions.
- In the following methods defined in utils.c:

```
2737 int hex2bin(const char *str, uint8_t *bytes, size_t blen) {
2738     size_t pos;
2739     uint8_t idx0;
2740     uint8_t idx1;
2741 
2742 ...
2743 
2744     memset(bytes, 0, blen);
2745     for (pos = 0; ((pos < (blen * 2)) && (pos < strlen(str))); pos += 2)
         ↪ {
2746         idx0 = (uint8_t) str[pos + 0];
2747         idx1 = (uint8_t) str[pos + 1];
2748         bytes[pos / 2] = (uint8_t)(hashmap[idx0] << 4) | hashmap[idx1];
2749 
2750 int bin2hex(const uint8_t *bytes, size_t blen, char *str, size_t slen) {
2751     char ch[17] = "0123456789abcdef";
2752     for (size_t i = 0; (i < blen) && (i * 2 + 1 < slen); ++i) {
2753         int low = bytes[i] & 0x0F;
2754         int high = (bytes[i] >> 4) & 0x0F;
2755         str[i * 2] = ch[high];
2756         str[i * 2 + 1] = ch[low];
2757     }
2758    return 1;
2759 }
```

- Finally, in the EncodeCompressed, DecodeCompressed, DecodeFull, EncodeFull, EncodeEdwardPoint and DecodeEdwardsPoint functions there is no input validation of the pointer pub33. If it is NULL, a NULL pointer dereference will happen. Moreover, there is no check to ensure that the point being decoded/compressed belongs to the curve.

Recommendation

Check that the input and output pointers are not NULL.

Notes

Hangzhou Xianbing Technology communicated to us that they have addressed this issue according to our recommendations.

2.6 KS-SBCF-F-06: Use Of mini-gmp Library Is Not Constant-Time

Severity: Medium

Status: Acknowledged

Location: All files in 3rdparty/mini-gmp/ and the functions imported from bn.cpp

Description

The GCD implementations provided by mini-gmp (e.g. mpn_gcd_11, mpn_gcd, mpn_gcdext), inversion, and exponentiation are not constant-time. Moreover, the GMP library is known for not providing constant-time arithmetic functions, as it is optimized for speed rather than security.

See for instance the exponentiation function below. It leaks the exponent via a timing side channel:

```
2737 void
2738 mpz_pow_ui(mpz_t r, const mpz_t b, unsigned long e) {
2739     unsigned long bit;
2740     mpz_t tr;
2741     mpz_init_set_ui(tr, 1);
2742
2743     bit = GMP_ULONG_HIGHBIT;
```

```
2744    do {
2745      mpz_mul(tr, tr, tr);
2746      if (e & bit)
2747        mpz_mul(tr, tr, b);
2748     bit >>= 1;
2749    } while (bit > 0);
2750
2751    mpz_swap(r, tr);
2752    mpz_clear(tr);
```

The arithmetic implementations provided by mini-gmp are imported from bn.cpp.

Recommendation

Use an arithmetic library that provides constant-time implementations.

Notes

Hangzhou Xianbing Technology communicated to us that they have addressed this issue according to our recommendations.

2.7 KS-SBCF-F-07: No Zeroization Of Sensitive Values

Severity: Medium

Status: Acknowledged

Location: mpc_hd_ecdsa/, 3rdparty/crypto/memzero.c, zkp/dlog_proof.cpp, zkp/pail_proof.cpp, /src/mpc_hd_ecdsa/sign/sign_once/round3.cpp, heg_proof.cpp.

Description

Sensitive values should be erased from memory after use, particularly:

- The secrets after share distribution in the VSS implementation.
- The variables containing the Paillier private key after use.
- In DLogProof::InternalProveWithR, the nonce r.
- The values generated during MtA in round 3 after they are sent e.g. the γ values.
- The intermediate values utilized during the ElGamal proof: s_1, s_2 and the witness structure.

Recommendation

You can use the memzero implementation of the Trezor library available at 3rd-party/crypto/memzero.c. There is a good explanation of this issue at https://www.cryptologie.net/article/419/zeroing-memory-compiler-optimizations-and-memset_s/.

Notes

Hangzhou Xianbing Technology communicated to us that they have addressed this issue according to our recommendations.

2.8 KS-SBCF-F-08: No Curve Point Validation In The VSS Implementation

Severity: Medium

Status: Acknowledged

Location: polynomial.cpp:108, curve_point.cpp:87

Description

For every operation where the party expects a curve point as a result of another party operation, checks that this point belongs to the curve are not performed as expected. Invalid points such as $(0, 0)$ and points that not belong to the curve can create interoperability and security problems.

Morever the CurvePoint constructor does not check if the input points belong to the curve. Further, the constructor contains the following comment:

```
87 // not suggested, not safe
```

Recommendation

Enable curve point validationy, also at the CurvePoint constructor in curve_point.cpp:87.

Notes

Hangzhou Xianbing Technology communicated to us that they will update the usage information for the constructor to avoid wrong use.

2.9 KS-SBCF-F-09: Unchecked Use Of malloc, calloc And New Operations

Severity: Medium

Status: Acknowledged

Location: encode/base64.cpp:11, encode/base64.cpp:22, rand/rand.cpp:27, rand/rand.cpp:35, zkp/pail_proof.cpp:43

Description

Not validating the result of a malloc/calloc/new operation and attempting to free a NULL pointer can create memory corruption issues.

Recommendation

Always check that the result of a malloc/calloc/new operation is not NULL. Otherwise, abort.

Notes

Hangzhou Xianbing Technology communicated to us that they have addressed this issue according to our recommendations.

2.10 KS-SBCF-F-10: Paillier Prover Modulus Not Checked

Severity: Medium

Status: Acknowledged

Location: pail_proof.cpp:111

Description

The verification part of the Paillier proof (according to https://eprint.iacr.org/2018/057.pdf, p. 6) requires checking the received Paillier modulus. However, this is not performed in the verification part of the proof in the code.

Recommendation

Ensure that the prover modulus is correct.

Notes

Hangzhou Xianbing Technology communicated to us that they have addressed this issue according to our recommendations.

2.11 KS-SBCF-F-11: Coordinate y is ignored in the computation of the challenge c in both prover and verifier implementation of Schnorr proof

Severity: Medium

Status: Acknowledged

Location: zkp/dlog_proof:35, zkp/dlog_proof:58

Description

The coordinate y of the points g_r, g and pk are not used in the computation of the challenge c and are ignored.

```
35 void DLogProof::InternalProveWithR(const BN &sk, const CurvePoint &g,
    ↪ const BN &order, const BN &r) {
36     g_r_ = g * r;
37     pk_ = g * sk;
38
39     // c = H(G || g^r || g^sk || UserID || OtherInfo)
40     uint8_t sha256_digest[SHA256_DIGEST_LENGTH];
41     SHA256_CTX sha_ctx;
42     sha256_Init(&sha_ctx);
43     uint8_t buf32[32];
44     g_r_.x().ToBytes32BE(buf32);
45     sha256_Update(&sha_ctx, buf32, 32);
46     g.x().ToBytes32BE(buf32);
47     sha256_Update(&sha_ctx, buf32, 32);
48     pk_.x().ToBytes32BE(buf32);
49     sha256_Update(&sha_ctx, buf32, 32);
50     sha256_Final(&sha_ctx, sha256_digest);
51     BN c = BN::FromBytesBE(sha256_digest, 32);
```

```
52
53      // res = r - sk * c mod n
54      BN skc = (sk * c) % order;
55      res_ = (r - skc) % order;
56  }
57
58  bool DLogProof::InternalVerify(const CurvePoint &g) const {
59      // c = H(G || g^r || g^sk || UserID || OtherInfo)
60      uint8_t sha256_digest[SHA256_DIGEST_LENGTH];
61      SHA256_CTX sha_ctx;
62      sha256_Init(&sha_ctx);
63      uint8_t buf32[32];
64      g_r_.x().ToBytes32BE(buf32);
65      sha256_Update(&sha_ctx, buf32, 32);
66      g.x().ToBytes32BE(buf32);
67      sha256_Update(&sha_ctx, buf32, 32);
68      pk_.x().ToBytes32BE(buf32);
69      sha256_Update(&sha_ctx, buf32, 32);
70      sha256_Final(&sha_ctx, sha256_digest);
71      BN c = BN::FromBytesBE(sha256_digest, 32);
```

Recommendation

Even if the probability of collision is small, include the coordinate y of the points in the computation of the challenge c in order to avoid interoperability problems.

Notes

Hangzhou Xianbing Technology communicated to us that they have addressed this issue according to our recommendations.

2.12 KS-SBCF-F-12: Deviation From Standard Paillier Bitsize Of GG18

Severity: Medium

Status: Acknowledged

Location: src/pail/pail.cpp:19

Description

[1] has been designed for 2,048 bits Paillier primes and hash functions of 256-bit output. However, pail.cpp, CreateKeyPair allows different key bits:

```
18 void CreateKeyPair(PailPrivKey &priv, PailPubKey &pub, int key_bits) {
19     assert(key_bits == 1024 || key_bits == 2048 || key_bits == 3072 ||
   ↪  key_bits == 4096);
```

This can introduce security issues: There is very strict correlations between the curve size, the Paillier modulo size, and the hash size. If these correlations are not respected, vulnerabilities are introduced that can leak secrets or make proofs invalid.

Recommendation

Only support the security level the scheme has been designed for.

Notes

Hangzhou Xianbing Technology communicated to us that they want to design a common library supporting different key sizes. They acknowledged that 2,048 bit are typically used in threshold schemes.

2.13 KS-SBCF-F-13: Lack Of Hardening Flags Against Binary Exploitation

Severity: Low

Status: Open

Location: CMakeLists.txt

Description

C/C++ is sensible to memory corruption bugs and exploitation techniques. However, in the last few years several protections been implemented at the operating system level. They should be enabled at compilation time.

Recommendation

Some of the hardening flags we recommend are:

- Compile-time protection against static sized buffer overflow
- Stack protector
- Full ASLR for executables
- Read-only segments after relocation

As a reference, you can check https://wiki.debian.org/Hardening#User_Space and https://developers.redhat.com/blog/2018/03/21/compiler-and-linker-flags-gcc.

2.14 KS-SBCF-F-14: Zero Witness

Severity: Low

Status: Acknowledged

Location: heg_proof.cpp:62

Description

In order to compute z_1 as $z_1 = s_1 + x \cdot e$, the witness x is compared against 0. If x is zero, $z_1 = s_1$. There are two issues here:

1. The zero witness should be checked and discarded as invalid.
2. This conditional branch introduces a timing side-channel that allows to detect when the witness is 0.

However, the probability of obtaining 0 in a randomly generated witness is negligible.

```
72      // z1 = s1 + x * e
73      BN z1 = s1;
74      if (witness.x_ != 0) {
75          z1 = s1 + witness.x_ * e;
76      }
```

Recommendation

We recommend discarding a zero witness as invalid, If this is not possible or undesirable, let the arithmetic layer deal with the computation of z_1 so it is not possible to guess when the witness value x is equal to 0.

Notes

Hangzhou Xianbing Technology communicated to us that they have addressed this issue according to our recommendations.

2.15 KS-SBCF-F-15: Modulo Bias In Mask Generation Of Paillier Proof

Severity: Low

Status: Acknowledged

Location: pail_proof.cpp:87

Description

There is a modulo bias in the generation of the Paillier Proof, where the output of the SHA-256 operation is reduced mod N. This can reduce the entropy of the proof.

Recommendation

Our recommendation is to use either a wider mask or rejection sampling as described in https://research.kudelskisecurity.com/2020/07/28/the-definitive-guide-to-modulo-bias-and-how-to-avoid-it/.

Notes

Hangzhou Xianbing Technology communicated to us that they have addressed this issue according to our recommendations.

2.16 KS-SBCF-F-16: Paillier Input Plaintext Size Not Checked

Severity: Low

Status: Acknowledged

Location: src/pail/pail_privkey.cpp:100

Description

If the plaintext it is too large, the reduction mod p can make the decryption process fail.

Recommendation

Validate the size of the plaintext before decryption.

Notes

Hangzhou Xianbing Technology communicated to us that, in the intended use of their product, it is the responsability of the invoker to confirm that the plaintext is not too long.

2.17 KS-SBCF-F-17: Number Of Parties and Threshold Can Be Zero Or Negative

Severity: Low

Status: Acknowledged

Location: mpc_context.h:86, mpc_round:18, polynomial.cpp:24

Description

The MPC context class allows to define a negative number of parties since it stores the number of parties in an int. Moreover, the method Polynomial::CreateRandomPolynomial allows a threshold 0 and possibly, negative numbers, thus creating an empty polynomial that is considered valid by the method. In this case, the user is not warned:

```
24 Polynomial Polynomial::CreateRandomPolynomial(const BN &secret, int
    ↪ threshold, const BN &prime) {
25     vector<BN> vecCoe;
26     for(int i = 1; i < threshold; ++i){
27         BN coe = Rand::RandomBNLt(prime);
28         vecCoe.push_back(coe);
29     }
30     return Polynomial(secret, vecCoe, prime);
```

```
31 }
```

Recommendation

Validate both the number of parties that can run the protocol and the threshold. Warn the user about incorrect values and abort.

Notes

Hangzhou Xianbing Technology communicated to us that they have addressed this issue according to our recommendations.

2.18 KS-SBCF-F-18: Small Subgroup Elements And Low Order Points

Severity: Low

Status: Acknowledged

Location: curve/curve_point.cpp

Description

The classes implemented in curve/curve_point.cpp and the CurveType::ED25519 definition suggest that the Ed25519 curve could be used in the MPC implementation. In this case, there is no check of small subgroup elements and low order points, which can lead to attacks.

Recommendation

Verify that the points belong to the prime-order subgroup.

References:

- https://www.blackhat.com/us-21/briefings/schedule/#baby-sharks-small-subgroup-attacks-to-disrupt-large-distributed-systems-22608
- https://eprint.iacr.org/2020/823.pdf

Further, Reject low order points, see for instance:

- https://github.com/Kuska-ssb/handshake/issues/39

Notes

Hangzhou Xianbing Technology communicated to us that they have addressed this issue according to our recommendations.

2.19 KS-SBCF-F-19: mpz_export, mpz_get_str Calls Not Required To Free Result

Severity: Low

Status: Acknowledged

Location: ntl/bn.cpp

Description

Across ntl/bn.cpp there are free calls to the output of the mpz_export and mpz_get_str functions. However, it is not clear if that must be freed by the user via free() or using the GMP API.

Recommendation

Ensure that the values returned by these functions required a call to free. More important, check that the values returned by the functions is not NULL before attempting to free memory.

Notes

Hangzhou Xianbing Technology communicated to us that they have addressed this issue according to our recommendations.

2.20 KS-SBCF-F-20: Serialization of Blinding Factors Truncate Representation in CreateComWithBlind

Severity: Low

Status: Acknowledged

Location: src/commitment/commitment.cpp:12, src/ntl/bn.cpp:500

Description

Big numbers are mapped into arrays of 32 bytes when creating the commitments e.g.

```
ntl::BN Commitment::CreateComWithBlind(ntl::BN &num, ntl::BN &blind_factor) {
    uint8_t sha256_digest[SHA256_DIGEST_LENGTH];
    SHA256_CTX sha_ctx;
    sha256_Init(&sha_ctx);
    uint8_t buf32[32];
    num.ToBytes32BE(buf32);
    sha256_Update(&sha_ctx, buf32, 32);
    blind_factor.ToBytes32BE(buf32);
    sha256_Update(&sha_ctx, buf32, 32);
    sha256_Final(&sha_ctx, sha256_digest);
    return BN::FromBytesBE(sha256_digest, 32);
}

ntl::BN Commitment::CreateComWithBlind(curve::CurvePoint &point, BN &blind_factor) {
    uint8_t sha256_digest[SHA256_DIGEST_LENGTH];
    SHA256_CTX sha_ctx;
    sha256_Init(&sha_ctx);
    uint8_t buf32[32];
    point.x().ToBytes32BE(buf32);
    sha256_Update(&sha_ctx, buf32, 32);
    point.y().ToBytes32BE(buf32);
    sha256_Update(&sha_ctx, buf32, 32);
    blind_factor.ToBytes32BE(buf32);
    sha256_Update(&sha_ctx, buf32, 32);
    sha256_Final(&sha_ctx, sha256_digest);
    return BN::FromBytesBE(sha256_digest, 32);
}
```

However, the BN types could be bigger than 32 bytes but this case is ignored by

ToBytes32BE:

```
500 void BN::ToBytes32BE(uint8_t * buf32) const {
501     size_t len = 0;
502     uint8_t * ch = (uint8_t ) mpz_export(NULL, &len, 1, sizeof(char), 1,
        ↪  0, mpz_data_);
503     memset(buf32, 0, 32);
504     if (len < 32) {
505       uint8_t * des = buf32 + 32 - len;
506       memcpy(des, ch, len);
507     } else {
508 
509        uint8_t *src = ch + len - 32;
510        memcpy(buf32, src, 32);
511     }
512     free(ch);
513 }
```

Since in that case, the size of ch is truncated to 32. This means, that when creating the commitment, hashing a value with a blinding factor of length 32 and another greater in length but sharing the same content in the first 32 bytes, will produce the same digest.

Recommendation

Do not truncate the length of the blinding factors to avoid collisions when generating commitments.

Notes

Hangzhou Xianbing Technology communicated to us that they have addressed this issue according to our recommendations.

2.21 KS-SBCF-F-21: Reduction mod q of the challenge c is not performed

Severity: Low

Status: Acknowledged

Location: zkp/dlog_proof:35, zkp/dlog_proof:58

Description

The challenge c after the hashing operation is not reduced mod q. This could create interoperability problems.

Recommendation

Reduce the challenge c after the hashing operation mod q.

Notes

Hangzhou Xianbing Technology communicated to us that they have addressed this issue according to our recommendations.

2.22 KS-SBCF-F-22: Enforce curve point validation of pk and g_r in DlogProof::InternalVerify

Severity: Low

Status: Acknowledged

Location: zkp/dlog_proof:58

Description

The verification function DLogProof::InternalVerify does not check if the input points belong to the curve. This could create interoperability problems if the prover is sending points that are not part of the curve.

Recommendation

Check that the input points belong to the curve, otherwise, abort.

Notes

Hangzhou Xianbing Technology communicated to us that they have addressed this issue according to our recommendations.

2.23 KS-SBCF-F-23: Use Of strcpy In Dependency

Severity: Low

Status: Acknowledged

Location: 3rdparty/crypto/bip39.c:85, 3rdparty/crypto/bip39.c:227, 3rdparty/crypto/bip39.c:228

Description

In C, the strcpy function does not check array boundaries and makes code prone to buffer overflows. Particularly, in bip39.c there are the following calls to strcpy:

```
2737 strcpy(p, wordlist[idx]);
2738 strcpy(bip39_cache[bip39_cache_index].mnemonic, mnemonic);
2739 strcpy(bip39_cache[bip39_cache_index].passphrase, passphrase);
```

Recommendation

If you plan to use this module in the future, adapt the implementation with a safe alternative to strcpy such as strlcpy. Otherwise, only take the implementations that you need from the Trezor cryptographic library.

Notes

Hangzhou Xianbing Technology communicated to us that they have addressed this issue according to our recommendations.

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: sign_key_gen, vault_key_gen and sign deviate from the GG18 protocol

Status: Acknowledged

Location: src/mpc_hd_ecdsa/sign, src/mpc_hd_ecdsa/sign_key_gen, src/mpc_hd_ecdsa/vault_key_gen

Description

Multiple operations described in [1] are performed in different rounds and/or not strictly followed, for instance:

- vault_key_gen: Paillier public keys are not broadcasted in round 0. In this case, the implementation deviate from the protocol. Paillier public keys are broadcasted before the MPC process.

In this respect, only the commitments are computed:

```
bool Round0::MakeMessage(std::vector<std::string> &out_msg_arr,
 ↪  std::vector<std::string> &out_des_arr) const {
    Context *ctx = dynamic_cast<Context *>(this->get_mpc_context());
    Vault &vault = ctx->vault_;

    out_msg_arr.clear();
    out_des_arr.clear();

    for (size_t i = 0; i < ctx->remote_parties_.size(); ++i) {
```

```
        out_des_arr.push_back(vault.remote_parties_[i].party_id_);

        Message0 message;
        message.kgc_chain_code_share_ =
↪   ctx->local_party_.kgc_chain_code_share_;
        message.kgc_y_ = ctx->local_party_.kgc_y_;
        string base64;
        message.ToBase64(base64);
        out_msg_arr.push_back(base64);
    }

    return true;
}
```

Moreover, in this operation, secret sharing is applied to the chain code (BIP32).

- Hangzhou Xianbing Technology integrates BIP32 and [1] in the MPC rounds. Particularly, round 0-2 derive child key shares from the parent shares according to BIP32. On the other hand, rounds 2-11 perform GG18, taking into account that the parties also deal with shares of the BIP32 offset.

Recommendation

Hangzhou Xianbing Technology deviates from the GG18 protocol. We haven't verified if the security assumption and proofs of [1] hold after the changes Hangzhou Xianbing Technology has introduced.

Notes

Hangzhou Xianbing Technology communicated to us that they will provide a security verification later.

3.2 KS-SBCF-O-02: The MPC implementation uses an outdated version of protobuf

Status: Acknowledged

Location: README.md

Description

According to README.md, the protobuf library version 3.7.1 is needed. However, the last version at the time of writing is 3.18.0.

```
### 2.1 Installation

$ sudo apt-get install autoconf automake libtool curl make g++ unzip

$ git clone https://github.com/protocolbuffers/protobuf.git
$ cd protobuf
$ git checkout v3.7.1
$ git submodule update --init --recursive
$ ./autogen.sh

$ ./configure
$ make
$ make check
$ sudo make install
$ sudo ldconfig # refresh shared library cache.
```

Recommendation

Update the implementation to rely on the last version of protobuf.

Notes

Hangzhou Xianbing Technology communicated to us that they will use this particular version.

3.3 KS-SBCF-O-03: README.md compilation instructions does not specify which operating system should be used

Status: Acknowledged

Location: README.md

Description

Several calls to apt-get suggest the utilization of a Debian-based OS, however this is not clarified in the compilation instructions.

Recommendation

Clarify the compilation process and detail the required OS.

Notes

Hangzhou Xianbing Technology communicated to us that the recommended operating systems are Ubuntu 18.04 and 20.04 versions.

3.4 KS-SBCF-O-04: Dependency installation instructions in README.md are incorrect

Status: Acknowledged

Location: README.md

Description

The installation instructions instruct the user to checkout the 3.7.1 version of protbuf. However, git checkout v3.7.1 is not a valid argument in git.

Recommendation

Fix the git argument.

Notes

Hangzhou Xianbing Technology communicated to us that they will improve the software installation instructions.

3.5 KS-SBCF-O-05: Missing dependency makes the compilation process fail

Status: Acknowledged

Location: README.md

Description

Compilation fails if the instructions are followed from README.md because GTest is not installed.

```
CMake Error at /usr/share/cmake-3.18/Modules/FindPackage
    HandleStandardArgs.cmake:165 (message):
Could NOT find GTest (missing: GTEST_LIBRARY GTEST_INCLUDE_DIR
GTEST_MAIN_LIBRARY)
Call Stack (most recent call first):
/usr/share/cmake-3.18/Modules/FindPackageHandle
    StandardArgs.cmake:458 (_FPHSA_FAILURE_MESSAGE)
/usr/share/cmake-3.18/Modules/FindGTest.cmake:205
     (FIND_PACKAGE_HANDLE_STANDARD_ARGS)
test/CMakeLists.txt:4 (find_package)
```

Recommendation

Add instructions in README.md for installing GTest e.g. apt install libgtest-dev.

Notes

Hangzhou Xianbing Technology communicated to us that GTest should be installed in order to run the unit tests.

3.6 KS-SBCF-O-06: Two tests fail

Status: Remediated

Location: build/test/base64-test, build/test/curve-point-test

Description

The following tests fail after compilation:

```
$ ./base64-test
[==========] Running 3 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 3 tests from Base64
[ RUN      ] Base64.ToBase64_FromBase64
```

```
munmap_chunk(): invalid pointer
Aborted (core dumped)

$ ./curve-point-test
[==========] Running 9 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 9 tests from CurvePoint
[ RUN      ] CurvePoint.CurveParameter
free(): double free detected in tcache 2
Aborted (core dumped)
```

Recommendation

Fix the memory corruption errors in both tests.

Notes

Hangzhou Xianbing Technology communicated to us that this issue has been fixed. However, we couldn't reproduce it using the following setup: Ubuntu 21.04, gcc version 10.3.0 (Ubuntu 10.3.0-1ubuntu1), googletest 1.10.0.20201025-1.1 and protobuf 3.7.1. Hangzhou Xianbing Technology communicated to us that Ubuntu 18.04 LTS/ Ubuntu 20.04 LTS are the recommended operating systems.

3.7 KS-SBCF-O-07: Pragma in random number generation implementation of dependency says NOT SUITABLE FOR PRODUCTION USE! during compilation

Status: Acknowledged

Location: 3rdparty/crypto/rand.c:28

Description

The random number generation code base in rand.c contains a pragma that warns the user about the code.

Recommendation

Determine if that part of the code is relevant for the MPC implementation. Otherwise, remove it.

3.8 KS-SBCF-O-08: Comparison between unsigned long and int

Status: Acknowledged

Location: 3rdparty/gmp/mini-gmp.c:100, 3rdparty/gmp/mini-gmp.c:3634

Description

In two parts of the mini-gmp dependency there is a comparison between two different types, unsigned long and int that can create problems where the latter is a negative number.

```
100
101 if (GMP_LIMB_BITS > LOCAL_SHIFT_BITS)                \
```

Recommendation

Decide if both variables should be unsigned long.

Notes

Hangzhou Xianbing Technology communicated to us that they have inspected this case.

3.9 KS-SBCF-O-09: Clang is not able to compile the MPC implementation

Status: Acknowledged

Location: 3rdparty/crypto/ed25519-donna/

Description

Clang fails to compile the MPC implementation with the following error:

```
/src/curve/../../3rdparty/crypto/ed25519-donna/../../crypto/ed25519-donna/
ed25519.h:45:5: note: candidate function not viable: no known conversion
from 'const     ed25519_public_key' (aka 'unsigned char const[32]') to
'unsigned char *' for 2nd argument6 int ed25519_cosi_
combine_two_publickeys(ed25519_public_key
res, CONST ed25519_public_key pk1, CONST ed    25519_public_key pk2);
```

Recommendation

Fix this error by using the right type in the function declaration.

Notes

Hangzhou Xianbing Technology communicated to us that this error is related to the third party library implementing Ed25519 and they will be solve this issue later.

3.10 KS-SBCF-O-10: The Trezor cryptographic library is not updated to last version

Status: Acknowledged

Location: 3rdparty/crypto/

Description

In https://github.com/trezor/trezor-crypto, the README.md file contains:

```
Don't use this repo, use the new monorepo instead:
github.com/trezor/trezor-firmware
```

This library has not been updated since 2019.

Recommendation

Update the dependency to the last version or use git submodule to maintain always the last version of the library in the repository.

3.11 KS-SBCF-O-11: The Trezor cryptographic library contains algorithms that are not used by the MPC implementation

Status: Acknowledged

Location: 3rdparty/crypto/

Description

Cryptographic primitives such as rc4 and blake are not used. However, they are compiled into the MPC implementation.

Recommendation

Only import what you really need from the Trezor cryptographic library.

3.12 KS-SBCF-O-12: It is not possible to easily update the mini-gmp library

Status: Acknowledged

Location: 3rdparty/gmp/

Description

Security fixes and updates of mini-gmp cannot be integrated into the MPC implementation.

Recommendation

Implement a way of updating mini-gmp or utilize, if possible, git for cloning and updating the dependency.

Notes

Hangzhou Xianbing Technology communicated to us that the library can be downladed from https://gmplib.org/repo/gmp-6.2/file/21b955fb2614/mini-gmp/mini-gmp.c.

3.13 KS-SBCF-O-13: Rely on an audited, constant-time, cryptographic library

Status: Acknowledged

Location: General

Description

The MPC implementation is based on C++ with dependencies written in C, particularly the Trezor cryptographic library.

Recommendation

We think that the cryptographic library utilized (Trezor) is less mature than alternatives such as OpenSSL (and variants such as BoringSSL and LibreSSL) and MbedTLS. In this

respect, the constant-time arithmetic layer of these libraries could be used in the MPC implementation instead of relying on the GMP library, which is a known non-constant-time library.

Notes

Hangzhou Xianbing Technology communicated to us that they will rely on the OpenSSL library in the future, particularly in the big number arithmetic implementation.

3.14 KS-SBCF-O-14: Multiple methods are commented in bip32.h

Status: Acknowledged

Location: bip32.h

Description

There are several methods commented in bips32.h.

Recommendation

Determine if they will be used, otherwise remove them.

3.15 KS-SBCF-O-15: Empty classes

Status: Acknowledged

Location: ntl.crt.h, bip32_ed25519.h, bip32_ecdsa.h, curve_enc.h, curve_eddsa.h and curve_ecdsa.h

Description

These classes are empty and/or do not implement any functionality.

Recommendation

Remove them if they are not used.

3.16 KS-SBCF-O-16: Make the wallet easy to configure according to the selected curve and/or add clarification on the curve used

Status: Acknowledged

Location: mta.cpp:34

Description

In general is not clear which curve is utilized in the MPC implementation. Only reading through the code e.g. via GetCurveParam is clear which curve is being used. However, the existence of files with names bip32_ed25519, curve_eddsa, etc. makes unclear which curve is used in the protocol.

Recommendation

Add clarification about the curve that is being used in the MPC implementation, either in README.md or in the code.

Notes

Hangzhou Xianbing Technology communicated to us that they will add more use cases in the future.

3.17 KS-SBCF-O-17: Utilize the original name of the rounds for clarification

Status: Acknowledged

Location: src/mpc_hd_ecdsa/

Description

Beyond round 5 of the signing protocol, the developers decided to name the rounds in increasing order instead of following [1].

Recommendation

Rename the rounds according to [1] to add clarity e.g. 5A, 5B, . . . , 5E.

Notes

Hangzhou Xianbing Technology communicated to us that [1] is only one part of their implementation and the rounds are not exactly numbered according to [1].

3.18 KS-SBCF-O-18: Comments in the SqrtM implementation

Status: Acknowledged

Location: ntl/bn.cpp:330, ntl/bn.cpp:504, ntl/bn.cpp:523

Description

In the SqrtM implementations there are the following comments:

```
// Tonelli-Shanks algorithm (Totally unoptimized and slow)
 ↪
```

```
// //It should never occur in maths theory that " tmp != 1 "
 ↪
```

Moreover, the functions ToBytes32BE and ToBytes32LE contain a redundant implementation commented.

Notes

Hangzhou Xianbing Technology communicated to us that they will remove the redundant comments in this part of the implemenetation.

Recommendation

Optimize the Tonelli-Shanks algorithm or find an alternative implementation. Determine if the commented implementation of ToBytes32BE and ToBytes32LE are needed.

3.19 KS-SBCF-O-19: The share conversion protocols (MtA and MtAwc) do not perform the range proofs

Status: Acknowledged

Location: src/mta/mta.cpp, src/mta/mta.h

Description

The first part of the MtA protocol requires that Alice proofs in ZK that $a < K$ using a range proof. However, only a is encrypted using Paillier:

```
void construct_message_a(BN &message_a, const pail::PailPubKey &pail_pub
, const BN &input_a) {
     BN r_lt_pailN = Rand::RandomBNLtGcd(pail_pub.n());
     construct_message_a_with_R(message_a, pail_pub, input_a,
↪  r_lt_pailN);
 }

void construct_message_a_with_R(BN &message_a, const pail::PailPubKey
 ↪  &pail_pub,
 const BN &input_a, co    nst BN &r_lt_pailN) {
     message_a = pail_pub.EncryptWithR(input_a, r_lt_pailN);
}
```

The second part of the MtA protocol requires proving that $b < K$. However, only is proved in ZK that Bob knows b, β'.

```
18 void construct_message_b(MessageB &message_b, BN &beta, const
    ↪  pail::PailPubKey &pail_pub, const BN &in    put_b, const BN
    ↪  &message_a) {
19      const curve::Curve * curv =
         ↪  curve::GetCurveParam(curve::CurveType::SECP256K1);
20      BN r0_lt_pailN = Rand::RandomBNLt(pail_pub.n());
21      BN r1_lt_curveN = Rand::RandomBNLt(curv->n);
22      BN r2_lt_curveN = Rand::RandomBNLt(curv->n);
23      construct_message_b_with_R(message_b, beta, pail_pub, input_b,
↪  message_a, r0_lt_pailN, r1_lt_curve    N, r2_lt_curveN);
24 }
25
26 void construct_message_b_with_R(MessageB &message_b, BN &beta, const
    ↪  pail::PailPubKey &pail_pub, const     BN &input_b, const BN
    ↪  &message_a, const BN &r0_lt_pailN,
```

```
27      const BN &r1_lt_curveN, const BN &r2_lt_curveN) {
28      const curve::Curve * curv =
          ↪  curve::GetCurveParam(curve::CurveType::SECP256K1);
29      const BN& c_a = message_a;
30      const BN& beta_tag = r0_lt_pailN;
31      string str;
32      //beta_tag.ToHexStr(str);
          ↪
33      //std::cout << "beta_tag: " << str << std::endl;
          ↪
34      BN bma = pail_pub.Mul(c_a, input_b);
35      BN c_b = pail_pub.AddPlain(bma, beta_tag);
36
37      DLogProof dlog_proof(curve::CurveType::SECP256K1);
38      dlog_proof.ProveWithR(input_b, r1_lt_curveN);
39      message_b.dlog_proof_b_ = dlog_proof;
40      dlog_proof.ProveWithR(beta_tag, r2_lt_curveN);
41      message_b.dlog_proof_beta_tag_ = dlog_proof;
42      message_b.c_b_ = c_b;
43      //beta_tag.ToHexStr(str);
          ↪
44      //std::cout << "beta_tag: " << str << std::endl;
          ↪
45      //curv->n.ToHexStr(str);
          ↪
46      //std::cout << "curve.n: " << str << std::endl;
          ↪
47      beta = beta_tag.Neg() % curv->n;
48  }
```

However, [1] says:

> However if the range proofs are not used, a malicious Alice or Bob can cause the protocol to ‘‘fail’’ by choosing large inputs. As a stand-alone protocol this is not an issue since the parties are not even aware that the reduction

```
mod N took place and no information is leaked about the other party's input.
However, when used inside our threshold DSA protocol, this attack will cause the
signature verification to fail, and this information is linked to the size of
 the other party's input.
```

According to [1], an alternative exists:

```
The above protocol is overwhelmingly correct, and hides $b$ perfectly. We
could modify it so that $\beta$' is always chosen uniformly at random in
$[0...N - K^2 ].$ This distribution is statistically close to the uniform
one over $Z_N$ (since $K > q$), therefore the value $b$ is now hidden in a
statistical sense. On the other hand the protocol is always correct.
```

However, in mta.cpp, β' is generated as:

```
BN r0_lt_pailN = Rand::RandomBNLt(pail_pub.n());
```

then

```
const BN& beta_tag = r0_lt_pailN;
string str;
//beta_tag.ToHexStr(str);
//std::cout << "beta_tag: " << str << std::endl;
BN bma = pail_pub.Mul(c_a, input_b);
BN c_b = pail_pub.AddPlain(bma, beta_tag);
```

Recommendation

You could implement the required range proofs in the MtA and MtAwc protocols. Otherwise, generate β' within the right range.

Notes

Hangzhou Xianbing Technology communicated us why the range proofs have not been implemented:

- They are expensive.

- Reduction could be recognized if happened, and then they could reset the system.
- The protocol is still secure enough if we remove the range proof as described in section 6 of [1].

3.20 KS-SBCF-O-20: Message class attribute is_share_encrypted_ is never used

Status: Acknowledged

Location: round0.cpp and other rounds that involve share generation

Description

Message classes contain a method to mark the shares as encrypted. However, it is not clear how this feature would be used in the code.

Recommendation

Add clarification in the code about how to use this attribute. Otherwise remove if there is no use case.

Notes

Hangzhou Xianbing Technology communicated to us that this feature won't be used in the code.

3.21 KS-SBCF-O-21: Missing reference to chosen number of iterations in Paillier proof

Status: Acknowledged

Location: zkp/pail_proof.h

Description

The parameter utilized for generating the masks required for the Paillier proof is fixed to 13 iterations.

However, is not clear which security parameters has been chosen by the authors in relation to https://eprint.iacr.org/2018/057.pdf.

```
17 class PailProof {
18 private:
19     void GenerateXs(std::vector<ntl::BN> &x_arr,     const ntl::BN
 ↪  &index, const ntl::BN &point_x, c    onst ntl::BN &point_y, const
 ↪  ntl::BN &N, uint pr    oof_iters = 13) const;
20 public:
```

Recommendation

We recommend Hangzhou Xianbing Technology to provide clarification in the code about how the number of iterations has been chosen.

Notes

Hangzhou Xianbing Technology provided us clarification about how the number of iterations has been chosen.

4 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

5 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.

5.1 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

5.2 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

5.3 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

5.4 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.

5.5 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.

5.6 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 TypeScript 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 TypeScript.

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.

6 APPENDIX C: DOCUMENT HISTORY

Version	Status	Date	Document Owner	Comments
0.1	Draft	4th October 2021	Kudelski Security Research Team	

Reviewer	Position	Date	Version	Comments
Nathan Hamiel	Head of Security Research	18th October 2021	Final	

Approver	Position	Date	Version	Comments
Nathan Hamiel	Head of Security Research	18th October 2021	Final	

Approver	Position	Date	Version	Comments
Nathan Hamiel	Head of Security Research	30th November 2021	Final	

Approver	Position	Date	Version	Comments
Nathan Hamiel	Head of Security Research	3rd December 2021	Final	

7 APPENDIX D: 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.

REFERENCES

[1]
Rosario Gennaro and Steven Goldfeder. 2019. Fast multiparty threshold ECDSA with fast trustless setup. *IACR Cryptol. ePrint Arch.* (2019), 114. Retrieved from https://eprint.iacr.org/2019/114

[2]
Yehuda Lindell. 2021. Fast secure two-party ECDSA signing. *J. Cryptol.* 34, 4 (2021), 44. DOI:https://doi.org/10.1007/s00145-021-09409-9