A Review of SPARC

Rigo Salazar, Freeman Slaughter, Luke Szramowski, Cypher Stack *

January 14, 2025

In this document, we present a review of Spend Proof and Anonymised Returns for CARROT (SPARC, v0.3), a zero-knowledge proving system for the cryptocurrency Salvium [Sal]. By extending the transaction scheme of CARROT (which is itself an addressing protocol), SPARC creates a *return functionality*, where a receiver may remit return funds to a user who previously sent them a transaction. This return functionality is entirely optional, and users may decline to utilize it without penalty or any third party being aware.

The specification for SPARC may be found at [Som], though this manuscript is based on a document that was communicated privately, which we recreate in Sections 3 and 4. The documentation for CARROT can be found at [jef].

Contents

*https://cypherstack.com

1 Executive Summary

Here, we collect a concise summary of our security findings. Overall, we find that SPARC is an impressive protocol that is correct, fitting for its intended purpose, and will be of benefit to users. Many of the issues we take with the protocol, which we list in Section 1.1, are relatively minor, and serve to mainly convince readers of the aptness of SPARC. All of the more major issues, such as the lack of formality for security proofs, we address by formalizing them ourselves in Section 5. We additionally organize some natural properties that we would expect an idealized return functionality to enjoy in Section 2.6, then prove that SPARC obtains them in Section 5.

1.1 Recommended Action

In this subsection, we outline some comments and suggestions for the SPARC document, which we feel will benefit the protocol.

1. State exactly how much communication overhead SPARC takes up. This way, the reader may be convinced that this protocol is genuinely more efficient than simply creating another entire transaction to send return funds.

2. State exactly which data is public, which is known only to the sender, only to the receiver, etc. This will only benefit the reader.

3. Clearly communicate what capabilities the expected adversary enjoys. The worst-case (and average-case, if computable) complexity for a computationally bounded adversary running the state-of-the-art classical solving methods, say with Pollard's rho algorithm [Pol10] or the number field sieve method [LL93], should be outlined. Additionally, Section 2.1 states "increased protection against a quantum adversary". This must be made more precise - how much protection is added? The same complexity calculation should be done for an adversary armed with an idealized quantum computer running Shor's algorithm [Sho97].

4. Formalize the non-return functionality, or Requirement 2. Yes, it's clear that if the user wishes not to utilize the return functionality, then random data may be appended, but this should be explicitly stated for implementers. Also, it should be made more explicit in Section 3.1 what is meant by "it is expected that $K_C = K_o$" - what are the instances where this is not the case?

5. Outline what desirable properties the hash function $\mathcal{H}_s$ has. In practice, which hash function should be implemented? Blake2b and Keccak256 are the natural choices, as these are already utilized in CARROT.

6. Address typos throughout paper. A few examples include when r_x is introduced in Section 3.2, K_O vs. K_o, and L_O vs. L_o.

7. Describe in the specification which elliptic curve has been selected for this protocol, as well as some of its benefits and vulnerabilities. We assume that ED25519 is being used, as this is the curve listed in the CARROT documentation, but this should be elucidated. A potential drawback of ED25519 is that even though it is the most common elliptic curve for Schnorr-type signature instantiations, it can fall victim to small subgroup attacks. This is traditionally addressed using key clamping [Mad], which we pointed out in our audit of the CARROT specification [SGS]. Another possible preventative measure for this attack is to follow Schnorrkel, which utilizes Ristretto to implement a prime-order group atop an existing cofactor-8 curve [bur]. Such measures, and why they are or are not implemented, should be justified in order for implementers to be confident in the scheme.

8. We urge the authors to consider generating r_x and r_y in a pseudorandom fashion, as the hash of the ephemeral Diffie-Hellman key for example, similar to what's done in EdDSA [BJL+15]. This will help to avoid some of the potential vulnerabilities listed in Section 6. Poor choice of randomness for the equivalent value in the EdDSA, also based on the Schnorr scheme, led to some Bitcoin users losing control of their wallets in 2013 [Bit], which is a vulnerability we wish to anneal SPARC against.

9. Define subprotocols that are intended strictly for the developers, as it would be a grave mistake to omit these details in the specification. As an example, implementers of SPARC would need to be careful about distinguishing between a group element and its serialization. This subtle feature has been neglected in the past and led to signature malleability attacks, resulting in funds being hemorrhaged for Bitcoin [DW14]. In the same vein, there should be a check in the protocol to make sure that certain values are indeed integers, and that they are in the allowed range for scalars: say $z \in [1, \ell - 1]$ for instance.

10. Ensure that all variables match the official CARROT documentation. Some variables, such as m_a^{SRA}, have been renamed. To prevent confusion, we recommend sticking with the established notation whenever possible.

2 Preliminaries

2.1 Adversary Capabilities

Two hard problems we highlight are the discrete logarithm problem and the problem of finding hash collisions. For this document, we assume that both are intractable to a computationally bound adversary, but will make this statement more exact later in the document.

To introduce the notions of security more formally, suppose we have access to a public group-generator algorithm GGen that accepts input $\mathbb{1}^\lambda$, where λ is the security parameter. This algorithm outputs a description of the group $\mathbb{G}$, which has prime order ℓ, as well as generator $g \in \mathbb{G}$ if needed. We denote the field of ℓ elements as $\mathbb{F}_\ell$.

A *negligible* function is simply a function f such that $|f(x)|$ decays faster than the reciprocal of any polynomial of x, for x large enough. We reserve the notation $x \leftarrow S$ to denote that the element x has been drawn from the set S in a possibly deterministic manner, and $r \xleftarrow{\$} S$ to mean that r has been sampled uniformly at random from S.

Definition 1. Let Δ and δ be two distribution algorithms which accept security parameter λ as input. We say that these algorithms are *indistinguishable* if, for any non-uniform, probabilistic, polynomial-time adversary $\mathcal{A}$, the following advantage is negligible:

$$\left|\Pr[1 \leftarrow \mathcal{A}(x) \,|\, x \leftarrow \Delta(\mathbb{1}^\lambda)] - \Pr[1 \leftarrow \mathcal{A}(x) \,|\, x \leftarrow \delta(\mathbb{1}^\lambda)]\right| \leq \mathsf{negl}(\lambda)$$

2.2 Discrete Logarithm

Problem 1 (Discrete Logarithm Problem)**.** The discrete logarithm problem is:

$$\text{given input } \{(g,\, g^{\alpha}) \,|\, (\mathbb{G},\, \ell,\, g) \leftarrow \mathsf{GGen}(\mathbb{1}^{\lambda}),\, \alpha \xleftarrow{\$} \mathbb{F}_{\ell}\},\ \text{recover } \alpha.$$

Assumption 1. We assume that the discrete logarithm problem is hard. More specifically, the following advantage is bounded by some negligible function for all probabilistic, polynomial-time adversaries $\mathcal{A}$:

$$\Pr[\alpha \leftarrow \mathcal{A}(g,\, g^{\alpha}) \,|\, (\mathbb{G},\, \ell,\, g) \leftarrow \mathsf{GGen}(\mathbb{1}^{\lambda}),\, \alpha \xleftarrow{\$} \mathbb{F}_{\ell}] \leq \mathsf{negl}(\lambda)$$

2.3 Hash Functions

We introduce a few desirable properties that we require for a cryptographic hash function, with definitions taken from [RS04]. We assume that for message space $\mathcal{M} = \{0,\, 1\}^k$ of length k, the hash function $\mathcal{H} : \mathcal{M} \to \{0,\, 1\}^n$. We also use $\mathcal{A}$ to denote a probabilistic, polynomial-time adversary for the rest of this document.

Assumption 2 (Collision Resistance)**.** We assume that finding hash collisions is hard:

$$\Pr[\mathcal{H}(m_1) = \mathcal{H}(m_2) \,|\, (m_1,\, m_2) \leftarrow \mathcal{A}(\mathbb{1}^{\lambda}) \text{ for } (m_1,\, m_2) \in \mathcal{M}^2 \text{ and } m_1 \neq m_2] \leq \mathsf{negl}(\lambda)$$

We also require preimage resistance, which states that it's difficult to calculate preimages. This is not implied by collision resistance.

Assumption 3 (Preimage Resistance)**.** We assume that finding a hash's preimage is hard:

$$\Pr[m \leftarrow \mathcal{A}(h,\, \mathbb{1}^{\lambda}) \,|\, h = \mathcal{H}(m) \text{ for } m \in \mathcal{M}] \leq \mathsf{negl}(\lambda)$$

Finally, we require *second* preimage resistance, which is implied by collision resistance. We note that second preimage resistance implies preimage resistance only if the length of the hash digests are allowed to be much longer than the length of the hash outputs.

Assumption 4 (Second Preimage Resistance)**.** Given a specific input, it's hard to find another hash digest that results in the same output:

$$\Pr[y \leftarrow \mathcal{A}(x,\, \mathbb{1}^{\lambda}) \,|\, \mathcal{H}(x) = \mathcal{H}(y) \text{ for } (x,\, y) \in \mathcal{M}^2 \text{ and } x \neq y] \leq \mathsf{negl}(\lambda)$$

2.4 Pedersen Commitments

Formally, a commitment scheme is comprised of two algorithms: the first is an efficient randomized setup algorithm setup which generates a commitment c, then also defines the message space $\mathcal{M}$, a randomness space $\mathcal{R}$, and a commitment space $\mathcal{C}$. We let pp denote public parameters, generated from $\mathsf{setup}(\mathbb{1}^{\lambda})$, where λ is some security parameter. The commitment function $\mathsf{commit} : \mathcal{M} \times \mathcal{R} \to \mathcal{C}$ uses a message and randomness to formulate a commitment. For a message $m \in \mathcal{M}$, the randomness $r \xleftarrow{\$} \mathcal{R}$ is selected uniformly, then the commitment is formed by $c := \mathsf{commit}(m;\, r)$.

Definition 2 (Hiding)**.** We say that a commitment scheme $(\mathsf{setup},\, \mathsf{commit})$ is *computationally hiding* if it is infeasible to determine the secret message m from $\mathsf{commit}(m;\, r)$. This is, for every probabilistic, polynomial-time adversary $\mathcal{A}$, the following holds:

$$\left| \Pr\left[\mathcal{A}(c) = b \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{setup}(\mathbb{1}^{\lambda}) \\ m_1,\, m_2 \leftarrow \mathcal{M} \\ b \xleftarrow{\$} \{1,\, 2\} \\ r \xleftarrow{\$} \mathcal{R} \\ c := \mathsf{commit}(m_b;\, r) \end{array} \right] - \tfrac{1}{2} \right| \leq \mathsf{negl}(\lambda)$$

If this probability is exactly $\frac{1}{2}$ for all $\mathcal{A}$, then we say the scheme is *perfectly* hiding.

Definition 3 (Binding)**.** We say that a commitment scheme (setup, commit) is *computationally binding* if the probability of finding $(m_1; r_1) \neq (m_2; r_2)$ such that $\mathsf{commit}(m_1; r_1) = \mathsf{commit}(m_2; r_2)$ is very low. This is, for all polynomial-time adversaries $\mathcal{A}$:

$$\Pr\left[\; c_1 = c_2 \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{setup}(\mathbb{1}^\lambda) \\ (m_1, m_2, r_1, r_2) \leftarrow \mathcal{A}(\mathsf{pp}) \text{ with } m_1 \neq m_2 \\ c_i = \mathsf{commit}(m_i; r_i) \text{ for } i = 1, 2 \end{array} \right] \leq \mathsf{negl}(\lambda)$$

If this probability is exactly 0 for all $\mathcal{A}$, then we say the scheme is *perfectly* binding.

Both of the above definitions assume a computationally bounded adversary. If we remove the "polynomial-time" condition on the adversary, then the definitions are *statistical* hiding and binding. Recall that the discrete logarithm problem is only *computationally* hiding, whereas the introduction of randomness r in the commitment scheme strengthens this condition to *perfect* hiding.

The most common commitment scheme used for zero-knowledge protocols is the Pedersen commitment, which relies on the hardness of the discrete logarithm problem from Problem 1. This type of commitment enjoys perfect hiding and computational binding, both of which are very nice to have for practical purposes. Pedersen commitments are used in many popular cryptocurrencies such as Monero [Noe15], Pinocchio coin [DFKP13], and Firo (formerly known as Zcoin) [MGGR], and additionally finds use in other frameworks like Zether [BAZB19], Pretty Good Confidential [CMTA19], and MERCAT [JLAA21]. The Pedersen commitment takes the form $\mathsf{commit}(m; r) = mG + rH$, with $r \xleftarrow{\$} \mathcal{R}$ and $m \in \mathcal{M}$, where G and H are a generators of group $\mathbb{G}$ specified in setup. The above commitment is in additive notation, but some authors prefer multiplicative notation, which then looks like $\mathsf{commit}(m; r) = g^m h^r$. Pedersen commitments are perfectly hiding, due to the fact that r is taken as a uniformly random value [Pro, KO11]. On the other hand, breaking the binding property is in fact equivalent to extracting discrete logarithms, which by Assumption 1 we assume cannot be done efficiently. Finally, Pedersen commitments enjoy the homomorphic property, which in additive notation states that $\mathsf{commit}(m_1; r_1) + \mathsf{commit}(m_2; r_2) = \mathsf{commit}(m_1 + m_2; r_1 + r_2)$.

For more background on these definitions and their use-cases, we cite [Blu83, Jon23, BCC+16, BBB+17, CHJ+20, EKRN22, BKM05].

2.5 Zero-Knowledge Proofs

A *zero-knowledge proof* (ZKP) is a game played between a prover and a verifier, where at the end of the game the verifier will be convinced that a particular statement is true, without learning any other information except its veracity.

Let $\mathcal{L}$ be some language, and let $\mathcal{R}$ be a relation that can be verified efficiently, in polynomial time perhaps, such that a statement s is in $\mathcal{L}$ if and only if there exists a witness w satisfying the relation $(s; w) \in \mathcal{R}$.

A ZKP can be seen as a triple of algorithms $\Pi = (\mathsf{Gen}, \mathcal{P}, \mathcal{V})$, all of which are probabilistic, polynomial time algorithms. These are representative of a generating algorithm Gen, the prover $\mathcal{P}$, and the verifier $\mathcal{V}$. Gen accepts an input of $\mathbb{1}^\lambda$ with security parameter λ and outputs a string σ. The prover's algorithm $\mathcal{P}(\sigma, (s; w)) = \pi$ accepts input of the string σ, a public statement s, and a secret witness w, then produces a proof π. The verifier's algorithm $\mathcal{V}(\sigma, s, \pi)$ takes in string σ, statement s, and proof π, then outputs $b \in \{0, 1\}$, where $b = 1$ notates that the proof has been accepted, and $b = 0$ indicates rejection. When the prover and verifier act on their inputs p and v, then they produce a transcript $\tau \leftarrow \langle \mathcal{P}(p), \mathcal{V}(v) \rangle$.

A protocol has three major properties that must be satisfies to be a ZKP: zero-knowledge, completeness, and soundness. We take these definitions from references such as [Jon23]

Definition 4 (Completeness)**.** A proof protocol Π for relation $\mathcal{R}$ is called *complete* if, given security parameter λ, for all $(s; w) \in \mathcal{R}$, the probability a valid proof is wrongly rejected is negligible:

$$\Pr\left[\; \mathcal{V}(\sigma, s, \pi) = \mathsf{rej} \;\middle|\; \begin{array}{l} \sigma \leftarrow \mathsf{Gen}(\mathbb{1}^\lambda) \\ (s; w) \in \mathcal{R} \\ \pi = \mathcal{P}(\sigma, (s; w)) \end{array} \right] \leq \mathsf{negl}(\lambda)$$

Definition 5 (Soundness)**.** A proof protocol Π for relation $\mathcal{R}$ is called *sound* if for all probabilistic polynomial-time adversaries $\mathcal{A}$, the probability that their proof get wrongly accepted is negligible:

$$\Pr\left[\begin{array}{l|l} \mathcal{V}(\sigma,\, s,\, \pi) = \mathsf{acc} & \begin{array}{l}\sigma \leftarrow \mathsf{Gen}(\mathbb{1}^\lambda) \\ (s;\, w) \notin \mathcal{R} \\ \pi \leftarrow \mathcal{A}(\sigma,\, (s;\, w))\end{array}\end{array}\right] \leq \mathsf{negl}(\lambda)$$

Definition 6 (Zero-Knowledge)**.** Let ρ be the verifier's random public coin. A public coin protocol is said to be *zero-knowledge* for relation $\mathcal{R}$ if there exists a probabilistic, polynomial-time simulator $\mathcal{S}$ such that for all polynomial time adversary $\mathcal{A}$, the following holds for some negligible function:

$$\left|\Pr\left[\begin{array}{l|l} \mathcal{A}(\tau) = 1 \\ (s;\, w) \in \mathcal{R} & \begin{array}{l}\sigma \leftarrow \mathsf{Gen}(\mathbb{1}^\lambda) \\ ((s;\, w),\, \pi) = \mathcal{A}(\sigma) \\ \tau \leftarrow \langle \mathcal{P}(\sigma,\, (s;\, w)),\, \mathcal{V}(\sigma,\, s,\, \pi)\rangle\end{array}\end{array}\right] - \Pr\left[\begin{array}{l|l} \mathcal{A}(\tau') = 1 \\ (s;\, w) \in \mathcal{R} & \begin{array}{l}\sigma' \leftarrow \mathsf{Gen}(\mathbb{1}^\lambda) \\ ((s;\, w),\, \rho) = \mathcal{A}(\sigma') \\ \tau' \leftarrow \mathcal{S}(s,\, \rho)\end{array}\end{array}\right]\right| \leq \mathsf{negl}(\lambda)$$

2.6 Return Functionality and Novel Security Definitions

Suppose that Alice sends funds to Bob. What we aim to formalize here is a *return functionality*, in which Alice can generate a transaction sending funds to Bob, then somehow append information that Bob can use later to generate a transaction sending return funds back to Alice. This should be doable in an efficient and secure manner. Such a functionality must satisfy a few natural requirements, which we outline here. To the best of our knowledge, these defining characteristics in conjunction are novel, and this is our attempt at standardizing them.

1. An adversary cannot distinguish a transaction containing return data from a transaction without it
2. If a user declines to include return data, this should be apparent to the intended recipient
3. Multiple transactions with return data to or from the same users should not reveal their identities
4. An adversary who knows a transaction contains return data should not be able to uncover the return address
5. An adversary should not be able to tamper with the return data in order for the change to go to them instead of the intended recipient

The following criteria are our attempts to formalize the casual conditions above: Let $\mathcal{E}$ denote an arbitrary enote, where $\mathcal{E}_{RD}$ denotes that this enote has return data attached and $\mathcal{E}_{\oslash}$ denotes that the user has declined to include return data. Let $\mathcal{A}$ denote Alice, $\mathcal{B}$ denote Bob, and $\mathcal{M}$ denote Mallory, a malicious probabilistic polynomial-time adversary. Then $\mathcal{A} \xrightarrow{\mathcal{E}_{RD}} \mathcal{B}$ is the notation we use to show that Alice sent an enote with return data to Bob. We use $\mathcal{M}(\mathcal{E})$ to denote Mallory's appraisal upon inspection of an enote, where this value equals 1 if Mallory correctly returns whether or not it contains return data, and 0 if she returns wrong.

Requirement 1 we call *return functionality indistinguishability*, in the sense of Definition 1. This requirement states that the probability Mallory can distinguish between enotes with return data and those without any better than chance is negligible. We break these up into two seperate conditions below:

$$\left|\Pr\left[\ \mathcal{M}(\mathcal{E}_{RD}) = 1 \ \middle|\ \mathcal{A} \xrightarrow{\mathcal{E}_{RD}} \mathcal{B}\ \right] - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda), \quad \text{and}$$

$$\left|\Pr\left[\ \mathcal{M}(\mathcal{E}_{\oslash}) = 1 \ \middle|\ \mathcal{A} \xrightarrow{\mathcal{E}_{\oslash}} \mathcal{B}\ \right] - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda).$$

Requirement 2 is *intended-recipient distinguishability*, where if Alice sends Bob an enote without return data, then Bob can recognize this.

$$\Pr\left[\ \mathcal{B}(\mathcal{E}_{\oslash}) = 1 \ \middle|\ \mathcal{A} \xrightarrow{\mathcal{E}_{\oslash}} \mathcal{B}\ \right] = 1.$$

Requirement 3 is *unlinkability*, where if n enotes with or without appended return data are sent, say $\mathcal{E}_i$ is sent by $\mathcal{A}_i$ to $\mathcal{B}_i$ for $i = 1, ..., n$, then Mallory cannot link any of the enotes to their sender or recipient.

$$\Pr\left[\ \mathcal{M}(\mathcal{E}_1, ..., \mathcal{E}_n) \text{ can link } \mathcal{E}_i \text{ to } \mathcal{A}_i \text{ or } \mathcal{B}_i \ \middle|\ \mathcal{A}_1 \xrightarrow{\mathcal{E}_1} \mathcal{B}_1, ..., \mathcal{A}_n \xrightarrow{\mathcal{E}_n} \mathcal{B}_n\ \right] \leq \mathsf{negl}(\lambda).$$

This functionality combines the notions of *sender unlinkability* and *receiver unlinkability*: the first is where we permit Mallory to inspect n enotes, where $\mathcal{E}_i$ is sent from Alice to $\mathcal{B}_i$ for $i = 1, ..., n$, then Mallory should be unable to determine that Alice was the sender with any non-negligible advantage. Similarly, the second is where the enote $\mathcal{E}_i$ is sent by $\mathcal{A}_i$ to Bob, and Mallory should be unable to determine that Bob was the recipient with any non-negligible advantage. We note that the conjunction of these two definitions is equivalent to the full definition of unlinkability given earlier:

$$\Pr\left[\ \mathcal{M}(\mathcal{E}_1, ..., \mathcal{E}_n) \text{ can link } \mathcal{A} \text{ to } \mathcal{E}_i \ \middle|\ \mathcal{A} \xrightarrow{\mathcal{E}_1} \mathcal{B}_1, ..., \mathcal{A} \xrightarrow{\mathcal{E}_n} \mathcal{B}_n\ \right] \leq \mathsf{negl}(\lambda).$$

$$\Pr\left[\ \mathcal{M}(\mathcal{E}_1, ..., \mathcal{E}_n) \text{ can link } \mathcal{B} \text{ to } \mathcal{E}_i \ \middle|\ \mathcal{A}_1 \xrightarrow{\mathcal{E}_1} \mathcal{B}, ..., \mathcal{A}_n \xrightarrow{\mathcal{E}_n} \mathcal{B}\ \right] \leq \mathsf{negl}(\lambda).$$

Requirement 4 is *return address hiding*, where even if Mallory knows that an enote contains return data, Alice's return address remains inaccessible.

$$\Pr\left[\ \mathcal{M}(\mathcal{E}_{RD}) \text{ can recover } RD \ \middle|\ \mathcal{A} \xrightarrow{\mathcal{E}_{RD}} \mathcal{B}\ \right] \leq \mathsf{negl}(\lambda).$$

This is dual in some sense to sender unlinkability in Requirement 3 above. To be exact, if Mallory can recover the return address, then she knows Alice's address, even if she doesn't know Alice's identity.

Requirement 5 is *non-swindling*, meaning that Mallory cannot intercept Alice's enote and replace the return data with her own, thus siphoning the return funds and swindling Alice.

$$\Pr\left[\ \mathcal{B} \text{ views } \mathcal{E}_{RD'} \text{ as valid from } \mathcal{A} \ \middle|\ \mathcal{A} \xrightarrow{\mathcal{E}_{RD}} \mathcal{M} \xrightarrow{\mathcal{E}_{RD'}} \mathcal{B}\ \right] \leq \mathsf{negl}(\lambda).$$

Here, we point out there's no benefit to Mallory, behaving as a man-in-the-middle, to remove Alice's return data and append just random values. This will still result in a valid enote, but will appear that Alice chose not to utilize the return functionality. As there's realistically no way to prevent Alice's return data from being stripped off by an intermediary and replaced with random values, we assume this won't happen in practice. We also note that the non-swindling property is distinct from *burning*, where a user is tricked into "burning" the funds contained in an enote, rendering them inaccessible to anyone [dEB, Par]. There are a number of ways to avoid this issue: Lelantus Spark [JF21], Jamtis [kj], and CARROT solve it by hard-coding a variable named $\mathsf{input\_context}$ into their enotes. While this check is valid for regular enotes, we discuss burning in relation to the return functionality later in Section 6.

For this document, we presume that the discrete logarithm problem is hard on the two curves implemented in CARROT: the Montgomery curve Curve25519 [Ber06] which is used for the initial Diffie-Hellman key exchange, and the twisted Edwards curve Ed25519 with 8ℓ total points [BDL+11] which is used for all other purposes. When we refer to $\mathbb{G}$, we mean the cyclic subgroup of the twisted Edwards curve with prime order $\ell = 2^{252} + 27742317777372353535851937790883648493$. We take this to imply that the adversarial advantage in Assumption 1 is negligible for all classically-armed probabilistic polynomial-time adversaries. We assume that the hash function $\mathcal{H}_s$ enjoys collision resistance from Assumption 2, preimage resistance from Assumption 3, and second preimage resistance from Assumption 4. Additionally, it should be indistinguishable from random values in the sense of Definition 1, and generally well-suited for cryptographic purposes. We also

take it as a given that all Pedersen commitments, which in CARROT and hence SPARC occur over the twisted Edwards curve [BDL+11], are hiding and binding, from Definition 2 and Definition 3 respectively. Finally, we assume that the above Requirements 1 - 5 hold for the SPARC return functionality, but defer our justification to Section 5.6.

3 Derived Values

For this section, we refer the unfamiliar reader to the specification for both SPARC and CARROT, as we draw the notation for the following section from these documents.

Recall that an enote's transaction output is defined as the point (K_o, C_a), where K_o is a one-time output key and $0 \leq a < 2^{64}$ is the amount in question. To generate this transactional output, Alice must know z and a such that $C_a = zG + aH$. To spend this output, Bob must know x, y, z, and a such that $K_o = xG + yT$ and $C_a = zG + aH$. Here, G, T, and H are generators for the cyclic subgroup $\mathbb{G}$ of prime order. Spending this output necessarily generates a *key image*, defined as $L = x\mathcal{H}(K_o)$.

3.1 Intermediate Values

Below are the values defined in SPARC.

$$
\begin{aligned}
m_a^{SRA} &= \text{SecretDerive}(\text{“Carrot encryption mask a”} \,||\, s_{sr}^{ctx} \,||\, K_o) \\
a_{enc}^{SRA} &= a \oplus m_a^{SRA} \\
sra_g &= \text{SecretDerive}(\text{“Carrot key extension G”} \,||\, s_{sr}^{ctx} \,||\, K_o) \\
sra_t &= \text{SecretDerive}(\text{“Carrot key extension T”} \,||\, s_{sr}^{ctx} \,||\, K_o) \\
L_o &\text{ is defined as the key image of the first input in the transaction} \\
K_C &\text{ is defined as the output one-time address for the change component}
\end{aligned}
$$

We note that some of these values are similar to those in the CARROT doc. For instance, m_a^{SRA} and a_{enc}^{SRA} are exactly m_a and a_{enc} from CARROT, while sra_g and sra_t are almost k_g^o and k_t^o, with the difference being that they have K_o in their hash digest instead of C_a. This makes them dual, in some sense, but also replaces the dependency on the amount with dependency on the one-time output key. We additionally note that it's anticipated that $K_C = K_o$, the one-time address output key.

Here, the shared secret s_{sr}^{ctx} is defined in the CARROT spec as

$$s_{sr}^{ctx} = \text{SecretDerive}(\text{“Carrot sender-receiver secret”} \,||\, s_{sr} \,||\, D_e \,||\, \mathsf{input\_context}).$$

The component s_{sr} can be defined in different ways depending on the type of enote, but is generally a shared secret derived from the ephemeral Diffie-Hellman public key D_e, which is a Montgomery curve point. The component input_context is intended to be unique for each transaction, and is derived from block height or key images.

3.2 Component Values

$$
\begin{aligned}
k_{rp} &= \text{ScalarDerive}(\text{“Carrot return address scalar”} \,||\, L_o) \\
&\quad \text{this is meant to replace the } y \text{ from the original scheme} \\
K_{SRA} &= K_C + sra_g \cdot G + sra_t \cdot T \\
&\quad \text{this is intended to be a pubkey that only Alice and Bob are able to calculate}
\end{aligned}
$$

3.3 Return Process

When Alice wishes to transfer funds to Bob, she must calculate K_{SRA}, then store the values K_{SRA} and $(a_{enc}^{SRA},\, m_a^{SRA})$ into a private hashtable for lookup later.

When Bob wishes to use the return functionality, he must calculate k_{rp}, then rederive the values sra_g and sra_t. This permits him to recompute K_{SRA} as defined above. Bob can then send his funds to the one-time address $K_R := k_{rp}^{-1} \cdot K_{SRA}$. Before going through with the standard enote scanning process, Alice calculates $k_{rp} \cdot K_R$ and searches through her private hashtable to check if that value can be found there. If it can be found, then Alice continues with the return as normal. If not, then she continues with the standard enote scanning process.

4 Proof Structure

SPARC is a zero-knowledge proof system that enables a return functionality, intended to permit a sender to demonstrate that they have the capability to spend a specific one-time address output key, i.e.: that they know the secrets x and y that form K_o. Here, the sender Alice assumes the role of a prover, akin to a sigma protocol. The verifier, who is usually denoted Bob, is represented in practice by autonomous code, which runs automatically before the transaction can be accepted into the pool.

The prover starts by sampling r_x and r_y as random scalars, then committing to them using the Pedersen commitment $R = r_x G + r_y T$. Then $c = \mathcal{H}(R \,||\, K_o)$ is the challenge associated with this commitment. The random scalars are also used to form the responses $z_x = r_x + cx$ and $z_y = r_y + cy$. This ensures the hiding of r_x and r_y, and the binding of c to the secrets x and y. Then the prover communicates R, z_x, and z_y, which the verifier will use to reconstruct the nominal challenge as $c' = \mathcal{H}(R \,||\, K_o)$. They will then check to see if $z_x G + z_y T - c' K_o$ is equal to R; if yes, then the prover has passed the verifier's check, and now the verifier can know that returned funds will be received by the prover.

Proof structure for SPARC		
Public: $G, T \in \mathbb{G}$, $\mathbb{F}_\ell$, $\mathcal{H}_s$, $K_o = xG + yT$ **Private:** $x, y \in \mathbb{F}_\ell$		
Alice		**Bob**
Sample $r_x, r_y \xleftarrow{\$} \mathbb{F}_\ell$ Commit with $R = r_x G + r_y T$ Calculate $c = \mathcal{H}_s(R \,||\, K_o)$		
Form $z_x = r_x + cx$ and $z_y = r_y + cy$	$\xrightarrow{R,\ z_x,\ z_y}$	Compute $c' = \mathcal{H}_s(R \,||\, K_o)$ Check $R \stackrel{?}{=} z_x G + z_y T - c' K_o$

Table 1: The verification process for SPARC.

The interactive version		
Public: $G, T \in \mathbb{G}$, $\mathbb{F}_\ell$, $\mathcal{H}_s$, $K_o = xG + yT$		
Private: $x, y \in \mathbb{F}_\ell$		
Alice		**Bob**
Sample $r_x, r_y \xleftarrow{\$} \mathbb{F}_\ell$		
Commit with $R = r_xG + r_yT$	$\xrightarrow{R}$	
	$\xleftarrow{c}$	$c \xleftarrow{\$} \mathbb{F}_\ell \setminus \{0\}$
Form $z_x = r_x + cx$ and $z_y = r_y + cy$	$\xrightarrow{z_x,\ z_y}$	Check $R \stackrel{?}{=} z_xG + z_yT - c'K_o$

Table 2: Three-round interactive version, which we use to prove certain security properties.

5 Security Analysis

5.1 Completeness

The verifier computes the purported challenge $c' = \mathcal{H}(R \mid\mid K_o)$, then performs the following check: $R \stackrel{?}{=} z_xG + z_yT - c'K_o$. If computed honestly, this check will be passed, as

$$\begin{aligned} z_xG + z_yT - c'K_o &= (r_x + cx)G + (r_y + cy)T - c'(xG + yT) \\ &= cxG - c'xG + cyT - c'yT + r_xG + r_yT \\ &= (c' - c)(xG + yT) + (r_xG + r_yT) \\ &= (c' - c)K_o + R \\ &= R \text{ if and only if } c' = c. \end{aligned}$$

If c' differs from c, then the check will not be passed. As the verifier's nominal recomputation c' is formed from information passed over a public channel, it will match c, and thus if the prover behaves in an honest manner then they will pass the verifier's check. Thus, SPARC is complete, as desired.

5.2 Soundness

One can argue that an adversary without knowledge of secrets x and y will only be able to pass the check in Table 1 by sheer chance, unless they can violate the discrete logarithm assumption of Assumption 1. As the group $\mathbb{G}$ is very large, one can argue that this will happen with expected probability on the order of $1/\ell$, to the point that we may assume it will never happen in standard practice. In the specification, the authors argue that SPARC obtains soundness, from Definition 5, however we can do better with special soundness:

Definition 7 (Special Soundness)**.** A proof Π for relation $\mathcal{R}$ achieves *special soundness* if there exists a deterministic polynomial-time extractor algorithm Ext with the following property: given a statement $s \in \mathcal{L}$ and two accepting transcripts $(R,\ c,\ z_x,\ z_y)$ and $(R,\ c',\ z'_x,\ z'_y)$, which share the same commitment R, then Ext will output a witness w such that $(s;\ w) \in \mathcal{R}$.

Because the challenge space where c is selected is large, we have that special soundness implies soundness from Definition 5.

In Table 2, suppose that the protocol runs twice, then outputs transcripts $(R,\ c,\ z_x,\ z_y)$ and $(R,\ c',\ z'_x,\ z'_y)$. This protocol has the *special soundness* property if these transcripts can be used to recover the secrets x and y. This clearly holds, as one may compute

$$\frac{z_x - z'_x}{c - c'} = x \quad \text{and} \quad \frac{z_y - z'_y}{c - c'} = y,$$

using the two transcripts to recover the secrets. Hence the protocol in Table 2 achieves special soundness, thus it is also sound [AFR23]. The non-interactive SPARC from Table 1 inherits this soundness from the interactive version, via the standard reduction applying the Fiat-Shamir heuristic.

5.3 Zero-Knowledge

In the specification, it's argued that SPARC achieves the zero-knowledge property from Definition 6. We present a proof that the interactive version of SPARC in Table 2 achieves the special honest-verifier zero-knowledge property, which we use as a proxy for the standard definition of zero-knowledge. This can be shown in a more straightforward fashion, by constructing a simulator algorithm which is indistinguishable from an honest interaction but will always appear to be accepted.

Definition 8 (Special Honest-Verifier Zero-Knowledge)**.** A proof protocol Π for relation $\mathcal{R}$ is said to have *special honest-verifier zero-knowledge* if there exists a probabilistic polynomial-time simulator algorithm Sim such that the following two properties both hold:

- For all statements $s \in \mathcal{L}$ and challenges c, then $\mathsf{Sim}(s, c)$ outputs (R, z) such that (R, c, z) is an accepting transcript for s.
- For all $(s; w) \in \mathcal{R}$, let c be a random challenge and (R, z) the output from $\mathsf{Sim}(s, c)$. Then (R, c, z) is identically distributed, in the vein of Definition 1, to a genuine interaction between $\mathcal{P}$ and $\mathcal{V}$.

This simulator Sim can be constructed readily as follows, by permitting it and the verifier to agree on challenge c ahead of time - say, by a shared coin. Specifically, it will select c, z_x, z_y uniformly at random, with c being a secret shared by Sim and the verifier. Then R will be defined as $z_x G + z_y T - cK_o$, which will always pass the verifier's challenge, thus appears to be an accepting transcript. Due to the hiding and indistinguishability properties of Pedersen commitments, and that these values were chosen uniformly at random, they will be indistinguishable from the transcript produced in a genuine interaction.

As both properties are satisfied, the interactive version in Table 2 achieves special honest-verifier zero-knowledge, thus it is zero-knowledge, hence SPARC from Table 1 inherits this zero-knowledge property.

5.4 Hiding and Binding

Proper usage of the Pedersen commitments ensures the hiding of r_x and r_y in the commitment R, from Definition 2. This is not violated by sending values containing the secrets, like $z_x = r_x + cx$ and $z_y = r_y + cy$, since a naive adversary is forced to solve for multiple unknowns as once (e.g: for r_x and x) which is an intractable task. Given the large size of the cyclic subgroup $\mathbb{G}$, a computationally bounded adversary will be forced to spend an intractably long time simply guessing at values, reinforcing the hiding property of the protocol.

This functionality additionally achieves binding, from Definition 3, as the challenge c is bound to the secrets x and y. We also note that these values, since r_x and r_y were selected uniformly at random, will also appear uniformly random, which gives an additional indistinguishability property akin to Definition 1. Subsequently, we achieve both the hiding and binding property.

5.5 Unforgeability

From the collision resistance given in Assumption 2 and preimage resistance given in Assumption 3, we may surmise that an adversarial prover cannot predict, manipulate, reuse, or rederive the challenge c. Additionally, any tampering with the intermediate values, like r_x or r_y, will be immediately identifiable and permit users to terminate communication. Due to the cryptographic properties of the chosen hash function $\mathcal{H}_s$, we may be rest assured that an adversary attempting to manipulate values for their own use must be able to solve a computationally intractable problem. As such, we attain the desired property of unforgeability.

5.6 Return Functionality Requirements

5.6.1 Indistinguishability

To see that this protocol has return functionality indistinguishability, Requirement 1, we note that Alice - who does not wish to include return data - can simply append a uniformly at random value. If Alice were

to neglect this and append no value at all, then Mallory would be able to inspect the enote and recognize this. So, by Alice selecting a random value when she does not wish to include return data, Mallory will be unable to distinguish between these two cases, since the honestly-computed return data appears uniformly at random. We note that if Mallory can violate this indistinguishability, then she can discriminate between return data and random values; since the return data is derived as a hash digest, this implies that the hash function of choice is not well-suited for this purpose - contrary to our assumptions. Thus, ignoring any advantage that can be obtained from metadata, we find it to be generally intractable for an adversary to distinguish between enotes with return data and those without, giving us return function indistinguishability.

5.6.2 Intended-Recipient Distinguishability

This protocol achieves intended-recipient distinguishability, Requirement 2. This can be seen readily, since if Alice does not wish to take advantage of the return functionality, she will simply append a random value. Only Bob will have the ability to determine that this return data is not genuine, as otherwise any adversary with the ability to distinguish will violate Requirement 1. It would, of course, be disappointing if Bob were unable to distinguish between enotes with return data and those without, so we highlight the purpose of Requirements 1 and 2: that Bob, and only Bob, should be able to distinguish.

5.6.3 Unlinkability

The property of unlinkability, Requirement 3, applies because of the unlinkability properties of CARROT - see the specification document for more details. Additionally, as Mallory will be unable to compute Alice's one-time address, she will not be able to use this to connect Alice to her address. This follows from the hiding property of Pedersen commitments, Definition 2.

5.6.4 Return Address Hiding

Requirement 4, return address hiding, is achieved by SPARC because if Mallory could inspect an enote and recover the one-time address K_R, which is defined as $k_{rp}^{-1} \cdot K_{SRA}$, then she must have knowledge of k_{rp} and K_{SRA} independently. As these are never presented in plaintext, it must be that Mallory was able to recover their constituent defining variables, which would violate the properties of Pedersen commitments or else Mallory can calculate hash collisions. Thus, from inspecting only the return data, Mallory cannot hope to recover the return address, which belongs to Alice and can be derived by Bob, providing us with Return Address Hiding.

5.6.5 Non-Swindling

The final Requirement 5, non-swindling, follows from the spend authority proof in Table 1. If the verifier's check is successfully passed, then Bob can be rest assured that his return funds will be directed towards Alice, and not to Mallory, except with near-zero probability.

6 Possible Attack Avenues

6.1 Nonce Reuse Attack

This is an attack method where an adversarial prover reuses one-time randomness, referred to as a nonce, from a previous transactions permitting them to falsify proofs. This is recognized as a common weakness, similar to a rewind attack, and thus the random scalars r_x and r_y "should be used for the present occasion and only once" `https://cwe.mitre.org/data/definitions/323.html` as a best-use practice. Besides outright reuse, biased nonces can also introduce vulnerabilities: if r_x and r_y are not uniform, then the secret key can be extracted.

6.2 Janus Attack

The Janus attack (or pordo attack when specified to Monero, from the Esperanto word for "door"; see [SGS]) is a framework in the legacy addressing protocol where a curious user, not necessarily malicious, can link a private subaddress to an individual in a way that they cannot immediately detect. More exactly, suppose that Bob publicly own subaddress $(K_s^i,\, K_v^i)$, but Alice suspects that Bob secretly owns a subaddress $(K_s^j,\, K_v^j)$. Instead of publishing the transactional pubkey rK_s^i and output pubkey $\mathcal{H}(rK_v^i)G + K_s^i$ for some sender-chosen random r, Alice publishes transactional pubkey rK_s^j and output pubkey $\mathcal{H}(rK_v^j)G + K_s^i$. Bob cannot recognize this, but if they affirm that they received the funds, they confirm to Alice that they do in fact own that subaddress.

CARROT prevented Janus attacks by introducing a *Carrot anchor* that forces an adversary attempting to leverage the Janus attack to solve a computationally intractable problem. More specifically, without knowledge of the recipient's private view key k_v, it's intractable to construct an external enote where the nominal address spend pubkey $K_s^{i\prime} = K_o - k_g^o G - k_t^o T$ matches the shared secret $s_{sr} = 8r\text{ConvertPointE}(K_s^{i\prime})$. This effectively relies on the discrete logarithm problem, Assumption 1.

We find that SPARC is not vulnerable to the Janus attack, because of the non-swindling property, Requirement 5. This forces an adversary aiming to replace Alice's return data with their own to siphon the return funds to solve computationally intractable problems, that of the discrete logarithm, Assumption 1, and finding hash collisions, Assumption 2.

6.3 Burning Bug

The burning bug is a method where an adversary can send two enotes that are linked, both of which appear valid but one of them for an amount of 0, and if the user spends this empty enote first then the funds in the other enote become inaccessible. This comes at only a marginal cost to the adversary, so can hypothetically have significant repercussions to an unsuspecting victim. This issue was patched in Jamtis and CARROT by binding the output public key to input_context, forcing a user attempting to burn funds to solve a computationally intractable problem.

We find that SPARC remains resistant to this vulnerability, meaning that Mallory cannot trick a user into burning their return funds, due to the return process not being linked to the amount a. Instead of being reliant on C_a, a Pedersen commitment to a, the return functionality is built upon K_o, preventing fund burning.

6.4 Frozen Heart

"FoRging Of ZEro kNowledge proofs" [Jon] is an attack avenue, usually utilized alongside the weak Fiat-Shamir heuristic [FS87, BPW16], where an adversary can generate false proofs by leveraging that the hash function only contains commitments, but omits group generators and public data. As an example, the Schnorr protocol [Sch91] falls victim to this - see Table 4 for how this attack works on the Schnorr scheme, which can be thought of as the progenitor for SPARC. In fact, many proof systems fall victim to this vulnerability: we refer to [DMWG23, NHB24] for more details. While it's not stated explicitly, in essence SPARC utilizes the Fiat-Shamir heuristic on the Schnorr protocol, so this should be considered as a potential method for an adversary to generate fake proofs and inflate the supply.

Luckily, we find that this attack avenue is not feasible, due to the fact that the generators $G, T \in \mathbb{G}$ are fixed in the CARROT specification. If these were selected "at random" by the prover, as is the case in the original Schnorr protocol, then an adversarial prover could select a malicious generator G', which will appear perfectly random, in order to falsify proofs. As the true generator G is decided by the protocol and is immutable, this attack cannot be mounted, so SPARC is annealed against Frozen Heart.

The non-interactive Schnorr protocol		
Public: $\mathbb{F}_\ell$, $\mathcal{H}_s$ **Private:** $x \in \mathbb{F}_\ell$		
Alice		**Bob**
Select $G \xleftarrow{\$} \mathbb{G}$ Compute $K_o = xG$ Sample $r \xleftarrow{\$} \mathbb{F}_\ell$ Commit with $R = rG$ Calculate $c = \mathcal{H}_s(R \mid\mid K_o)$ Form $z = r + cx$	$\xrightarrow{G,\ K_o,\ R,\ z}$	Compute $c' = \mathcal{H}_s(R \mid\mid K_o)$ Check $R \stackrel{?}{=} zG - c'K_o$

Table 3: The Schnorr signature scheme under the weak Fiat-Shamir transform

Frozen Heart attack on the Schnorr protocol		
Public: $\mathbb{F}_\ell$, $\mathcal{H}_s$ **Private:** $x \in \mathbb{F}_\ell$		
Alice		**Bob**
Sample $R \xleftarrow{\$} \mathbb{G},\ z \xleftarrow{\$} \mathbb{F}_\ell$ Compute $K_o = xG'$ Calculate $c = \mathcal{H}_s(R \mid\mid K_o)$ Compute $G' = z^{-1}(R + cK_o)$	$\xrightarrow{G',\ K_o,\ R,\ z}$	Compute $c' = \mathcal{H}_s(R \mid\mid K_o)$ Check $R \stackrel{?}{=} zG' - c'K_o$

Table 4: A Frozen Heart attack on the Schnorr signature scheme

We note that the traditional method of preventing this vulnerability in the Schnorr protocol is to include the generator G in the hash that defines c, along with the commitments R and K_o, as well as, ideally, salt.

7 A Protocol with Better Overhead

We propose an improvement to the SPARC protocol, which permits the verifier to calculate 2 fewer elliptic curve point computations. The verifier must compute r_xG and r_yT given scalars r_x and r_y, and generators G and T. Despite using multiplicative notation in this document, realistically these point computations will be exponentiations like g^x, and performing these calculations independently is not efficient. While computing double-scalar multiplication can be sped up by implementing Shamir's trick from `https://www.lirmm.fr/~imbert/talks/laurent_Asilomar_08.pdf`, we present a more efficient, computationally friendly solution.

The key difference here is that we shift the double-scalar multiplication's computational overhead to the prover, which can be done as a preprocessing step. This will result in less live time being devoted to point computation, which will speed up the verification process. We present the interactive version in Table 5, and the non-interactive variety, as it would appear in practice, in Table 6. This adaptation also helps to protect against poor randomness selection of r_x and r_y, as Z_G and Z_T hide them under the discrete logarithm assumption. As this does not alter the underlying protocol in any significant fashion, all the security properties of SPARC will carry over, so we present this protocol without repeating such proofs.

An interactive version with better computational overhead		
Public: $G, T \in \mathbb{G}$, $\mathbb{F}_\ell$, $\mathcal{H}_s$, $K_o = xG + yT$		
Private: $x, y \in \mathbb{F}_\ell$		
Alice		**Bob**
Sample $r_x, r_y \xleftarrow{\$} \mathbb{F}_\ell$		
Commit with $R = r_xG + r_yT$	$\xrightarrow{R}$	
	$\xleftarrow{c}$	$c \xleftarrow{\$} \mathbb{F}_\ell \setminus \{0\}$
Form $z_x = r_x + cx$ and $z_y = r_y + cy$		
Compute $Z_G = z_xG$ and $Z_T = z_yT$	$\xrightarrow{R,\ Z_G,\ Z_T}$	Check $R \stackrel{?}{=} Z_G + Z_T - cK_o$

Table 5: An improved three-round interactive protocol, with the differences from Table 2 highlighted.

An improvement for SPARC		
Public: $G, T \in \mathbb{G}$, $\mathbb{F}_\ell$, $\mathcal{H}_s$, $K_o = xG + yT$		
Private: $x, y \in \mathbb{F}_\ell$		
Alice		**Bob**
Sample $r_x, r_y \xleftarrow{\$} \mathbb{F}_\ell$		
Commit with $R = r_xG + r_yT$		
Calculate $c = \mathcal{H}_s(R \mid\mid K_o)$		
Form $z_x = r_x + cx$ and $z_y = r_y + cy$		
Calculate $Z_G = z_xG$ and $Z_T = z_yT$	$\xrightarrow{R,\ Z_G,\ Z_T}$	Compute $c' = \mathcal{H}_s(R \mid\mid K_o)$
		Check $R \stackrel{?}{=} Z_G + Z_T - c'K_o$

Table 6: An improvement to the computational overhead of SPARC.

References

[AFR23] Thomas Attema, Serge Fehr, and Nicolas Resch. Generalized special-sound interactive proofs and their knowledge soundness. Cryptology ePrint Archive, Paper 2023/818, 2023. https://eprint.iacr.org/2023/818.

[BAZB19] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. Zether: Towards privacy in a smart contract world. Cryptology ePrint Archive, Paper 2019/191, 2019. https://eprint.iacr.org/2019/191.

[BBB+17] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. Cryptology ePrint Archive, Paper 2017/1066, 2017. https://eprint.iacr.org/2017/1066.

[BCC+16] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. Cryptology ePrint Archive, Paper 2016/263, 2016. https://eprint.iacr.org/2016/263.

[BDL+11] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. Cryptology ePrint Archive, Paper 2011/368, 2011. https://eprint.iacr.org/2011/368.

[Ber06] Daniel J. Bernstein. Curve25519: New diffie-hellman speed records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography - PKC 2006*, pages 207–228, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. https://doi.org/10.1007/11745853_14.

[Bit] Bitcoin. Android security vulnerability. https://bitcoin.org/en/alert/2013-08-11-android. Accessed: 2025-01-14.

[BJL+15] Daniel J. Bernstein, Simon Josefsson, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. EdDSA for more curves. Cryptology ePrint Archive, Paper 2015/677, 2015. https://eprint.iacr.org/2015/677.

[BKM05] Adam Bender, Jonathan Katz, and Ruggero Morselli. Ring signatures: Stronger definitions, and constructions without random oracles. Cryptology ePrint Archive, Paper 2005/304, 2005. https://eprint.iacr.org/2005/304.

[Blu83] Manuel Blum. Coin flipping by telephone a protocol for solving impossible problems. *SIGACT News*, 15(1):23–27, January 1983. https://doi.org/10.1145/1008908.1008911.

[BPW16] David Bernhard, Olivier Pereira, and Bogdan Warinschi. How not to prove yourself: Pitfalls of the fiat-shamir heuristic and applications to helios. Cryptology ePrint Archive, Paper 2016/771, 2016. https://eprint.iacr.org/2016/771.

[bur] burdges. Schnorrkel. https://github.com/w3f/schnorrkel. Accessed: 2025-01-12.

[CHJ+20] Heewon Chung, Kyoohyung Han, Chanyang Ju, Myungsun Kim, and Jae Hong Seo. Bulletproofs+: Shorter proofs for privacy-enhanced distributed ledger. Cryptology ePrint Archive, Paper 2020/735, 2020. https://eprint.iacr.org/2020/735.

[CMTA19] Yu Chen, Xuecheng Ma, Cong Tang, and Man Ho Au. PGC: Pretty good decentralized confidential payment system with auditability. Cryptology ePrint Archive, Paper 2019/319, 2019. https://eprint.iacr.org/2019/319.

[dEB] dEBRUYNE. A Post Mortem of The Burning Bug. https://www.getmonero.org/2018/09/25/a-post-mortum-of-the-burning-bug.html. Accessed: 2024-10-23.

[DFKP13] George Danezis, Cedric Fournet, Markulf Kohlweiss, and Bryan Parno. Pinocchio coin: building zerocoin from a succinct pairing-based proof system. In *Proceedings of the First ACM Workshop on Language Support for Privacy-Enhancing Technologies*, PETShop '13, page 27–30, New York, NY, USA, 2013. Association for Computing Machinery. https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/pinocoin.pdf.

[DMWG23] Quang Dao, Jim Miller, Opal Wright, and Paul Grubbs. Weak fiat-shamir attacks on modern proof systems. Cryptology ePrint Archive, Paper 2023/691, 2023. https://eprint.iacr.org/2023/691.

[DW14] Christian Decker and Roger Wattenhofer. Bitcoin transaction malleability and mtgox. In Mirosław Kutyłowski and Jaideep Vaidya, editors, *Computer Security - ESORICS 2014*, pages 313–326, Cham, 2014. Springer International Publishing. https://doi.org/10.1007/978-3-319-11212-1_18.

[EKRN22] Liam Eagen, Sanket Kanjalkar, Tim Ruffing, and Jonas Nick. Bulletproofs++: Next generation confidential transactions via reciprocal set membership arguments. Cryptology ePrint Archive, Paper 2022/510, 2022. https://eprint.iacr.org/2022/510.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology — CRYPTO' 86*, pages 186–194, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-47721-7_12.

[jef] jeffro256. Carrot. https://github.com/jeffro256/carrot/blob/master/carrot.md. Accessed: 2025-01-14.

[JF21] Aram Jivanyan and Aaron Feickert. Lelantus Spark: Secure and Flexible Private Transactions. Cryptology ePrint Archive, Paper 2021/1173, 2021. https://eprint.iacr.org/2021/1173.

[JLAA21] Aram Jivanyan, Jesse Lancaster, Arash Afshar, and Parnian Alimi. MERCAT: Mediated, encrypted, reversible, SeCure asset transfers. Cryptology ePrint Archive, Paper 2021/106, 2021. https://eprint.iacr.org/2021/106.

[Jon] Marvin Jones. Vac 101: Transforming an interactive protocol to a noninteractive argument. https://vac.dev/rlog/vac101-fiat-shamir/. Accessed: 2024-10-15.

[Jon23] Marvin Jones. *Zero-Knowledge Reductions and Confidential Arithmetic*. PhD thesis, Clemson University, 2023. https://tigerprints.clemson.edu/all_dissertations/3472.

[kj] koe and jeffro256. Implementing Seraphis (WIP). https://raw.githubusercontent.com/UkoeHB/Seraphis/master/implementing_seraphis/Impl-Seraphis-0-0-4.pdf. Accessed: 2024-10-23.

[KO11] Takeshi Koshiba and Takanori Odaira. Non-interactive statistically-hiding quantum bit commitment from any quantum one-way function, 2011. https://arxiv.org/abs/1102.3441.

[LL93] Arjen K. Lenstra and Hendrik W. Lenstra. The development of the number field sieve. In *Springer: Lecture Notes in Mathematics*, 1993. https://api.semanticscholar.org/CorpusID:123670445.

[Mad] Neil Madden. What's the Curve25519 clamping all about? https://neilmadden.blog/2020/05/28/whats-the-curve25519-clamping-all-about/. Accessed: 2025-01-12.

[MGGR] Ian Miers, Christina Garman, Matthew Green, and Aviel Rubin. Zerocoin: Anonymous Distributed E-Cash from Bitcoin. https://www.allcryptowhitepapers.com/zcoin-whitepaper/. Accessed: 2024-11-13.

[NHB24] Hieu Nguyen, Uyen Ho, and Alex Biryukov. Fiat-shamir in the wild. Cryptology ePrint Archive, Paper 2024/1565, 2024. https://eprint.iacr.org/2024/1565.

[Noe15] Shen Noether. Ring signature confidential transactions for monero. Cryptology ePrint Archive, Paper 2015/1098, 2015. https://eprint.iacr.org/2015/1098.

[Par] Luke "kayabaNerve" Parker. Remove the burning bug as a class of attack with a modified shared key definition. https://github.com/monero-project/research-lab/issues/103. Accessed: 2024-10-23.

[Pol10] By J. M. Pollard. Monte carlo methods for index computation (mod p). In *American Mathematical Society: Mathematics of Computation*, 2010. https://api.semanticscholar.org/CorpusID:235457090.

[Pro] Zecrey Protocol. Pedersen commitment in zecrey. https://zecrey.medium.com/pedersen-commitment-in-zecrey-170981f71b86. Accessed: 2024-11-13.

[RS04] Phillip Rogaway and Thomas Shrimpton. Cryptographic hash-function basics: Definitions, implications and separations for preimage resistance, second-preimage resistance, and collision resistance. Cryptology ePrint Archive, Paper 2004/035, 2004. https://eprint.iacr.org/2004/035.

[Sal] Salvium. Salvium. https://salvium.io/. Accessed: 2025-01-14.

[Sch91] C. P. Schnorr. Efficient signature generation by smart cards. *J. Cryptol.*, 4(3):161–174, January 1991. https://doi.org/10.1007/BF00196725.

[SGS] Freeman Slaughter, Brandon Goodell, and Rigo Salazar. An audit of the FCMP++ addressing protocol: CARROT. https://github.com/cypherstack/carrot-audit. Accessed: 2025-01-14.

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997. https://doi.org/10.1137/S0097539795293172.

[Som] Some Random Crypto Guy. Spend proof and anonymised returns for CARROT (SPARC). https://github.com/somerandomcryptoguy/carrot/blob/master/carrot.md#8-spend-proof-and-anonymised-returns-for-carrot-sparc. Accessed: 2025-01-14.