End-to-End Encrypted Backups Security Assessment

WhatsApp

October 27, 2021 – Version 1.2

Prepared by

Gérald Doussot

Marie-Sarah Lacharité

Eric Schorn

Table of Contents

Executive Summary

Synopsis

During the summer of 2021, WhatsApp engaged NCC Group's Cryptography Services team to conduct an independent security assessment of its End-to-End Encrypted Backups project. End-to-End Encrypted Backups is an hardware security module (HSM) based key vault solution that aims to primarily support encrypted backup of WhatsApp user data. This assessment was performed remotely, as a 35 person-day effort by three NCC Group consultants over the course of five weeks. NCC Group and the WhatsApp team scheduled the retesting of findings, and preparation of this public report a few weeks later, following the delivery of the initial security assessment.

Project Scope and Approach

This assessment was part of a larger program of work, articulated around five phases:

- **Phase 1**: End-to-End Encrypted Backups review:
  - Overall architecture review, with a focus on the cryptographic implementation.
- **Phase 2**: "opaque-ke" open-source library review:
  - Review of the server-side cryptographic implementation of the OPAQUE protocol.
- **Phase 3**: HSM and key management:
  - HSM consensus protocol review.
  - Key management implementation review.
  - Key ceremony guidance.
- **Phases 4 and 5**: Retesting and public report creation:
  - Testing of any gaps from earlier rounds of testing, to ensure completeness of coverage.
  - Retesting of any recommended fixes, mitigations, or improvements.
  - Public reports:
    - End-to-End Encrypted Backups report.
    - "opaque-ke" report.

This public report discusses WhatsApp's End-to-End Encrypted Backups only. The implementation of the server-side OPAQUE protocol by the "opaque-ke" library will be covered separately. For the End-to-End Encrypted Backups assessment phases, NCC Group's evaluation included:

- **End-to-End Encrypted Backups HSM Secure Execution Environment (SEE)**: Embedded, custom security software that runs within an HSM.
- **End-to-End Encrypted Backups Server**: Thrift server colocated on an HSM host. The server connects SEE to the world outside the HSM.
- **HSM common utilities**: Used by SEE and End-to-End Encrypted Backups server.
- **HSM configuration**: HSM initialization scripts.
- **Chatd End-to-End Encrypted Backups module**: Relay between external mobile clients and internal End-to-End Encrypted Backups Server endpoints.
- **WA-MSYS**: OPAQUE protocol client library.
- **WhatsApp iOS and Android Clients End-to-End Encrypted Backups interface**: Mobile client interface to End-to-End Encrypted Backups service.
- **Merkle library interface**: Library used by End-to-End Encrypted Backups to protect data hosted on untrusted storage.

The consultants reviewed the above components starting at `fbsource` changeset `D28877521`, which included End-to-End Encrypted Backups related changeset `D28875781`.

The WhatsApp team further requested NCC Group to identify any gaps between the End-to-End Encrypted Backups whitepaper[1] technical claims, and its actual implementation, for the components in scope at the end of the project. NCC Group delivered this additional scope item as part of phase 5 of the program of work.

[1] https://www.whatsapp.com/security/WhatsApp_Security_Encrypted_Backups_Whitepaper.pdf

NCC Group assessed End-to-End Encrypted Backups for common cryptographic and software vulnerabilities using mostly code review; the consultants performed some dynamic testing, leveraging existing unit test cases.

Limitations

The following components were not in scope; NCC Group was therefore unable to evaluate and identify issues with them:

- Third-party and proprietary HSM vendor implementation.
- Backup encryption implementation.
- Side-channels in the access, creation, modification and deletion of backup data on third-party cloud storage.
- End-to-End Encrypted Backups Reverse Proxy.
- OPAQUE security protocol.

Nevertheless, NCC Group provided good test coverage of all the components in scope for this review, as listed in the previous section, in the context of the End-to-End Encrypted Backups service.

Key Findings

The assessment uncovered a number of common cryptography and application flaws. The most notable findings were:

- **Weak 512 bits RSA key signing key**, allowing attackers to impersonate the HSM and OPAQUE services, and to decrypt user backups.
- **Insufficient validation of OPAQUE protocol data**, that may permit attackers, in the worst case, to recover a user's WhatsApp password and encrypted backup information.
- **Key material in lower integrity environment**, may facilitate a complete bypass of the HSM and thus of the End-to-End Encrypted Backups service security assurances that the HSM provides.
- **Several weaknesses in the handling of passwords**, which may allow an attacker in general to more easily brute-force user passwords.
- **Insufficient access controls** for the End-to-End Encrypted Backups service from the internal WhatsApp infrastructure may permit unauthorized large scale operation on user data such as account deletion. This issue also results in an increase in the attack surface of the End-to-End Encrypted Backups service in general.
- **Denial of service attacks** potential due to the lack of enforcement of certain limits.

NCC Group also compared the technical claims stated in the WhatsApp End-to-End Encrypted Backups whitepaper version 1.0 with its implementation for the components in scope for this review, and found no discrepancies at the time of the assessment.

Retest Summary

The WhatsApp team implemented a number of changes to address NCC Group's findings. NCC Group retested these changes at the end of August 2021. During the assessment, NCC Group identified:

- one (1) high severity vulnerability;
- nine (9) medium severity vulnerabilities;
- seven (7) low severity vulnerabilities;
- six (6) informational findings.

Upon completion of the assessment, all findings were reported to WhatsApp, along with recommendations. After retesting, and before the solution was rolled out to users, fifteen findings were found to be fully fixed. There were eight remaining findings, which NCC Group reported to WhatsApp. WhatApp provided details about these findings and their response to them in WhatsApp's Response to NCC Group Findings on page 22 of this report.

Other Remarks

All findings are listed in Table of Findings on page 21, along with their retest status. WhatsApp's response to several findings can be found in WhatsApp's Response to NCC Group Findings on page 22. NCC Group also provided a high-

level security evaluation of the End-to-End Encrypted Backups architecture design in WhatsApp Encrypted Backups Secrets Management Solution Architecture on the next page including key design security assumptions, and recommendations for establishing and managing key ceremonies in Trusted Setup Ceremony on page 16. An interesting attack, generalizable to any consumers of the OPAQUE protocol, was described in some details in Generic Attacks on OPAQUE on page 12. Finding details and other informational sections were omitted from the public document. Note that missing finding numbers in Table of Findings on page 21 are an artefact of NCC Group's internal reporting tools and do not imply missing findings; NCC Group did not omit any findings from this report.

Introduction

WhatsApp is a cross-platform messaging and calling service, with more than 2 billion users in over 180 countries. WhatsApp claims[2] that its application is end-to-end encrypted (E2EE), thus ensuring that only intended senders and recipients can access the contents of their communication. E2EE was born out of the need to prevent infrastructure and service providers from reading or tampering with user messages, especially in the case where communications are terminated at a third-party server and meddling-in-the-middle is trivial.

E2EE protects data "in-flight"; any E2EE mobile application may further process exchanged messages present on the device for various purposes, including re-sharing received messages' media contents on social networks, or exporting them to other applications. When messages are processed outside of the original E2EE channel in which they were sent or received, users may lose some of the security and privacy guarantees that they enjoyed so far. One such scenario is the backup of messages to a third-party cloud service, where the service can read, tamper with, or share the messages in the absence of other security controls.

WhatsApp designed and implemented an encrypted backup solution to prevent third-party cloud services, and itself, from accessing the contents of WhatsApp messages backed-up to their infrastructure, under the umbrella of the End-to-End Encrypted Backups project. This section describes the WhatsApp encrypted backup solution that came out of this project. It first presents its security architecture at a high level, then analyzes its prominent security and privacy features.

This review focused on the secure management of secrets underpinning the encrypted backup solution, including user passwords, and symmetric and asymmetric cryptographic keys. Actual encryption and decryption of backup data on mobile clients was not in scope, as the focus of the project was mostly key storage; however, NCC Group captured several notes and findings in relation to the underlying cipher primitives.

Encrypted Backup Architecture

Goals

WhatsApp provides end-to-end encryption for all messages, thus only users and their interlocutors can read what is sent. This same protection does not apply to a user's message backups. If a user chooses to backup their messages, the backups are stored on a third-party cloud service where there is a risk that the backups can either be read or tampered with, or shared with other parties.

The goal of this project is to extend WhatsApp's commitment to privacy to the user's backup. This project enables users to encrypt their backups in such a way that nobody, including WhatsApp, can access their data.

Overview

Users can choose to enable encrypted backups. When they do, the WhatsApp mobile client generates a random backup key `K`. `K` is stored locally on the device for future encryption/decryption of user backup data. Nobody can decrypt message backups without access to the plaintext backup key `K`.

Backup key `K` is itself backed-up outside of the device, in case the user loses access to it (e.g. lost mobile device). To achieve that, backup key `K` is stored in encrypted form by the WhatsApp service; however controls are in place to prevent the service from decrypting the backup key. The following is a high level description of how the backup key `K` is protected from WhatsApp, and third-party cloud services.

Users are asked to provide a PIN or passphrase. This user secret is mixed with a per-user hardware security module secret derived from a secret seed (`oprf_seed`), to produce an export key, using the OPAQUE[3] asymmetric PAKE protocol. The export key is known by the WhatsApp client only, and is created during registration, and regenerated at login time; it can only be regenerated with the knowledge of the PIN/passphrase and with contribution from the HSM secret seed.

[2] https://www.whatsapp.com/security/
[3] https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-opaque-05

Backup key `K` is first encrypted using the user export key to produce `aes_k`. `aes_k` is then encrypted using the HSM public key, along with other required information, to produce an encrypted registration payload. This encrypted registration payload is stored server-side. Decrypting the backup key stored server-side would require the user PIN/passphrase, the secret seed and the HSM private key. The last two cannot be accessed outside of the HSM. `aes_k` can only be decrypted by the HSM upon a successful OPAQUE login from the client. Furthermore, the HSM limits the amount of incorrect login attempts using the user PIN/passphrase to ten, at which point the account is irremediably locked by the HSM software, and the backup data cannot be retrieved in plaintext. Once deployed to production, the HSM code becomes immutable, thus limiting the ability to tamper with the HSM code incorrect login counter and logic, in the absence of vulnerabilities.

A Merkle tree, with the root hash stored in the HSM, ensures that encrypted registration payloads cannot be tampered with, e.g., one payload cannot be replaced with another perhaps older but valid payload.

Security Assumptions

Below is a list of key security assumptions that underpin the security of the overall architecture:

1. After destroying the hardware security module[4] (HSM) and management cards, it is impossible to extract HSM managed secrets. Specifically, there is no API or management functionality to expose and extract these secrets. There are no vulnerabilities in the third-party HSM implementation that allows extracting these secrets.
2. The End-to-End Encrypted Backups service is authenticated by clients against a public key obtained when the app is first downloaded, or when updated (e.g. when the baked-in HSM fleet keys are installed). The End-to-End Encrypted Backups project service was originally trusted on first use[5] (TOFU), specifically at registration time, to a large extent (but controls existed to make the security of this process harder to circumvent). The change happened as a result of the security assessment.
3. HSM release management, configuration and key ceremony processes produce an immutable HSM configuration that does not permit exfiltration of HSM managed secret material before, during or after commissioning the devices to production.
4. WhatsApp configuration and release management processes produce a client build that only trusts securely commissioned HSMs.
5. Compromise of any aspect of a client device or software (e.g. operating system), is out of scope.
6. The Chatd service component adequately extracts and annotates client user ID when forwarding requests from clients to the End-to-End Encrypted Backups project service.
7. Side-channels in transmitting, accessing, creating, modifying or deleting encrypted backup information on cloud services are out of scope.
8. There are processes in place to continuously ensure that the WhatsApp mobile application does not include any malicious code or data, including but not limited to rogue HSM certificates or PIN exfiltration code.

[4] https://en.wikipedia.org/wiki/Hardware_security_module
[5] https://en.wikipedia.org/wiki/Trust_on_first_use

Components Landscape

Below is an illustration of the main components of the encrypted backups solution.

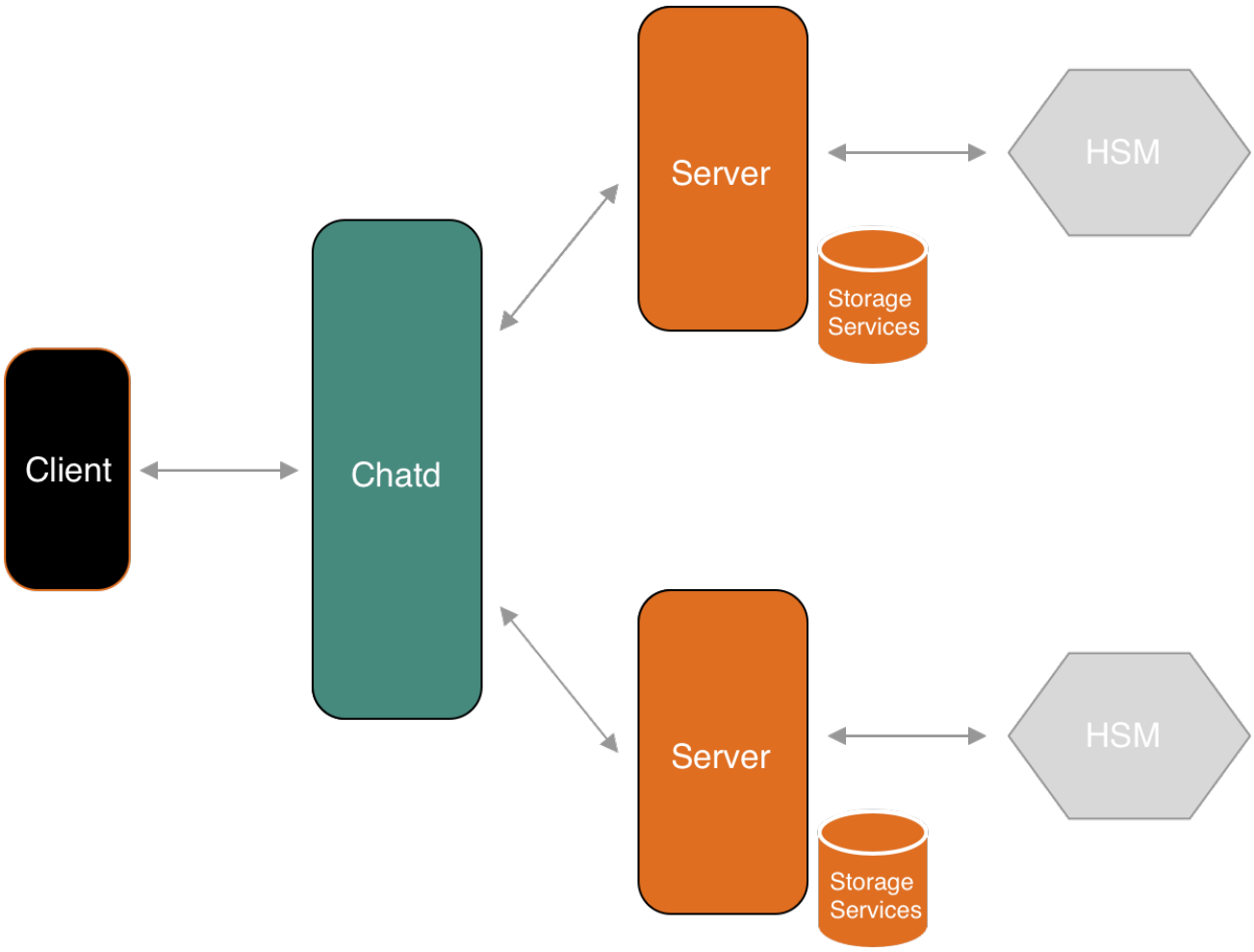

Figure 1: Encrypted Backups Architecture Components

We describe the role and security features of these components in the following sections.

Hardware Security Module

The HSM (hardware security module) is foundational to the security of the solution. It is composed of a hardware module performing cryptographic operations, and custom embedded software code developed by WhatsApp that underpins many of the security features of the encrypted backups solution. The custom embedded software code is written in a memory-safe language in order to reduce risks of catastrophic vulnerabilities in that component. It exposes an API via host shared memory to register and delete accounts and to authenticate users via the OPAQUE protocol.

The HSM manages and does not externally expose the OPAQUE `oprf_seed` secret, used to randomize the user PIN/passphrase using OPAQUE. It encrypts/decrypts registration payloads, including the client-encrypted backup key `aes_k`, using its own non-exportable asymmetric keys (`HK_pub`).

It enforces a maximum number of user login attempts, in order to prevent brute-force of a user PIN/passphrase; after

ten unsuccessful attempts to log into an account, the account is locked and the backup data is irremediably lost.

It stores registration payloads, including keys, in an encrypted format on external untrusted storage. It verifies that externally stored data is not compromised, using a Merkle tree, whose root hash is maintained within the HSM embedded software and whose leaves are hashes of the registration payloads.

Communication to the HSM is performed via shared memory from the host. HSMs are organized into islands that share state using the Raft[6] consensus algorithm for redundancy and performance reasons. Once commissioned to production, it is not possible to add more, potentially malicious, nodes to an island. HSM management cards are destroyed during the key ceremony, thus making alterations to the HSM configuration and embedded code impossible, in the absence of vulnerabilities.

End-to-End Encrypted Backups Server

The End-to-End Encrypted Backups Server is a server software component that provides an interface between the HSM embedded software, the WhatsApp mobile clients and internal resources such as storage services. It resides on the same machine as the HSM card and communicates with the HSM using shared memory. It is designed to not provide critical security and privacy guarantees by delegating these features to a hardened HSM, to resist internal compromises of the infrastructure.

The End-to-End Encrypted Backups Server is accessed over HTTPS. It exposes APIs to register and delete End-to-End Encrypted Backups user accounts, and to authenticate users.

End-to-End Encrypted Backups Storage Services

The End-to-End Encrypted Backups Storage Services stores HSM managed and encrypted information. The HSM is the only component that can decrypt this information. As explained above, information contained in End-to-End Encrypted Backups Storage Services is hashed and added to a Merkle tree managed by the HSM. Tampering with the End-to-End Encrypted Backups Storage Services data is detected by the HSM.

Chatd

Chatd is the main WhatsApp server which handles all external client requests. Chatd forwards requests from clients to the End-to-End Encrypted Backups service and can be considered as a simple relay. Of note, it extracts authenticated user IDs from client requests and provides these to the End-to-End Encrypted Backups Server APIs, along with the actual client requests.

iOS/Android Clients

The WhatsApp client runs on iOS and Android devices. It stores the encryption key K, which is established when enabling encrypted backups, and then used to encrypt backups before they are sent to cloud services. The WhatsApp client also contains HSM public key material, critical in establishing a trust relationship with legitimate HSMs. If K is lost, it can only be retrieved from WhatsApp and decrypted with knowledge of the user PIN/passphrase and the HSM secret seed.

OPAQUE Security Protocol Usage

The design relies on a number of common security controls and protocols, such as TLS and the Noise protocol framework. The OPAQUE protocol is utilized to create a trust relationship between HSMs and WhatsApp mobile clients upon registration, then use it for mutual authentication upon ulterior login. OPAQUE is an asymmetric password-authenticated key exchange (aPAKE), that provides a number of interesting features, such as the ability to hide the user password from the server, even during password registration. The solution employs two implementations of the protocol, opaque-ke which is open-source and runs on the WhatsApp HSM embedded environments, and a proprietary implementation that runs on potentially resource-constrained mobile devices. NCC Group reviewed the opaque-ke implementation, as part of a separate project.

Typically, the OPAQUE protocol requires the server secret to be split so that a compromise of one server does not allow

[6] https://raft.github.io/

brute-forcing users' credentials in an offline manner. The encrypted backup solution entrusts HSMs to ensure that the server secret does not leak. WhatsApp staff and "traditional" hosts do not have access to this per-user secret.

The OPAQUE protocol does not prevent online brute-force attacks. End-to-End Encrypted Backups mitigates such attacks through the HSM embedded code, which implements an unsuccessful login maximum count, leading to permanent account lockout when reached.

OPAQUE registration must be performed over a secure channel. In this solution, this secure channel uses the Noise protocol framework from the mobile clients to Chatd, then TLS to the End-to-End Encrypted Backups Server.

Design Security Analysis

This section considers the security design and not the implementation of the encrypted backups solution. Implementations issues were reported separately and highlighted in Table of Findings on page 21.

- The compromise of an HSM would allow performing a number of damaging attacks, including brute-force of user PIN/passphrase. The efficiency of such attacks would be roughly correlated to the strength of the PIN/passphrase. Recovery of the PIN/passphrase would in turn permit to decrypt `aes_k` to obtain `K`, and decrypt user backups. More powerful attacks may be possible in the presence of other vulnerabilities, which may for instance force the OPAQUE export key to a known value. The primary defense is the assumption that HSMs are secure. WhatsApp takes additional steps to reduce risks, including forcing the immutability of the HSM devices and embedded custom software, and ensuring a reduced attack surface at the service and API level.
- A compromised End-to-End Encrypted Backups Server on its own cannot decrypt user backups or encrypted backup keys `K`. Attackers may at most pre-register, delete or lock large numbers of user accounts (denial of service attack).
- A compromise of End-to-End Encrypted Backups Server Storage Services would at most result in the inability to access backup data. Attackers may at most delete data or change data causing the HSM to stop processing requests (denial of service attack).
- A compromise of the Chatd component may result at most in the inability to retrieve or store backup data. Attackers may at most pre-register, delete or lock large numbers of user accounts (denial of service attack).
- A compromise of a client platform would result in the ability to obtain its backup key, and to decrypt its backup data. WhatsApp cannot address all client threats, especially with regard to mobile device platform security. It uses secure mobile device platform security APIs when available, such as iOS' keychain.
- A compromise of WhatsApp client source or binary files at any point of the software lifecycle may allow an attacker to ultimately retrieve users' backups and possibly users' PINs/passphrases in a number of ways, e.g. capturing PINs, pushing rogue HSM certificates, etc. It is assumed there are controls in place to mitigate this risk.
- A compromise of a user PIN/passphrase, including via the reuse of credentials across different services, would result in the ability to obtain the user's backup key and to decrypt the corresponding backup data.

Conclusion

The design provides strong privacy guarantees for backup data stored in third-party cloud storage services from compromised or malicious third-party service providers who manage them. It provides a number of controls to strongly reduce the risk that WhatsApp accesses user backup data and/or PIN/passphrase, especially after HSMs have been deployed to production.

WhatsApp encrypted backups' security and privacy controls are based on the assumptions that HSMs are hard to compromise and that there are processes in place to ensure that HSM custom embedded code, and WhatsApp mobile app code, are free of malicious code. The design relies on these premises to provide a strong level of assurance that most compromise scenarios do not affect the privacy of backup data and of the user PIN/passphrase. Note that the third-party HSM platform was not in scope for this review, because it is a proprietary vendor solution. However, NCC Group reviewed the HSM End-to-End Encrypted Backups code developed by WhatsApp, which implements the APIs supporting encrypted backups.

WhatsApp cannot and does not make claims about the security of backup data and user credentials, when aspects that are not under its control are compromised, such as the mobile application operating platform.

The design initially assumed that the registration process had to be trusted, as it established the relationship between an HSM and the mobile application. During the security assessment, it was decided to bundle the HSM public keys with the app when installing or updating the app. This thus removed the need to trust the registration process, but one still needs to trust the client application to handle data, be it for day-to-day use or backup purposes.

The solution is more forgiving of weak PIN/passphrase in most compromise scenarios, when compared to typical password-based authentication implementations. However, in this design, users should still ensure that their PIN or passphrase cannot be guessed too easily (i.e. within the ten attempts allowed by the service).

The OPAQUE protocol is central to WhatsApp's End-to-End Encrypted Backups solution. While NCC Group's security assessment did not include analyzing the OPAQUE protocol itself, it did include analyzing WhatsApp's use of OPAQUE. In particular, NCC Group assessed how known, generic attacks on OPAQUE impact the security of WhatsApp's End-to-End Encrypted Backups solution. Three such attacks are summarized in this section. The third attack, server impersonation during credential retrieval, is presented in detail along with a discussion of how OPAQUE could be made more resistant to it.

Background

Augmented password-authenticated key exchange (APAKE) protocols like OPAQUE are inherently (and unavoidably) vulnerable to certain classes of attacks due to their reliance on passwords. For a PAKE to be considered "secure" given this limitation, it must be designed such that an adversary needs to interact with an honest client or server in order to check every single password guess. (For a discussion of the security of PAKEs, see RFC 8125, section 4.[7])

A few of these attacks are described here in terms of generic OPAQUE as specified in the draft.[8] The discussion about the third attack expands upon a class of attack described in Finding 20: Compromised Backend or Infrastructure Can Force Non-Randomized Client Password Value and Weak OPAQUE Keys. (This finding's details are not included in this public report.)

1. Honest-but-Curious Server Attack

The credential file is made of a number of records with primary keys of the format `credential_identifier` and records of the format `(client_public_key, masking_key, envelope=(nonce, inner_env, auth_tag))`. The server knows the `oprf_seed` value, therefore it can re-derive the OPRF key for a specific user's `credential_identifier`, evaluate the OPRF at the guessed password, derive the hardened `randomized_pwd`, and then derive the `masking_key`. If the derived `masking_key` matches the stored `masking_key` for that `credential_identifier`'s record, then their guess of the password was correct. This attack requires knowing the `oprf_seed` value, obtaining the credential file, and computing the hardened randomized password once per `client_identity`-guessed password pair.

2. Client Impersonation Attack during Credential Retrieval

The adversary pretends to be the client with some identity `client_identity` and performs the OPAQUE log-in/credential retrieval protocol with a guess of that client's password. The adversary is able to recover the enveloped credentials (the client's private keys) if and only if their guess of the password was correct. This attack requires interacting with the server and computing the hardened randomized password once per `client_identity`-guessed password pair.

3. Server Impersonation Attack During Credential Retrieval

The server is not authenticated during OPAQUE log-in/credential retrieval. Since client records are derived entirely from a client's identity and their password, an adversary could craft a record corresponding to a `client_identity` -guessed password pair and use it to impersonate the server when that particular client initiates OPAQUE log-in/credential retrieval. The adversary can then monitor the client's behavior to determine whether the MAC check involved in credential recovery was successful or not. It is expected that this is easy to do, since if the check passed, the client proceeds with the protocol, while if the check failed, then the client aborts and/or re-starts log-in/credential retrieval. This attack is slightly cheaper than the client impersonation attack: while it still requires an honest client to initiate OPAQUE log-in/credential retrieval, it requires only one hardened randomized password computation per guessed password (not per client-guessed password pair).

In more detail, the server impersonation attack involves the following steps.

First, the attacker chooses a commonly used, low-entropy password — `common_password`. It performs a one-time computation to craft a log-in/credential retrieval response, then monitors the network for log-in/credential retrieval requests. Next, every time it observes a credential request, it simply responds with the pre-computed response.

[7] https://datatracker.ietf.org/doc/html/rfc8125#section-4
[8] https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-opaque-06

Essentially, the adversary behaves like a server that uses an OPRF private key of 1 and has stored the credential envelope of a client whose password is `common_password`. Finally, it monitors the client's behavior to determine whether the MAC check in step 6 of `RecoverEnvelope` (OPAQUE draft section 4.2[9]) was successful or not.

Note: There is nothing special about using an OPRF key of 1; it simply makes the adversary's one-time computation easier. It could pretend to have an OPRF key `k` and the attack would work with one additional scalar inversion and scalar multiplication in the first step: the adversary would simply compute `y' = Finalize(common_password, k^(-1), GG.SerializeElement(GG.HashToGroup(common_password)))`, since `Finalize` internally calls `Unblind(k^(-1), GG.SerializeElement(GG.HashToGroup(common_password)))`.

In response to a `CredentialRequest` struct `request`, the adversary responds with the following `CredentialResponse` struct:

```
{ data:            request.data,       // copied from the request
  masking_nonce:   masking_nonce,
  masked_response: masked_response
}
```

Re-using exactly the same `masked_response` requires that `client_identity`s are not used and that their default value of being equal to `client_public_key` is used instead. If `client_identity`s *are* used, then the adversary has to compute a MAC tag over the `CleartextCredentials` struct including this `client_identity` value, so *some* per-client computation is necessary, but computing the hardened randomized password is still done only once.

The adversary's steps to craft `masked_response` (when `client_identity`s are not used) are the following:

1. Compute `y' = Finalize(common_password, 1, GG.SerializeElement(GG.HashToGroup(common_password)))` (as the client does in step 1 of `RecoverCredentials`).
2. Compute `randomized_pwd' = Extract("", Harden(y', params))` (as the client does in step 2 of `RecoverCredentials`).
3. Pick at random a nonce `random_envelope_nonce` (as the client does in step 1 of `CreateEnvelope`).
4. (For external mode) Compute an inner envelope `inner_env` (as the client does in `BuildInnerEnvelope` in draft section 4.3.2[10]):
   - Pick at random an AKE private key `client_private_key` and compute its corresponding `client_public_key`.
   - Compute `pseudorandom_pad = Expand(randomized_pwd', concat(random_envelope_nonce, "Pad"), len(client_private_key))`.
   - Compute `inner_env` as the `InnerEnvelope` struct `{ encrypted_creds: xor(client_private_key, pseudorandom_pad) }`.
5. (For internal mode) Compute the client keys (as the client does in `BuildInnerEnvelope` in draft section 4.3.1[11]):
   - Compute `seed = Expand(randomized_pwd', concat(random_envelope_nonce, "PrivateKey"), Nsk)`.
   - Compute `(client_private_key, client_public_key) = DeriveAuthKeyPair(seed)`.
   - Set `inner_env = { }` to be empty.
6. Compute `auth_key = Expand(randomized_pwd', concat(random_envelope_nonce, "AuthKey"), Nh)`.
7. Pick at random an AKE public key `random_server_public_key`.
8. Compute a `cleartext_creds` struct (as in `CreateCleartextCredentials` in draft section 4[12]):

   ```
   { server_public_key: random_server_public_key,
     server_identity:   random_server_public_key,  // or any domain name
     client_identity:   random_client_public_key   // separate client_identity must not be used
   }
   ```

9. Compute `auth_tag = MAC(auth_key, random_envelope_nonce || inner_env || cleartext_creds)`.

[9] https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-opaque-06#section-4.2
[10] https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-opaque-06#section-4.3.2
[11] https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-opaque-06#section-4.3.1
[12] https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-opaque-06#section-4

10. Compute `envelope = random_envelope_nonce || inner_env || auth_tag`.
11. Compute `masking_key' = Expand(randomized_pwd', "MaskingKey", Nh)` (as the client would have done in step 4 of `CreateEnvelope`).
12. Pick at random a `masking_nonce` (as the server would do in step 4 of `CreateCredentialResponse`).
13. Compute `credential_response_pad' = Expand(masking_key', concat(masking_nonce, "CredentialResponsePad"), Npk + Ne)`.
14. Compute `masked_response = xor(credential_response_pad', random_server_public_key || envelope)`.

The adversary sends the `CredentialResponse` struct `{ request.data, masking_nonce, masked_response }`. The client proceeds with `RecoverCredentials` (as in draft section 6.1[13]). If the adversary correctly guessed the client's password, then the client will derive the same `y = y'`, `randomized_pwd = randomized_pwd'`, `masking_key = masking_key'`, and `credential_response_pad = credential_response_pad'` (steps 1-4 of `RecoverCredentials`) as the adversary. Hence, if the adversary correctly guessed the client's password, then when the client calls `RecoverEnvelope` (step 6 of `RecoverCredentials`), the MAC verification will succeed and the client will proceed with the AKE protocol.

How to Address the Properties that Enable the Attack on Generic OPAQUE

A few properties separately make this very cheap attack possible. They are explored in turn here, and the trade-offs involved in addressing them are briefly discussed. Addressing some properties makes the attack impossible, while addressing other properties makes the attack at least as expensive as other generic attacks on OPAQUE.

1. **The server's OPRF output is not verifiable.** This means that the client has no reason to believe that the OPRF element it received in the `data` field of the `CredentialResponse` struct is the correct output of the OPRF function. That is, the client does not know whether the response includes the value of the OPRF evaluated at the blinded point they sent in the `CredentialRequest` with the private key derived from the server's OPRF seed *and* the client's `credential_identifier`. This is what allows the adversary to send a `CredentialResponse` struct where the OPRF element is the same as what the client sent.

   Modifying OPAQUE's use of OPRF to be verifiable would be complex; the VOPRF specified in a separate draft document[14] is not directly applicable to OPAQUE. Since OPAQUE's OPRF uses different keys for each client, it would be necessary to design a new proof that (i) the OPRF output is correct with respect to the server's OPRF public key, and (ii) the OPRF public key was derived from the server's seed and the client's identity or their credential identifier. Designing such a zero-knowledge proof should be possible and it **could guarantee that the server's response is correct and authentic.**

   This approach was suggested by NCC Group in the initial recommendation for Finding 20, because it would completely rule out the possibility of carrying out this attack. However, it is unclear whether the proof could be designed in a way that is compatible with OPAQUE's goal of a "PKI-free" log-in/credential retrieval process where the client does not need to store any server public keys after registration. In particular, it is **unclear whether it would be sufficient** for the OPRF proof verification to use a public key that the server provides as part of the `CredentialResponse` message itself. If this were not sufficient for security, then this solution would not be in line with OPAQUE's goal to be PKI-free after registration, as the only alternative appears to be that the client stores the server's OPRF public key after registration.

2. **The `client_identity` component of the `CleartextCredentials` struct is only *optional***; when it is not used, the `client_public_key` is used in its place. This means that during log-in/credential retrieval, when the client calls `RecoverEnvelope` (draft section 4.2), the MAC check is completely independent of the `client_identity`; the MAC covers a `CleartextCredentials` struct that includes only the client's `client_public_key`, which the client gets from the adversary. This makes the attack very cheap, as the same `auth_tag` can be used for different

[13] https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-opaque-06#section-6.1
[14] https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-voprf-07

`client_identity`s.

If the use of a `client_identity` were *not* optional (i.e. clients were responsible for remembering their password *and* their identity, and had to use their identity in the `CleartextCredentials` struct), then the adversary would not be able to re-use the same `masked_response` for every request it intercepted. Instead, it would have to (i) know the `client_identity` associated with the request and (ii) compute a different MAC tag for each client. Nevertheless, **the attack would still be cheap** (the adversary would still need to call the memory-hard function only once per guessed password), but not *very* cheap (since the adversary would have to compute a different MAC tag for each client). Thus, simply enforcing the inclusion of the client's identity would not be sufficient.

3. **The client does not incorporate their `client_identity` into the hardened OPRF output `randomized_pwd`**; only the server is responsible for doing this (by deriving the OPRF key from the `credential_identifier`, which the server derives from the `client_identity`). So, during log-in/credential retrieval, when the server is unauthenticated, an adversary can craft a response that has a valid MAC tag if and only if they have guessed the client's password.

   If the clients were responsible for inputting their `client_identity` to the `Harden` function used to construct the `randomized_pwd`, then **the attack would be more expensive**, as the adversary would have to (i) know the `client_identity`, (ii) compute a different hardened `randomized_pwd` for each client, and (iii) compute a different MAC tag for each client.

4. **The server is unauthenticated during log-in/credential retrieval.** This is what allows an adversary to respond to a client's log-in/credential retrieval request.

   Not requiring the server to be authenticated during log-in/credential retrieval is one of the major goals/features of OPAQUE: it is a "PKI-free" protocol, i.e. server authentication is required only during registration. Authenticating the server during log-in/credential retrieval **would not be in line with the "spirit" of OPAQUE**. However, if the client could obtain an authentic copy of the OPAQUE server's public key, and that key remains secure, then this **could rule out the attack.**

The customer team requested guidance in relation to implementing Trusted Setup Ceremonies. The NCC Group team provided the contents below. It is presented for information purposes only. The customer team evaluated the guidance, and stated that they implemented the guidance below as appropriate. Some initial observations from the NCC Group team no longer apply; WhatsApp implemented changes to the solution in several areas, following feedback from the consultants during the course of the project. In particular, the server signing keys used in the initial commissioning process were later replaced with native HSM keys, and the public counterparts of these HSM keys are now hardcoded into the client.

The security of the HSMs, cryptographic key material, and user data stored in the HSMs, are crucial to End-to-End Encrypted Backups security: their access must be tightly controlled from provisioning to decommission for security, transparency, and auditability.

As another example, the SEE code running within the HSMs must not be modifiable, as replacing this code could compromise security in a number of ways. For instance, if the derivation of the OPRF key from the OPRF seed were changed to be universal rather than client-specific, then the protocol would be susceptible to pre-computation attacks based on common passwords, as two clients with the same password would obtain the same "`randomized_pwd`" after OPRF evaluation and consequently derive the same "`masking_key`" (OPAQUE draft 04 and later only).

WhatsApp provided NCC Group with HSM setup scripts and access to a page on the internal wiki about HSM fleet setup "HSM prod setup (fleet commission)". These materials did not address procedures around the server signing key (used for issuing certificates the HSMs, so that clients may authenticate legitimate HSMs during registration).

This section suggests best practices for End-to-End Encrypted Backups trusted setup and HSM fleet commissioning ceremonies. It is organized as follows:

- Trusted Setup General Advice
- Trusted Setup Examples
- HSM Configuration and Island Creation
- Server Signing Keys

Trusted Setup General Advice

The End-to-End Encrypted Backups trusted setup ceremonies should address the following items:

- **Ceremony participants.** Formalize a set of roles for participants in the ceremony, e.g., ceremony administrator, trusted participants, internal witnesses, external witnesses, auditors. See https://www.iana.org/help/key-ceremony-roles for more possible roles. Involving trusted participants external to WhatsApp may help strengthen user confidence in the service. In particular, since End-to-End Encrypted Backups is designed to protect user data even from WhatsApp itself, it is recommended that the ceremony participants not consist solely of engineers involved in its design and implementation.
- **Ceremony script and procedure.** Write a detailed script for the ceremony administrator to read during the ceremony. The administrator can annotate each step with their initials and the current time as it is completed. Writing such a script before the ceremony allows performing a "test run" of the ceremony, which may help identify otherwise unforeseen issues. Maintain revision history of all documents.
- **Ceremony recordings.** Record a video of the ceremony from multiple angles. Consider live-streaming it. Ensure that entering passphrases is not captured on video.
- **Handling of HSMs, cards, and card readers.** Perform all unboxing of HSMs and related hardware during the ceremony. Inspect each item for evidence of tampering. These devices should have a chain of custody beginning with the manufacturer and ending at the data centers in which they will reside for daily operations.
- **Hosts and operating environment.** Where possible, all systems involved should be hardened machines that are running trusted operating system images on read-only media and are isolated from public networks. This also applies to the system on which the SEE binary is built.
- **Availability of ceremony materials.** Collect all logs, scripts, annotated documents, audio-visual recordings, attendance sheets, etc. into a package of ceremony materials. Inviting the public to review these artifacts can strengthen

users' confidence in the service.

Trusted Setup Examples

This section summarizes public trusted setup ceremonies. Although not all of them are similar to End-to-End Encrypted Backups (e.g., not all include HSMs), they can be useful as references regarding how to conduct, record, and share the ceremony, as well as for how to improve the public's confidence in the ceremony.

Example: DNSSEC Key Signing Ceremonies

One of the responsibilities of the Internet Assigned Numbers Authority (IANA) is managing the DNS root zone. This includes responsibility for all operations involving the root zone's Zone Signing Key (ZSK) and Key-Signing Key (KSK). The root KSK is the trust anchor for DNSSEC and signs all top-level domains' ZSKs. It is generated and stored in an HSM, and used every three months in a root KSK ceremony. The following quote from the IANA ceremonies webpage (the first bullet point in the list after the quote) describes these ceremonies:

> *Ceremonies are usually conducted four times a year to perform operations using the Root Key Signing Key, and involving Trusted Community Representatives. In a typical ceremony, the KSK is used to sign a set of operational ZSKs that will be used for a three month period to sign the DNS root zone. Other operations that may occur during ceremonies include installing new cryptographic officers, replacing hardware, or generating or replacing a KSK.*

- Materials of past KSK ceremonies: https://www.iana.org/dnssec/ceremonies
- Policies and procedures: https://www.iana.org/dnssec/procedures

Example: Zcash Public Parameter Ceremonies

The zero-knowledge proofs used in Zcash require a set of public parameters. Zcash designed a one-time multi-party trusted setup ceremony to collaboratively generate these public parameters and to ensure that any sensitive values involved in their generation ("toxic waste") are destroyed. The ceremony involved five main participants in different locations and was designed to be secure as long as at least one participant was not compromised. The ceremony was preceded by three "dress rehearsals," which uncovered issues that were addressed with changes to the protocol software before the actual ceremony. The ceremony was video recorded and witnessed by a journalist at one of the locations and the public was invited to review ceremony details and assets after it happened.

- Overview of Zcash parameter generation: https://z.cash/technology/paramgen/
- Zcash ceremony details and assets: https://github.com/zcash/mpc#zcash-ceremony
- External blog post criticizing requirement of at least one non-compromised participant: https://blog.okturtles.org/2016/09/how-to-compromise-zcash-and-take-over-the-world/

Example: WebTrust Certification Authority Audits

As roots of trust for the web PKI, Certification Authorities (CAs) must safely handle signing keys. For example, commercial CAs wishing to be considered trusted roots by Microsoft must undergo yearly audits by qualified, independent auditors.[15] Similarly, audits are required by Mozilla's CA Certificate Program, which governs which root CAs are trusted by the Firefox browser.[16] One major audit framework is WebTrust for CAs, managed by Chartered Professional Accountants of Canada. It covers, among other aspects, key lifecycle management, certificate lifecycle management, and environmental controls. The following quote from the main WebTrust webpage (the first bullet point in the list after the quote) describes the program:

> *The WebTrust for Certification Authorities program was developed to increase consumer confidence in the Internet as a vehicle for conducting ecommerce, and to increase consumer confidence in the application of PKI technology. This program, which was originally developed jointly by the AICPA and CICA, is now managed by Chartered Professional Accountants of Canada. Public accounting*

[15] https://www.docs.microsoft.com/en-us/security/trusted-root/audit-requirements
[16] https://www.mozilla.org/en-US/about/governance/policies/security-group/certs/policy/#3-documentation

> *firms and practitioners who are enrolled by CPA Canada, can provide assurance engagements to report on the disclosure of relevant policies and effectiveness of controls by a certification authority (CA) using the relevant principles and criteria.*

- WebTrust seal program for Certification Authorities: https://www.cpacanada.ca/en/business-and-accounting-resources/audit-and-assurance/overview-of-webtrust-services
- WebTrust Principles and Criteria for Certification Authorities, Version 2.2.1: https://www.cpacanada.ca/-/media/site/operational/ms-member-services/docs/webtrust/wt100awebtrust-for-ca-221-110120-finalaoda.pdf
- Sample WebTrust CA audit report: https://www.cpacanada.ca//GenericHandlers/CPACHandler.ashx?AttachmentID=257007

HSM Configuration and Island Creation

The End-toEnd Encrypted Backups architecture includes a number of islands. An "island" is a group of 5 HSMs that are configured to access the same cryptographic material. In the HSM manufacturer's terminology, this "island" is a set of HSMs that share the same "Security World". HSMs in the same Security World can be administered by the same Administrator Card Set (ACS) and share the same Security World Key, which encrypts all key material produced by the HSMs and stored on their host(s). In End-to-End Encrypted Backups, each HSM in an island has its own host, that handles storage of data (in encrypted format).

The main steps of HSM configuration and island creation are discussed in the following subsections.

1. Configuring the Security World

The commands for the creation of one island are gathered in the script `setup_security_world.rs`, including Security World creation, provisioning of Administrator Card Set (ACS) cards, and provisioning of Operator Card Set (OCS) cards. At the time of review, this script had not been updated to reflect changes in the "HSM prod setup (fleet commission)" wiki page. For instance, the wiki page describes the ACS and OCS requiring quorums of 3 of 5 users, while the script says 1 of 1.

A number of configuration options are passed to the `new-world` utility, which creates the Security World and ACS and sets the quorum numbers. The Security World is configured to require ACS authorization to allocate NVRAM regions, set the real-time clock, and enable foreign token operations.

- **Recommendation:** The `new-world` utility could have the `--no-recovery` option set, which prevents a quorum of ACS cards from authorizing the recovery of lost or damaged OCS and softcards. Currently, since the `recovery=no` option is also passed to `generatekey` for the two OCS-protected keys, setting this property at the world level would not change their recoverability. However, setting it at the world level offers protection in the case of accidental omission of the `recovery=no` option when creating OCS-protected keys.
- **Recommendation:** The `new-world` command could use the `--pp-min=X` option to set a non-zero default passphrase length for ACS and OCS cards. The default minimum is `0`, which allows cards to have no passphrase protection. Currently, all ACS and OCS cards are wiped at the end of the ceremony, so this minimum passphrase length is not very important. Also note that, according to the nShield Solo User Guide, the length specified here is used only to *warn* users when they are using short passphrases (and ultimately does not prevent their use).

Next, the OCS is created with the `create-ocs` utility, which sets its quorum numbers and gives the cardset the name "`codesign`."

- **Recommendation:** The `create-ocs` utility could have the `--no-pp-recovery` option set, which prevents a quorum of ACS cards from authorizing the recovery of lost or damaged OCS cards.
- **Recommendation:** The `create-ocs` utility could have the `timeout=X` option set, which automatically removes the OCS after a specified time period.

Then, the `generatekey` utility is used to create two 3072-bit RSA signing keys of type "`seeinteg`" (key size is defined as

RSA_LENGTH_BITS in opaque.rs.). One is used for signing the SEE binary ("seesign") and the other is used for signing "userdata" ("configsign").

- **Recommendation:** The generatekey utility could have the cardset=codesign option set, which explicitly specifies the OCS that protects the key, rather than the default of using the OCS of the card currently in the slot.

2. Generating Island Keys

The commands for generating an island's RSA key pair, OPAQUE key pair, and AES key are also in the script setup_security_world.rs. However, they are not created with command-line utilities, but through a C program (key_gen.c) using the nCore API provided by the HSM manufacturer.

The OPAQUE key ("opaque") is an X25519 key. Its Access Control List (ACL) specifies that it is module protected (rather than cardset protected, which would require cards to be inserted for every operation) and configured to be accessible only by SEE code signed with the "configsign" key, identified by its key hash retrieved with get_codesign_cert() . The ACLs of the public and private components also specify that they can be exported in plaintext and used as a blob key. These permissions seem necessary to allow the OPAQUE key to be exported as a keypair blob and stored on the host. The keypair blob's creation has an identseeinteg parameter that appears to limit its access to only SEE programs signed by the "configsign" key.

The island AES key ("islandaes") is a 128-bit AES key (length is hardcoded on line 180 of key_gen.c). Its ACL also specifies that it is module protected, accessible only by SEE code signed with the "configsign" key, and can be exported in plaintext and used as a blob key. The AES key is exported as a key blob and stored on the host, with the same identseeinteg parameter set to "configsign."

The island's RSA key ("islandrsa") is 3072 bits (hardcoded on line 294 of key_gen.c). Its ACL also specifies that it is module protected and accessible only by SEE code signed with the "configsign" key. The RSA key is exported as a keypair blob and stored on the host, with the same identseeinteg parameter set to "configsign."

- **Note:** According to the nCore Developer Tutorial, the NFKM_NKF_SEEAppKey flag, which is set for all three keys, has been superseded by the NFKM_NKF_SEEAppKeyHashAndMech flag.
- **Recommendation:** The ACL of the RSA private key specifies that it can be used to sign and decrypt; the RSA public key, to verify and encrypt. The key pair should be used either for signing and verification, or encryption and decryption, but not both, and the public and private ACLs should be changed to reflect this. (See lines 256–265 of key_gen.c.)

3. Adding HSMs to the Island

Once the keys have been generated on the host of the HSM used to create the Security World, the key directory on the host must be copied to all other HSM hosts. The ceremony script should specify exactly how these keys are copied and transferred. Each HSM is then joined to the same Security World as the first, which requires ACS authorization. The NVRAM allocation script (allocate_nvram_regions.rs) is run on each HSM host. It uses the nvram-sw utility, for which no detailed documentation was available, and passes the --key=seeinteg,configsign parameter, which is expected to restrict access to the configured NVRAM regions to only the signed SEE binary.

The "HSM prod setup (fleet commission)" wiki page mentions that an HSM configuration verification script (not available at the time of NCC Group's review) will then be run.

4. Building, Signing, and Loading the SEE binary

The five ceremony participants each build the SEE binary on their own machine and compute its SHA-256 hash. This computation of the expected hash could be done before the ceremony, as soon as there is agreement on which version of the SEE to build.

During the ceremony, on one of the HSM hosts, the SEE binary is built. Next, the script create_sar_files.rs is run, which calls the tct2 Trusted Code Tool utility to create the SEE binary (signed by the seesign key) and config

file (signed by the `configsign` key, and accessible only to SEE machines signed by the `seesign` key). This will require authorization from the OCS.

- **Recommendation:** The "HSM prod setup (fleet commission)" wiki page describes verifying the SHA-256 hash *after* the binary is signed. This hash should be verified *before*.
- **Note:** The "HSM prod setup (fleet commission)" wiki page states that the OCS cardset has size 5 and that fleet commissioning should be "robust to a minority [...] of the 5 commissioners," which implies that the quorum is set to 3 of 5. However, the page also states that *all commissioners* should sign the SEE binary. If enforcing this is desirable, consider raising the quorum when creating the OCS.

Lastly, ACS and OCS cards are wiped by formatting them with the `slotinfo --format` utility.

- **Recommendation:** If there is any doubt about whether the `slotinfo` utility truly zeroizes the key material on ACS and OCS cards, the smartcards' chips should be physically destroyed (e.g., using a hammer, scissors, or shredder).

Server Signing Keys

If the End-to-End Encrypted Backups server private key were compromised, it could be used to create certificates for non-legitimate HSMs. End users, who authenticate HSMs by verifying certificates issued (signed) by the server key on the HSMs' public keys, would not distinguish compromised or fake HSMs from legitimate End-to-End Encrypted Backups HSMs, and may proceed to register with them. To restrict use of the server private key, it is destroyed after the HSM fleets are commissioned during a "trusted setup ceremony." This is described in the Threat Model page of an internal WhatsApp wiki.

No material was provided around the handling of the End-to-End Encrypted Backups server keys that sign the HSMs' keys.

- **Recommendation:** Document the policies and procedures in place around the server signing key.

Miscellaneous Resources

This section contains some additional resources that may facilitate a smooth trusted setup ceremony.

- **Tamper-evident bags.** Note that smartcards require additional protection, as it may be possible to read a smartcard through such plastic tamper-resistant bags.[17]
- **Verbal hash comparison.** Hashes or long strings of binary data are easier to share and compare as words. One utility is `sha2wordlist`,[18] which translates a SHA-256 hash to a list of words from the PGP wordlist.
- **Read-only storage media.** "R" format DVDs (not "RW") are writable only once.

[17] https://www.kumari.net/index.php/projects/random-projects/105-reading-a-smart-card-through-a-tamper-evident-bag
[18] https://www.github.com/kirei/sha2wordlist

Table of Findings

For each finding, NCC Group uses a composite risk score that takes into account risk severity, application exposure, user population, technical difficulty of exploitation, and other factors. For an explanation of NCC Group's risk rating and finding categorization, see Finding Field Definitions on page 24. A response from WhatsApp on each of the remaining "Reported" findings can be found in WhatsApp's Response to NCC Group Findings on the next page.

Title	Status	ID	Risk
Weak 512 bits RSA Key Signing Key	Fixed	10	High
Overly Permissive Remote Access Controls for Vesta HSM Server	Fixed	3	Medium
Weak Password Complexity Requirements	Fixed	4	Medium
Missing Unicode Normalization on Password	Fixed	5	Medium
Key Signing File Accessibility In Lower Integrity Environment May Facilitate Complete HSM Bypass	Fixed	7	Medium
Insufficient Input Validation During OPRF Group Element Deserialization	Fixed	13	Medium
Compromised Backend Can Force the Computation of a Known Backup Export and Other Keys in WhatsApp Clients	Fixed	14	Medium
Unlimited Raft Max Message Size Limit	Fixed	19	Medium
Compromised Backend or Infrastructure Can Force Non-Randomized Client Password Value and Weak OPAQUE Keys	Fixed	20	Medium
Missing Bincode Deserialization Limits on Maximum Size	Fixed	22	Medium
Non Constant-Time GHASH Operation in WhatsApp iOS Encrypted Backup Implementation	Reported	2	Low
Weak Password Hashing	Reported	6	Low
Weak Error Handling via Panic	Fixed	8	Low
Vesta Server Containerized Process Runs With Root Privileges	Reported	9	Low
User Account Enumeration and Disabling	Reported	15	Low
One-Time Device Compromise Results in Irreparable Loss of Confidentiality	Reported	16	Low
Insufficient Protection of Backup Key on Device	Reported	18	Low
OPRF Blinding Scalar Can Be Chosen At Random To Be The Zero Element In GF(p)	Fixed	1	Informational
Non Constant-Time MAC Comparisons of Envelope Auth Tag and Protocol Transcript	Fixed	17	Informational
Brittle Use of `i8` and `i16` Types in Restore History Involving Signature Generation	Reported	21	Informational
Non Constant-Time Registration Challenge Comparison	Fixed	23	Informational
Potential for Unsigned Overflow on Subtraction	Fixed	24	Informational
HSM Firmware and SEE Code and Configuration Updates Impossible Once Commissioned in Production (as Designed and Intended)	Reported	25	Informational

WhatsApp's Response to NCC Group Findings

WhatsApp provided a response for each of NCC Group's findings, whose status is "Reported" in the Table of Findings on the preceding page. WhatsApp's responses are reproduced below as-is:

Name	ID	Risk	WhatsApp Response
Non Constant-Time GHASH Operation in WhatsApp iOS Encrypted Backup Implementation	2	Low	WhatsApp did not deem this to be an issue because the impact and likelihood of any attack is very low and WhatsApp has built other ways of mitigating this denial of service attack should it happen. WhatApp did report this upstream to the library owner in hopes that they will update their library accordingly.
Weak Password Hashing Risk	6	Low	WhatsApp uses 100,000 iterations of PBKDF2 as a password hashing algorithm, as part of the password protections, which is industry standard. We don't use a memory hard algorithm because many of our users have phones with low memory and storage. We thought carefully about the trade off for reliability, user experience and security, and in this case believe that we provided reasonable algorithmic protections with these tradeoffs in mind.
Vesta Server Containerized Process Runs With Root Privileges Risk	9	Low	WhatsApp's system is designed to protect the confidentiality of user keys even against attackers with full internal system access - including full root access to the server hosting the HSM. It is necessary to run this configuration to talk to the HSM and we have other controls in place to manage escalation of privilege attacks.
User Account Enumeration and Disabling Risk	15	Low	API access for WhatsApp's backend is appropriately restricted through standard internal ACL mechanisms, and so the risk of an internal attacker abusing this API to disable or delete user accounts en masse is limited to a very small set of system administrators.
One-Time Device Compromise Results in Irreparable Loss of Confidentiality Risk	16	Low	There is a way for a user to change the key: disable encryption and enable it back. WhatsApp didn't want to provide an easier way to do this because key change will result in a whole backup reupload which could create additional financial costs for WhatsApp users who may pay for network bandwidth to upload information.

Name	ID	Risk	WhatsApp Response
Insufficient Protection of Backup Key on Device	18	Low	On device protection for end-to-end encrypted backups is secure and relies on operating system storage for protections. WhatsApp uses the Keychain on iOS, which is backed by Apple's Secure Enclave hardware key manager on supported devices. WhatsApp has found that secure enclave storage for Android users is not available on all Android devices that WhatsApp supports. As it becomes available and reliable on all Android devices that we support, we will consider implementing this solution.
Brittle Use of i8 and i16 Types in Restore History Involving Signature Generation Risk	21	Informational	WhatsApp appreciates NCC Group bringing this to our attention. The standard internal APIs limit us from using unsigned types ubiquitously. All casts are guaranteed to fall within the corresponding numeric limits, and thus pose no risk to the system.
HSM Firmware and SEE Code and Configuration Updates Impossible Once Commissioned in Production (as Designed and Intended	25	Informational	WhatsApp is intentionally limiting our access to these systems once deployed so that we can uphold the end-to-end encryption guarantees.

Finding Field Definitions

The following sections describe the risk rating and category assigned to issues NCC Group identified.

Risk Scale

NCC Group uses a composite risk score that takes into account the severity of the risk, application's exposure and user population, technical difficulty of exploitation, and other factors. The risk rating is NCC Group's recommended prioritization for addressing findings. Every organization has a different risk sensitivity, so to some extent these recommendations are more relative than absolute guidelines.

Overall Risk

Overall risk reflects NCC Group's estimation of the risk that a finding poses to the target system or systems. It takes into account the impact of the finding, the difficulty of exploitation, and any other relevant factors.

Critical Implies an immediate, easily accessible threat of total compromise.

High Implies an immediate threat of system compromise, or an easily accessible threat of large-scale breach.

Medium A difficult to exploit threat of large-scale breach, or easy compromise of a small portion of the application.

Low Implies a relatively minor threat to the application.

Informational No immediate threat to the application. May provide suggestions for application improvement, functional issues with the application, or conditions that could later lead to an exploitable finding.

Impact

Impact reflects the effects that successful exploitation has upon the target system or systems. It takes into account potential losses of confidentiality, integrity and availability, as well as potential reputational losses.

High Attackers can read or modify all data in a system, execute arbitrary code on the system, or escalate their privileges to superuser level.

Medium Attackers can read or modify some unauthorized data on a system, deny access to that system, or gain significant internal technical information.

Low Attackers can gain small amounts of unauthorized information or slightly degrade system performance. May have a negative public perception of security.

Exploitability

Exploitability reflects the ease with which attackers may exploit a finding. It takes into account the level of access required, availability of exploitation information, requirements relating to social engineering, race conditions, brute forcing, etc, and other impediments to exploitation.

High Attackers can unilaterally exploit the finding without special permissions or significant roadblocks.

Medium Attackers would need to leverage a third party, gain non-public information, exploit a race condition, already have privileged access, or otherwise overcome moderate hurdles in order to exploit the finding.

Low Exploitation requires implausible social engineering, a difficult race condition, guessing difficult-to-guess data, or is otherwise unlikely.

Category

NCC Group categorizes findings based on the security area to which those findings belong. This can help organizations identify gaps in secure development, deployment, patching, etc.

Access Controls	Related to authorization of users, and assessment of rights.
Auditing and Logging	Related to auditing of actions, or logging of problems.
Authentication	Related to the identification of users.
Configuration	Related to security configurations of servers, devices, or software.
Cryptography	Related to mathematical protections for data.
Data Exposure	Related to unintended exposure of sensitive information.
Data Validation	Related to improper reliance on the structure or values of data.
Denial of Service	Related to causing system failure.
Error Reporting	Related to the reporting of error conditions in a secure fashion.
Patching	Related to keeping software up to date.
Session Management	Related to the identification of authenticated users.
Timing	Related to race conditions, locking, or order of operations.