Fine penetration tests for fine websites

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

Audit-Report Privy.io Shamir Secret Sharing 02.2023

Cure53, Dr.-Ing. M. Heiderich, Dr. M. Conde

Index

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

Introduction

“Easily onboard your users to web3 with a simple, robust library. Add beautiful authentication flows in minutes and serve every user, whether they have a wallet or not, across mobile and desktop.”

From https://www.privy.io/

This report describes the results of a cryptography review and source code audit targeting the Privy.io Shamir secret-sharing TypeScript implementation. The work was requested by Horkos, Inc., in February 2023.

Cure53 carried out the assessment shortly after it was requested, namely in CW06 2023. Regarding resources allocated to this project, registered as *PVY-01,* it should be clarified that a total of two days had been invested to reach the coverage expected for this project. Further, a team of two senior testers were assigned to this project’s preparation, execution and finalization. The work was structured within a single work package (WP):

- **WP1**: Crypto review and code audit against the Privy.io SSS implementation

White-box methodology was employed to complete the goals of this assignment. Cure53 was provided with the sources via GitHub, as well as received all other means of access required to complete the review. All preparations were done in late January and early February, namely in CW05, to foster a smooth transition into the testing phase. Over the course of the engagement, the communications were done using a private, dedicated and shared Slack channel set up between Horkos, Inc., and Cure53. All involved personnel from both parties could join the discussions on Slack.

The discussions throughout the test were very good and productive and not many questions had to be asked. Ongoing interactions positively contributed to the overall outcomes of this project. The scope was well-prepared and clear, greatly contributing to the fact that no noteworthy roadblocks were encountered during the test. Cure53 offered frequent status updates about the test and the emerging findings. Live-reporting was offered and executed in the aforementioned Slack channel.

The Cure53 team managed to get very good coverage over the WP1 scope items. Among three security-relevant discoveries, one was classified to be a security vulnerability and two should be considered general weaknesses with lower exploitation potential. It needs to be noted that the time assigned for this review was relatively short, which already led to an expectation towards a small number of findings. It is nevertheless a good sign that this prediction has become reality.

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

In the following sections, the report will first shed light on the scope and key test parameters, as well as the structure and content of the WPs. A dedicated chapter pertaining to the results of the cryptographic review is then included in this report.

Next, all three findings will be discussed in grouped vulnerability and miscellaneous categories, then following a chronological order in each group. Alongside technical descriptions, PoC and mitigation advice are supplied when applicable. Finally, the report will close with broader conclusions pertinent to this February 2023 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 Privy.io Shamir secret-sharing TypeScript implementation 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

- **Crypto review & code audit against the Privy.io Shamir secret-sharing implementation**
  - **WP1**: Cryptography review & Code audit of the Privy.io Shamir secret-sharing TypeScript implementation
    - **Sources:**
      - https://github.com/privy-io/shamir-secret-sharing/blob/main/src/index.ts
    - **Commit:**
      - 3383ad927211615f3d70b5ceb5489f8a149be191

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

Cryptography review

This section discusses the premise and outcomes of the cryptography adopted by Privy.io. It begins with some general clarifications pertaining to the project, so as to facilitate a more detailed understanding of subsequent comments and claims made by Cure53 in the frames of this *PVY-01* crypto-review.

A *(t,n)*-threshold secret sharing scheme - as devised by Shamir - makes it possible to split a secret among *n* parties in such a way that two conditions are met. First, any subset of *t* or more parties can jointly combine their shares and reconstruct the secret. Second, no group of less than *t* parties - i.e., the threshold, can learn anything about the secret.

The mathematical foundation of Shamir's secret-sharing scheme is rather simple, relying on the fact that any polynomial *f* of degree *t-1* over a finite field is uniquely determined by *t* evaluations and can be reconstructed from those evaluations using Lagrange interpolation. In particular, if a dealer wishes to split a secret, they can choose a random polynomial *f* of the form:

$$f(x) = secret + a_1 x + ... + a_{t-1} x^{t-1}$$

Based on that, they can distribute *n* shares of the form $(x_i, f(x_i))$ among the parties, with $i \in \{0, ..., t-1\}$.

Choice of the finite field

The mathematical foundation of the Shamir's secret-sharing scheme holds, irrespective of the finite field chosen. In other words, this choice does not affect the security of the scheme as such, but has bearing on efficiency.

By design, the secret needs to be represented as an element of the finite field, which might mean that several executions of the Shamir's secret-sharing scheme must be run, depending on the length of the secret. This clearly adds complexity.

The finite field chosen by the developers of Privy is GF(2^8), which effectively means that if the secret consists of more than one byte, say *len* bytes, Shamir's secret sharing scheme will have to be run *len* number of times independently to generate *len* shares for each of the parties. It should be noted that GF(2^8) constitutes a common choice in cryptography due to the speed of its field operations, so it is a good trade-off between complexity and efficiency.

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

Functional correctness of field operations

Aside from the addition operation (*add*), which is trivially implemented as a *xor,* multiplication and division operations (*mult* and *div*) have been facilitated with look-up tables, which is a good choice in terms of speed.

It has been verified that the look-up tables fixed in the code (*LOG_TABLE* and *EXP_TABLE*) are correct in regard to the use of the irreducible polynomial:

$$x^8+x^4+x^3+x+1.$$

This polynomial is not explicitly mentioned in the code documentation, and it is recommended to include it, so as to ensure a better readability.

Choice of other relevant parameters

This subsection focuses on the correctness of the choices behind certain parameters. These parameters were reviewed, as they might negatively impact the security of Shamir's secret-sharing scheme when executed poorly.

Construction of the polynomials involved in function *split*

As stated before, if the secret consists of *len* bytes, Shamir's secret-sharing scheme will have to be run *len* number of times independently. There is a requirement for this to be done in such a way that it does not break the security of the scheme. This requirement states that the polynomials chosen to generate the share need to be chosen independently at random in each execution of the scheme This has been implemented correctly by the developers of Privy.

Construction of x_i in function *newCoordinates*

The x_i can be chosen in different ways without a security impact as long as:

- No x_i is equal to zero, as this would trivially reveal the secret to the party receiving the *share.*
- All x_i are pairwise-different.

Both conditions are met and guaranteed in the code.

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

Resistance to side-channel attacks

As the *shamir-secret-sharing* library is written in JavaScript and JavaScript is commonly just-in-time compiled, it is not possible to guarantee constant-time operations. As a result of this, Cure53 did not focus on reporting any findings related to non-constant time operations.

Furthermore, it should be noted that field operations are implemented with look-up tables, which constitute a very performant option in terms of speed, but introduce a poor resistance against cache side-channel attacks.

Future work & considerations

The Shamir's secret-sharing scheme - as implemented by Privy - can be considered as secure under the following assumptions:

- The dealer is honest: A dishonest dealer could corrupt the *shares* and the parties would have no way of knowing it prior to the secret reconstruction.
- The parties are honest: A dishonest party could corrupt their *share* and convince *t-1* parties to jointly reconstruct the secret, which would result in an incorrect reconstruction of the secret output by the function *combine*.
  - In this regard, it shall be remarked that the *combine* function can output an incorrect secret and there is no way to check whether that is the case or not solely on the secret output by the function. In essence, this is something that users of the library need to take care of. Thus, it is recommended to clearly state that these limitations are present, doing so in the code and documentation to avoid misuse. In case the developers of Privy wish to include the option for parties to detect whether or not their *shares* reconstruct the correct secret solely based on the output of the function *combine*, it would be possible to do so by using some identifying information of the secret (e.g., a hash).
- The secret is uniformly distributed at random: This is the typical scenario, as encryption keys are usually split. If a non-random secret shall be split, it is recommended to encrypt it and split the encryption key instead.

Once again, Cure53 recommends acknowledging these considerations in the documentation to avoid potential problems.

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

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. *PVY-01-001*) for the purpose of facilitating any future follow-up correspondence.

PVY-01-002 WP1: Degree of polynomial might be less than t-1 *(High)*

***Note**: This issue has been fixed and the fix was verified by Cure53, the issue as described no longer exists in the implementation.*

While reviewing the *shamir-secret-sharing* repository, it was observed that the polynomial generated at random in the function *split* might not satisfy the necessary conditions to instantiate a *(t,n)*-threshold Shamir's secret sharing scheme, as it is expected from the function. As can be recalled from the cryptography review, such a polynomial must be of the following form.

$$f(x) = secret + a_1 x + ... + a_{t-1} x^{t-1},$$

In particular, the degree of the polynomial needs to be exactly *t-1*, as this is the condition that guarantees that no group of less than *t* parties can jointly recover the secret. To be more clear, if the degree of the polynomial generated is actually *t-2*, this means that *t-1* parties, i.e., a number of parties under the threshold, could jointly reconstruct the secret. As a consequence, the function *split* would not be behaving as it is supposed to, producing an outcome that happens to be a (*t-1,n)*-threshold scheme instead of a *(t,n)*-threshold scheme.

The probability of generating the coefficient a_{t-1} to be non-zero is $1-\frac{1}{2^8}$, which approximately equals 0.996. Hence, it is expected that for approximately every 1000 executions of the *split* function, four of the executions will yield an incorrect *(t,n)*-threshold scheme, resulting in a *(t-1,n)*-threshold scheme instead.

To illustrate it further, if the *split* function is called in order to split an *AES-128* key into *shares*, the secret scheme runs 16 times (as 16 bytes are being split). In this case, the probability of generating the coefficients of the 16 polynomials correctly is $\left(1-\frac{1}{2^8}\right)^{16}$,

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

which comes up to ca. 0.94. Therefore, if the *split* function is executed 100 times in order to generate a *(t,n)*-threshold scheme and share an *AES-128* key, approximately 4 times the resulting scheme will not be a correct *(t,n)*-threshold scheme. In practice, *t-1* parties - which is one fewer than expected as a demand - can jointly recover at least one byte of the secret, hence learning something from said secret. This does not meet the definition of a *(t,n)*-threshold scheme and represents an undesired behavior of the *split* function.

Affected file:
shamir-secret-sharing/src/index.ts

Affected code:

```
function newCoefficients(intercept: number, degree: number):
Readonly<Uint8Array> {
  const coefficients = new Uint8Array(degree + 1);
  coefficients[0] = intercept;
  coefficients.set(getRandomBytes(degree), 1);
  return coefficients;
}
```

The function *newCoefficients* that implements the generation of the coefficients of the polynomial f fails to check that the coefficient a_{t-1} is not a zero, which might result in incorrect instantiations of a *(t,n)*-threshold scheme.

To avoid this issue, it is necessary to add a check that prevents generating a polynomial that is of a degree *strictly less* than what is actually input as the degree.

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

Miscellaneous Issues

This section covers those noteworthy findings that did not lead to an exploit but might aid an attacker in achieving their malicious goals in the future. Most of these results are vulnerable code snippets that did not provide an easy way to be called. Conclusively, while a vulnerability is present, an exploit might not always be possible.

PVY-01-001 WP1: Computationally suboptimal recovery of secrets *(Info)*

***Note**: This issue has been fixed and the fix was verified by Cure53, the issue as described no longer exists in the implementation.*

While reviewing the *shamir-secret-sharing* repository, it was observed that the function *interpolatePolynomial* provides functionality in order to evaluate Lagrange's interpolation polynomial at a generic point *x*. However, this function is later called exclusively on input x = 0, since it is solely this evaluation that is relevant for reconstructing the secret in Shamir's secret-sharing scheme.

The general expression of the Lagrange interpolation polynomial evaluated at a generic point *x* is as follows, where $(x_i, f(x_i))$ are the *t* evaluations given:

$$\sum_{i=0}^{t-1} y_i \prod_{m=0, m\neq i}^{t-1} \frac{x - x_m}{x_m - x_i} .$$

Considering that this evaluation needs to be performed only for *x=0*, the above expression can be slightly simplified to the following, when taking into account that $x_m = -x_m$ in GF(2⁸):

$$\sum_{i=0}^{t-1} y_i \prod_{m=0, m\neq i}^{t-1} \frac{x_m}{x_m - x_i} .$$

Directly coding the latter expression affords *t-1* calls to the function *add* (implementing a *xor* operation), which is a minor efficiency improvement. This is why the flaw has only been placed in the *Miscellaneous* category.

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

Affected file:
shamir-secret-sharing/src/index.ts

Affected code:

```
function interpolatePolynomial(xSamples: Uint8Array, ySamples: Uint8Array, x:
number): number {
  if (xSamples.length !== ySamples.length) {
    throw new Error('sample length mistmatch');
  }

  const limit = xSamples.length;

  let basis = 0;
  let result = 0;

  for (let i = 0; i < limit; i++) {
    basis = 1;

    for (let j = 0; j < limit; ++j) {
      if (i === j) {
        continue;
      }
      const num = add(x, xSamples[j]!);
      const denom = add(xSamples[i]!, xSamples[j]!);
      const term = div(num, denom);
      basis = mult(basis, term);
    }

    result = add(result, mult(ySamples[i]!, basis));
  }

  return result;
}
```

It is recommended to refactor the function *interpolatePolynomial*. The interpolation polynomial should be directly evaluated at *x=0,* which also avoids passing a value (*x*) that is fixed to zero as an input in calls to the function.

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

PVY-01-003 WP1: Non-resistance to cache side-channel attacks *(Info)*

***Note**: This issue has been fixed and the fix was verified by Cure53, the issue as described no longer exists in the implementation.*

As already indicated in the crypto-review section, field operations are implemented with look-up tables that are input-dependent (particularly dependent on *shares*). This very premise makes the implementation vulnerable to cache side-channel attacks.

Affected file:
shamir-secret-sharing/src/index.ts

Affected code:

```
const LOG_TABLE: Readonly<Uint8Array> = new Uint8Array([
  0x00, 0xff, 0xc8, 0x08, 0x91, 0x10, 0xd0, 0x36, 0x5a, 0x3e, 0xd8, 0x43, 0x99,
0x77, 0xfe, 0x18,
[...]
  0x3b, 0x52, 0x6f, 0xf6, 0x2e, 0x89, 0xf7, 0xc0, 0x68, 0x1b, 0x64, 0x04, 0x06,
0xbf, 0x83, 0x38,
]);

// Provides the exponentiation value at each index X.
const EXP_TABLE: Readonly<Uint8Array> = new Uint8Array([
  0x01, 0xe5, 0x4c, 0xb5, 0xfb, 0x9f, 0xfc, 0x12, 0x03, 0x34, 0xd4, 0xc4, 0x16,
0xba, 0x1f, 0x36,
[...]
  0x66, 0xb2, 0x76, 0x60, 0xda, 0xc5, 0xf3, 0xf6, 0xaa, 0xcd, 0x9a, 0xa0, 0x75,
0x54, 0x0e, 0x01,
]);
```

This class of side channel attacks can be prevented by implementing field operations without look-up tables. For example, one can review the *dsprenkels/sss* repository[1], which is designed to be resistant to cache side-channel attacks and side-channel attacks more broadly. Using a dedicated solution is recommended if preventing this class of attacks is desired.

[1] https://github.com/dsprenkels/sss

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

Conclusions

Drawing on the collected evidence, Cure53 argues that the Privy.io Shamir secret-sharing TypeScript implementation leaves a good impression. Two members of the Cure53 team completed this project in February 2023 and observed that the examined code has been well-written. Many common best practices were found to guard the scope, with only three findings ultimately reported in the frames of this *PVY-01* assessment. Yet, there is still some room for improvement and the herein discovered issues need to be addressed for the Privy.io to achieve even stronger security posture.

To reiterate the context of this assessment, the SSS-Typescript library is an implementation of Shamir's threshold secret-sharing scheme. As for its use-cases, it is a good solution for settings in which both the dealer and the participants are honest (i.e., they are assumed to not act maliciously by corrupting *shares,* for example). The disclaimers about the raised security considerations stemming from this and discussed in the dedicated section should be added to the code documentation. Privy.io needs to promote a consistently safe use of the library by consumers who rely on it.

As noted, Cure53 believes that safe choices were made for the parameters that might negatively impact the security of the scheme. To give one example, correct paths were selected for the generation of the polynomials independently at random for each run of the scheme. This means that the security of Shamir's secret-sharing scheme is well-understood by the maintainers. In terms of flaws, the library failed to validate a subtle condition in order to make sure that the coefficient of the term of the highest degree in the generated polynomials is non-zero. With a certain probability, this introduces a vulnerability in which fewer parties than intended can jointly reconstruct the secret. However, this should be a very easy fix.

Relatively major changes of the library would be required in order to include functionality to detect a dishonest dealer. Changes of such nature would involve implementing a verifiable version of Shamir's secret-sharing scheme, which would need to use zero-knowledge proofs. However, a nice addition to the library could be the detection of the potentially dishonest parties (with corrupt *shares*), caught at trying to reconstruct a secret. This could be realized by using a collision-resistant hash function for the purpose of generating a *validation check* for the reconstructed secret, as explained in the section dedicated to the review of the cryptography. Remaining potential room for improving the library also concerns including support to verify the validity of the secret recovered by the *combine* function. At the moment it does not meet this standard and the validity of the recovered secret has to be managed by a higher-level application.

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

It is hoped that the findings shared by Cure53 as a result of this February 2023 project can assist the Privy.io team in focused efforts towards the already robust implementation. Since the code is well-written, validity checks are usually enforced, edge conditions are managed well, and common implementation errors are avoided, the testing team is convinced that the Shamir secret-sharing TypeScript implementation by Privy.io is on the right path in terms of security.

Cure53 would like to thank Asta Li, Ben Reinhart and Henri Stern from the Horkos, Inc. team for their excellent project coordination, support and assistance, both before and during this assignment.