An Audit of the FCMP++ Addressing Protocol: CARROT

Freeman Slaughter, Brandon Goodell, Rigo Salazar, Cypher Stack*

November 22, 2024

In this document, we audit the framework of CARROT (Cryptonote Address on Rerandomizable-RingCT-Output Transactions). CARROT is an addressing protocol, intended to be compatible with the FCMP++ (Full Chain Membership Protocol + Spend Authorization + Linkability) upgrade that is coming to Monero. The specification document can be found at `https://github.com/jeffro256/carrot/blob/master/carrot.md`

This scheme is intended to patch two potential vulnerabilities in the FCMP protocol, namely the burning bug and Janus attack, which we discuss in greater detail in subsequent sections, while still remaining compatible with the current addressing protocol. CARROT introduces new security models intended to give the user more concrete security guarantees, such as a novel key hierarchy, while supporting indistinguishability to another addressing protocol Jamtis.

Contents

*`https://cypherstack.com`

1 Introduction

Monero is a privacy-oriented cryptocurrency based on the 2013 CryptoNote v2 whitepaper [vS] that offers many strong secrecy guarantees; indeed, no third party can efficiently reveal transaction addresses, amounts, or records. At times, Monero implements protocol upgrades in order to better serve their customer base and make sure that security is as tight as possible. In this document, we audit a potential future Monero upgrade CARROT and find that it is well-suited to patch two main vulnerabilities: the burning bug, where a malicious user could force an unsuspecting victim to throw away funds, and the Janus attack, where a curious adversary armed with public information about a known address could reveal that a user has ownership of a private address. Additionally, we find that CARROT obtains desirable features such as unlinkability, forward secrecy, and indistinguishability. While we do present suggestions to improve upon the protocol, largely we support CARROT's security claims, and believe it is well-suited for its intended purpose as an improvement to the Monero protocol.

1.1 Executive Summary

Here, we collect a concise summary of our security findings.

We were able to verify the security claims for completeness and binding, various unlinkability conditions, forward secrecy, and indistinguishability of values, as well as burning bug resistance and Janus attack resistance, where an adversary who can perform these attacks at-will can either efficiently find hash collisions or break the discrete logarithm problem, which we assume are computationally intractable - see Section 1.4 for a formal security overview. However, we believe that some of the statements for these security conditions could be made more exacting, such as upgrading the initial discrete logarithm to a Pedersen commitment for Transaction Output Random Indistinguishability in Section 3.4.1. This is a stronger statement, and fits more in line with the condition that the authors of CARROT wish to show.

Beyond some slight typos or mislabeling, we recommended certain actions be taken: for instance, the specification document would benefit from an in-depth discussion about key clamping - or lack thereof - since this concept is related to efficient implementation and confinement attack prevention. For a complete list of recommendations, navigate to Section 2.

1.2 Background

Monero is slated to adopt Seraphis [koea], which is a peer-to-peer electric currency protocol based on the RingCT (Ring Confidential Transactions) [kAN] paradigm, and which uses extremely efficient building blocks, like [BBB$^+$17, CHJ$^+$20]. Seraphis will enable much larger RingCT sizes, but is not backwards compatible with existing CryptoNote addresses. As such, the Monero Research Lab (`https://www.getmonero.org/resources/research-lab/`) views this as an auspicious opportunity to upgrade the current addressing protocol in order to improve key features and mitigate potential security problems. For an informal writeup of Seraphis, as well as its progenitor Triptych [NG20], which was an upgrade to CLSAG (Concise Linkable Spontaneous Anonymous Group) signatures [GNB19, Monb], which itself was an upgrade from MLSAG (Multilayered Linkable Spontaneous Anonymous Group) signatures [Noe15], we refer the interested reader to [koeb]. We note that Seraphis shares many features with Lelantus Spark [JF21a, JF21b], a Firo protocol

[Yap]. Despite being quite similar, Seraphis is strictly more efficient, as it uses a succinct version of "Grootle" proofs ([GK14] and [BCC+15]).

One option for a new addressing scheme is Jamtis [tev], which is tailor-made to be compatible with Seraphis. This protocol is intended to fix various potential issues with CryptoNote, such as divulging too much wallet information during third-party scanning and linking wallets via Janus attack - this will be discussed in greater depth in Section 1.3.2. The addressing protocol CARROT which we discuss in this document is distinct from Jamtis, though is designed to be indistinguishable from it, and they share some security features.

1.2.1 FCMP++

FCMP++ [Par24] is a double-upgrade to the original Full-Chain Membership Proofs, adding on + Spend Authorization and + Linkability, intended as a replacement to Monero's ring signatures, thus detaching Monero from its reliance on Seraphis. A few notable improvements over ring signatures are that FCMP++ is resistant to EAE attacks [Mona], it has greater resiliency against naive statistical analysis, and it uses an anonymity set that is many orders of magnitude larger ($\sim$ 5,000,000x larger) [Parb]. As a historical aside, two distinct protocols were suggested for Monero under the name FCMP: the first [Parc] was proposed to deploy alongside Seraphis, while the second [Par24] was intended to move the paradigm away from Seraphis. Here, we mean the second, and will refer to it throughout this document as FCMP++ without ambiguity. For more background on FCMP as well as future directions, we refer the interested reader to [CHAK22, Eag22, COPZ22, Pard, Para].

CARROT is the potential addressing protocol for Monero that we focus on in this manuscript, which layers on top of FCMP++ by defining the ruleset for addressing and sending funds, while the underlying FCMP++ defines how to validate transactions. This addressing protocol is intended to be practically indistinguishable from Jamtis, and moreover, the new addresses it generates are required to be computationally indistinguishable from legacy addresses. Naturally, CARROT also comes with backwards compatibility, so users with legacy CryptoNote addresses are not excluded from the new hierarchy. CARROT offers security features like burning bug resistance, unlinkability, indistinguishability, and forward secrecy through the use of dedicated outgoing view keys and RingCT Rerandomization abstraction.

1.3 Undesirable Vulnerabilities

In this section, we highlight two attack vectors that appeared as artifacts of the legacy enote scanning process: the burning bug [dEB] and the Janus attack [Jus].

1.3.1 Burning Bug

The burning bug [dEB] is named for the outcome of the vulnerability, where a user is tricked into "burning" the funds in an enote. Specifically, in Monero's legacy scan process, a malicious sender can copy an existing transaction using some of the same public key information, then send this poisoned enote to a receiver. This recipient will successfully scan both enotes, but because key images are generated from the output public keys, and repeat key images are not permitted, they will only be able to spend the first one scanned - thus the other is burned. This malicious user can copy a legitimate enote and create a tainted enote corresponding to an amount of 0 moneroj, so that if the receiver spends this poisoned enote first, then the funds in the original one will be inaccessible. This comes at a nominal cost for the malicious user, but could potentially be quite damaging to an unsuspecting recipient.

Originally, a patch for this issue was quietly issued as a source code pull request [Ric], and exchange services were notified. This initial patch simply warned users of suspected burning bug enotes with a warning message, as opposed of somehow preventing its occurrence in the first place. Later patches dealt with this by recognizing the burning bug's poisoned enote clique, then discarding duplicate enotes unless they

corresponded to the greatest transactional amount in the clique. Unfortunately, these workarounds were rather delicate, and required a user to have knowledge of the record of all previous enote scans in the chain.

Jamtis, one of the first protocols to prevent burning bugs, achieved this by "baking an input_context into enotes" [kj], forcing burning bug attackers to solve a computationally intractable problem. We note that this isn't the only method of preventing the burning bug attack [kay]. Seraphis attempted to address this by requiring uniqueness of the the enote's ephemeral public key $\mathsf{D_e}$ (see [kj, Section 8.2.2]), which is generally computed from the receiver's address key.
CARROT solves this problem in a similar manner to Jamtis, by binding output public keys to the value input_context; uniqueness then follows from consensus rules in the chain, so inspecting previous enote scanning history is not required.

1.3.2 Janus Attack

The Janus attack is an identifying attack, aimed at discerning whether or not two addresses belong to the same wallet [Jus, Wor]. The name Janus refers to the two-faced Roman deity representing duality, new beginnings, and doorways, loaning us the name of the month "January" and the occupation "janitor," among others. The duality in this vulnerability is that two different addresses can potentially be linked to the same wallet by a user. As a linguistic aside, we tongue-in-cheek propose that the Janus attack for Monero should be referred to as the "pordo" attack, from the Monero word for door!

In CryptoNote, the balance recovery is split into two parts: using the view key with ECDH (elliptic curve Diffie Hellman) to create a shared secret, then using this shared secret to recover the spend key [vS]. As the two constituent sub-processes are implemented separately, both parts can come from alternate subaddress, hence a curious - but not necessarily malicious - user can create an enote using a subaddress that they believe is owned by a recipient, but that appears to be sent to their public address. When this recipient confirms that their public address received the enote, then they inadvertently confirm that the putatively owned address subaddress belongs to them.

Suppose that a victim has a public address P and a secret address S. An adversary knows that this individual owns address P, but suspects they also own S. To duplicitously authenticate this suspicion, they construct a transaction using only public information about S, but that the victim will view as being sent to address P. When the victim naively confirms the transaction from address P with a receipt, the adversary then has confirmation that the victim owns address S.

This issue was patched in Jamtis by permitting users to recognize Janus output [tev]. The 3-key variant of Jamtis provides protection against Janus as well as improve the third party enote scanning process, since they are related concepts. Using a similar concept in the burning bug mitigation, Jamtis prevents Janus attacks by baking the subaddress index j into the ECDH shared secret (specifically the second secret $\mathsf{s}_2^{\mathsf{sr}}$; see [kj, Section 8.4.3] for more details), thus connecting Janus attack prevention to solving a hard problem. We note that this is not the only method of patching this vulnerability [Sar].

CARROT addresses this issue by introducing a Janus anchor, aptly named "anchor," which is used to rederive an enote's ephemeral private key $\mathsf{d_e}$ and verify the ephemeral public key $\mathsf{D_e}$. The anchor is (usually, see [jef, Section 7.4] for details) a uniformly random 16-byte array that is then encrypted by XORing with a random encryption mask. This value guarantees that either Janus attacks are prevented, or the sender knows the private view key $\mathsf{k_v}$. In this second case, there is no need for Janus output, as they can easily determine if two addresses are in the same wallet. Additionally, the view tag vt has the input_context baked into it, so that a third party cannot simply copy values into a new enote, as then the view tag will only match if a hash collision has been found, which we assume occurs with negligible probability due to collision resistance.

1.4 Preliminaries

1.4.1 Notation

For this manuscript, we denote honest elements like t and dishonest, corrupted, or proxy values as $\mathbf{t}$. So a sender might attempt to send a receiver t, but it's intercepted by an adversary, and this malicious party sends the receiver $\mathbf{t}$; separately, if two senders are interacting with the same receiver, one would send t and the other $\mathbf{t}$. We also denote t' as the receiver's attempted reconstruction of the sender's t from the limited information available to them. We reserve the notation $r \xleftarrow{\$} S$ to mean that the element r has been sampled uniformly at random from the set S.

When context is clear, we suppress hash input domain-separating tags in our notation. This is for brevity and pedagogical reasons solely, and is not good practice on the implementation side, so please take care when reading this document. As an example, if the hash in question is something like

$$\mathsf{SecretDerive}(\mathsf{domain\_separating\_tag} \,||\, \mathsf{important\_stuff}),$$

we simply write $\mathsf{SecretDerive}(\mathsf{important\_stuff})$. We suppress this input because when it comes to hash padding, we just don't CARROT-all.

1.4.2 Adversary Capabilities

Some hard problems we highlight are the discrete log problem, the Diffie-Hellman problem, and the problem of finding hash collisions. We assume that these are intractable for a computationally bound adversary. The supposed difficulty of these problems (ie: the security level at practical parameters, expected time for generic attacks like Pollard's rho algorithm [Pol10] or the general number field sieve [LL93], etc.) should be stated somewhere in the CARROT documentation (see comment 5 in Section 2).

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 p, as well as generator $g \in \mathbb{G}$ if needed. We denote the field of p elements as $\mathbb{F}_p$. 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.

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

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

Definition 1. We let $\mathcal{E}$ denote a hypothetical extractor algorithm which can break Assumption 1 with impunity. That is, $\mathcal{E}$ can accept input $(g,\, g^\alpha)$ and efficiently extract α for any values of g and α.

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

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

The security of the next problem relies on Assumption 1, in that if one can efficiently solve the discrete logarithm problem, then they can solve Problem 2 below. This assumption is necessary because a Diffie-Hellman key exchange occurs in CARROT to compute the private view key. This exchange happens exclusively on the Montgomery elliptic curve, Curve25519 [Ber06].

Problem 2 (Decisional Diffie-Hellman Problem)**.** The decisional Diffie-Hellman problem is:

$$\text{given input } \{(g,\, g^\alpha,\, g^\beta,\, g^\gamma) \,\big|\, (\mathbb{G},\, q,\, g) \leftarrow \mathsf{GGen}(\mathbb{1}^\lambda),\, \alpha, \beta, \gamma \xleftarrow{\$} \mathbb{F}_p\},\, \text{determine if } \gamma = \alpha\beta.$$

If this problem were to instead ask the adversary to compute $g^{\alpha\beta}$ given g^α and g^β, this would be the *computational* Diffie-Hellman problem. We elected to go with the decisional variant here, because it's the "harder" problem, in the sense that if an algorithm exists to solve the computational variant, then it can also solve the decisional variant.

This next assumption states that an adversary cannot determine if a given point is part of a Diffie-Hellman key exchange or if it's simply randomly selected, with any probability greater than by chance.

Assumption 2 (Diffie-Hellman Assumption)**.** We say that the decisional Diffie-Hellman assumption holds if for all probabilistic, polynomial-time adversaries $\mathcal{A}$, the following advantage is negligible:

$$\left|\Pr\Big[\mathcal{A}(g,\, g^\alpha,\, g^\beta,\, g^{\alpha\beta}) = 1 \,\Big|\, (\mathbb{G},\, q,\, g) \leftarrow \mathsf{GGen}(\mathbb{1}^\lambda),\, \alpha, \beta \xleftarrow{\$} \mathbb{F}_p\Big] - \Pr\Big[\mathcal{A}(g,\, g^\alpha,\, g^\beta,\, g^\gamma) = 1 \,\Big|\, (\mathbb{G},\, q,\, g) \leftarrow \mathsf{GGen}(\mathbb{1}^\lambda),\, \alpha, \beta, \gamma \xleftarrow{\$} \mathbb{F}_p\Big]\right| \leq \mathsf{negl}(\lambda)$$

For the purposes of CARROT, we require a hash function $\mathcal{H}$ to be *collision resistant*, meaning that it should be difficult for an adversary to find distinct messages m_1 and m_2 such that $\mathcal{H}(m_1) = \mathcal{H}(m_2)$.

Assumption 3 ((Strong) Collision Resistance)**.** We assume that finding hash collisions is hard; that given a message space $\mathcal{M} = \{0,\, 1\}^k$ of length k and a hash function $\mathcal{H} : \mathcal{M} \to \{0,\, 1\}^n$, the following advantage is bounded by some negligible function for all probabilistic, polynomial-time adversaries $\mathcal{A}$:

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

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

Assumption 4 ((First) Preimage Resistance)**.** We assume that finding a hash's preimage is hard; that given a message space $\mathcal{M} = \{0,\, 1\}^k$ and a hash function $\mathcal{H} : \mathcal{M} \to \{0,\, 1\}^n$, the following advantage is bounded by some negligible function for all probabilistic, polynomial-time adversaries $\mathcal{A}$:

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

We also recall properties of Pedersen commitments.

Formally, a commitment scheme is defines as two parts: the first is an efficient randomized setup algorithm CGen 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 $\mathsf{CGen}(\mathbb{1}^\lambda)$, where λ is some security parameter. The commitment function $\mathsf{Com} : \mathcal{M} \times \mathcal{R} \to \mathcal{C}$ 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 $c := \mathsf{Com}(m,\, r)$.

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

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

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 (CGen, Com) is *computationally binding* if the probability of finding $(m_1, r_1) \neq (m_2, r_2)$ such that $\mathsf{Com}(m_1, r_1) = \mathsf{Com}(m_2, r_2)$ is very low. This is, for all polynomial-time adversaries $\mathcal{A}$:

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

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 log problem is only *computationally* 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. Pedersen commitments are used in many popular cryptocurrencies such as Monero [Noe15], Pinocchio coin [DFKP13], and Firo (then 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 $\mathsf{Com}(m, r) = mG + rH$, with $r \xleftarrow{\$} \mathcal{R}$ and $m \in \mathcal{M}$, where G and H are a generators of group $\mathbb{G}$ specified in CGen. The above commitment is in additive notation, which is what we use throughout this document, but other authors prefer multiplicative notation, which takes the form $\mathsf{Com}(m, r) = g^m h^r$. 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 we assume cannot be done efficiently. Finally, Pedersen commitments enjoy the homomorphic property, which states that $\mathsf{Com}(m_1, r_1) + \mathsf{Com}(m_2, r_2) = \mathsf{Com}(m_1 + m_2, r_1 + r_2)$, in additive notation.

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

For this document, we presume that the discrete log problem is hard on the two curves implemented in CARROT, the Montgomery curve [Ber06] and the twisted Edwards curve [BDL$^+$11], meaning that the adversarial advantage in Assumption 1 is negligible for all probabilistic polynomial-time adversaries. We also take it as fact that Assumption 2, the Diffie-Hellman assumption, holds for both curves, meaning that an adversary cannot in general determine if a value is a Diffie-Hellman key or simply random. Additionally, we suppose that the hash functions used here, Blake2b [ANWOW13] and Keccak [BDH$^+$], enjoy collision resistance from Assumption 3 and preimage resistance from Assumption 4. We also take it as a given that all Pedersen commitments, which in CARROT occur over the twisted Edwards curve [BDL$^+$11], are hiding and binding, from Definition 2 and Definition 3 respectively.

These next definitions are presented in full abstraction, but make the most sense when applied later on in Section 3 of this document. We present them here for completeness.

Definition 4. Let $\mathcal{S}$ 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

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

In Section 3, we take $\mathcal{S}$ to be a simulator algorithm which simulates the transcript of a valid CARROT interaction, and Δ to be a randomized algorithm which selects parameters uniformly at random from their respective sample spaces. This permits us to make claims such as "the ephemeral public key of a CARROT interaction is indistinguishable from a random point on the Montgomery curve," as the probability ensembles do not permit an adversary to distinguish with any probability greater than sheer chance.

Definition 5. We say that an encryption protocol obtains *forward secrecy* if inspection of the key agreement transcript in plaintext does not permit decryption of the remainder of the data.

Less formally, this states that if a single old key is compromised, then the remainder of the newer keys which were derived from it remain protected [NIS]. The purpose of this definition becomes more clear in Section 3, where it enables us to make claims like "even if certain keys are compromised, an attacker cannot break the CARROT protocol."

2 Recommended Action

In this section, we outline some comments and suggestions for the CARROT spec document. In no way are these intended to be mandatory changes, we simply outline improvements that we believe can increase readability, exactness, and operability.

1. In Section 7.3.5 the anchor is set to an "HMAC" of the ephemeral key $\mathsf{D_e}$ in special enotes, which can then be authenticated with private view key $\mathsf{k_v}$. We do not find that this fits the technical definition of an HMAC (see FIPS 198-1, `https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.198-1.pdf`). The $\mathsf{anchor_{sp}}$ is defined in Section 7.4.1 as $\mathsf{SecretDerive}(\mathsf{D_e} \,||\, \mathsf{input\_context} \,||\, \mathsf{K_o} \,||\, \mathsf{k_v} \,||\, \mathsf{K_s})$, which does not strictly fit the criteria for an HMAC as defined below.

   An HMAC of message m with secret key k as defined in a 1997 RFC by Bellare, Canetti, and Krawczyk (`https://www.rfc-editor.org/rfc/pdfrfc/rfc2104.txt.pdf`) looks like

$$\mathsf{HMAC}(\mathsf{m},\, \mathsf{k}) = \mathsf{H}\Big(\mathsf{h} \oplus \mathsf{out\_pad} \;\; || \;\; \mathsf{H}\big(\mathsf{h} \oplus \mathsf{in\_pad} \;\; || \;\; \mathsf{m}\big)\Big)$$

$$\mathsf{h} = \begin{cases} \mathsf{H(k)} & \text{if k is longer than block size} \\ \mathsf{k} & \text{else} \end{cases}$$

   where in_pad and out_pad are padding of repeated values.

   So, we find that $\mathsf{anchor_{sp}}$ is a MAC with secret view key $\mathsf{k_v}$ and not an HMAC, though could be redefined into one at the cost of an additional hash. We do not expect this will confer major security benefits, and would likely slow down the protocol. We point out that common argument for the use of HMACs is protection against length extension attacks - Blake2b [ANWOW13] and Keccak [BDH+] enjoy this protection for free, so ScalarDerive and its legacy version are not susceptible to length-extension attacks.

2. In Section 7.4.1 the padding for $\mathsf{m_{anchor}}$ should read "jamtis_encryption_mask_anchor" in our opinion, not "jamtis_encryption_mask_j," as j is the subaddress index, which the anchor does not rely on. There is a typo in Section 1.3, where one paragraph claims "unconditional forward secrecy" and the other claims "conditional forward secrecy."

3. In Section 7.4.2 the view tag vt is defined as the hash of $\mathsf{s_{sr}}$, input_context, and $\mathsf{K_o}$. Yet in Section 7.3.4 it's mentioned that for efficient scanning purposes, vt is truncated - only the first few bytes are communicated. This should be made clear in definition.

4. In Section 8.1, readers benefit from explicitly labeling data, so it is clear which information is public or private, known only to the sender or the receiver, etc. While this would assist with readability of the protocol, it would also clarify what information is only available to which party, which would allow an interested reader to work though the protocol themselves.

5. In Section 8.1, line 31 - 33 are repeats of 27 - 29, so we suggest combining them into a single if-else statement. While this is not strictly necessary, but it may improve readability of the scanning process.

6. We would welcome a paragraph in the intro explaining what hard problems the security of this protocol relies on, what properties we assume our hash functions have, what security factor we take in practice, etc. What about computational vs unconditional vs perfect hiding, and similarly for binding? What capabilities does our idealized adversary have? At practical parameters, what is the expected runtime for generic algorithms like Pollard's rho [Pol10] or the general number field sieve [LL93]?

7. We recommend more be said about clamped keys, for instance by formalizing Section 9.4.2, since clamped keys are common for X25519 and Ed25519. Clamped keys are connected to non-constant-time implementation and guarding against small subgroup attacks. It should also be pointed out that without clamping, keys can only be referred to as indistinguishable from other keys, instead of from random values. This can be dealt with by using Elligator [BHKL13, Vai] to hide curve points, though that is definitively a future work.

8. How exactly is the master secret $\mathsf{s_{sm}}$ derived, and what is its length? We assume it's randomly selected from $\mathbb{F}_\ell$, but this is not directly stated in Section 5.2. Without a recommended length, it may admit attacks aimed at determining the value of short $\mathsf{s_{sm}}$, say from side-channel or fault-injection attacks. If possible (it might not be, due to compatibility constraints) we would recommend adding salt to keys in the novel key hierarchy. As it stands, subordinate keys like $\mathsf{k_{ps}}$ are computed simply as a hash of the master key $\mathsf{s_{sm}}$ along with padding, so it's perhaps possible for these to be put to malicious use - such as if the master key is imprudently reused, or if two users have short $\mathsf{s_{vb}}$ keys that match by chance. To this end, it may be helpful to consider formally using a key derivation function `https://en.wikipedia.org/wiki/Key_derivation_function` for the subsequent keys. Though endorsing a standard way to generate $\mathsf{s_{sm}}$ and selecting a key derivation function are likely unenforceable, it would be a best-practice recommendation.

3 Security Proofs

In this section, we recapitulate the security claims in the CARROT specification document and present arguments for those claims. After reviewing the security properties in Section 9, we propose some slight alterations to a few statements, which we believe strengthens or makes more precise their assertions. Below each number and title in bold, the indented paragraphs are what we aim to show, with justification immediately following.

3.1 Balance Recovery Security

3.1.1 Completeness

> An honest sender sending amount a and payment ID pid to the address $(\mathsf{K_s^j},\ \mathsf{K_v^j})$ may be assured that an honest receiver who derived the address will, without requiring additional information,
>
> 1. recover the same values a, pid, and $(\mathsf{K_s^j},\ \mathsf{K_v^j})$
> 2. recover proper $\mathsf{x},\ \mathsf{y},\ \mathsf{z}$ such that $\mathsf{K_o} = \mathsf{xG} + \mathsf{yT}$ and $\mathsf{C_a} = \mathsf{zG} + \mathsf{aH}$.

To address point 1. we note that in the honest sender - honest receiver model, the shared secret $\mathsf{s_{sr}}$ (and thus $\mathsf{s_{sr}^{ctx}}$) obtained from the ephemeral pubkey $\mathsf{D_e}$ will be accurate. From this, the amount mask $\mathsf{m_a}$ will be correct, giving that the receiver's nominal amount $\mathsf{a'}$ will match the true amount a. The only conceivable way that the receiver could recover a non-matching amount would be that if they or the sender have deviated from the protocol, or else an error has been introduced (in communication, elliptic curve calculation, etc).

In the same vein, the receiver's reconstruction of $\mathsf{pid'}$ will match the true pid, as its encryption mask $\mathsf{m_{pid}}$ derived from $\mathsf{s_{sr}^{ctx}}$ will be accurate. So the payment ID will be recovered correctly.

Because s_{sr}^{ctx} matches the genuine value, $k_a{}'$ will match k_a, giving that the nominal $C_a{}'$ matches the true C_a. Hence, both $k_g^{o\prime}$ and $k_t^{o\prime}$ will be authentic, meaning $K_s^{j}{}'$ will equal K_s^j. From this, $K_v^{j}{}'$ will equal K_v^j also, so the receiver will recover the same address.

As for point 2. since $K_s^{j}{}'$ has been reconstructed correctly, and $K_s^{j}{}' = K_o - k_g^{o\prime}G - k_t^{o\prime}T$, the receiver will recover proper $x,\ y$ such that $K_0 = xG + yT$.

We saw above that the the reconstructed values $k_a{}'$ and a' are true. Thus, since $C_a{}' = k_a{}'G + a'T$, the receiver will recover proper values for the amount commitment.

3.1.2 Spend Binding

> If an honest receiver recovers $x,\ y$ such that $K_0 = xG + yT$ for an enote, then no adversary without knowledge of k_{ps} (for new key hierarchy; k_s if on legacy key hierarchy) will be able to recover $x,\ y$ up to a security factor.

Since K_0 is a Pedersen commitment, we automatically get unconditional hiding (even a computationally unbounded adversary cannot efficiently recover $r_1,\ r_2$ such that $K_0 = r_1G + r_2T$). The "security factor" statement is probably related to discrete log problem, though we don't think it's necessary for Pedersen. Note that if this adversary knows the prove-spend key k_{ps} (resp. k_s) then this can be used to spend enotes, which necessitates recovering $x,\ y$ such that $K_0 = xG + yT$.

3.1.3 Enote Scan Binding

> No two honest receivers using different values of k_v (for external scans; s_{vb} for internal scans) can implement enote scan process for the same enote and return successfully, even if constructed dishonestly.

Suppose $\mathcal{R}_1$ uses k_v and $\mathcal{R}_2$ uses $\mathbf{k_v}$, distinct. If performing an internal scan, then the scan processes will diverge on Section 8.1, line 26, where $K_v^{j}{}' = k_v K_{base}$ and $\mathbf{K_v^j}{}' = \mathbf{k_v} K_{base}$. As d_e' relies on the hash of $K_v^{j}{}'$, we obtain two very different values for d_e' and $\mathbf{d_e'}$. If this discrepancy results in both $\mathcal{R}_1$ and $\mathcal{R}_2$ passing the test $D_e' == D_e$ on line 29, then it must be that $D_e' = \mathbf{D_e'}$, implying that $d_e' = \mathbf{d_e'}$; this violates collision resistance.

If the scan is internal, then by Section 7.6 s_{sr} and s_{vb} are equal. Suppose $\mathcal{R}_1$ uses s_{sr} and $\mathcal{R}_2$ uses $\mathbf{s_{sr}}$, distinct. Then Section 8.1, line 2 gives vt' as the hash of s_{sr}, so will have a very different value from $\mathbf{vt'}$. If both pass the test on line 3, then $vt' = \mathbf{vt'}$ and we have a hash collision.

3.1.4 Burning Bug Resistance

> For any K_o, it is computationally intractable to find distinct input_context and **input_context** such that an honest receiver views both as spendable.

For the scan process, two distinct values for input_context and **input_context** will lead to distinct vt' and $\mathbf{vt'}$. Then Section 8.1, line 3 will return ABORT, and can only continue the scan process if a hash collision has been found.

To spend, a receiver must know $z,\ a$ such that $C_a = zG + aH$. Yet, Section 8.1, line 7-9 mean that s_{sr}^{ctx} will rederive m_a (and thus a') and $k_a{}'$. This will then be used to generate $C_a{}'$, which will only avoid the ABORT on line 15 if $C_a = C_a{}'$. Since the same holds for $\mathbf{s_{sr}^{ctx}}$ after rederiving $\mathbf{a'}$ and $\mathbf{k_a}{}'$, and hence $\mathbf{C_a}{}'$, both can only pass the test if $C_a{}' = \mathbf{C_a}{}'$. Thus either a hash collision has occurred, or a relationship between linearly independent generators has been found - in contradiction to the hardness of the discrete log problem. Thus, users can be assured that incoming transactions cannot "burn" their funds.

3.1.5 Janus (or Pordo) Attack Resistance

There exists no algorithm $\mathcal{A}$ without knowledge of the receiver's private view key $\mathsf{k_v}$ that permits a sender to construct an enote with two honestly-derived, non-integrated addresses satisfying the following properties:

1. the enote successfully passes the scan process when the addresses were derived from the same account
2. the enote fails the scan process when the addresses are unrelated.

ie: it is computationally intractable to construct an *external* enote which passes the scan such that the Recipient's reconstructed address spend pubkey $\mathsf{K_s^j}' = \mathsf{K_o} - \mathsf{k_g^o G} - \mathsf{k_t^o T}$ doesn't match the shared secret $\mathsf{s_{sr}} = 8\ \mathsf{r}\ \mathsf{ConvertPointE}(\mathsf{K_s^j})$ for some sender-chosen r.

Assume the addresses were derived from the same account, and suppose the sender selects $\mathsf{r} \in \mathbb{F}_p$ at random, then forms $\mathbf{s_{sr}} = 8\ \mathsf{r}\ \mathsf{ConvertPointE}(\mathsf{K_s^j})$. Here, r serves as a replacement for the enote's ephemeral private key $\mathsf{d_e}$, which we should view as uniformly random anyways. (External enotes use Diffie-Hellman solely for incoming private view key $\mathsf{k_v}$, which the receiver then uses with $\mathsf{d_e}'$ to nominally construct ephemeral pubkey $\mathsf{D_e}'$.) This is used to construct $\mathbf{s_{sr}^{ctx}}$ on Section 8.1, line 4, which is used to create $\mathbf{K_s^j}'$, then hashed with $\mathsf{K_o}$ to form $\mathbf{m_{anchor}}$, and this is XORed to nominally reconstruct $\mathbf{anchor}'$. Since the ephemeral private key $\mathsf{d_e}$ is a hash digest of values including anchor', the nominal rederivation $\mathbf{d_e}'$ will be very different from $\mathsf{d_e}$.

The attempted reconstruction of the ephemeral public key will then be $\mathbf{D_e}' = \mathbf{d_e}'\ \mathsf{ConvertPointE}(\mathbf{K_s^j}')$. Note that $\mathbf{D_e}' \neq \mathsf{D_e}$, unless either a hash collision or a discrete log relation has been found, so Section 8.1, line 34 results in $\mathbf{anchor_{sp}}$ being distinct from $\mathbf{anchor}'$, thus returning ABORT. Hence, users are protected from the duality of the Janus attack.

As an aside, we point out that along the way we find the check:

$$\mathsf{SecretDerive(x)} = \mathbf{anchor_{sp}} \stackrel{?}{=} \mathbf{anchor}' = \mathsf{anchor_{enc}} \oplus \mathbf{m_{anchor}} = \mathsf{anchor_{enc}} \oplus \mathsf{SecretDerive(y)},$$

for inputs x and y, which can be rewritten as

$$\mathsf{SecretDerive(x)} \oplus \mathsf{SecretDerive(y)} \stackrel{?}{=} \mathsf{anchor_{enc}} \quad \leftarrow \text{ a given value.}$$

This looks *almost* like a hash collision, but is not exactly. See Appendix A of the Micciancio paper [BM97] for an example breaking the injectivity of the hash function $H(x_1 \,||\, x_2 \,||\, ... \,||\, x_n) := h(x_1) \oplus h(x_2) \oplus ... \oplus h(x_n)$. Also see [hoy] for a use-case.

Let us consider an idealized binary hash function $h : \{0,1\}^r \to \{0,1\}^b$, then build up $\mathcal{H} : \{0,1\}^{nr} \to \{0,1\}^b$ by $\mathcal{H}(x_1 \,||\, x_2 \,||\, ... \,||\, x_n) := h(x_1) \oplus h(x_2) \oplus ... \oplus h(x_n)$. Suppose that we've collected $m = 2^{b/n}$ hashes, where n is the number of hashes being XORed together and b is the hash's output length. Since the hash's range has cardinality 2^b, but we've accrued m^n sums of the form $h(x_1) \oplus ... \oplus h(x_n)$, we can guarantee a collision of the form $\mathcal{H}(x_1' \,||\, ... \,||\, x_n') = \mathcal{H}(x_1 \,||\, ... \,||\, x_n)$. Moreover, we've collected a basis for the vector space $\{0,1\}^b$, so can "solve" arbitrary hashes just by taking linear combinations of the $h(x_i)$'s. For the case $n = 2$, this is simply the birthday paradox [STKT06], and Wagner's algorithm [Wag02] presents a sub-exponential algorithm solving for any n. This argument needs better formalizing, but this back-of-the-notebook calculation demonstrates that an adversary collecting enough hashes could potentially be problematic for the CARROT protocol.

3.2 Unlinkability

3.2.1 Computational Address-Address Unlinkability

Any third party can't determine if two non-integrated addresses share the same $\mathsf{k_v}$ better than simple guessing.

With integrated addresses, the payment IDs pid and $\mathbf{pid}$ won't match up, so this is readily seen. If non-integrated, then the view key $\mathsf{k_v}$ is hashed to create the address $(\mathsf{K_s^j},\ \mathsf{K_v^j})$, so if any third party can identify this to another address $(\mathbf{K_s^j},\ \mathbf{K_v^j})$ with non-negligible probability, either they can find hash collisions or have broken the discrete log problem.

3.2.2 Computational Address-Enote Unlinkability

> Any third party can't determine if an address is the destination of a chosen enote better than simple guessing, even if they know the enote's destination address.

An enote is sent to the address $(\mathsf{K_s^j},\ \mathsf{K_v^j})$, but contains the output pubkey $\mathsf{K_o}$ - full knowledge of the opening allows for spending, and partial knowledge permits uncovering the key image which reveals where the enote was spent. As $\mathsf{K_o} = \mathsf{K_s^j} + \mathsf{k_g^o G} + \mathsf{k_t^o T}$, if any third party is able to link $\mathsf{K_o}$ and $\mathsf{K_s^j}$, then ostensibly they can open the Pedersen commitment and recover $\mathsf{x},\ \mathsf{y}$ such that $\mathsf{K_o} - \mathsf{K_s^j} = \mathsf{xG} + \mathsf{yT}$. Thus, either they've broken the commitment's unconditional hiding or broken the discrete log problem.

3.2.3 Computational Enote-Enote Unlinkability

> Any third party can't determine if two enotes share the same address better than simple guessing, even if they know the destination address.

Suppose two enotes are sent to the same address, but have different master keys, $\mathsf{s_{sm}}$ and $\mathbf{s_{sm}}$. Unless a hash collision has been found, this will result in distinct values for $\mathsf{k_{sub\_ext}^j}$ and $\mathbf{k_{sub\_ext}^j}$. This means that $\mathsf{K_s^j}$ and $\mathbf{K_s^j}$ will not be the same, except in violation of the discrete log problem. As $\mathsf{k_v}$ is a hash digest of $\mathsf{s_{vb}}$, and ultimately the master key $\mathsf{s_{sm}}$, the two users will have $\mathsf{k_v} \neq \mathbf{k_v}$, resulting in distinct $\mathsf{K_v^j}$ and $\mathbf{K_v^j}$. By working backwards, if a third party is able to connect these two addresses, they must either violate the hardness of the discrete log problem or be able to compute hash collisions.

3.2.4 Computational Enote-Key Image Unlinkability

> Any third party can't determine if a given key image is *the* key image for an enote better than simple guessing, even if they know the destination address.

Recall that the key image is computed as $\mathsf{L} = (\mathsf{k_{gi}} \cdot \mathsf{k_{sub\_scal}^j} + \mathsf{k_g^o})\ \mathsf{H_p^2(K_o)}$ where $\mathsf{k_{sub\_scal}^j} = 1$ if $\mathsf{j} = 0$ and $\mathsf{ScalarDerive}(\mathsf{s_{gen}^j} \,||\, \mathsf{K_s} \,||\, \mathsf{K_v} \,||\, \mathsf{j})$ else. Here, $\mathsf{H_p^2}(\cdot)$ is *not* the same hash function used in $\mathsf{SecretDerive}$, rather it is the second hash-to-point function on the twisted Edwards curve Ed25519.

Suppose an adversary can view the public data of an enote ($\mathsf{K_o}$, vt, $\mathsf{C_a}$, etc.), then determine correctly if said enote is linked to key image $\mathsf{L} = \mathsf{x}\ \mathsf{H_p^2(K_o)}$ for $\mathsf{K_o} = \mathsf{xG} + \mathsf{yT}$. Then from the information available, they must be able to break the discrete log problem or find a hash-to-point collision, neither of which we assume can be done with any degree of efficiency.

3.3 Forward Security

3.3.1 Address-Conditional Forward Secrecy

> Without knowing the receiver's public address, an adversary armed with the extractor algorithm $\mathcal{E}$ from Definition 1 still cannot determine the receiver, the amount sent, or the destination address.

Because

$$\mathsf{K_o} = \mathsf{K_s^j} + \mathsf{k_g^o G} + \mathsf{k_t^o T} = \mathsf{K_s^j} + \mathsf{ScalarDerive}(\mathsf{s_{sr}^{ctx}} \,||\, \mathsf{C_a})\mathsf{G} + \mathsf{ScalarDerive}(\mathsf{s_{sr}^{ctx}} \,||\, \mathsf{C_a})\mathsf{T},$$

(where recall we suppress the hash padding) we can see that $\mathsf{C_a}$ is protected by hash collisions even if the adversary knows the address. Even if uncovered, the amount commitment $\mathsf{C_a}$ is masked by blinding factor $\mathsf{k_a}$ (excepting in Coinbase transactions).

The destination address $(\mathsf{K_s^j},\ \mathsf{K_v^j})$ is also safe, because of collision resistance.

3.3.2 Internal Forward Secrecy

> Even knowing $\mathsf{s_{ga}},\ \mathsf{k_{ps}},\ \mathsf{k_{gi}},\ \mathsf{k_v}$, an adversary armed with extractor algorithm $\mathcal{E}$ from Definition 1 - but without knowledge of $\mathsf{s_{vb}}$ - cannot recover internal enote addresses, where or if they have been spent, or the transaction amounts any better than simple guesswork.

We point out that the prove-spend key $\mathsf{k_{ps}}$ and view-balance secret $\mathsf{s_{vb}}$ are defined as a hash digest of the master secret $\mathsf{s_m}$, while $\mathsf{s_{ga}},\ \mathsf{k_{gi}},$ and $\mathsf{k_v}$ are defined as hash digests of $\mathsf{s_{vb}}$. If an adversary is armed with the view-all tiered $\mathsf{s_{vb}}$ (which is equal to $\mathsf{s_{sr}}$ for internal enotes), then they can find and decode incoming and outgoing enotes, so we assume that they do not know the view-balance secret. In this case, for an internal enote, then an adversary cannot reform a valid $\mathsf{vt'}$ that passes the check at Section 8.1, line 3 without finding a hash collision.

Suppose that an adversary has knowledge of $\mathsf{k_{ps}},\ \mathsf{s_{ga}},\ \mathsf{k_{gi}},$ and $\mathsf{k_v}$.

We start by considering the enote's amount. For an internal enote, $\mathsf{s_{vb}}$ is used to derive $\mathsf{s_{sr}^{ctx}}$, so without the view-balance secret an adversary will have an incorrect value for their $\mathbf{s_{sr}^{ctx}}$, unless a hash collision has been found. Thus, $\mathbf{m_a}$ will be reconstructed incorrectly, thus the nominal amount $\mathbf{a'}$ will not match the true amount, unless a hash collision has occurred.

As for an internal enote's address, the incorrect value of $\mathbf{s_{sr}^{ctx}}$ will result in $\mathbf{k_g^{o\prime}}$ and $\mathbf{k_t^{o\prime}}$ being different from the true values, thus $\mathbf{K_s^{j\prime}}$ in Section 8.1, line 18 will be incorrect, unless hash collisions have been found.

To determine where or if it's been spent, one would inspect the key image. By the above, the enote will not appear to an adversary to be spendable. Since the key image is calculated as $\mathsf{L} = (\mathsf{k_{gi}} \cdot \mathsf{k_{sub\_scal}^j} + \mathsf{k_g^o})\,\mathsf{H_p^2}(\mathsf{K_o})$, but the nominal $\mathbf{k_g^{o\prime}}$ differs from the sender's value, the enote will not appear to have been spent.

3.4 Indistinguishability

3.4.1 Transaction output random indistinguishability

> Sample $\mathsf{r_1},\ \mathsf{r_2} \xleftarrow{\$} [0,\ \ell)$ then form $\mathbf{K_o} = \mathsf{r_1 G}$ and $\mathbf{C_a} = \mathsf{r_2 G}$, where G is a generator of $\mathbb{G}_\mathsf{E}$ and $\ell \approx 2^{252}$. Then $\mathbf{K_o}$ and $\mathbf{C_a}$ are indistinguishable from the $\mathsf{K_o}$ and $\mathsf{C_a}$ obtained by a genuine interaction.

We believe the proper formulation of this condition is: "sample $\mathsf{r_1},\ \mathsf{r_2},\ \mathsf{s_1},\ \mathsf{s_2} \xleftarrow{\$} [0,\ \ell)$ then form $\mathbf{K_o} = \mathsf{r_1 G} + \mathsf{s_1 T}$ and $\mathbf{C_a} = \mathsf{r_2 G} + \mathsf{s_2 H}$." This is an important distinction, in our opinion, because Pedersen commitments *do* appear random, especially given that the genuine $\mathsf{K_o}$ and $\mathsf{C_a}$ have the form $\mathsf{xG} + \mathsf{yT}$ and $\mathsf{zG} + \mathsf{aH}$ (for non-Coinbase transactions) where $\mathsf{x},\ \mathsf{y},\ \mathsf{z}$ are obtained as hashes - so will appear uniformly at random (up to some hash bias, which we assume is negligible). If an adversary can distinguish between these, they can either break the discrete log problem or find a hash collision.

3.4.2 Ephemeral pubkey random indistinguishability

> Sample $\mathsf{r} \xleftarrow{\$} [0,\ \ell)$ then form $\mathbf{D_e} = \mathsf{rB}$ where B is the generator of the Montgomery curve Curve25519 corresponding to $x = 9$. Then this is indistinguishable from a genuine ephemeral pubkey $\mathsf{D_e}$.

Recall that $\mathsf{D_e}$ will be derived differently depending on if the enote is normal to the main address, normal to a subaddress, internal, or special (external, self-send in a 2-out transaction).

Normal, to main address	$\mathsf{d_e B}$
Normal, to subaddress	$\mathsf{d_e ConvertPointE}(\mathsf{K_s^j})$
Internal	$\xleftarrow{\$} \mathbb{G}_\mathsf{M}$
Special, with other enote normal	$\mathsf{D_e^{other}}$, the other enote's ephemeral pubkey
Special, with other enote also special	$\xleftarrow{\$} \mathbb{G}_\mathsf{M}$

One can immediately see that two of these directly sample $\mathsf{D_e}$ randomly, so will be indistinguishable. An additional two use $\mathsf{d_e}$, the ephemeral privkey, which is a hash digest, so appear random. The last one left is $\mathsf{D_e^{other}}$, which - being the ephemeral public key of another enote - will fall under one of the previous possibilities, thus appear random.

We would welcome a more formalized description about the effects key clamping, or lack thereof, has on this protocol.

3.4.3 Other enote component random indistinguishability

The following Carrot components will be indistinguishable from their counterparts obtained via a genuine interaction:

$$\begin{aligned} \mathsf{a_{enc}} &= \mathsf{RandBytes}(8) \\ \mathsf{anchor_{enc}} &= \mathsf{RandBytes}(16) \\ \mathsf{vt} &= \mathsf{RandBytes}(3) \\ \mathsf{pid_{enc}} &= \mathsf{RandBytes}(8). \end{aligned}$$

Because vt is obtained as a hash digest, we may conclude that it will appear uniformly at random. The other three values here are obtained via masking the true values by XORing with a hash digest. ie: $\mathsf{a_{enc}} = \mathsf{a} \oplus \mathsf{SecretDerive}(\mathsf{s_{sr}^{ctx}} \,||\, \mathsf{K_o})$. This masking with an apparently random value means that the values themselves will appear random, thus all of the genuine values above will be indistinguishable from those acquired through $\mathsf{RandBytes}(\cdot)$.

References

[ANWOW13] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. BLAKE2: simpler, smaller, fast as MD5. Cryptology ePrint Archive, Paper 2013/322, 2013. `https://eprint.iacr.org/2013/322`.

[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+15] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Essam Ghadafi, Jens Groth, and Christophe Petit. Short accountable ring signatures based on DDH. Cryptology ePrint Archive, Paper 2015/643, 2015. `https://eprint.iacr.org/2015/643`.

[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.

[BDH+] Guido Bertoni, Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Keccak. https://keccak.team/keccak.html. Accessed: 2024-11-1.

[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.

[BHKL13] Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. Elligator: Elliptic-curve points indistinguishable from uniform random strings. Cryptology ePrint Archive, Paper 2013/325, 2013. https://eprint.iacr.org/2013/325.

[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.

[BM97] Mihir Bellare and Daniele Micciancio. A new paradigm for collision-free hashing: Incrementality at reduced cost. Cryptology ePrint Archive, Paper 1997/001, 1997. https://eprint.iacr.org/1997/001.

[CHAK22] Matteo Campanelli, Mathias Hall-Andersen, and Simon Holmgaard Kamp. Curve trees: Practical and transparent zero-knowledge accumulators. Cryptology ePrint Archive, Paper 2022/756, 2022. https://eprint.iacr.org/2022/756.

[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.

[COPZ22] Melissa Chase, Michele Orrù, Trevor Perrin, and Greg Zaverucha. Proofs of discrete logarithm equality across groups. Cryptology ePrint Archive, Paper 2022/1593, 2022. https://eprint.iacr.org/2022/1593.

[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.

[Eag22] Liam Eagen. Zero knowledge proofs of elliptic curve inner products from principal divisors and weil reciprocity. Cryptology ePrint Archive, Paper 2022/596, 2022. https://eprint.iacr.org/2022/596.

[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.

[GK14] Jens Groth and Markulf Kohlweiss. One-out-of-many proofs: Or how to leak a secret and spend a coin. Cryptology ePrint Archive, Paper 2014/764, 2014. https://eprint.iacr.org/2014/764.

[GNB19] Brandon Goodell, Sarang Noether, and Arthur Blue. Concise linkable ring signatures and forgery against adversarial keys. Cryptology ePrint Archive, Paper 2019/654, 2019. https://eprint.iacr.org/2019/654.

[hoy] hoytech. XOR Hash Collision Generator. https://github.com/hoytech/xor-hash-collisions. Accessed: 2024-11-15.

[jef] jeffro256. CARROT. https://github.com/jeffro256/carrot/blob/master/carrot.md. Accessed: 2024-10-18.

[JF21a] 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.

[JF21b] 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.

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

[Jus] Justin "knaccc" Ehrenhofer. Advisory note for users making use of subaddresses. https://web.getmonero.org/2019/10/18/subaddress-janus.html. Accessed: 2024-10-23.

[kAN] koe, Kurt M. Alonso, and Sarang Noether. Zero to Monero. https://www.getmonero.org/library/Zero-to-Monero-2-0-0.pdf. Accessed: 2024-10-18.

[kay] kayabaNerve. 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.

[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.

[koea] koe. Seraphis: A Privacy-Preserving Transaction Protocol Abstraction. https://raw.githubusercontent.com/UkoeHB/Seraphis/master/seraphis/Seraphis-0-0-18.pdf. Accessed: 2024-10-18.

[koeb] koe. What is Seraphis, and Why Should You Care? https://www.getmonero.org/2021/12/22/what-is-seraphis.html. Accessed: 2024-10-18.

[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.

[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.

[Mona] Monero Community Workgroup. Breaking Monero Episode 09: Poisoned Outputs (EAE Attack). https://www.youtube.com/watch?v=iABIcsDJKyM&list=PLsSYUeVwrHBnAUre2G_LYDsdo-tD0ov-y&index=10. Accessed: 2024-10-23.

[Monb] Moneropedia. CLSAG. https://www.getmonero.org/resources/moneropedia/clsag.html. Accessed: 2024-10-23.

[NG20] Sarang Noether and Brandon Goodell. Triptych: ogarithmic-sized linkable ring signatures with applications. Cryptology ePrint Archive, Paper 2020/018, 2020. https://eprint.iacr.org/2020/018.

[NIS] NIST. Perfect Forward Secrecy (PFS). https://csrc.nist.gov/glossary/term/perfect_forward_secrecy. Accessed: 2024-11-15.

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

[Para] Luke "Kayaba" Parker. Full-Chain Membership Proofs + Spend Authorization + Linkability Development CCS. https://ccs.getmonero.org/proposals/fcmp++-development.html. Accessed: 2024-11-12.

[Parb] Luke "Kayaba" Parker. Full-Chain Membership Proofs Development. https://www.getmonero.org/2024/04/27/fcmps.html. Accessed: 2024-10-18.

[Parc] Luke "Kayaba" Parker. Full chain membership proofs: Solving one of monero's last privacy hurdles. https://www.youtube.com/watch?v=vrCAiLPfXlg. Accessed: 2024-11-12.

[Pard] Luke "Kayaba" Parker. Retroactive funding for work on full-chain membership proofs. https://ccs.getmonero.org/proposals/kayabaNerve-fcmp-retroactive.html. Accessed: 2024-11-12.

[Par24] Luke "Kayaba" Parker. FCMP++, 2024. https://moneroresearch.info/index.php?action=attachments_ATTACHMENTS_CORE&method=downloadAttachment&id=220&resourceId=227&filename=95effbc6d6ad425edb788df9480e0dc64d4d3713.

[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.

[Ric] Riccardo "fluffypony" Spagni. DoS/RPC fixes. https://github.com/monero-project/monero/pull/4438/files. Accessed: 2024-10-23.

[Sar] Sarang Noether. Janus mitigation. https://github.com/monero-project/research-lab/issues/62. Accessed: 2024-10-23.

[STKT06] Kazuhiro Suzuki, Dongvu Tonien, Kaoru Kurosawa, and Koji Toyota. Birthday paradox for multi-collisions. In Min Surp Rhee and Byoungcheon Lee, editors, *Information Security and Cryptology – ICISC 2006*, pages 29–40, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. https://doi.org/10.1007/11927587_5.

[tev] tevador. JAMTIS. https://gist.github.com/tevador/50160d160d24cfc6c52ae02eb3d17024#1-introduction. Accessed: 2024-10-18.

[Vai] Loup Vaillant. Elligator: Hiding cryptographic key exchange as random noise. https://elligator.org/. Accessed: 2024-10-30.

[vS] Nicolas van Saberhagen. CryptoNote v2.0. https://bytecoin.org/old/whitepaper.pdf. Accessed: 2024-10-18.

[Wag02] David Wagner. A generalized birthday problem. In *Annual International Cryptology Conference*, pages 288–304. Springer, 2002. https://iacr.org/archive/crypto2002/24420288/24420288.pdf.

[Wor] Monero Community Workgroup. Breaking Monero Episode 14: Subaddress Association (Janus Attack). https://www.youtube.com/watch?v=M_IYzzC5Zqk&list=PLsSYUeVwrHBnAUre2G_LYDsdo-tD0ov-y&index=14. Accessed: 2024-10-23.

[Yap] Reuben Yap. Lelantus Spark is live on Firo. https://firo.org/2024/01/18/spark-is-live.html. Accessed: 2024-10-18.