Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Audit-Report MetaMask key-tree Interface 04.2024

Cure53, Dr.-Ing. M. Heiderich, Dr. M. Conde

Index

- Introduction
- Scope
- Test Methodology
- Miscellaneous Issues
  - MM-04-001 WP1: Unchecked entropy size in entropyToCip3MasterNode (Info)
  - MM-04-002 WP1: Unneeded modular reduction in private key derivation (Low)
  - MM-04-003 WP1: Failure to reject invalid extended BIP32-Ed25519 keys (Medium)
- Conclusions

Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Introduction

“MetaMask provides the simplest yet most secure way to connect to blockchain-based applications. You are always in control when interacting on the new decentralized web.”

From https://metamask.io/

This report describes the results of a cryptography review and source code audit against the MetaMask *key-tree* interface, performed by Cure53 in Q2 2024.

To give some context regarding the assignment’s origination and composition, ConsenSys Software Inc. contacted Cure53 in March 2024. The test execution was scheduled for CW17 April 2024, whereby four work days were invested to reach the coverage expected for this project. A team of two senior testers were assigned to this project’s preparation, execution and finalization.

The work was structured using a single work package (WP), defined as:

- **WP1**: Crypto review & code audit against the MetaMask *key-tree* interface

It should be noted that this audit specifically focused on changes implemented since the last audit of the MetaMask *key-tree* interface, which was performed by Cure53 in February 2023 (see MM-02). In particular, the main focus of this review was directed to the added support for the Cardano key derivation as specified by CIP3.

The methodology conformed to a white-box strategy, whereby assistive materials such as sources, an overview of all implemented changes since the last audit, and all further means of access required to complete the tests were provided to facilitate the undertakings.

All preparations were completed in April 2024, specifically during CW16, to ensure a smooth start for Cure53. Communication throughout the test was conducted through a dedicated and shared Slack channel, established to combine the teams of MetaMask and Cure53. All personnel involved from both parties were invited to participate in this channel. Communications were smooth, with few questions requiring clarification, and the scope was well-defined and clear. No significant roadblocks were encountered during the test. Cure53 provided frequent status updates and shared their findings. Live-reporting was not specifically requested for this audit.

The Cure53 team achieved good coverage over the scope items, and identified a total of three findings. Of these three findings, all were classified to be general weaknesses with lower exploitation potential.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

This second review of the MetaMask *key-tree* interface revealed only a very small sum of weaknesses. While the identified issues are minor, they do warrant attention in order to ensure optimal security. Cure53 can confirm that the newly implemented Cardano key derivation function adheres to the CIP3-Icarus standard.

The report will now shed more light on the scope and testing setup, as well as providing a comprehensive breakdown of the available materials. Next, the report will detail the *Test Methodology* used in this exercise. This chapter will show which areas of the software in scope have been covered and what tests have been executed, despite the limited number of findings made during the course of the exercise. Following this, the report will list all findings identified in chronological order, starting with the *Identified Vulnerabilities* and followed by the *Miscellaneous Issues* unearthed. Each finding will be accompanied by a technical description, Proof-of-Concepts (PoCs) where applicable, plus any fix or preventative advice to action.

In summation, the report will finalize with a *Conclusions* chapter in which the Cure53 team will elaborate on the impressions gained toward the general security posture of the MetaMask *key-tree* interface.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Scope

- **Cryptography review & source code audit against new MetaMask *key-tree* interface**
  - **WP1:** Crypto review & code audit against the MetaMask *key-tree* interface
    - Support of Cardano key derivation (main focus):
      - https://github.com/MetaMask/key-tree/commit/bfa679ecc8a9e56516c92ab63bb3b4e5f7556577
    - Changes implemented since the last audit (see MM-02).
      - https://github.com/MetaMask/key-tree/compare/39397d1...bfa679e
  - **Test-supporting material was shared with Cure53**
  - **All relevant sources were shared with Cure53**

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Test Methodology

This section documents the testing methodology applied by Cure53 during this project, and discusses the resulting coverage, shedding light on the numerous methods by which the various components were examined. Further clarification concerning areas of investigation subject to deep-dive assessment is offered, especially due to the absence of any significant security vulnerabilities being detected.

The assignment had a single work package that consisted of a cryptography review and source code audit of the changes implemented in the *key-tree* interface since it was last audited (MM-02 for reference). The main focus was the added support for Cardano key derivation as indicated by CIP3-Icarus.

The functions in *ed25519Bip32.ts* that perform elliptic curve operations were inspected first, as they underlie the implementation of the key derivation as mandated by CIP3[1]. Since *key-tree* imports a third-party library[2] to perform the basic elliptic curve operations (such as point addition or scalar multiplication), it was checked that the library is correctly leveraged. In particular, it was seen that the exported function *publicAdd* is correct and that it does not allow the addition of invalid curve points (i.e. those that do not satisfy the curve equation), which could be problematic. The function that computes the public key that corresponds to a given private key (namely *multiplyWithBase*) also behaves adequately. However, any extended private key is declared as valid without checking that certain bits of it are appropriately set and cleared (see MM-04-003), which could eventually cause issues to applications making use of the library.

The derivation of the master node from an input entropy was reviewed next. It was observed that the function that derives the master node from an input entropy (namely *entropyToCip3MasterNode*) lacks a check of the size of the entropy (see MM-04-001). This is not an issue for *key-tree* itself, as the function is called with an entropy of the expected length. However, it was reported as a miscellaneous issue as the function is exported.

Finally, the implementation of the derivation of the child private and public keys was reviewed, and it was found to adhere to the CIP3 specification[3]. Note that a redundant modular reduction was spotted in the private child key derivation function (see MM-04-002). It must be noted that it is the result of a particular addition that needs to be modularly reduced, and not one of the summands alone as it is done in the code, as the summand remains unaltered. However, this does not actually imply a deviation from the standard, as it turns out that the modular reduction of said addition is computed (although not by calling the function that performs modular reduction). Therefore, the private key derivation function behaves as expected, but this should be addressed for performance and clarity reasons.

[1] https://input-output-hk.github.io/adrestia/static/Ed25519_BIP.pdf
[2] https://github.com/paulmillr/noble-ed25519
[3] https://input-output-hk.github.io/adrestia/static/Ed25519_BIP.pdf

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Additionally, the specification establishes that when deriving an extended private key, denoted by (K_L, K_R), the child key should be discarded if K_L is divisible by the curve order (see Section V, C.2). This check is not present in the code, but it was observed that the condition can't possibly be satisfied, so it can be skipped. Similarly, the specification establishes that when deriving a public child key it should be discarded if it is the identity point (see Section V, D.2). This check is not included in the code either, but this happens with an extremely negligible probability, so in practice it is also safe to skip it (although it could be added for the sake of clarity).

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Miscellaneous Issues

This section covers any and all noteworthy findings that did not incur an exploit but may assist an attacker in successfully achieving malicious objectives in the future. Most of these results are vulnerable code snippets that did not provide an easy method by which to be called. Conclusively, while a vulnerability is present, an exploit may not always be possible.

MM-04-001 WP1: Unchecked entropy size in *entropyToCip3MasterNode* *(Info)*

While auditing the function that derives the master node from the entropy according to CIP3, it was found that the size of the entropy is implicitly assumed to be as mandated by the standard (i.e. a multiple of 32 bits within the range 128-256). As such, the function never ensures that the entropy size is valid.

Affected file:
key-tree/src/derivers/bip39.ts

Affected code #1:
```
export async function entropyToCip3MasterNode(
  entropy: Uint8Array,
  curve: Extract<Curve, { masterNodeGenerationSpec: 'cip3' }>,
): Promise<SLIP10Node> {
  const rootNode = pbkdf2(sha512, curve.secret, entropy, {
    c: 4096,
    dkLen: 96,
  });
```

Note that this does not result in a problem within the repo *key-tree*. This is because the function is invoked with the entropy that results from the execution of the function *mnemonicToEntropy()*, as can be seen in the following code excerpt, and that function does ensure that the size of the entropy is valid.

Affected code #2:
```
export async function deriveChildKey({
  path,
  curve,
}: DeriveChildKeyArgs): Promise<SLIP10Node> {
  switch (curve.masterNodeGenerationSpec) {
    case 'slip10':
      return createBip39KeyFromSeed(
        await mnemonicToSeed(path, englishWordlist),
        curve,
      );
    case 'cip3':
      return entropyToCip3MasterNode(
        mnemonicToEntropy(path, englishWordlist),
```

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

```
        curve,
      );
    default:
      throw new Error('Unsupported master node generation spec.');
  }
}
```

Given the crucial importance of the master node for the overall security of an HD wallet and considering that the function *entropyToCip3MasterNode* is exported, it is recommended to ensure that the size of *entropy* is as expected (i.e. 16, 20, 24, 28, 32 bytes).

MM-04-002 WP1: Unneeded modular reduction in private key derivation *(Low)*

While reviewing the private child key derivation, it was noticed that a reduction of K_R (as denoted in the specification) modulo 2^{256} is performed. However, K_R is a 32-bytes integer and as a consequence it is strictly less than 2^{256}. As a consequence, the modular reduction leaves K_R unaltered and serves no purpose.

Affected file:
key-tree/src/derivers/cip3.ts

Affected code #1:

```
export const derivePrivateKey = async ({
  parentNode,
  childIndex,
  isHardened,
}: DeriveWithPrivateArgs) => {
  const extension = isHardened
    ? getKeyExtension(
        Z_TAGS.hardened,
        parentNode.privateKeyBytes,
        childIndex + BIP_32_HARDENED_OFFSET,
      )
    : getKeyExtension(Z_TAGS.normal, parentNode.publicKeyBytes,
childIndex);

  [...]

  const zl = entropy.subarray(0, 32);
  const zr = entropy.subarray(32);

  const parentKl = parentNode.privateKeyBytes.subarray(0, 32);
  const parentKr = parentNode.privateKeyBytes.subarray(32);

  // 8[ZL] + kPL
  const childKl = add(trunc28Mul8(zl), parentKl);
  // ZR + kPR
```

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

```
  const childKr = add(zr, mod2Pow256(parentKr));

  return concatBytes([childKl, childKr]);
};
```

Note that it is the sum Z_R+K_R that needs to be reduced modulo 2^{256}, not the summands K_R or Z_R alone. However, even though the sum is not explicitly reduced by calling the function *mod2Pow256*, the addition is still reduced modulo 2^{256}. Note that such a step is performed by the function *add*, as shown in the following code excerpt.

Affected code #2:

```
export const add = (left: Uint8Array, right: Uint8Array): Uint8Array => {
  const added = bytesToBigInt(left) + bytesToBigInt(right);
  return padEnd32Bytes(bigIntToBytes(added)).slice(0, 32);
};
```

Note that although the overall behavior of the function is correct, it is recommended to remove the redundant modular reduction of K_R for clarity and performance reasons. Besides, since the modular reduction is performed in the *add* function, the auxiliary function *mod2Pow256* serves no purpose and should be removed.

MM-04-003 WP1: Failure to reject invalid extended BIP32-Ed25519 keys *(Medium)*

While auditing the validation of BIP32-Ed25519 extended private keys, it was observed that any 64-byte array is declared as a valid extended private key. However, valid extended BIP32-Ed25519 private keys (K_L, K_R) need to have the lowest 3 bits of the first byte of K_L cleared, the highest bit of the last byte of K_L cleared, and the second highest bit of the last byte of K_L set.

Note that the above properties are explicitly forced during the generation of the master node. Besides, derived private child keys also satisfy those requirements by design.

Affected file:
key-tree/src/curves/ed25519Bip32.ts

Affected code:

```
export const isValidPrivateKey = (_privateKey: Uint8Array | string |
bigint) =>
  True;
```

Nevertheless, the method *fromExtendedKey* accepts extended private keys that do not satisfy those properties, and the impact of this entirely depends on how the MetaMask *key-tree* library is leveraged by an application making use of it.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Affected file:
key-tree/src/SLIP10Node.ts

Affected code:

```
static async fromExtendedKey({
  depth,
  masterFingerprint,
  parentFingerprint,
  index,
  privateKey,
  publicKey,
  chainCode,
  curve,
}: SLIP10ExtendedKeyOptions) {
  const chainCodeBytes = getBytes(chainCode, BYTES_KEY_LENGTH);

  [...]
  if (privateKey) {
    const privateKeyBytes = getBytesUnsafe(
      privateKey,
      curveObject.privateKeyLength,
    );
    assert(
      curveObject.isValidPrivateKey(privateKeyBytes),
      `Invalid private key: Value is not a valid ${curve} private key.`,
    );
  [...]
}
```

To mitigate this issue, it is recommended to ensure that the lowest 3 bits of the first byte of K_L and the highest bits of the last byte of K_L satisfy the desired properties.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Conclusions

The cryptography review and source code audit against the MetaMask *key-tree* interface described in this report was performed by Cure53 in April 2024. The audit identified a total of three findings, all of which were classified as general weaknesses with lower exploitation potential.

From a contextual perspective, four working days were allocated to reach the coverage expected for this project. A white-box testing methodology was used to assess the specified scopes, whereby the Cure53 team of two senior testers was granted access to sources, an overview of all implemented changes since the last audit, and all further means of access required to complete the tests. The assignment consisted of a single WP, a cryptography review and audit of the *key-tree* library. This library was previously audited by Cure53 in February 2023 (see MM-02). In particular, ConsenSys Software Inc. defined the focus to be the added support for Cardano key derivation according to CIP3-Icarus.

The functions involved in the child key derivation were reviewed. As the basic cryptographic operations are performed by a third-party library, the potential for flaws is reduced. In this sense, it was confirmed that the library is adequately leveraged. However, the validation of BIP32-Ed25519 private keys is not strict (MM-04-003) and could lead to issues.

Next, the added key derivation as indicated by CIP3 was audited. The derivation of the master node was reviewed first, and was found to be compliant with the standard. However, the Cure53 team noticed that the function that derives the master node does not ensure that the size of the input entropy is valid (see MM-04-001). Although the function is called with an input of a valid length in the *key-tree* library, this finding was reported as a miscellaneous issue because the function is exported.

The implementation of the child key derivation follows the standard and behaves as expected. When auditing the private child key derivation it was noticed that there is a function which is defined with the sole purpose of performing a modular reduction. However, the code invokes the function on a summand instead of invoking it on an addition (see MM-04-002), which would be the correct approach. Despite this, the correct modular reduction is performed in the *add* function itself, and so the private key derivation is correct. This might have occurred due to an oversight of the specification, but it can easily be addressed to achieve better clarity and performance.

In the timeframe of this assignment, the added support for the Cardano key derivation was found to be compliant with the CIP3-Icarus standard. The spotted issues are not major, but Cure53 still recommends that they be addressed.

Cure53 would like to thank Christian Montoya, Frederik Bolding, and Maarten Zuidhoorn from the ConsenSys Software Inc. team for their excellent project coordination, support and assistance, both before and during this assignment.