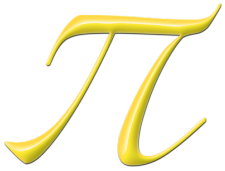

Paragon Initiative Enterprises

NaclKeys

Source Code Audit
18 July 2015

I. Introduction

This document describes the security audit of the NaclKeys library by Paragon Initiative Enterprises for Christian Hermann (bitBeans).

Our audit targeted git commit `e19089397bb73cad4f11d4a2c038a3bc81867d7f`, which was committed on June 2, 2015.

This report was prepared by Scott Arciszewski, CDO, and reviewed by Robyn Terjesen, CEO.

Audit Results Summary

After a comprehensive code review of the NaclKeys project, we did not identify any security vulnerabilities or non-exploitable security concerns.

II. Audit Scope

We have limited the scope of this audit to focus specifically on the contents of the NaclKeys source code repository on Github.

We excluded the following from our scope (although, where appropriate, we did verify that their features were being used safely):

- Base58Check
- The .NET bindings for libsodium
- Zxcvbn

III. Issues

No security vulnerabilities were discovered in the NaclKeys project.

IV. Other Findings

Note: The findings in this section are not necessarily vulnerabilities.

1. Checksum Calculation Can Be Optimized

The `CalculateBytejailChecksum()` method in `NaclKeys.KeyGenerator` calculates a BLAKE2b hash (provided by libsodium) of the version and the public key, then calculates a second hash and returns the first 4 bytes of the second hash:

```
    var hashRound1 = GenericHash.Hash(ArrayHelpers.ConcatArrays(version,
publicKey), null, 64);
    var hashRound2 = GenericHash.Hash(hashRound1, null, 64);
```

However, only the first 4 bytes of `hashRound2` is returned by this method. The obvious patch is to remove the 6, thus changing the second line to look like this:

```
    var hashRound2 = GenericHash.Hash(hashRound1, null, 4);
```

This results in a `BytesOutOfRangeException` being thrown. One of the decisions made in the libsodium is to enforce a minimum output size of 16 bytes.

Consequently, we offered pull request #1 which changed this line to only return 16 bytes instead of 64 bytes. However, due to how the hashing is implemented in libsodium, the checksum this generates is not backwards compatible with the version that produced a full 32 byte output.

Given that there are no significant real-world performance implications for discarding 60 bytes versus discarding 12 bytes, and it would break backwards compatibility, we do not strongly recommend our pull request be merged.

2. Base58 Encoding *Might* Contain Side-Channels

We are not aware of any practical cache-timing attacks on base-N encoding at this time.

However, most implementations use an index lookup based on the raw bytes to construct an encoded string, which leaks timing information about the binary data due to processor caching. It is for this very reason that libsodium offers its own cache-timing-safe binary-to-hexadecimal string and hexadecimal-to-binary string utilities.

More academic research is needed into the risk of cache-timing side-channels in encoding functions before we consider them a valid security concern. For now, they bear mentioning but no action is needed.

V. Conclusion

After a thorough analysis of the NaclKeys library, we did not discover any security vulnerabilities or other non-exploitable security concerns. We can attribute this result to two factors:

1. All cryptographic operations (hashing, scrypt key derivation, generating a keypair from an scrypt hash output, etc.) are handled by libsodium, which is designed to prevent implementors from shooting themselves in the foot.

2. NaclKeys is a lightweight library that doesn't attempt to do too much. The code is conservative and apparently consistent with the UNIX philosophy of doing one thing, and doing it very well.

We can say with confidence that this library will not be the weak point of any project that builds atop it.