CARROT Audit Follow-Up

Brandon Goodell, Rigo Salazar, Freeman Slaughter, Luke Szramowski

Cypher Stack

September 10, 2025

Change log

- August 28, 2025: Initial preprint.
- September 10, 2025: Added Addendum section.
- September 24, 2025: Improved clarity of Addendum and moved section to top.

0 Addendum

0.1 Key Exchange

Based on [jef25], the potential ramifications of clearing the cofactor in $\mathsf{s_{sr}}$, from the initial ECDH key exchange, are now clear. The example given in [jef25] has an enote creator set $\mathsf{D_e} = O$, where O is the point at infinity for the Montgomery curve [Ber06]. We generalize this scenario to the point where $\mathsf{D_e}$ is specifically constrained in a subgroup of order 8. In this case, $8\ \mathsf{D_e} = O$, which means that for $\mathsf{s_{sr}} = \mathsf{k_v}\ \mathsf{D_e}$ (in the Recipient, external case), where we are assuming that the hash-derived $\mathsf{k_v}$ is distributed uniformly at random, that we can expect 12.5% of the subsequent $\mathsf{s_{sr}}$ variables would end up being O. This outcome would lead to a vulnerability in the enote scanning process.

The rederivation of $\mathsf{D_e}$ in the *Janus* would not act as a suitable deterrent for computing $\mathsf{D_e}$ maliciously. In Section 8.1 of [jefa], lines 23 - 34 are relevant. Let us say that an adversary chose $\mathbf{D_e}$ to be in a subgroup of order 8. Step 27 calculates $\mathsf{D'_e} = \mathsf{d'_e}\ \mathsf{ConvertPointE}(\mathsf{K_{base}})$. Both of the possible values of $\mathsf{K_{base}}$ (Step 25) are elements of a subgroup with order ℓ, and because the $\mathsf{ConvertPointE}$ function preserves the group structure, the check at Step 28 (ensure $\mathsf{D'_e} = \mathbf{D_e}$) will fail. Then $\mathsf{pid'}$ is recalculated, which will change the value of $\mathsf{D'_e}$, but that won't change the fact that $\mathsf{D'_e}$ and $\mathbf{D_e}$ reside in different subgroups. Thus, Step 32 will fail because it's the same check as Step 28.

At this point, the final roadblock between the adversary and a successful enote scan is Step 34, which compares the two following variables (we are not assuming a normal anchor):

$$\begin{aligned} \mathsf{anchor_{sp}} &= \mathsf{SecretDerive}(\text{“Carrot janus anchor special”} \ \|\ \mathbf{D_e} \ \|\ \mathsf{input\_context} \ \|\ \mathsf{K_o} \ \|\ \mathsf{k_v})[:16] \\ \mathsf{anchor'} &= \mathsf{anchor_{enc}} \ \oplus\ \mathsf{m_{anchor}} = [\mathsf{anchor_{sp}} \ \oplus\ \mathsf{m_{anchor}}] \ \oplus\ \mathsf{m_{anchor}} = \mathsf{anchor_{sp}} \end{aligned}$$

In the derivation of $\mathsf{anchor'}$, the term $\mathsf{anchor_{enc}}$ has already been computed with $\mathbf{D_e}$ contained inside the hash, so the check at Step 34 will show that $\mathsf{anchor_{sp}} = \mathsf{anchor'}$ by construction! To mitigate this risk, we recommend verifying that $\mathsf{D_e}$ resides within a subgroup of order ℓ via a preprocessing check before Step 1. Additionally, we recommend performing the same check for all pertinent public values, which we assume can be done in an efficient manner. Because of the cascading dependencies in the CARROT protocol, we expect this can be achieved with negligible overhead. As an example, this check is already performed on $\mathsf{K}_\mathsf{s}^\mathsf{j}$ in Step 19, but it would suffice to check K_o before Step 1. Once K_o is verified as a group point in a valid subgroup, then $\mathsf{K}_\mathsf{s}^\mathsf{j}$ is a sum of sound group points, which implies that it itself must be a legitimate group point. We want to emphasize that it is insufficient to check that the points are **not** in a subgroup of order 8, rather it is imperative to verify that the points **are** in the subgroup of order ℓ. This is relevant, because there are other subgroups which can introduce vulnerabilities, of orders $1, 2, 4, 8, 2\ell, 4\ell$ and 8ℓ.

0.2 Linguistics

The exact terminology of $\mathsf{anchor}_\mathsf{sp}$ remains a low priority in the audit. Regardless, we would argue that some of the security properties of a MAC have not quite been justified in the CARROT documentation. Namely, per NIST, the only approved general-purpose MACs are HMAC algorithms, those utilizing Keccak, or those leveraging block ciphers `https://csrc.nist.gov/projects/message-authentication-codes`. We do not believe this meets the criteria for an HMAC, and it clearly does not utilize Keccak or block ciphers, so cannot be a MAC in the strictly defined sense. However, the label of $\mathsf{anchor}_\mathsf{sp}$ is ultimately the decision of the protocol author; we simply aim to prevent any misunderstandings regarding the properties of $\mathsf{anchor}_\mathsf{sp}$.

0.3 Dependency Diagram

To better understand the relationships between the variables in the CARROT protocol, we found it helpful to construct a dependency diagram. We include it in our communication, in the hopes that any potential implementers might benefit from its clarity.

1 Introduction

1.1 Executive Summary

In [Sta], we audited the framework of CARROT (Cryptonote Address on Rerandomizable-RingCT-Output Transactions), an addressing protocol to be used in the upcoming FCMP++ (Full Chain Membership Proof + Spend Authorization + Linkability) upgrade for Monero. Our audit was based on the specification document [jefa] that has since undergone slight alterations, summarized in [jefc]. In this document, we find that the "tweaks" result in no significant changes to CARROT or its security. The discussion contained in this document justifies this conclusion, as well as provides some measured feedback on potential drawbacks. Lastly, we discuss some points of interest in our prior audit.

1.2 Scope of Work

See `https://gist.github.com/jeffro256/12f4fcc001058dd1f3fd7e81d6476deb` for the follow-up documentation. The changes are in the CARROT repository from commit dbb04d91d40b68b2a8b82b895acf762c864b4cbc, where the scope of work is itemized in the beginning. We list them here:

1. The removal of the cofactor (8) clearing multiply for the initial X25519 ECDH key exchange: s_sr
2. Assignment change of the amount commitment blinding factor: k_a
3. Assignment change of the ephemeral private key: d_e
4. Assignment change of the special Janus anchor: $\mathsf{anchor}_\mathsf{sp}$
5. Addressing previous comments in [Sta], regarding the size of the CARROT view tag vt

1.3 Notation and Preliminary Definitions

To be straightforward, we will carry the same notation over from [Sta]. Additionally, all adversary capabilities found in Section 1.4.2 from [Sta] will be carried over, including all **Problems**, **Assumptions**, and **Definitions**

2 CARROT Changes and Subsequent Consequences

2.1 Cofactor Removal

The derivation of $\mathsf{s_{sr}}$ has been changed to remove the cofactor in the external case. Below is the original derivation and the updated derivation with an underbrace emphasizing the removed cofactor:

$$\mathsf{s_{sr}} = \underbrace{8}\ \mathsf{d_e}\ \mathsf{ConvertPointE}(\mathsf{K_v^j}) \rightarrow$$
$$\mathsf{s_{sr}} = \mathsf{d_e}\ \mathsf{ConvertPointE}(\mathsf{K_v^j})$$

This will not pose any complications. Although delegated to a future point in time, the check still occurs, so there should be no issue with small subgroup attacks. Since the computations made throughout the scanning process operate without the cofactor removal, all computations will occur normally and, given that the check passes, the result will retain the soundness guaranteed in the case where the cofactor is cleared.

As an aside, we recommend editing CARROT **Section 7.3.5**, which references $\mathsf{anchor_{sp}}$ as a deterrent for small subgroup attacks. We find that the bulk of the security actually comes from the aforementioned check, so any mention of subgroup attacks is unnecessary.

2.2 Amount Commitment Blinding Factor

Two additional variables have been added into the ScalarDerive function. Below is the original and the updated assignment with an underbrace emphasizing said variables:

$$\mathsf{k_a} = \mathsf{ScalarDerive}(\text{“Carrot Commitment mask”} \mid\mid \mathsf{s_{sr}^{ctx}} \mid\mid \mathsf{enote\_type}) \rightarrow$$
$$\mathsf{k_a} = \mathsf{ScalarDerive}(\text{“Carrot Commitment mask”} \mid\mid \mathsf{s_{sr}^{ctx}} \mid\mid \underbrace{\mathsf{a} \mid\mid \mathsf{K_s^j}} \mid\mid \mathsf{enote\_type})$$

where a is a plaintext integer amount, and

$$\mathsf{K_s^j} = \begin{cases} \mathsf{k^j_{sub\_scal}} \mathsf{K_s} & \text{(subaddress key)} \\ \mathsf{K_s} + \mathsf{k^j_{sub\_ext}} \mathsf{G} & \text{(legacy subaddress key)} \end{cases}$$

The purpose of $\mathsf{k_a}$ is to act as a blinder for the amount commitment $\mathsf{C_a}$ in the CARROT addressing protocol. It should be noted that in Section 4 of [jefa], $\mathsf{C_a}$ uses a different blinder, z, but this “overloading” of $\mathsf{C_a}$ is clear from context. For clarity, modifying **Section 4** of CARROT to reflect that the old blinder will eventually be replaced, would be welcome. As is traditional for a Pedersen commitment [Pro], the primary criteria for any blinding factor is that it is random within the appropriate field. Once that is assured, the Pedersen commitment relies on the hardness of the discrete logarithm problem which holds for the elliptic curves [BDL$^+$11, Ber06] that CARROT implements. Recall from [jefa], ScalarDerive is derived by deserializing a 512-bit little endian integer from the 64-bit output of a Blake2b hash function [ANWOW13] and then reducing it modulo ℓ, where ℓ is the prime order of the relevant subgroups.

Concatenating two more terms into the input of a hash function is functionally similar to adding “salt” to the hash. Therefore the output of the hash function is virtually indistinguishable from another output in the sense that they are both assumed to be random. If the hash function output is still random, then the deserialization will also be random. Since $\mathsf{k_a}$ is still random, then the amount commitment $\mathsf{C_a}$ will remain perfectly hiding as a result. The ultimate consensus is that this particular assignment change will have no effect on the CARROT protocol.

2.3 Ephemeral Private Key

The updated CARROT specification now withholds a variable that was once added into the ScalarDerive function for the derivation of $\mathsf{d_e}$. Below is the original assignment and the updated assignment with an underbrace emphasizing the removed variable:

$$\mathsf{d_e} = \mathsf{ScalarDerive}(\text{“Carrot sending key normal”} \mathbin{||} \mathsf{anchor_{norm}} \mathbin{||} \mathsf{input\_context} \mathbin{||} \mathsf{K_s^j} \mathbin{||} \underbrace{\mathsf{K_v^j}} \mathbin{||} \mathsf{pid}) \rightarrow$$

$$\mathsf{d_e} = \mathsf{ScalarDerive}(\text{“Carrot sending key normal”} \mathbin{||} \mathsf{anchor_{norm}} \mathbin{||} \mathsf{input\_context} \mathbin{||} \mathsf{K_s^j} \mathbin{||} \mathsf{pid})$$

where

$$\mathsf{K_v^j} = \begin{cases} \mathsf{k^j_{sub\_scal} K_v} & \text{(subaddress key)} \\ \mathsf{k_v K_s^j} & \text{(legacy subaddress key)} \end{cases}$$

A naive view on this change would be to look at Section 2.2 and quickly conclude that the same reasoning would apply. However, removing variables from a hash function can have dramatically different outcomes regarding security than adding variables. In the context of a Pedersen commitment, improper hash computation (i.e. not including the complete transcript up until that point) can introduce a Frozen Heart vulnerability [Mil] when applying a Fiat-Shamir transform. While we aren't applying a Fiat-Shamir in this case, removing variables from a hash is generally considered bad form cryptographically. Hashes are often used to enforce a specific order of operations in an interaction, so by neglecting terms, a malicious adversary isn't forced to use the proper order when crafting a hash, potentially giving them an advantage.

An example of this is in our proof of **Enote Scan Binding** in [Sta] which relied on the test on line 28 (which was line 29 at the time [jefb]). Said test hinged on $\mathsf{K_v^j} = \mathsf{k_v K_{base}}$ to ensure that no two honest receivers could successfully implement an enote scan process for the same enote with different values of $\mathsf{k_v}$. Because $\mathsf{d_e}$ is no longer dependent on $\mathsf{K_v^j}$ (and $\mathsf{k_v}$ as a result), the original proof no longer holds exactly as written. For external transactions, the security property of **Enote Scan Binding** still holds because $\mathsf{s_{sr}} = \mathsf{k_v D_e}$ and $\mathsf{s_{sr}}$ is part of the the construction of $\mathsf{s_{sr}^{ctx}}$. This means distinct $\mathsf{k_v}$ and $\mathbf{k_v}$ would still flag the **ABORT** conditions when checking if $\mathsf{K_s^j} = \mathsf{K_s}$ (for coinbase enotes) or $\mathsf{C'_a} = \mathsf{C_a}$ (otherwise). Regarding an internal transaction, there is no dependency on $\mathsf{k_v}$ as $\mathsf{s_{sr}} = \mathsf{s_{vb}}$. This shouldn't be a problem, though, because there's not a large use-case for scanning an internal transaction when a user knows it's already theirs. The only odd case would be if an auditor needed to look through transactions, but they would already have the root secret $\mathsf{s_{vb}}$, so the point remains moot.

We are of two minds for this change. In general, removing elements from a hash is ill-advised, with how cheap hashes are to compute. For this case, it appears that the dependencies still work out with this removal. We just want to voice that unforeseen consequences could arise later on. The code simplicity and efficiency reasons provided by [jef25] do carry weight, though. Streamlining code reduces insecure implementation risks after all. All that being said, the assignment change of $\mathsf{d_e}$ seems to be fine.

2.4 Special Janus Anchor

Seemingly as a combination of the prior changes, for the derivation of $\mathsf{anchor_{sp}}$, a variable has again been removed from the hash function, in this case we are using the SecretDerive function. Below is the original assignment (as of [jefb]) and the updated assignment with an underbrace emphasizing the removed variable:

$$\mathsf{anchor_{sp}} = \mathsf{SecretDerive}(\text{“Carrot janus anchor special”} \mid\mid \mathsf{D_e} \mid\mid \mathsf{input\_context} \mid\mid \mathsf{K_o} \mid\mid \mathsf{k_v} \mid\mid \underbrace{\mathsf{K_s}}) \rightarrow$$

$$\mathsf{anchor_{sp}} = \mathsf{SecretDerive}(\text{“Carrot janus anchor special”} \mid\mid \mathsf{D_e} \mid\mid \mathsf{input\_context} \mid\mid \mathsf{K_o} \mid\mid \mathsf{k_v})[:16]$$

where

$$\mathsf{K_s} = \begin{cases} \mathsf{k_{gi}G} + \mathsf{k_{ps}T} & \text{(spend key)} \\ \mathsf{k_sG} & \text{(legacy spend key)} \end{cases}$$

As stated earlier, removing a term from a hash function can introduce complications to a protocol where senders and receivers need to agree on an output. In this case, we also have the additional hurdle in that the hash functions are different. Indeed, while both ScalarDerive and SecretDerive involve the Blake2b hash function, the former takes a 64-byte output and proceeds to deserialize then reduce modulo ℓ, while the latter only delivers a 32-byte hash output.

Based on [jef25], this particular change was made to prevent continuous checks for accounts with several main addresses (hybrid-key hierarchies). If $\mathsf{anchor_{sp}}$ needed to be checked for every sub address, the efficiency loss would be inconsequential. Since there is a chain of hash-induced dependencies between $\mathsf{anchor_{sp}}$ and $\mathsf{K_s}$, this change will also not have an effect on the overall protocol. As stated earlier, though, there could be subtle problems that arise later which aren't immediately clear.

We would like to make a special note regarding this change. In the commit at the time of the original audit, the derivation of $\mathsf{anchor_{sp}}$ did not make it abundantly clear that the hash was truncated. In [jefc], the showcased "protocol tweaks" seemed to imply that the truncation was a part of the definition at the time of our audit, with the only change being the removal of $\mathsf{K_s}$. Because this change was rather subtle, we ran a difference check between [jefa] and [jefb] to identify any other small changes and found inconsequential edits.

3 Prior CARROT Audit Comments

3.1 Size of vt

The reasoning supplied in [jefc] regarding the truncation of vt not applying to security is sound. After further review, checking vt for an early abortion cannot be easily verified; only fully honest (not even curious) verifiers can be assumed to have performed the check at all. Thus, vt is a matter of convenience only. The two instances in [Sta] referencing vt in the context of security should therefore be disregarded. Thankfully, this does not affect the overall protocol. In both of these instances, the usage of vt was only supplementary, and additional justification was also given.

3.2 Typo

In Section 3.1.5 of [Sta], we incorrectly listed $\mathsf{s_{sr}} = 8\ \mathsf{r}\ \mathsf{ConvertPointE}(\mathsf{K_s^j})$ twice, when it should have been $\mathsf{s_{sr}} = 8\ \mathsf{r}\ \mathsf{ConvertPointE}(\mathsf{K_v^j})$. The factor of 8 was obviously based on the commit at the time, [jefb], so we understand that using new terminology, we would have $\mathsf{s_{sr}} = \mathsf{r}\ \mathsf{ConvertPointE}(\mathsf{K_v^j})$ based on the cofactor removal. The erroneous swapping of the spend and view keys only nominally alters the proof, as the same line of reasoning still applies.

3.3 HMACs

In Section 2 of [Sta], we found that $\mathsf{anchor_{sp}}$ did not fit the criteria of an HMAC, and instead found it to be a MAC. Upon review, we find that using the language of MACs is inexact and potentially confusing for readers. MACs are used for message authentication, but in this case we are verifying knowledge of a secret, which is a different use-case. We would recommend changing the language to say that $\mathsf{anchor_{sp}}$ is verified by checking its image under a one-way function, then remove any mention of MACs.

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`.

[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`.

[jefa] jeffro256. CARROT. `https://github.com/jeffro256/carrot/blob/master/carrot.md`. Accessed: 2025-8-19.

[jefb] jeffro256. CARROT. `https://github.com/jeffro256/carrot/blob/0a07dfe4f6549a33b06511666667fb31c2016ceb/carrot.md`. Accessed: 2024-10-18.

[jefc] jeffro256. CARROT Follow-Up Audit. `https://gist.github.com/jeffro256/12f4fcc001058dd1f3fd7e81d6476deb`. Accessed: 2025-8-19.

[jef25] jeffro256. Personal communication, 2025.

[Mil] Jim Miller. The Frozen Heart Vulnerability in Bulletproofs. `https://blog.trailofbits.com/2022/04/15/the-frozen-heart-vulnerability-in-bulletproofs/`. Accessed: 2025-8-20.

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

[Sta] Cypher Stack. CARROT Audit. `https://github.com/cypherstack/carrot-audit`. Accessed: 2025-8-19.