Audit of Micro-Starknet

Starkware

04 septembre 2023

Version: 1

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

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

For public release

TABLE OF CONTENTS

1 EXECUTIVE SUMMARY

Kudelski Security ("Kudelski", "we"), the cybersecurity division of the Kudelski Group, was engaged by Starkware ("the Client") to conduct an external security assessment in the form of a code audit of the cryptographic library Micro-Starknet ("the Product"). The assessment was conducted remotely by the Kudelski Security Team and coordinated by Sylvain Pelissier, Cryptography Expert and Antonio De La Piedra, Senior Cybersecurity Engineer. The audit took place from July 11, 2023 to July 25, 2023 and involved 10 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

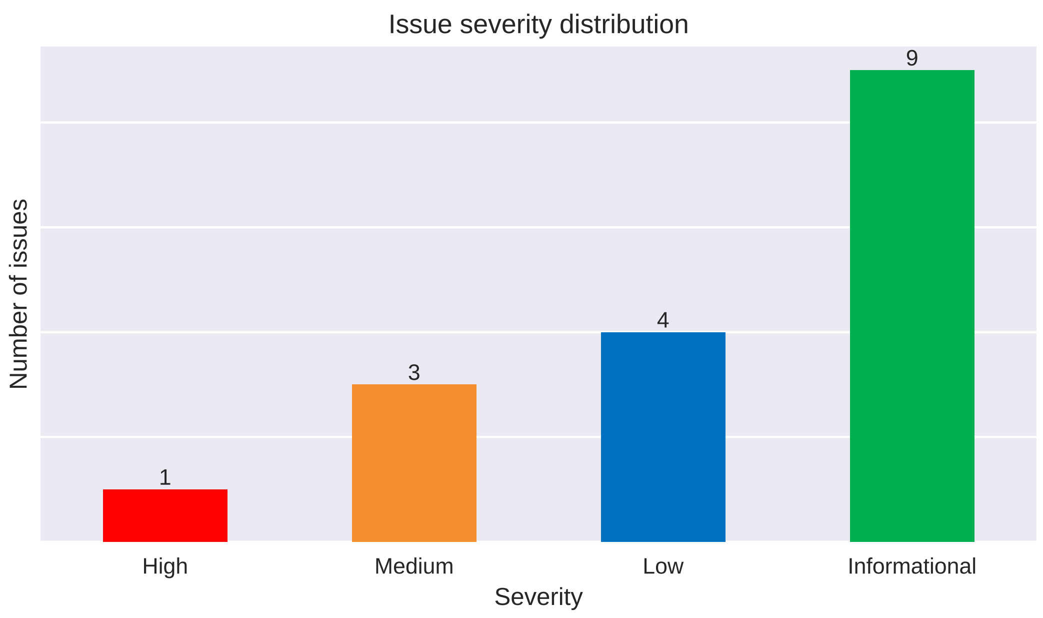
The scope of the audit was a code audit of Micro-Starknet written in Typescript. Starknet provides Zero-knowledge rollups built on top of the Ethereum blockchain. It allows to optimize changes on the layer 1 blockchain by providing a summary of the changes and a proof that the changes were valid. Micro-Starknet is a library implementing the Starknet cryptography used for key generation, signature verification, and hash computation. The audit was done with particular attention to the safe implementation of hashing, randomness generation, protocol verification, and the potential for misuse and leakage of secrets.

The target of the audit was the cryptographic code located in the micro-starknet Github repository: https://github.com/paulmillr/micro-starknet. We audited the commit number: `07b25e9997b45a0c0d83ced2c0272306143f0660`. Particular attention was given to side-channel attacks and in particular constant timeness.

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**, **3 Medium**, **4 Low** and **9 Informational** findings.



1.3 Issue Summary List

The following security issues were found:

ID	Severity	Finding	Status
KS-SBCF-F-01	**High**	Hash function is not second image resistant	**Remediated**
KS-SBCF-F-02	**Medium**	No check on Poseidon MDS matrix generation	**Acknowl-edged**
KS-SBCF-F-03	**Medium**	Hardcoded power map for undefined inputs	**Remediated**

ID	Severity	Finding	Status
KS-SBCF-F-04	**Medium**	Possible infinite loop in Tonelli-Shanks implementation	**Remediated**
KS-SBCF-F-05	**Low**	Pedersen hash is not time constant	**Acknowl-edged**
KS-SBCF-F-06	**Low**	The Poseidon round constant generation is not following the specifications	**Acknowl-edged**
KS-SBCF-F-07	**Low**	reversePartialPowIdx can be used to apply the non-linear layer to the first element of the state and to the last one	**Acknowl-edged**
KS-SBCF-F-08	**Low**	weierstrass: Bias in random private key generation	**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**	Lack of parameters generation documentation.
KS-SBCF-O-03	**Informational**	Invalid signature test are commented
KS-SBCF-O-04	**Informational**	Undefined and unused field Fp253 in Poseidon hash
KS-SBCF-O-05	**Informational**	Improvement: Add support for domain separation in Poseidon
KS-SBCF-O-06	**Informational**	Improvement: Poseidon permutation optmization
KS-SBCF-O-07	**Informational**	The Poseidon implementation doesn’t abort when the number of full rounds is odd
KS-SBCF-O-08	**Informational**	Not constant-time arithmetic methods in noble-curves
KS-SBCF-O-09	**Informational**	Inconsistent results from batch inversion

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: Hash function is not second image resistant

Severity: High

Status: Remediated

Location: micro-starknet/index.ts:165

Description

The function `hashChain` is built upon the Pedersen hash function and used to hash an array of values. According to the documentation [4], the hash chain is used to compute the contract storage address of a variable. However, the hashChain function does not include the length of the data neither the starting value in the hash computation. Thus, is prone to second pre-image attack. Here is a proof of concept:

```
> var starknet = require('micro-starknet');
undefined
> h1 = starknet.hashChain([1, 2, 3])
'0x5d9d62d4040b977c3f8d2389d494e4e89a96a8b45c44b1368f1cc6ec541891
↪  5'
> h3 = starknet.hashChain([1, "0x5774fa77b3d843ae9167abd61cf80365
↪  a9b2b02218fc2f628494b5bdc9b33b8"])
'0x5d9d62d4040b977c3f8d2389d494e4e89a96a8b45c44b1368f1cc6ec541891
↪  5'
> h1 === h2
true
```

We added the resulting hash of the array `[2,3]` in the the second array and we obtain a second pre-image of for the array `[1,2,3]`. Even though the malleability is limited, it allows an attacker to compute the same storage address for two different variables. This property is undesirable for this feature.

Recommendation

Implement the Array hashing method as define in the **Hash functions** documentation [5] and how it is done with `computeHashOnElements` function. The previous example can also be added to the tests to avoid regressions.

Status details

The function was removed from the library in version 0.3.0.

2.2 KS-SBCF-F-02: No check on Poseidon MDS matrix generation

Severity: Medium

Status: Acknowledged

Location: micro-starknet/index.ts:245

Description

The current implementation of Poseidon does not rely on the methods and validation checks provided by the authors of Poseidon [7] for creating the MDS matrix utilized in the linear layer. Further, the following method is provided for creating alternative MDS matrices:

```
// NOTE: doesn't check eiginvalues and possible can create unsafe
↪ matrix. But any filtration here will break compatibility with
↪ starknet
// Please use only if you really know what you doing.
// https://eprint.iacr.org/2019/458.pdf Section 2.3 (Avoiding
↪ Insecure Matrices)
export function _poseidonMDS(Fp: IField<bigint>, name: string, m:
↪ number, attempt = 0) {
  const x_values: bigint[] = [];
  const y_values: bigint[] = [];
  for (let i = 0; i < m; i++) {
    x_values.push(poseidonRoundConstant(Fp, `x`, attempt * m +
↪ i));
    y_values.push(poseidonRoundConstant(Fp, `y`, attempt * m +
↪ i));
```

```
  }
  if (new Set([...x_values, ...y_values]).size !== 2 * m)
    throw new Error('X and Y values are not distinct');
  return x_values.map((x) => y_values.map((y) => Fp.inv(Fp.sub(x,
  ↪   y))));
}
```

The MDS matrix is part of the linear layer of the Poseidon permutation. The different checks described in [7] are not performed during the generation, it is possible to end up using insecure MDS matrices. Moreover, this could result in an implementation of the Poseidon hash that is vulnerable to cryptanalytic attacks.

The `_poseidonMDS` function is utilized every time `poseidonCreate` is called:

```
export function poseidonCreate(opts: PoseidonOpts, mdsAttempt = 0)
 ↪  {
  const m = opts.rate + opts.capacity;
  if (!Number.isSafeInteger(mdsAttempt)) throw new Error(`Wrong
  ↪   mdsAttempt=`);
  return poseidonBasic(opts, _poseidonMDS(opts.Fp, 'HadesMDS', m,
  ↪   mdsAttempt));
}
```

Recommendation

We recommend the Client to rely on the recommendations in the papers for generating safe MDS matrices as well as using the scripts created by the authors of Poseidon, available at https://extgit.iaik.tugraz.at/krypto/hadeshash/-/blob/master/code/generate_params_poseidon.sage. That would mean to implement the `algorithm_1`, `algorithm_2` and `algorithm_3` methods utilized in the `generate_matrix` function of the script.

Status details

Not fixed.

2.3 KS-SBCF-F-03: Hardcoded power map for undefined inputs

Severity: Medium

Status: Remediated

Location: noble-curves/src/abstract/poseidon.ts:28

Description

The option validation function `validateOpts` defined in the noble-curves implementation for Poseidon choses a power map $\alpha = 5$ when the exponent is not defined.

```
let sboxPower = opts.sboxPower;
if (sboxPower === undefined) sboxPower = 5;
if (typeof sboxPower !== 'number' ||
 ↪  !Number.isSafeInteger(sboxPower))
  throw new Error(`Poseidon wrong sboxPower=`);

const _sboxPower = BigInt(sboxPower);
let sboxFn = (n: bigint) => FpPow(Fp, n, _sboxPower);
// Unwrapped sbox power for common cases (195->142µs)
if (sboxPower === 3) sboxFn = (n: bigint) => Fp.mul(Fp.sqrN(n),
 ↪  n);
else if (sboxPower === 5) sboxFn = (n: bigint) =>
 ↪  Fp.mul(Fp.sqrN(Fp.sqrN(n)), n);
```

However, arithmetization-oriented constructions such as Poseidon [7], which relies on power maps, require that the exponent α satisfies gcd$(\alpha, p-1) = 1$. For that reason, the validation function must not create a non-linear layer for a parameter $\alpha = 5$ that is not suitable for a field p. That would mean using a parameter that doesn't guarantee invertibility and that might not provide non-linearity.

Particularly, for the Starkware curve field $p = 2^{251} + 17 \cdot 2^{192} + 1$, a value of 5 for the exponent would be an invalid exponent, since gcd$(5, p-1) = 5$. For instance, in Python:

```
>>> import math
>>> p = 2**251+17*2**192+1
>>> math.gcd(5, p-1)
```

```
5
```

Further, for clarity, we also recommend the Client to elaborate on the following comment:

```
  // Default is 5, but by some reasons stark uses 3
```

Recommendation

We recommend the client to perform an exhaustive parameter validation procedure to instantiate the Poseidon hash function, abort in case of incorrect and/or undefined parameters and never default to a power map.

Further, the reference implementation of Poseidon (available at https://extgit.iaik.tugraz.at/krypto/zkfriendlyhashzoo/-/blob/master/plain_impls/src/poseidon/poseidon.rs), prohibits values for α different from $[3, 5, 7]$. We recommend the client to restrict the values the power map exponent can take:

```
    fn sbox_p(&self, input: &S) -> S {
        let mut input2 = *input;
        input2.square();

        match self.params.d {
            3 => {
                let mut out = input2;
                out.mul_assign(input);
                out
            }
            5 => {
                let mut out = input2;
                out.square();
                out.mul_assign(input);
                out
            }
            7 => {
                let mut out = input2;
                out.square();
                out.mul_assign(&input2);
                out.mul_assign(input);
                out
            }
            _ => {
                panic!()
            }
        }
    }
```

Status details

Fixed, sboxPower is limited to 3, 5, 7 and the default value 5 was removed in noble-curves 1.2.0

2.4 KS-SBCF-F-04: Possible infinite loop in Tonelli-Shanks implementation

Severity: Medium

Status: Remediated

Location: noble-curves/src/abstract/modular.ts:121

Description

The Tonneli-Shanks algorithm assumed that p is a prime number to be able to finish the computation. However, if p is not prime, the algorithm may enter in an infinite loop. The current implementation can enter into an infinite loop for certain parameters e.g.

```
> mod = await import('@noble/curves/abstract/modular')
> mod.FpSqrt(BigInt(1), BigInt(0))
[Infinite loop]
```

In addition the finite field constructor does not check that p is prime thus this problem can be trigger through this interface:

```
> Fp = mod.Field(12n)
> Fp.sqrt(7n)
[Infinite loop]
```

Since the code doesn't validate if the parameter P is not prime nor odd and the following while loop is executed for ever:

```
    while (!Fp.eql(b, Fp.ONE)) {
      if (Fp.eql(b, Fp.ZERO)) return Fp.ZERO; //
       ↪  https://en.wikipedia.org/wiki/Tonelli%E2%80%93Shanks_
       ↪  algorithm (4. If t = 0, return r = 0)
      // Find m such b^(2^m)==1
      let m = 1;
      for (let t2 = Fp.sqr(b); m < r; m++) {
        if (Fp.eql(t2, Fp.ONE)) break;
        t2 = Fp.sqr(t2); // t2 *= t2
      }
      // NOTE: r-m-1 can be bigger than 32, need to convert to
       ↪  bigint before shift, otherwise there will be overflow
      const ge = Fp.pow(g, _1n << BigInt(r - m - 1)); // ge =
       ↪  2^(r-m-1)
      g = Fp.sqr(ge); // g = ge * ge
      x = Fp.mul(x, ge); // x *= ge
      b = Fp.mul(b, g); // b *= g
      r = m;
    }
```

This problem which leads to a denial of service in Tonneli-Shanks implementations, is known and has been first detected in OpenSSL https://www.openssl.org/news/secadv/20220315.txt and in the Go cryptography library https://github.com/golang/go/issues/51747.

Recommendation

We recommend the Client to catch invalid parameters P to avoid an infinite loop situation in critical code depending on noble-curves.

Status details

The issue has been documented and library mentions the input must be a prime. The primeness check is not done in the library for performance since provable primally test is slow.

2.5 KS-SBCF-F-05: Pedersen hash is not time constant

Severity: Low

Status: Acknowledged

Location: micro-starknet/index.ts:202

Description

The function `pedersenSingle` implements a double and add scalar multiplication with precomputed value. Each time the value bit is one an additional point addition is performed. This is not time constant and the time difference depends directly on the input value.

```
let x = pedersenArg(value);
let x = pedersenArg(value);
for (let j = 0; j < 252; j++) {
    const pt = constants[j];
    if (pt.equals(point)) throw new Error('Same point');
    if ((x & 1n) !== 0n) point = point.add(pt);
    x >>= 1n;
  }
```

Even though we did not find any usage where the input of the Pedersen hash is secret if this library is used in this way, a timing attack may be done to recover the input value.

Recommendation

To have a time constant function, a fake point addition may be added. At least the time sensitivity of the function should be documented to avoid misusage in the future.

Status details

A comment was added that mentions it’s not constant time.

2.6 KS-SBCF-F-06: The Poseidon round constant generation is not following the specifications

Severity: Low

Status: **Acknowledged**

Location: micro-starknet/index.ts:240

Description

According to [7], the Poseidon round constants are generated using the Grain LFSR taking as parameters, the number of rounds, the type and the size of the field and the SBox number. It makes the round constants unique per Poseidon instances.

In the repository, the Poseidon round constants are generated via the `poseidonRoundConstant` function as the SHA256 hash of a string and the S-Box index:

```
function poseidonRoundConstant(Fp: IField<bigint>, name: string,
 ↪  idx: number) {
  const val = Fp.fromBytes(sha256(utf8ToBytes(`${name}${idx}`)));
  return Fp.create(val);
}
```

This function is called inside `poseidonBasic` with the string `"Hades"`:

```

  for (let i = 0; i < rounds; i++) {
    const row = [];
    for (let j = 0; j < m; j++)
     ↪  row.push(poseidonRoundConstant(opts.Fp, 'Hades', m * i +
     ↪  j));
    roundConstants.push(row);
  }
```

This leads to deviation of the hash function which does not follow the paper specification and generate constants which stay the same between different instances of the hash function. It also makes harder for independent implementations to be compatible with Starknet.

Recommendation

The round constants and matrices should be generated using the Grain LFSR. The authors of Poseidon have made available the implementation of the Grain LFSR and of the methods for generating round constants at **https://extgit.iaik.tugraz.at/krypto/hadeshash/-/blob/master/code/generate_params_poseidon.sage**. Consequently, the correct manner of generating the round constants for Poseidon relies on the Grain LFSR, defined in the script. We recommend the Client to follow the recommendations of the authors of the Poseidon paper for generating the round constants as the authors claimed in the paper: *"Indeed, letting the attacker freely choose round constants and/or matrices can lead to attacks"*

Status details

Not fixed.

2.7 KS-SBCF-F-07: reversePartialPowIdx can be used to apply the non-linear layer to the first element of the state and to the last one

Severity: Low

Status: Acknowledged

Location: micro-starknet/index.ts:292 and noble-curves/src/abstract/poseidon.ts:24 and 82

Description

In order to select the the position of the state where the sbox is applied in a partial round, the parameter `reversePartialPowIdx` is used, being the position `t-1` for a value `true`, that is, for Poseidon using the starkware curve, and the position `0` otherwise.

```
export function poseidon(opts: PoseidonOpts) {
  const { t, Fp, rounds, sboxFn, reversePartialPowIdx } =
   ↪  validateOpts(opts);
  const halfRoundsFull = Math.floor(opts.roundsFull / 2);
  const partialIdx = reversePartialPowIdx ? t - 1 : 0;
  const poseidonRound = (values: bigint[], isFull: boolean, idx:
   ↪  number) => {
    values = values.map((i, j) => Fp.add(i,
 ↪  opts.roundConstants[idx][j]));
```

```
  if (isFull) values = values.map((i) => sboxFn(i));
  else values[partialIdx] = sboxFn(values[partialIdx]);
```

The parameter `reversePartialPowIdx` is set to `true` for Poseidon in index.ts:292:

```
const res: Partial<PoseidonFn> = poseidon({
  ...opts,
  t: m,
  sboxPower: 3,
  reversePartialPowIdx: true, // Why?!
  mds,
  roundConstants,
});
```

However, the first element of the state is used instead in the reference implementation of Poseidon at https://extgit.iaik.tugraz.at/krypto/zkfriendlyhashzoo/-/blob/master/plain_impls/src/poseidon/poseidon.rs:

```
    pub fn permutation_not_opt(&self, input: &[S]) -> Vec<S> {
        let t = self.params.t;
        assert_eq!(input.len(), t);

        let mut current_state = input.to_owned();

        for r in 0..self.params.rounds_f_beginning {
            current_state = self.add_rc(&current_state,
    ↪   &self.params.round_constants[r]);
            current_state = self.sbox(&current_state);
            current_state = self.matmul(&current_state,
    ↪   &self.params.mds);
        }
        let p_end = self.params.rounds_f_beginning +
            ↪   self.params.rounds_p;
        for r in self.params.rounds_f_beginning..p_end {
            current_state = self.add_rc(&current_state,
    ↪   &self.params.round_constants[r]);
```

```
            current_state[0] = self.sbox_p(&current_state[0]);
            current_state = self.matmul(&current_state,
↪   &self.params.mds);
        }
        for r in p_end..self.params.rounds {
            current_state = self.add_rc(&current_state,
↪   &self.params.round_constants[r]);
            current_state = self.sbox(&current_state);
            current_state = self.matmul(&current_state,
↪   &self.params.mds);
        }
        current_state
    }
```

It is not clear why Starkware allows to apply the non-linear layer in internal rounds to the first element of the state and to the last one.

Recommendation

- Clarify in the source code and documentation why the `reversePartialPowIdx` is required in the Poseidon implementation.
- Add more information to the following comment:

```
    reversePartialPowIdx: true, // Why?!
```

Status details

Not fixed.

2.8 KS-SBCF-F-08: weierstrass: Bias in random private key generation

Severity: Low

Status: Remediated

Location: noble/curves/src/abstract/weierstrass.ts:848

Description

The function `randomPrivateKey` generates a random number between 0 and the curve order. To achieve that, a random number in $[0, 2^{(\log 2(p))+64}[$ is generated and then reduced modulus the order of the curve.

```
randomPrivateKey: (): Uint8Array => {
  const rand = CURVE.randomBytes(Fp.BYTES + 8);
  const num = mod.hashToPrivateScalar(rand, CURVE_ORDER);
  return ut.numberToBytesBE(num, CURVE.nByteLength);
},
```

According to [6] this gives give a number with a bias of about $\frac{1}{2^{64}}$. The implementation is based on the FIPS 186-4 [1] Appendix B.4.1 called "Key Pair Generation Using Extra Random Bits". However this algorithm has been replaced in FIPS 186-5 [3] in Appendix A.2.1. The new algorithm define the extra bits to add before the modular reduction only for specific curves and it does not apply for the Starknet curve. The function is not currently used by the micro-starknet library but may be used by a user to generate their keys and thus would lead to value generated non-uniformly with a bias bigger than the level of security requires.

Recommendation

Use a larger number of random bytes like `Fp.BYTES + Fp.BYTES/2` before the modular reduction to keep the same level of security as the curve.

Status details

The issue have been reported to the maintainer of noble-curves and corrected in version 1.2.0. The bias is now 2^{-128} and matches curve security level.

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

Description

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

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 or the SECURITY.md file of https://github.com/paul-millr/noble-curves

Status details

A `SECURITY.md` have been added.

3.2 KS-SBCF-O-02: Lack of parameters generation documentation.

Location: https://docs.starkware.co/starkex/pedersen-hash-function.html

Description

The Pedersen hash uses five different points on the curve. This is critical to ensure that they have been generated in a way that nobody knows the discrete logarithm of one point regarding another [2]. The documentation gives only the point values without any further explanation.

Recommendation

According to the script **https://github.com/starkware-libs/starkex-for-spot-trading/blob/master/src/starkware/crypto/starkware/crypto/signature/nothing_up_my_sleeve_gen.py**, the parameters of the Pedersen hash are generated from the constant π. The x-coordinate of each point is a chunk of 76 decimal digit of π modulo p. If it is a quadratic residue then the point is valid else the x-coordinate coordinate is incremented by one.

This method of generated points should be described in the documentation and explained how it avoid to know the discrete logarithm between each points.

Status details

The generated points have been commented.

3.3 KS-SBCF-O-03: Invalid signature test are commented

Location: src/test/stark.test.js:66

Description

A lot of test regarding invalid rejection signature are commented in the test file stark.test.js. For example, there is a test with a message bigger that the curve order:

```
// Test invalid message length.
    expect(() =>
    starkwareCrypto.verify(maxStarkKey,
 ↪  maxMsgHash.add(oneBn).toString(16), {
        r: maxR,
        s: maxS
    })
```

Recommendation

Those tests make sense and should be added back to the test suite to avoid regression.

Status details

Test have been brought back and allowed to spot invalid signature verification cases which have been fixed as well.

3.4 KS-SBCF-O-04: Undefined and unused field Fp253 in Poseidon hash

Location: src/index.ts:233

Description

The Poseidon hash parameters also include an alternative field Fp253:

```
// Poseidon hash
export const Fp253 = Field(
  BigInt('14474011154664525231415395255581126252639794253786371766033694892385558855681')
); // 2^253 + 2^199 + 1
```

which is not defined and not used in the current implementation.

Recommendation

Since the current provided parameters for the default field, Fp251 could be different for the Fp253 field, we recommend the Client to provided specific parameters and documentation for this field too.

Status details

The field has been removed.

3.5 KS-SBCF-O-05: Improvement: Add support for domain separation in Poseidon

Description

The Client could support a domain separation parameter in the Poseidon implementation. The authors of [7] suggest the support of a domain separation in Poseidon hash implementation for applications that require different instances of the hashing function. They propose to encode it in the capacity element in Section 4.2.

3.6 KS-SBCF-O-06: Improvement: Poseidon permutation optmization

Description

The authors of [7] proposed an efficient implementation of the permutation in Appendix B that the client could adopt. The approach is based on reducing the size of the round constants utilized in each partial round. The Client could rely on the permutation function of the Poseidon reference implementation available at https://extgit.iaik.tugraz.at/krypto/zkfriendlyhashzoo/-/blob/master/plain_impls/src/poseidon/poseidon.rs, line 22.

3.7 KS-SBCF-O-07: The Poseidon implementation doesn't abort when the number of full rounds is odd

Location: noble-curves/src/abstract/poseidon.ts:82

Description

The Poseidon implementation performs the `Math.floor` operator on the division by 2 of the number of full rounds (`opts.roundsFull`).

```
export function poseidon(opts: PoseidonOpts) {
  const { t, Fp, rounds, sboxFn, reversePartialPowIdx } =
   ↪  validateOpts(opts);
  const halfRoundsFull = Math.floor(opts.roundsFull / 2);
```

Then:

```
    let round = 0;
    // Apply r_f/2 full rounds.
    for (let i = 0; i < halfRoundsFull; i++) values =
     ↪  poseidonRound(values, true, round++);
    // Apply r_p partial rounds.
    for (let i = 0; i < opts.roundsPartial; i++) values =
     ↪  poseidonRound(values, false, round++);
    // Apply r_f/2 full rounds.
    for (let i = 0; i < halfRoundsFull; i++) values =
     ↪   poseidonRound(values, true, round++);
```

On the other hand, the reference implementation enforces that the number of full rounds is even when validating the parameters at https://extgit.iaik.tugraz.at/krypto/zkfriendlyhashzoo/-/blob/master/plain_impls/src/poseidon/poseidon_params.rs at line 35:

```
impl<S: PrimeField> PoseidonParams<S> {
    #[allow(clippy::too_many_arguments)]
    pub fn new(
        t: usize,
        d: usize,
        rounds_f: usize,
        rounds_p: usize,
        mds: &[Vec<S>],
        round_constants: &[Vec<S>],
    ) -> Self {
        assert!(d == 3 || d == 5 || d == 7);
        assert_eq!(mds.len(), t);
        assert_eq!(rounds_f % 2, 0);
        let r = rounds_f / 2;
```

Since having an odd number of full rounds could mean that the parameters for Poseidon are wrong, we recommend the Client to warn the user and/or abort in this case.

Status details

`validateOpts` is now checked.

3.8 KS-SBCF-O-08: Not constant-time arithmetic methods in noble-curves

Location: noble-curves/src/abstract/modular.ts:24

Description

There are several methods at modular.ts that are not constant time:

- Modular exponentiation:

```
/**
 * Efficiently raise num to power and do modular division.
 * Unsafe in some contexts: uses ladder, so can expose bigint
 ↪  bits.
 * @example
 * pow(2n, 6n, 11n) // 64n % 11n == 9n
 */
// TODO: use field version && remove
export function pow(num: bigint, power: bigint, modulo: bigint):
 ↪  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;
}
```

- Field exponentiation:

```

// Generic field functions
export function FpPow<T>(f: IField<T>, num: T, power: bigint): T {
  // Should have same speed as pow for bigints
  // TODO: benchmark!
  if (power < _0n) throw new Error('Expected power > 0');
  if (power === _0n) return f.ONE;
  if (power === _1n) return num;
  let p = f.ONE;
  let d = num;
  while (power > _0n) {
    if (power & _1n) p = f.mul(p, d);
    d = f.sqr(d);
    power >>= _1n;
  }
  return p;
}
```

Status details

The issue was documented.

3.9 KS-SBCF-O-09: Inconsistent results from batch inversion

Location: src/abstract/modular.ts:280

Description

Modular library implements the batch modular inversion to speed up the modular inversion of element lists. However the handling of error seems inconsistent with the `inv` function which invert a single element. For example, the inversion of 0 gives an exception with `inv`:

```
> starknet.Fp251.inv(0n)
Uncaught:
Error: invert: expected positive integers, got n=0 mod=36185027...
```

Whereas with `invertBatch` it gives a list containing an empty item without raising any exception:

```
> starknet.Fp251.invertBatch([1n,0n,2n])
[
  1n,
  <1 empty item>,
  18092513...936010241n
]
```

But inverting a multpiple of the prime raise an exception:

```
> starknet.Fp251.invertBatch([1n, 36185027...72020481n])
Uncaught:
Error: invert: expected positive integers, got n=0 mod=36185027...
```

The behavior of the function seems confusing and is prone to further error when implementing elliptic curve arithmetic.

Recommendation

The `invertBatch` function should behave like the `inv` function to avoid confusion.

Status details

The issue was documented but not fixed. It is left to the user to throw an error for incorrect values.

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.

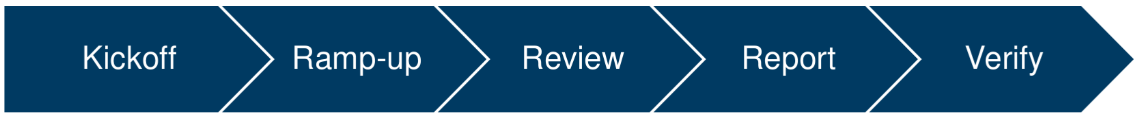


Figure 1: Methodology flow

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: 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] Elaine Barker. 2013. Digital signature standard (DSS). DOI:https://doi.org/https://doi.org/10.6028/NIST.FIPS.186-4

[2] Paul Bottinelli. 2023. Breaking pedersen hashes in practice. In *NCC group research blog*.

[3] Lily Chen, Dustin Moody, Andrew Regenscheid, and Angela Robinson. 2023. Digital signature standard (DSS). DOI:https://doi.org/https://doi.org/10.6028/NIST.FIPS.186-5

[4] Starknet community. 2023. Starknet documentation: Contract storage. Retrieved from https://docs.starknet.io/documentation/architecture_and_concepts/Contracts/contract-storage/

[5] Starknet community. 2023. Starknet documentation: Hash functions. Retrieved from https://docs.starknet.io/documentation/architecture_and_concepts/Hashing/hash-functions/

[6] Armando Faz-Hernandez, Sam Scott, Nick Sullivan, Riad S. Wahby, and Christopher A. Wood. 2022. *Hashing to Elliptic Curves*. Internet Engineering Task Force; Internet Engineering Task Force. Retrieved from https://datatracker.ietf.org/doc/draft-irtf-cfrg-hash-to-curve/16/

[7] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. 2019. Poseidon: A new hash function for zero-knowledge proof systems. Retrieved from https://eprint.iacr.org/2019/458