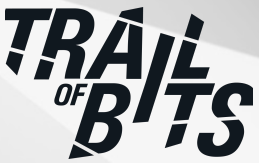

SimpleX Chat

Security Assessment

November 3, 2022

Prepared for:
Evgeny Poberezkin
SimpleX Chat

Prepared by: **Artur Cygan and Jim Miller**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

Table of Contents

Executive Summary

Engagement Overview

SimpleX engaged Trail of Bits to review the security of SimpleX Chat. From October 11 to October 14, 2022, a team of two consultants conducted a security review of the client-provided source code, with one person-week of effort. Details of the project's timeline, test targets, and coverage are provided in subsequent sections of this report.

Project Scope

Our testing efforts were focused on the identification of flaws that could result in a compromise of confidentiality, integrity, or availability of the target system. We conducted this audit with full knowledge of the system. We had access to the source code and documentation. We performed manual review and testing of the target system and its codebase.

Summary of Findings

The audit uncovered two significant flaws that could impact system confidentiality, integrity, or availability. A summary of the findings and details on notable findings are provided below.

EXPOSURE ANALYSIS

Severity	*Count*
Medium	2
Low	2

CATEGORY BREAKDOWN

Category	*Count*
Cryptography	1
Data Exposure	1
Data Validation	2

Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

- **TOB-SMP-1**
  The X3DH implementation does not apply HKDF to the three Diffie-Hellman outputs, which worsens the impact of key compromise and affects the protocol's forward secrecy.

- **TOB-SMP-4**
  The key material is generated and processed in unpinned memory and is not cleared out after its lifetime. This increases the key exposure.

Project Summary

Contact Information

The following managers were associated with this project:

Dan Guido, Account Manager
dan@trailofbits.com

Anne Marie Barry, Project Manager
annemarie.barry@trailofbits.com

The following engineers were associated with this project:

Artur Cygan, Consultant
artur.cygan@trailofbits.com

Jim Miller, Consultant
james.miller@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
October 6, 2022	Pre-project kickoff call
October 18, 2022	Delivery of report draft and report readout meeting
November 3, 2022	Delivery of final report

Project Goals

The engagement was scoped to provide a security assessment of the SimpleX Chat. Specifically, we sought to answer the following non-exhaustive list of questions:

- Does the end-to-end encryption protocol implementation conform with the Signal specification?
- Is the implementation vulnerable to any known cryptographic attacks?
- Is the key material stored and processed in a way that minimizes its exposure?
- Do the codebases adhere to Haskell programming best practices?

Project Targets

The engagement involved a review and testing of the targets listed below.

SimpleXMQ

Repository	https://github.com/simplex-chat/simplexmq
Version	`413aad5139acee28033404aed2e5516fc71c337c`
Type	Haskell
Platform	Native

SimpleX

Repository	https://github.com/simplex-chat/simplex-chat
Version	`07d2c9ff49034520effdf247f022c03b5a890150`
Type	Haskell, Kotlin, Swift
Platform	iOS, Android, Linux, MacOS, Windows

Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches include the following:

- A manual review of the SimpleXMQ reference implementation written in Haskell. We focused on the client/server interaction in `Simplex.Messaging.Server`, `Simplex.Messaging.Client`, `Simplex.Messaging.Agent` modules and reviewed the cryptography implementation in `Simplex.Messaging.Crypto` and `Simplex.Messaging.Crypto.Ratchet` modules.
- A review of the SimpleXMQ end-to-end encryption protocol and its adherence to Signal's Double Ratchet algorithm and the X3DH key agreement protocol.

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. The following list outlines the coverage limitations of the engagement and indicates system elements that may warrant further review:

- The SimpleXMQ notifications code was not reviewed.
- Other than the modules specified above, the Haskell code was reviewed on a best-effort basis.
- The simplex-chat repository was not prioritized for this review as that code performs only business logic and delegates cryptography and networking to the SimpleXMQ library.

Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

Category	Summary	Result
Arithmetic	We identified integer overflow that could impact the security of the system (TOB-SMP-2). We consider the `fromInteger` casting function that is used by SimpleXMQ to be unsafe.	**Moderate**
Auditing	The SimpleXMQ library performs logging of events in relevant places without exposing sensitive information.	**Satisfactory**
Authentication / Access Controls	Although there is no user authentication due to the design of SimpleX platform (no user accounts), the client authorization is performed with anonymous, client-generated signature keys, which are used to sign commands.	**Strong**
Complexity Management	The code is organized in well defined modules and functions. There are occasional complex functions that are harder to audit.	**Satisfactory**
Cryptography and Key Management	We identified one issue related to a missing cryptographic primitive in the X3DH protocol implementation (TOB-SMP-1). We also included recommendations for the secure erasure of cryptographic secrets (TOB-SMP-4). Otherwise,, we found that implementation's cryptographic choices adhere to the recommendations of Signal's specification and other cryptographic specifications.	**Moderate**

Documentation	The SimpleXMQ codebases are well documented with a specification and inline documentation. We found that the specification largely complies with the implementation.	**Satisfactory**
Memory Safety and Error Handling	The memory safety is guaranteed by the Haskell language. There is little use of C FFI in the dependencies. The errors are handled correctly using the standard Haskell conventions and enforced by the type system. Some libraries are throwing exceptions that are not encoded in the type system and are easy to miss, as detailed in TOB-SMP-3.	**Satisfactory**
Testing and Verification	The system is tested with high-level integration tests using the HSpec library. The unit simple tests, however, are scarce and could easily detect some simpler issues such as TOB-SMP-2 and TOB-SMP-3.	**Moderate**

Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

ID	Title	Type	Severity
1	X3DH does not apply HKDF to generate secrets	Cryptography	**Medium**
2	The pad function is incorrect for long messages	Data Validation	**Low**
3	The unPad function throws exception for short messages	Data Validation	**Low**
4	Key material resides in unpinned memory and is not cleared after its lifetime	Data Exposure	**Medium**

Detailed Findings

1. X3DH does not apply HKDF to generate secrets

Severity: **Medium**	Difficulty: **High**
Type: Cryptography	Finding ID: TOB-SMP-1
Target: `simplexmq/src/Simplex/Messaging/Crypto/Ratchet.hs`	

Description

The extended triple Diffie-Hellman (X3DH) key agreement protocol works by computing three separate Diffie-Hellman computations between pairs of keys. In particular, each party has a longer term private and public key pair as well as a more short-term private and public key pair. The three separate Diffie-Hellman computations are performed between the various pairs of long term and short term keys. The key agreement is performed this way to simultaneously authenticate each party and provide forward secrecy, which limits the impact of compromised keys.

When performing the X3DH key agreement, the final shared secret is formed by applying HKDF to the concatenation of all three Diffie-Hellman outputs. The computation is performed this way so that the shared secret depends on the entropy of all three Diffie-Hellman computations. If the X3DH protocol is being used to generate multiple shared secrets (which is the case for SimpleX), then these secrets should be formed by computing the HKDF over all three Diffie-Hellman outputs and then splitting the output of HKDF into separate shared secrets. However, as shown in Figure 1.1, the SimpleX implementation of X3DH uses each of the three Diffie-Hellman outputs as separate secrets for the Double Ratchet protocol, rather than inputting them into HKDF and splitting the output.

```
x3dhSnd :: DhAlgorithm a => PrivateKey a -> PrivateKey a -> E2ERatchetParams a ->
RatchetInitParams
x3dhSnd spk1 spk2 (E2ERatchetParams _ rk1 rk2) =
  x3dh (publicKey spk1, rk1) (dh' rk1 spk2) (dh' rk2 spk1) (dh' rk2 spk2)

x3dhRcv :: DhAlgorithm a => PrivateKey a -> PrivateKey a -> E2ERatchetParams a ->
RatchetInitParams
x3dhRcv rpk1 rpk2 (E2ERatchetParams _ sk1 sk2) =
  x3dh (sk1, publicKey rpk1) (dh' sk2 rpk1) (dh' sk1 rpk2) (dh' sk2 rpk2)

x3dh :: DhAlgorithm a => (PublicKey a, PublicKey a) -> DhSecret a -> DhSecret a ->
DhSecret a -> RatchetInitParams
```

```
x3dh (sk1, rk1) dh1 dh2 dh3 =
  RatchetInitParams {assocData, ratchetKey = RatchetKey sk, sndHK = Key hk,
rcvNextHK = Key nhk}
  where
    assocData = Str $ pubKeyBytes sk1 <> pubKeyBytes rk1
    (hk, rest) = B.splitAt 32 $ dhBytes' dh1 <> dhBytes' dh2 <> dhBytes' dh3
    (nhk, sk) = B.splitAt 32 rest
```

Figure 1.1: simplexmq/src/Simplex/Messaging/Crypto/Ratchet.hs#L98-L112

Performing the X3DH protocol this way will increase the impact of compromised keys and have implications for the theoretical forward secrecy of the protocol. To see why this is the case, consider what happens if a single key pair, (sk2, spk2), is compromised. In the current implementation, if an attacker compromises this key pair, then they can immediately recover the header key, hk, and the ratchet key, sk. However, if this were implemented by first computing the HKDF over all three Diffie-Hellman outputs, then the attacker would not be able to recover these keys without also compromising another key pair.

Note that SimpleX does not perform X3DH with long-term identity keys, as the SimpleX protocol does not rely on long-term keys to identify client devices. Therefore, the impact of compromising a key will be less severe, as it will affect only the secrets of the current session.

Exploit Scenario

An attacker is able to compromise a single X3DH key pair of a client using SimpleX chat. Because of how the X3DH is performed, they are able to then compromise the client's header key and ratchet key and can decrypt some of their messages.

Recommendations

Short term, adjust the X3DH implementation so that HKDF is computed over the concatenation of dh1, dh2, and dh3 before obtaining the ratchet key and header keys.

2. The pad function is incorrect for long messages

Severity: **Low**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-SMP-2
Target: `simplexmq/src/Simplex/Messaging/Crypto.hs`	

Description

The `pad` function from the `Simplex.Messaging.Crypto` module uses the `fromIntegral` function, resulting in an integer overflow bug that leads to incorrect length encoding for messages longer than 65535 bytes (Figure 2.1). At the moment, the function appears to be called only with messages that are less than that; however, due to the general nature of the module, there is a risk of using a `pad` with longer messages as the message length assumption is not documented.

```
pad :: ByteString -> Int -> Either CryptoError ByteString
pad msg paddedLen
  | padLen >= 0 = Right $ encodeWord16 (fromIntegral len) <> msg <> B.replicate
padLen '#'
  | otherwise = Left CryptoLargeMsgError
  where
    len = B.length msg
    padLen = paddedLen - len - 2
```

Figure 2.1: simplexmq/src/Simplex/Messaging/Crypto.hs#L805-L811

Exploit Scenario

The `pad` function is used on messages longer than 65535 bytes, introducing a security vulnerability.

Recommendations

Short term, change the `pad` function to check the message length if it fits into 16 bits and return `CryptoLargeMsgError` if it does not.

Long term, write unit tests for the `pad` function. Avoid using `fromIntegral` to cast to smaller integer types; instead, create a new function that will safely cast to smaller types that returns `Maybe`.

3. The unPad function throws exception for short messages

Severity: **Low**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-SMP-3
Target: `simplexmq/src/Simplex/Messaging/Crypto.hs`	

Description

The unPad function throws an undocumented exception when the input is empty or a single byte. This is due to the `decodeWord16` function, which throws an `IOException` if the input is not exactly two bytes. The unPad function does not appear to be used on such short inputs in the current code.

```
unPad :: ByteString -> Either CryptoError ByteString
unPad padded
  | B.length rest >= len = Right $ B.take len rest
  | otherwise = Left CryptoLargeMsgError
  where
    (lenWrd, rest) = B.splitAt 2 padded
    len = fromIntegral $ decodeWord16 lenWrd
```

Figure 3.1: simplexmq/src/Simplex/Messaging/Crypto.hs#L813-L819

Exploit Scenario

The unPad function takes a user-controlled input and throws an exception that is not handled in a thread that is critical to the functioning of the protocol, resulting in a denial of service.

Recommendations

Short term, validate the length of the input passed to the unPad function and return an error if the input is too short.

Long term, write unit tests for the unPad function to ensure the validation works as intended.

4. Key material resides in unpinned memory and is not cleared after its lifetime

Severity: **Medium**	Difficulty: **High**
Type: Data Exposure	Finding ID: TOB-SMP-4
Target: `simplexmq`	

Description
The key material generated and processed by the SimpleXMQ library resides in unpinned memory, and the data is not cleared out from the memory as soon as it is no longer used. The key material will stay on the Haskell heap until it is garbage collected and overwritten by other data. Combined with unpinned memory pages where the Haskell's heap is allocated, this creates a risk of paging out unencrypted memory pages with the key material to disk. Because the memory management is abstracted away by the language, the manual memory management required to pin and zero-out the memory in garbage-collected language as Haskell is challenging.

This issue does not concern the communication security; only device security is affected.

Exploit Scenario
The unencrypted key material is paged out to the hard drive, where it is exposed and can be stolen by an attacker.

Recommendations
Short term, investigate the use of mlock/mlockall on supported platforms to prevent memory pages that contain key material to be paged out. Explicitly zero out the key material as soon as it is no longer needed.

Long term, document the key material memory management and the threat model around it.

Summary of Recommendations

The SimpleX Chat is a work in progress with multiple planned iterations. Trail of Bits recommends that SimpleX Chat address the findings detailed in this report and take the following additional steps prior to deployment:

- Cover all the modules from the SimpleXMQ with unit or property tests. This will help catch simpler errors and regressions while developing the codebase.
- The cryptonite library is considered state of the art for cryptography within the Haskell ecosystem. However, the library's maintenance should be strengthened and its test coverage expanded, given that it is a language standard. If the project is still developed in Haskell, consider investing in or contributing to the library and performing a security audit on it.
- This audit covered only the most critical parts of the SimpleXMQ Haskell implementation. Perform an audit of the rest of the SimpleXMQ library as well as the mobile applications.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	**Description**
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Code Maturity Categories

The following tables describe the code maturity categories and rating criteria used in this document.

Code Maturity Categories	
Category	**Description**
Arithmetic	The proper use of mathematical operations and semantics
Auditing	The use of event auditing and logging to support monitoring
Authentication / Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system
Complexity Management	The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions
Cryptography and Key Management	The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution
Documentation	The presence of comprehensive and readable codebase documentation
Memory Safety and Error Handling	The presence of memory safety and robust error-handling mechanisms
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage

Rating Criteria	
Rating	**Description**
Strong	No issues were found, and the system exceeds industry standards.
Satisfactory	Minor issues were found, but the system is compliant with best practices.
Moderate	Some issues that may affect system safety were found.
Weak	Many issues that affect system safety were found.
Missing	A required component is missing, significantly affecting system safety.
Not Applicable	The category is not applicable to this review.
Not Considered	The category was not considered in this review.

Further Investigation Required	Further investigation is required to reach a meaningful conclusion.

C. Non-Security-Related Findings

The following recommendations are not associated with specific vulnerabilities. However, they enhance code readability and may prevent the introduction of vulnerabilities in the future.

1. The `sign'` function has a type that indicates it can fail, but in practice it never fails. Its return type can be changed to `IO (Signature a)`.
2. The `mkProtocolClient` function partially fills the ProtocolClient record by using `undefined` for some fields. The same pattern is used in the `mkHTTPS2Client` function. Evaluating the undefined function results in a program crash. Consider changing the records structure so that no partial filling is required.
3. A couple of modules have TODO comments, indicating the code is unfinished. The code should be finished and the TODO comments removed.