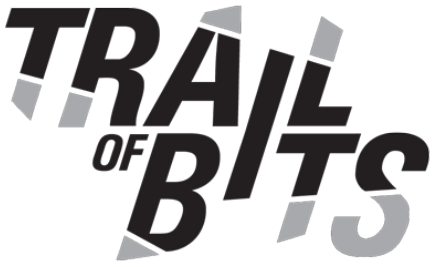

RenVM

Security Assessment

August 17, 2020

Prepared For:
Michael Burgess | *Ren*
michael@renproject.io

Loong Wang | *Ren*
loong@renproject.io

Ross Pure | *Ren*
ross@renproject.io

Prepared By:
Jim Miller | *Trail of Bits*
james.miller@trailofbits.com

Will Song | *Trail of Bits*
will.song@trailofbits.com

Executive Summary

From August 3 through August 17, 2020, Ren engaged with Trail of Bits to review the security of the RenVM secure multi-party computation protocol. Trail of Bits conducted this assessment over the course of four person-weeks with two engineers reviewing the `mpc`, `secp256k1`, and `shamir` repositories.

Week one: First, we familiarized ourselves with the RenVM specification and the codebases. After that, we performed manual review on the implementation, concentrating mainly on the logic inside of `mpc` and `shamir`. Specifically, this review verified that the logic conforms with the specification and that invalid inputs are handled properly.

Week two: We continued our manual review of all the codebases, including a deeper review of the logic inside of `mpc` and `shamir`, and a closer look at the code responsible for marshaling and unmarshaling. We also reviewed the `secp256k1` repository, verifying that the underlying library is wrapped safely.

Our review resulted in six total findings. We report one high-severity issue (TOB-RVM-001) related to adding a check for indices in secret shares. We classified the remaining five issues as having informational severity. Three of these findings (TOB-RVM-002, TOB-RVM-003, TOB-RVM-006) relate to adding checks to ensure that proper behavior is achieved and security guarantees are not violated. The other two informational findings (TOB-RVM-004, TOB-RVM-005) relate to assumptions on function inputs that are not clearly stated or enforced.

Overall, we found that the implementation generally complies with the protocol specification; the issues reported do not highlight any discrepancies between these two. Further, the reported issues largely represent recommendations for strengthening the system overall, as the issues we raised do not seem immediately exploitable. We encourage Ren to add the necessary checks and clearly state the assumptions each function has on its inputs as recommended.

Project Dashboard

Application Summary

Name	RenVM

Version	`mpc` commit: `6c55fe37`
	`secp256k1` commit: `43cb0b3f`
	`shamir` commit: `72890748`

Type	Cryptographic library
Platforms	Go

Engagement Summary

Dates	August 3–August 17, 2020
Method	Whitebox
Consultants Engaged	2
Level of Effort	4 person-weeks

Vulnerability Summary

Total High-Severity Issues	1	■
Total Medium-Severity Issues	0	
Total Low-Severity Issues	0	
Total Informational-Severity Issues	5	■■■■■
Total	6	■■■■■■

Category Breakdown

Cryptography	4	■■■■
Data Validation	2	■■
Total	6	■■■■■■

Code Maturity Evaluation

Category Name	Description
Access Controls	**Not Applicable**
Arithmetic	**Moderate.** We reported various findings related to arithmetic. However, these largely represent improvements and are not currently exploitable issues.
Assembly Use	**Not Applicable.**
Centralization	**Not Applicable.**
Contract Upgradeability	**Not Applicable.**
Function Composition	**Satisfactory.** We noted a few ways in which the code could be improved (Appendix C), but otherwise, we found the code to be straightforward.
Front-Running	**Not Applicable.**
Key Management	**Strong.** We did not report any issues related to key management.
Monitoring	**Not Applicable.**
Specification	**Satisfactory.** The supplied documentation was fairly thorough. Additional comments could be added in `mpc` to improve readability.
Testing & Verification	**Satisfactory.** We found the codebases to have strong testing overall, and we encourage Ren to achieve full testing coverage.

Engagement Goals

The engagement was scoped to provide a security assessment of the `mpc`, `secp256k1`, and `shamir` repositories.

Specifically, we sought to answer the following questions:

- Are edge-case and malicious behaviors being handled properly? Does the implementation panic when given bad input?
- Is `libsecp256k1` wrapped correctly? Does it expose any unintentional C bugs?
- Is Shamir Secret Sharing implemented correctly? Do additive homomorphic secrets present any additional attack surfaces?
- Does the `mpc` implementation match its corresponding specification?
- Does `mpc` use the `shamir` and `secp256k1` libraries correctly?
- Does the code correctly implement what is documented in the comments?
- Are data structures being marshaled and unmarshaled correctly?

Coverage

`mpc`. An extensive manual review was conducted on the `mpc` module, guided by comments. We assessed the code for implementation correctness, and verified that invalid inputs are handled properly. The `mpc` computation was also analyzed for any potential cryptographic attacks, but we do not report any issues related to these.

`secp256k1`. The code was manually reviewed for correctness with respect to `libsecp256k1`. Several informational findings were produced. Our review focused on verifying that `libsecp256k1` is safely wrapped, ensuring that unsafe inputs are not passed into it, and that errors are properly caught.

`shamir`. An extensive manual review was conducted on the `shamir` module, guided by comments. A high-severity issue was discovered that allowed incorrect library usage to give away the secret for free by evaluating polynomials at 0. Several informational findings also emerged.

Recommendations Summary

This section aggregates all the recommendations made during the engagement. Short-term recommendations address the immediate causes of issues. Long-term recommendations pertain to the development process and long-term design goals.

Short term

□ **Add a check inside of the `ShareAndGetCoeffs` function in `shamir` to ensure that secret shares are never generated with an index of zero.** Alternatively, add this check to each of the `brng`, `rng`, `rkpg`, and `open` components. An index of zero would allow the secret to be revealed. TOB-RVM-001

□ **Add checks in the `New` functions inside of `brng`, `rng`, `rkpg`, and `open` that do not allow `h` to be the point at infinity.** If this parameter is set to the point at infinity, the security guarantees of the verifiable secret sharing scheme can be violated. TOB-RVM-002

□ **Either add a call `IsOnCurve` inside of `NewPointFromXY` or specify clearly in its surrounding comments that this check is not performed.** Currently, users may incorrectly assume that this function will verify the point is on the curve. TOB-RVM-003

□ **Either normalize the input field element before negating or add a comment surrounding `Negate` that states the input element is assumed to be normalized.** All assumptions on inputs should be clearly stated or enforced. TOB-RVM-004

□ **Either add a call to the `removeLeadingZeros` function inside both `IsZero` and `Eq`, or add a comment surrounding both `IsZero` and `Eq` that states it's assumed that leading zeros have been removed from the input polynomial.** All assumptions on inputs should be clearly stated or enforced. TOB-RVM-005

□ **Adjust XY to panic when called on the point at infinity.** TOB-RVM-006

Long term

□ **Ensure that indices are selected properly whenever using `mpc`.** Poorly selected indices could compromise the secret. TOB-RVM-001

□ **Ensure that `h` is generated safely in a manner that ensures its discrete log relative to `g` is unknown.** If `h` is not generated safely, the security guarantees of the verifiable secret sharing scheme could be violated. TOB-RVM-002

 If the `IsOnCurve` check is not performed, consider adjusting the name of `NewPointFromXY`. Currently, users may incorrectly assume that this function will verify the point is on the curve. TOB-RVM-003

 If a function operates with assumptions on any of its inputs, specify these assumptions clearly in surrounding comments. All assumptions on inputs should be clearly stated or enforced. TOB-RVM-004, TOB-RVM-005

 Consider using the `result, error` return pattern to avoid panicking when retrieving the coordinates of the point at infinity. TOB-RVM-006

Findings Summary

#	Title	Type	Severity
1	Index of zero is allowed for secret shares	Cryptography	High
2	Sanity checks missing for second secret sharing generator	Cryptography	Informational
3	NewPointFromXY does not verify the point is on the curve	Cryptography	Informational
4	NegateUnsafe assumes field element representation is normalized	Cryptography	Informational
5	IsZero and Eq assumes leading zeros have been removed	Data Validation	Informational
6	XY returns the wrong result for the point at infinity	Data Validation	Informational

1. Index of zero is allowed for secret shares

Severity: High | Difficulty: Undetermined
Type: Cryptography | Finding ID: TOB-RVM-001
Target: `shamir/shamir.go`, `brng.go`, `rng.go`, `rkpg.go`, `open.go`

Description
The RenVM protocol integrates Pedersen verifiable secret shares. Secret shares from Shamir's secret sharing scheme are used to distribute secret values to other parties in the protocol. Shamir secret shares are defined to be points on a polynomial. The secret value is defined as the evaluation of the polynomial at zero, and the shares that are distributed are any other non-zero points. It is imperative that these distributed shares are non-zero because evaluating the polynomial at zero immediately reveals the secret value.

Each of the `brng`, `rng`, `rkpg`, and `open` components assign indices to each party in the protocol. These indices correspond to the x-coordinate at which the polynomial will be evaluated. It is critical that no member of the protocol is assigned an index of zero, because their corresponding secret share would reveal the secret itself.

Currently, there are no checks in any of `brng`, `rng`, `rkpg`, or `open` components that ensure all of the indices are non-zero. Each of these components use `shamir` to compute secret shares, and there are currently no checks inside of `shamir` to ensure non-zero indices, either.

Exploit Scenario
The `mpc` component is used in a manner that allows parties to obtain an index of zero. An attacker, Eve, discovers a way to assign herself index zero; as a result, Eve immediately recovers all of the secret values in the protocol.

Recommendation
Short term, add a check inside of the `ShareAndGetCoeffs` function in `shamir` to ensure that secret shares are never generated with an index of zero. Alternatively, add this check to each of the `brng`, `rng`, `rkpg`, and `open` components.

Long term, ensure that indices are selected properly whenever using `mpc`.

2. Sanity checks missing for second secret sharing generator

Severity: Informational | Difficulty: N/A
Type: Cryptography | Finding ID: TOB-RVM-002
Target: `shamir/shamir.go`, `brng.go`, `rng.go`, `rkpg.go`, `open.go`

Description
The RenVM protocol integrates Pedersen verifiable secret shares. This scheme requires two generators, `g` and `h`, where the discrete log is unknown. The default generator is used for the `g` parameter, and the `h` parameter is taken as input in the `New` functions inside of `brng`, `rng`, `rkpg`, and `open`. These, in turn, call various functions inside of `shamir`, some of which operate under the assumption that points like `h` are not the point at infinity.

Setting this `h` parameter to the point at infinity would violate the security guarantees of the verifiable secret sharing scheme. However, currently there are no checks to ensure that `h` does not take unsafe values, such as the point at infinity.

Recommendation
Short term, add checks in the `New` functions inside of `brng`, `rng`, `rkpg`, and `open` that do not allow `h` to be the point at infinity.

Long term, ensure that `h` is generated safely in a manner that ensures its discrete log relative to `g` is unknown.

3. NewPointFromXY does not verify the point is on the curve

Severity: Informational
Type: Cryptography
Target: `secp256k1/point.go`

Difficulty: N/A
Finding ID: TOB-RVM-003

Description
The `secp256k1` repository wraps an underlying `libsecp256k1` C library and contains a series of functions responsible for initializing and operating on curve points. This includes `NewPointFromXY`, a function for initializing new curve points from `x` and `y` inputs. Specifically, this function initializes an empty `Point` object, then calls the `SetXY` function to apply the `x` and `y` inputs.

As seen from its surrounding comments, the `SetXY` function sets the curve point to the given `x` and `y` coordinates, but it does not verify if those coordinates actually correspond to a point on the curve. There is a separate `IsOnCurve` function in the repository that performs this check; however, it is not called inside either `SetXY` or `NewPointFromXY`.

The fact that `SetXY` does not verify whether the point is on the curve is mentioned in its surrounding comments. However, this is not mentioned in the comments surrounding the `NewPointFromXY` function. The name of this function could mislead users to think that this check is performed.

Recommendation
Short term, either add a call `IsOnCurve` inside of `NewPointFromXY` or specify clearly in its surrounding comments that this check is not performed.

Long term, if the `IsOnCurve` check is not performed, consider adjusting the name of `NewPointFromXY`.

4. NegateUnsafe assumes field element representation is normalized

Severity: Informational
Type: Cryptography
Target: `secp256k1/fp.go`

Difficulty: N/A
Finding ID: TOB-RVM-004

Description

The `secp256k1` repository contains several functions responsible for operating on field elements. The `Negate` function, which computes the additive inverse of a field element (see Figure 4.1), is one such function. In order to ensure consistent field representation, the repository also includes a function for normalizing field elements.

```
// Negate computes the additive inverse of the field element and stores the
// result in the receiver.
func (x *Fp) Negate(a *Fp) {
	if a == nil {
		panic("expected first argument to be not be nil")
	}
	x.NegateUnsafe(a)
}

// NegateUnsafe computes the additive inverse of the field element and stores
// the result in the receiver.
//
// Unsafe: If this function receives nil arguments, the behaviour is
// implementation dependent, because the definition of the NULL pointer in c is
// implementation dependent. If the NULL pointer and the go nil pointer are the
// same, then the function will panic.
func (x *Fp) NegateUnsafe(a *Fp) {
	// NOTE: The final argument is set to 0 because it is assumed that the
	// representation is normalized.
	C.secp256k1_fe_negate(&x.inner, &a.inner, 0)
	x.normalize()
}
```

Figure 4.1: The `Negate` *and* `NegateUnsafe` *functions (secp256k1/fp.go#L335-357).*

As seen in figure 4.1, the `Negate` function calls the `NegateUnsafe` function. The `NegateUnsafe` function takes an input field element and calls the underlying C library with the assumption that the input field element is normalized. This assumption is not stated in the comments surrounding `Negate`.

Recommendation

Short term, either normalize the input field element before negating, or add a comment surrounding `Negate` that states the input element is assumed to be normalized.

Long term, if a function operates with assumptions on any of its inputs, specify these assumptions clearly in surrounding comments.

5. IsZero and Eq assume leading zeros have been removed

Severity: Informational
Type: Data Validation
Target: `shamir/poly/poly.go`

Difficulty: N/A
Finding ID: TOB-RVM-005

Description
The `shamir` repository introduces the `Poly` type, which represents a polynomial in the field defined by the `secp256k1` elliptic curve. Therefore, a polynomial corresponds to an array of integers modulo `n`, the order of the `secp256k1` group.

The degree of a polynomial corresponds to the largest exponent in the polynomial with a non-zero coefficient. In `shamir`, the degree of the polynomial is determined based on the length of the array of coefficients. To handle zero coefficients, the function `removeLeadingZeros` is defined, which removes zero coefficients from the array altogether.

This repository also includes an `IsZero` function, which returns a boolean value indicating whether the input polynomial is the zero polynomial. Specifically, this function will return `false` if the input polynomial has any non-zero degree. As mentioned above, the degree is computed based on the length of the array of coefficients. Therefore, this `IsZero` computation is only valid when the `removeLeadingZeros` function has been called on the input polynomial. However, this is not expressed in the comments surrounding `IsZero`.

The same is also true for the function `Eq`. This function compares two polynomials by comparing first their degrees and then their coefficients. This comparison is only valid when both polynomials have had their leading zeros removed from their coefficients. This is also not expressed in the comments surrounding `Eq`.

Recommendation
Short term, either add a call to the `removeLeadingZeros` function inside both `IsZero` and `Eq`, or add a comment surrounding both `IsZero` and `Eq` stating that it is assumed leading zeros have been removed from the input polynomial.

Long term, if a function operates with assumptions on any of its inputs, specify these assumptions clearly in surrounding comments.

6. XY returns the wrong result for the point at infinity

Severity: Informational
Type: Data Validation
Target: `secp256k1/point.go`

Difficulty: N/A
Finding ID: TOB-RVM-006

Description
The `secp256k1` repository wraps an underlying `libsecp256k1` C library, and its `Point` struct represents a point in Jacobian coordinates with a flag for the point at infinity. However, the point at infinity does not have any legitimate `X`, `Y` representations. Furthermore, retrieving the `X`, `Y` coordinates calls the `XY` function, which in turn calls the `secp256k1_ge_set_gej` function, which divides the `X`, `Y` by `Z`. Default initialization of the point at infinity will set `X = Y = Z = 0`, and if any other result produces the point at infinity, `Z` will probably also be `0`. Depending on how `libsecp256k1` handles such operations, it will either return the wrong result or produce a runtime exception.

Exploit Scenario
A developer does not properly check for the point at infinity when retrieving X, Y coordinates and the application now has an obscure failure mode.

Recommendation
Short term, panic when `XY` is called on the point at infinity.

Long term, consider using the `result, error` return pattern to avoid panicking when retrieving the coordinates of the point at infinity.

A. Vulnerability Classifications

Vulnerability Classes	
Class	**Description**
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 protecting the privacy or integrity of 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
Testing	Related to test methodology or test coverage
Timing	Related to race conditions, locking, or order of operations
Undefined Behavior	Related to undefined behavior triggered by the program

Severity Categories	
Severity	**Description**
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or Defense in Depth
Undetermined	The extent of the risk was not determined during this engagement
Low	The risk is relatively small or is not a risk the customer has indicated is important

Medium	Individual user's information is at risk, exploitation would be bad for client's reputation, moderate financial impact, possible legal implications for client
High	Large numbers of users, very bad for client's reputation, or serious legal or financial implications

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploit was not determined during this engagement
Low	Commonly exploited, public tools exist or can be scripted that exploit this flaw
Medium	Attackers must write an exploit, or need an in-depth knowledge of a complex system
High	The attacker must have privileged insider access to the system, may need to know extremely complex technical details, or must discover other weaknesses in order to exploit this issue

B. Code Maturity Classifications

Code Maturity Classes	
Category Name	**Description**
Access Controls	Related to the authentication and authorization of components.
Arithmetic	Related to the proper use of mathematical operations and semantics.
Assembly Use	Related to the use of inline assembly.
Centralization	Related to the existence of a single point of failure.
Upgradeability	Related to contract upgradeability.
Function Composition	Related to separation of the logic into functions with clear purpose.
Front-Running	Related to resilience against front-running.
Key Management	Related to the existence of proper procedures for key generation, distribution, and access.
Monitoring	Related to use of events and monitoring procedures.
Specification	Related to the expected codebase documentation.
Testing & Verification	Related to the use of testing techniques (unit tests, fuzzing, symbolic execution, etc.).

Rating Criteria	
Rating	**Description**
Strong	The component was reviewed and no concerns were found.
Satisfactory	The component had only minor issues.
Moderate	The component had some issues.
Weak	The component led to multiple issues; more issues might be present.
Missing	The component was missing.

Not Applicable	The component is not applicable.
Not Considered	The component was not reviewed.
Further Investigation Required	The component requires further investigation.

C. Non-Security-Related Findings

This appendix contains findings that do not have immediate or obvious security implications.

- **Various functions in `shamir/poly/poly.go` will panic when the destination polynomial does not have sufficient capacity (e.g., `ScalarMul`, `Add`, and `Mul`).** Consider increasing the capacity of this polynomial when its current capacity is insufficient to prevent unnecessary panics.

- **`Fp` assumes normalized preconditions.** Each implemented `Fp` operation normalizes at the end of the function, but it would make sense to use `normalize` before the operations that require it to remove the precondition.

- **Consider expanding comments for `mpc`.** Ren supplied strong documentation for the `mpc` protocol. However, even with strong documentation, this implementation was somewhat challenging to follow. We recommend adding comments that connect each step in the implementation to its corresponding numbered step in the RenVM `mpc` wiki.