Spend Proof and Anonymised Returns for CARROT (SPARC)

Salvium
Version 0.4
17 Dec 2024

Table of Contents

Version History

Version	Date	Author	Comment
0.1	05/12/2024	SRCG	Initial draft
0.2	12/12/2024	SRCG	Revised k_{rp} and K_{SRA} calculations; added version history
0.3	12/12/2024	SRCG	Revised calculations; removed erroneous TX pool comment
0.4	17/12/2024	SRCG	Fixed typo as reported by Cypherstack

1. Introduction

This document describes technical information relevant to the cryptocurrency Salvium, and in particular the release Salvium One which is notably built with support for the CARROT transaction scheme. SPARC is described by Salvium as “a suite of extensions to the CARROT transaction scheme”, and specifically identifies the support of “anonymised returns” and “spend authority proof”. Each of these are considered in turn in the remainder of this document.

All terminology is assumed to be drawn from the official CARROT documentation[1] unless otherwise stated. The reader is assumed to be fully conversant with the content of that documentation.

2. Anonymised Returns

2.1 Overview

A “return address scheme” was first proposed for Monero and its derivations by knaccc[2] in 2019. Salvium Zero implemented the mechanism, along with some innovation in order to support the Salvium “protocol_tx” functionality (a necessary precursor to staking and accrued yield in Salvium).

However, Salvium One development has revealed that the original scheme is vulnerable to a quantum adversary capable of solving the discrete logarithm problem, and therefore needed to evolve. Anonymised Returns is the next generation return-address scheme. Fully CARROT-compatible, it addresses the “cascade” vulnerability present in the original design, and provides increased resistance against a quantum adversary.

The new scheme has been suggested by a Monero developer, and is adopted almost in its entirety.

2.2 Scheme Derivations

2.2.1 Intermediate Values

- $m_a^{SRA} = SecretDerive(\text{"}Carrot\ encryption\ mask\ a\text{"} \mid\mid s_{sr}^{ctx} \mid\mid K_0)$
- $a_{enc}^{SRA} = a \oplus m_a^{SRA}$ (where a is the amount of the output)
- $sra_g = SecretDerive(\text{"}Carrot\ key\ extension\ G\text{"} \mid\mid s_{sr}^{ctx} \mid\mid K_0)$
- $sra_t = SecretDerive(\text{"}Carrot\ key\ extension\ T\text{"} \mid\mid s_{sr}^{ctx} \mid\mid K_0)$
- L_0 = key image of first input to the transaction
- K_C = output onetime address of change component

2.2.2 Component Values

- $k_{rp} = ScalarDerive(\text{"}Carrot\ return\ address\ scalar\text{"} \mid\mid L_0)$
  (a derived scalar intended to replace y from the original scheme)
- $K_{SRA} = K_C + sra_g \cdot G + sra_t \cdot T$
  (a public key that only the sender and receiver can calculate)

[1] https://github.com/jeffro256/carrot/blob/master/carrot.md
[2] https://github.com/monero-project/research-lab/issues/53

2.3 Return Process

When sending funds to Bob, Alice calculates K_{SRA} and then stores $K_{SRA} \Rightarrow \{a_{enc}^{SRA}, m_a^{SRA}\}$ into a private hashmap table for later lookup.

If Bob wishes to return the payment to Alice, he must calculate k_{rp}, rederive the values sra_g, and sra_t to recompute K_{SRA}. Bob then sends to the one-time address $K_R = k_{rp}^{-1} \cdot K_{SRA}$ (where k_{rp}^{-1} is the multiplicative inverse of k_{rp}). Prior to performing the existing enote scan process, Alice recalculates $k_{rp} \cdot K_R$ and checks to see if that value is present in the private hashmap. If the value is found, then Alice proceeds to process the return directly. If the value is not found, the existing enote scan process is performed.

3. Spend Authority Proof

3.1 Overview

The purpose of the "spend authority proof" is to establish, in zero knowledge, that the prover knows the secret values x, y for a given key K_o, such that $K_o = x \cdot G + y \cdot T$. This requirement comes from the fact that knowledge of the x, y values permits an individual to be able to spend the output with the one-time address output key K_o.

The proof is designed for use with the cryptocurrency Salvium, which requires a means to prove that a one-time address output key is able to be spent by the sender of the transaction. The sender assumes the role of prover in this scenario, with the verifier being the recipient of any transaction output. The goal of the proof is to allow the verifier to know whether returned funds would be received by the prover. It is expected that $K_O = K_C$ (the one-time address output key for the transaction).

3.2 Proof Structure

The proof structure includes:

- Commitments: $R = r_x G + r_y T$, where r_x and r_y are random scalars.
- Responses:
  $z_x = r_x + c \cdot x$
  $z_y = r_y + c \cdot y$
  where $c = H_s(R \,||\, K_o)$

This structure ensures that:

- The commitment R hides the random values r_x and r_y.
- The responses z_x and z_y bind the challenge c to the secrets x and y.

3.3 Verification Process

The verifier performs the following checks:

1. Recomputes the challenge $c' = H_s(R \,||\, K_o)$.
2. Validates the commitments by testing the validity of the following calculation:

$$z_x G + z_y T = cK_o + R$$

Which expands as:

$$z_x G + z_y T - cK_o = r_x G + r_y T$$

$$r_x G + cxG + r_y T + cyT - cK_o = r_x G + r_y T$$

$$cxG + cyT = cK_o$$

Thus, the proof is valid if $z_x G + z_y T - cK_o$ matches R.

3.4 Security Analysis

To ensure the prover cannot manipulate the proof:

3.4.1 Unforgeability

- The challenge c is a cryptographic hash of the commitment and K_o. As a result:
  - The prover cannot predict or manipulate c because cryptographic hashes are resistant to pre-image attacks.
  - Any tampering with r_x,r_y, or K_o changes c, making it impossible to construct valid responses z_x and z_y without knowing x and y.

3.4.2 Binding of Responses

- The responses $z_x = r_x + c \cdot x$ and $z_y = r_y + c \cdot y$ bind c to the secrets x and y:
  - If x or y are incorrect, the responses will not satisfy the verification equation $z_x G + z_y T - cK_o = R = r_x G + r_y T$
  - The responses inherently depend on the random scalars r_x,r_y and the challenge c, ensuring there is no way to “fake” them.

3.4.3 Zero-knowledge

- The commitment $R = r_x G + r_y T$ is independent of the secrets x and y thanks to the random values r_x and r_y.
- The verifier learns nothing about x or y because:
  - The challenge c is deterministic but unpredictable, derived from a cryptographic hash.
  - The responses z_x and z_y are randomised by r_x and r_y, ensuring no direct leakage of x or y.

3.4.4 Soundness

- The verification equation ensures soundness because:

$$z_x G + z_y T - cK_o = r_x G + r_y T$$

  If the prover does not know x and y, they cannot produce responses z_x and z_y.

3.5 Assumptions

1. The hash function H_s used to calculate c is cryptographically secure and resistant to collisions.
2. The random scalars r_x and r_y are truly random and kept secret.
3. The scalar multiplication operations $r_x G$,$r_y T$, and others are performed correctly in the elliptic curve group.