A CONSENSYS DILIGENCE AUDIT REPORT

Thesis - Cryptographic Review

Date	March 2020
Auditors	John G. Brainard

1 Overview

This document is the result of a design and code review of the cryptographic constructions and algorithms used in the keep network. This cryptographic review was performed as part of the TBTC and keep Audit.

1.1 Scope

The design review is based on the documents in the keep-core repository [KC20] and the Keep Random Beacon Yellow Paper [KN20]. The review evaluates the soundness of the protocol and the appropriate use of cryptographic elements. It is not a detailed cryptanalysis of the cryptographic algorithms used.

The code review is based on the source code in the keep-core repository. The goal of the code review is to ensure that cryptographic primitives are being used appropriately, and with proper cryptographic hygiene. It does not constitute a formal proof of correctness or an assurance that the code is free of defects. Note that the ECDSA implementation used [KD20] is excluded from this review, and is assumed to be correct.

2 Design Review

2.1 Design Overview

The primary reference for the design review is the Keep Network Yellow Paper [KN20], which describes both the architecture of the random beacon scheme and the ticketing protocol based on the output of the random beacon. The document delineates a number of possible attacks and demonstrates the effectiveness of the protocol at preventing these attacks.

2.2 Design Elements

GJKR Protocol

The Keep Random Beacon is implemented using the GJKR distributed key generation protocol [GJKR99]. While earlier protocols for distributed key generation exist, GJKR was the first to insure that malicious participants cannot influence the final value of the generated private key. GJKR has been widely studied and implemented for over twenty years, with no significant flaws found.

The variant of the GJKR protocol used in the Keep Network modifies the original specification to work over a single shared channel, rather than separate channels between participants. It also adds the ability to explicitly identify participants who violate the protocol. Neither of these changes affect the overall security properties of the protocol which should be comparable in security to the original GJKR scheme.

ECDSA

The signature algorithm used as the basis for the GJKR protocol used here is the Elliptic Curve Digital Signature Algorithm [NIST13]. ECDSA is the established standard for digital signatures on elliptic curves and has been studied and used for many years.

During the design review, it was suggested that the request for a signature should include the actual data being signed instead of just the digest value. Our conclusion is that if an attacker could create false digest and signature pairs from a valid pair that either the underlying signature scheme, or the hash function used, must be broken. Since the security of the protocol assumes the soundness of both the hash algorithm and signature schemes, there is no need to include the preimage.

AltBN_128 Curve

The `AltBN_128` Elliptic Curve [YCKS19] is used widely in existing Ethereum applications. This is an efficient choice of curve and has the appropriate field size to correspond with the other cryptographic algorithms used in the protocol.

No specific cryptographic attacks against the `AltBN_128` curve have been published, but recent mathematical advances [KB16] have raised concerns about its security level. An improved version of the Number Field Sieve specific to the moduli used in pairing-friendly curves has led to the conclusion that `AltBN_128` and similar curves offer approximately 96 bits of security, rather than the expected 128.

This is not a catastrophic attack, and it has yet to be implemented in practice, but it does suggest changes for the future. The broader ethereum community is looking at other pairing-friendly curves with a larger underlying field size, such as the BLS12-381 curve [YCKS19].

There is no need to immediately abandon the `AltBN_128` curve, but cryptographic prudence suggests making the curve implementation as modular as possible. This allows a new curve to be substituted in the future, without difficult modifications to the design and code base.

SHA2-256

The signature algorithm relies on the use of a secure hash function, which is assumed to be resistant to both preimage and collision attacks. In this design, the hash function used is the 256-bit variant of the SHA-2 algorithm, part of NIST's Secure Hash Standard [NIST15].

The use of a 256-bit hash function is appropriate when paired with the `AltBN_128` Elliptic Curve. While collision attacks have been discovered against earlier algorithms in the SHA series [WYY05], SHA2-256 has, to date, provided the expected level of resistance to both preimage and collision attacks.

2.3 Design Recommendations

1. The use of the `AltBN_128` curve should be modularized, as much as possible, to allow an alternate curve to be used in future implementations. This may be required if the use of the `AltBN_128` curve is deprecated in Ethereum at some point in the future.

2. In the Yellow Paper section labeled “Interactive protocol”, the second option for a non-interactive version of the protocol is said to violate requirement 5 for the protocol, but the document only lists 4 requirements. I believe it should specify requirement 4.

3 Code Review

The keep-core repository contains code, in both go and solidity, that implements the system described in the Keep Network Yellow Paper [KN20]. No critical issues were discovered in the course of this review.

The source code modules which use or implement cryptographic operations are listed below, along with some observations on each module.

3.1 Code Modules

`/pkg/beacon/relay/gjkr/protocol.go`

This module is an implementation, in go, of the GJKR distributed key generation protocol as specified in the Yellow Paper.

1. The `GenerateEphemeralKeyPair()` function corresponds properly with Phase 1 of the protocol as defined in the Yellow Paper.
2. The `GenerateSymmetricKeys()` function corresponds properly with Phase 2 of the protocol as defined in the Yellow Paper.
3. The `CalculateMembersSharesAndCommitments()` function corresponds properly with Phase 3 of the protocol as defined in the Yellow Paper.
4. The `VerifyReceivedSharesAndCommitmentsMessages()` function corresponds properly with Phase 4 of the protocol as defined in the Yellow Paper.

5. The `ResolveSecretSharesAccusationsMessages()` function corresponds properly with Phase 5 of the protocol as defined in the Yellow Paper.
6. The `CombineMemberShares()` function corresponds properly with Phase 6 of the protocol as defined in the Yellow Paper.
7. The `CalculatePublicKeySharePoints()` function corresponds properly with Phase 7 of the protocol as defined in the Yellow Paper.
8. The `VerifyPublicKeySharePoints()` function corresponds properly with Phase 8 of the protocol as defined in the Yellow Paper.
9. The `ResolvePublicKeySharePointsAccusationsMessages()` function corresponds properly with Phase 9 of the protocol as defined in the Yellow Paper.
10. The `RevealMisbehavedMembersKeys()` function corresponds properly with Phase 10 of the protocol as defined in the Yellow Paper.
11. The `ReconstructMisbehavedIndividualKeys()` function corresponds properly with Phase 11 of the protocol as defined in the Yellow Paper.
12. The `findPublicKey()` function corresponds properly with Phase 12 of the protocol as defined in the Yellow Paper.

`/pkg/beacon/relay/dkg/result/signing.go`

This module computes the shared signature.

1. The `SignDKGResult()` function does proper error checking on both the hash and signature computations

`/pkg/altbn128/altbn128.go`

This module implements all the elliptic curve operations, on the `altbn_128` curve.

1. The Cloudflare library is used for big number arithmetic.
2. The `AltBN_128` Curve parameters are properly defined.
3. The curve operations appear to be correctly implemented.

`/pkg/bls/bls.go`

This module implements the BLS distributed signature algorithm.

1. The `Sign()` and `Verify()` functions invoke the corresponding operations in `altbn128`.

2. The `RecoverSignature()` function does proper validation on the individual shares and threshold testing on the number of shares.

`/pkg/operator/key.go`

This module generates and manages ECDSA keys.

1. The `GenerateKeyPair()` function generates the ECDSA key pair for the participants in the protocol.

`/pkg/chain/local/signing.go`

This module does local signatures using the ECDSA algorithm.

1. Local signing and verification of ECDSA signatures is done by the `Sign()` and `Verify()` functions.
2. Inputs are properly validated, and proper error checking is done in both functions.

`/contracts/solidity/contracts/cryptography/AltBn128.sol`

This module implements all the elliptic curve operations, in solidity, on the `altbn_128` curve.

1. An internal library (`/contracts/solidity/contracts/utils/ModUtils.sol`) is used for big number arithmetic.
2. The `AltBN_128` Curve parameters are properly defined.
3. The curve operations appear to be correctly implemented.

`/contracts/solidity/contracts/cryptography/BLS.sol`

This module implements the BLS distributed signature algorithm in solidity.

1. The `sign()` and `verify()` functions invoke the corresponding operations in `AltBn128.sol`.

3.2 Code Recommendations

No significant issues were identified in this review that would mandate modifications to the code. The design review recommends modularizing the elliptic curve operations, so that new curves might be used in the future. This could be done by creating a “wrapper class” for curve operations, with a curve ID as a class member. This would allow the use of new curves without major modification to all the code using the curves.

4 Conclusions

No significant issues were found in the cryptographic review of the design and code. The design is clearly and completely specified in the Yellow Paper, and it is a sound application of the widely-studied GJKR signature scheme. The cryptographic primitives used are appropriate and conform to industry best practices. The code is clearly written and the invocations of cryptographic algorithms follow proper cryptographic hygiene in both parameter checking and cleanup.

Appendix 1 - References

- **[GJKR99]** R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Advances in Cryptology — EUROCRYPT ’99: International Conference on the Theory and Application of Cryptographic Techniques Prague, Czech Republic, May 2–6, 1999 Proceedings, chapter Secure Distributed Key Generation for Discrete-Log Based Cryptosystems, pages 295–310. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999 ; http://groups.csail.mit.edu/cis/pubs/stasio/vss.ps.gz

- **[KB16]** T. Kim and R. Barbulescu, “Extended tower number field sieve: A new complexity for medium prime case”, Advances in Cryptology – CRYPTO 2016, LNCS 9814 (2016), 543–571.

- **[KC20]** https://github.com/keep-network/keep-core

- **[KD20]** https://github.com/keep-network/keep-ecdsa

- **[KN20]** “The Keep Random Beacon: An Implementation of a Threshold Relay”, https://docs.keep.network/random-beacon/

- **[NIST13]** National Institute of Standards and Technology, “Digital Signature Standard”, FIPS PUB 186-4, https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf, July 2013

- **[NIST15]** National Institute of Standards and Technology, “Secure Hash Standard”, FIPS PUB 180-4, https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf, August 2015
- **[YCKS19]** Yonezawa, Chikara, Kobyashi, and Saito, “Pairing-Friendly Curves”, Internet Draft, https://tools.ietf.org/id/draft-yonezawa-pairing-friendly-curves-00.html, January 2019
- **[WYY05]** Wang, Yin, and Yu, “Finding Collisions in the Full SHA-1”, https://www.iacr.org/archive/crypto2005/36210017/36210017.pdf, August 2005

Appendix 2 - Disclosure