Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Review-Report Request Network Codebase 05.2020

Cure53, Dr.-Ing. M. Heiderich, Dr. N. Kobeissi

Index

Introduction

> *“Request enables your users with a common way to request payments while providing full control over their financial data. From simple peer-to-peer payment requests to full business invoices.”*
>
> From https://request.network/en/

This report describes the results of a security audit and cryptography review carried out against the Request network complex, specifically targeting its *TypeScript* codebase and cryptographic architecture. The project was completed by Cure53 in May 2020.

To give some context, it can be clarified that two members of the Cure53 executed this assessment, following a specific timeline for the work being delivered in Calendar Weeks 20 and 21 of 2020. The tasks have been completed over the course of six person-days budget allocated to Cure53 by the Request team.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

In order to make sure that all areas are covered and various goals are fulfilled, Cure53 split the work into two Work Packages. In WP1, the testing team performed a cryptographic review and audited the *TypeScript* codebase of the Request network. Then, in WP2, an explicit focus was placed on the questions that the Request team formulated and needed to get answers to from a security perspective. In this realm, Cure53 zoomed in on the cryptography-specific matters.

The project was well-prepared, started on time and progressed efficiently. The Cure53 team could leverage the advantages of a white-box methodology. Under this premise, Cure53 was granted access to all relevant material from the Request network team. Among other data, Cure53 received a detailed scope document (see below), several crypto-specific questions, as well as the codebase, which is generally available as open source software.

The communication during the assessment and review was done via Slack, the Request network team created a test-related channel on their Slack workspace and invited relevant members of the Cure53 team to join it. All exchanges were productive and the project was finalized with a debrief call. During this meeting, the results were presented and remaining questions have been discussed in detail.

Seven findings have been spotted during the project. Three items were classified to be security vulnerabilities, two scored *Medium* in terms of risks and one had only *Low* impact. Four *Info*-only findings were documented as miscellaneous, with possible recommendations to consider. The Request network team was informed about all issues in the aforementioned briefing. In addition to individual items, Cure53 answered four questions posed by the Request network team prior to the audits and reviews happening. All of them can be seen as general recommendations with corresponding headlines.

In the following sections, the report will first shed light on the scope and key test parameters, including also a list of questions asked by the Request network team. Next, all findings will be discussed in a chronological order alongside technical descriptions, as well as PoC and mitigation advice when applicable. Finally, the report will close with broader conclusions about this May 2020 project. Cure53 elaborates on the general impressions and reiterates the verdict based on the testing team’s observations and collected evidence. Tailored hardening recommendations for the Request complex are also incorporated into the final section.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Scope

- **Cryptographic Review & Security Audit against Request Network Codebase**
  - **WP1**: Crypto Review and Audit against the Request network's *TypeScript* codebase
    - A scope document was shared with Cure53
      - https://docs.google.com/document/d/1R9b_dZRif9Z4CG6brDRDmvAgDd1Gvct8Ds6lgcYNhJ4/edit?usp=sharing
  - **WP2**: Q&A-based section, written in response to crypto-specific questions from the Request team
    - See *Miscellaneous Issues* section in this document

Identified Vulnerabilities

The following sections list both vulnerabilities and implementation issues spotted during the testing period. Note that findings are listed in chronological order rather than by their degree of severity and impact. The aforementioned severity rank is simply given in brackets following the title heading for each vulnerability. Each vulnerability is additionally given a unique identifier (e.g. *RN-01-001*) for the purpose of facilitating any future follow-up correspondence.

RN-01-001 WP1: Insecure PRNG for cryptographic material *(Low)*

It was found that the *crypto.ts* provider uses *Math.random()* in order to provide the *generates4randomBytes()* and *generate8randomBytes()* functions. *Math.random()* does not expose a cryptographically secure pseudorandom number generator in most runtime environments, and as such bytes obtained through these functions should not be considered suited for a variety of cryptographic operations, such as private key generation.

However, it was also observed that *generate4randomBytes()* and *generate-8randomBytes()* were only being used to generate salts, which is an operation that does not require a cryptographically secure pseudorandom number generator. As such, the impact of this issue is set to *Low.* While it does not pose any harm at present, the presence of these functions within Request network's cryptographic provider could lead to them being used for generating sensitive cryptographic material in the future.

Affected File:
packages/utils/src/crypto.ts

Affected Code:

```
function generate8randomBytes(): string {
  const base16 = 16;
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
  const generate4randomBytes = (): string => {
    // A 4 byte random integer
    const randomInteger = Math.floor(Math.random() * Math.pow(2, 4 * 8));

    // Convert to hexadecimal and padded with 0
    return randomInteger.toString(base16).padStart(8, '0');
  };

  // Do it in 2 passes because an integer doesn't have enough bits
  const high = generate4randomBytes();
  const low = generate4randomBytes();
  return high + low;
}
```

It is recommended to replace the usage of *Math.random()* with either WebCrypto's *Crypto.GetRandomValues()*[1] or with Node's *crypto.randomBytes()*[2] function.

RN-01-006 WP1: Encryption scheme reveals plain-text oracle *(Medium)*

It was found that Request network's request encryption protocol communicated a simple cryptographic hash of the plain-text, along with each message. Given that the plain-text follows a highly predictable, standardized and normalized JSON structure, this could allow the attacker to use the hash as a high-speed oracle for guessing the plain-text contents of the Request network messages.

For example, consider the following sample Request network plain-text, which has been provided by Request network.

```
{"meta":{"format":"rnf_invoice","version":"0.0.2"},"creationDate":"2020-02-
11T05:00:00.000Z","invoiceNumber":"1","purchaseOrderId":"1","note":"\
n","sellerInfo":{"businessName":"Decipher Data Consultancy","address":
{}},"buyerInfo":{"businessName":"MakerDAO","address":{}},"invoiceItems":
[{"name":"Grant (Presentation, Consulting, Modeling) - June
2019","quantity":1,"unitPrice":"55000000000000000000","currency":"DAI","delive
ryDate":"2020-02-11T05:00:00.000Z"},{"name":"Grant (Presentation, Consulting,
Modeling) - July
2019","quantity":1,"unitPrice":"55000000000000000000","currency":"DAI","delive
ryDate":"2020-02-11T05:00:00.000Z"}],"paymentTerms":{},"miscellaneous":
{"builderId":"app.request.network"}}
```

A reasonably well-informed attacker could obtain, on top of the easily guessable JSON structure and field-names, information regarding the transaction dates and parties. Thereby, they would be shortening the number of unknown bits to a restricted subset for which the hash can then be used as an oracle at up to 23,000 SHA-2

[1] https://developer.mozilla.org/en-US/docs/Web/API/Crypto/getRandomValues
[2] https://nodejs.org/api/crypto.html#crypto_crypto_randombytes_size_callback

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

MegaHash/second-guessing rates with standard consumer equipment. Given a sufficiently restricted number of unknown bits, which are almost certain to be within the alpha-numeric ASCII range, this could allow the attacker to reliably confirm a suspected plain-text or even obtain an unknown plain-text. This is connected to well-informed guesses on predictable structural elements of the payload.

As documented in RN-01-007, it is recommended that authenticated symmetric encryption primitives (such as *AES-GCM*) get included in the design. This would eliminate the requirement for this hash, allowing for it to be removed altogether from the protocol.

RN-01-007 WP1: No integrity from symmetric encryption *(Medium)*

The request encryption protocol for the Request network employs two primitives:

- **AES-CBC** is used for symmetric payload encryption, with a randomly generated key that is unique per payload.
- **ECIES** is used with *secp256k1* as the underlying curve in order to provide authenticated public-key encryption of the unique AES-CBC key that was used to encrypt the payload.

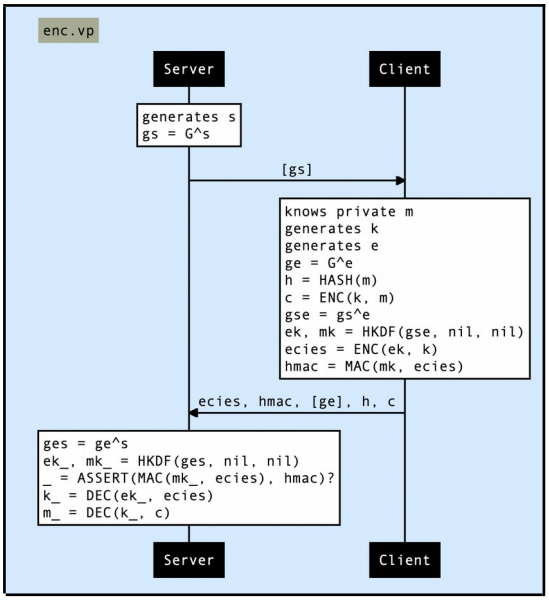

Fig.: RequestNetwork protocol, modeled in Verifpal.[3]

[3] https://verifpal.com

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Since AES-CBC does not provide integrity guarantees or authenticated encryption on the cipher-text, it was found that, in fact, Request network only provided cryptographic integrity on the encryption key. The cipher-text, however, can be tampered with by the attacker undetectably.

Since Request network uses AES-CBC, the precision with which the attacker can produce malicious tampering is greatly reduced. Were Request network to be using a stream cipher such as AES-CTR, the impact of this attack would have been catastrophic. In this alternative scenario, it would have allowed the attacker to modify arbitrary cipher-texts with bit-level precision on the changes taking effect on the underlying plain-text. This weakness was modeled for and discovered using the Verifpal protocol analysis software.

It is recommended for AES-CBC to be replaced with an authenticated encryption primitive such as AES-GCM. In addition to resolving this issue, an authenticated symmetric primitive would also render the hashing which causes RN-01-006 unnecessarily, thereby helping to resolve that issue as well.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Miscellaneous Issues

This section covers low-priority issues as well as requests for information and discussion points that arose from the audit.

RN-01-002 WP2: Using keys for signing and encryption *(Info)*

The Request network team asked for recommendations on whether using the same key pair for signing and encryption can cause problems.

Strictly speaking, using the same private key for both signing and encryption key pairs does not pose issues, especially if "encryption" is meant to signify Diffie-Hellman key agreement. Popular encryption protocols, such as the Signal protocol, convert a *X25519* Diffie-Hellman key pair to an *Ed25519* signing key pair.[4] The age encryption utility does the reverse, converting an *Ed25519* signing key pair into an *X25519* Diffie-Hellman key pair.[5] A similar approach may be adopted by Request network if necessary.

RN-01-003 WP2: Encrypting short messages *(Info)*

Request network asked for recommendations on whether it is a problem to encrypt short messages. So long as messages are padded up to the correct block's cipher length, encrypting many short messages should not pose any issues, except if encryption keys last for an extremely long time (upwards of 2^{32} encryptions). Given that the Request network protocol generates a fresh encryption key for each message, this should not be an issue.

RN-01-004 WP2: Encrypting same message with different keys *(Info)*

The Request network team asked for recommendations on whether it is a problem to encrypt copies of the same message separately with different keys. There is no issue with this approach, given the Request network's security goals and protocol design.

RN-01-005 WP2: General post-quantum cryptography Issues *(Info)*

The Request network team asked for recommendations regarding the necessity and workability of switching over to post-quantum cryptography in the future. Efforts have recently been made in order to organize and improve the available knowledge on the four main branches of cryptographic primitives that are thought to be resistant to quantum algorithms.[6] Among these, one of the more workable candidates for post-quantum Diffie-Hellman appears to be CSIDH.[7] However, even in the case of relatively

[4] https://signal.org/docs/specifications/xeddsa/
[5] https://docs.google.com/document/d/11yHom20CrsuX8KQJXBBw04s80Unjv8zCg_A7sPAX_9Y/preview
[6] https://pqcrypto.org/
[7] https://csidh.isogeny.org/index.html

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

efficient and implementation-friendly post-quantum primitives, widespread compatibility and performance remain completely incomparable to “standard” elliptic-curve Diffie-Hellman primitives, such as those used by Request network (e.g. *Secp256k1* and similar).

Request network uses ECIES as a public-key encryption construction. A recent alternative to ECIES, namely HPKE[8], provides very much the same functionality as ECIES while allowing for an easier definition of underlying cryptographic primitives. As such, a migration away from ECIES and into HPKE could be considered. With a revised approach, the HPKE implementation could be reconfigured to use post-quantum primitives once those are ready for prime-time and, moreover, doing so would not render previous cipher-texts “undecryptable”.

It should also be noted that there is currently no evidence to suggest that quantum attacks on cryptographic primitives are likely to occur in the near future. Similarly, it cannot be said that they would be catastrophic in terms of impact on the Request network. In fact, their implications might be negligible when compared to the consequences they would have for the world’s entire technical infrastructure.

Conclusions

Cure53 generally gained a rather positive impression of the audited code and architecture provided by the Request network team. After spending six days on the scope in May 2020, two members of the Cure53 have identified both strengths and weaknesses in the Request network complex. On the one hand, the team can conclude that efforts have clearly been made towards good security and reliability of the code. On the other hand, the protocol was found to suffer from serious issues that must be fixed as soon as possible.

It should be noted that the Request network’s protocol design was fully reviewed, starting from the provided specifications and API documentations, then moving onto the code itself. The code should be considered well-readable and no major findings were detected. One minor recommendation has been made in RN-01-001 and pertains to the exposure of an insecure pseudorandom number generator via the cryptographic API of the Request network. However, since that functionality is not used in a security-critical context, it does not weaken the effective Request network’s security client.

More urgency should be given as regards two medium-severity findings detected in the protocol design, potentially exposing protocol messages in the Request network to significant cryptographic weaknesses. These weaknesses were confirmed with the

[8] https://tools.ietf.org/html/draft-barnes-cfrg-hpke-00

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Request network team and are documented in RN-01-006 and RN-01-007, respectively. The tickets include advice on the proposed fixes, which are easy to implement.

Further, the involved Request network team offered a set of general questions and requests for recommendations as regards topics in applied cryptography. Some of these questions were fielded during a conference call while others were answered in this report, specifically under RN-01-002, RN-01-003, RN-01-004 and RN-01-005.

Cure53 would like to thank Yoann Marion from the Dragon Research B.V. team for his excellent project coordination, support and assistance, both before and during this assignment.