RESEARCH BULLETIN MRL-0006

An efficient implementation of Monero subaddresses

Sarang Noether* and Brandon Goodell†
Monero Research Lab
October 3, 2017

Abstract

Users of the Monero cryptocurrency who wish to reuse wallet addresses in an unlinkable way must maintain separate wallets, which necessitates scanning incoming transactions for each one. We document a new address scheme that allows a user to maintain a single master wallet address and generate an arbitary number of unlinkable subaddresses. Each transaction needs to be scanned only once to determine if it is destinated for any of the user's subaddresses. The scheme additionally supports multiple outputs to other subaddresses, and is as efficient as traditional wallet transactions.

1 Introduction

Privacy within Monero transactions is achieved by three primary constructions: ring signatures, one-time keys, and amount commitments. The use of ring signatures ensures that an attacker cannot determine the actual input public key used in the transaction, as it is obscured by the presence of randomly chosen input public keys [3]. Amount commitments use homomorphic properties to guarantee that while a third party is not able to determine the amount of a transaction output, it can prove that the transaction inputs and outputs are balanced. When combined with a range proof to ensure that the output is within a defined and valid range, commitments mask transaction amounts while avoiding misuse by a malicious spender. One-time keys are generated using transaction parameters and the recipient's published wallet address, and are intended to make it impossible for anyone but the recipient to identify the destination of transactions or spend the resulting funds.

An issue not addressed (pun intended) by these privacy guards is that of recipient addresses. Bob may wish to receive Monero into a wallet for personal donations, but also wish to receive Monero for purchases from his business. If Bob is conscious of his privacy, he may not wish to use the same wallet address, since this links his personal and business online presence to the same individual. An obvious solution is for Bob to create two wallets and publish the addresses separately, one to his personal blog and the other to his business site. However, this means that Bob must scan each incoming transaction twice to determine if it was sent to one of his two wallets. This problem is compounded linearly when Bob creates additional addresses for separate uses.

What is needed is a method for allowing Bob to publish different and unlinkable addresses in a way that does not adversely affect computations applied to incoming transactions. In this whitepaper, we document an efficient solution to the problem: a *subaddress* scheme [1]. The scheme allows Bob to produce as many addresses as he wishes and distribute them in any way he sees fit. These subaddresses cannot be linked to each other, nor to Bob's original wallet address. When scanning incoming transactions, the computations required scale in constant time with the number of subaddresses, meaning there is no additional computational complexity.

*sarang.noether@protonmail.com
†surae.noether@protonmail.com

2 Construction

Suppose that Alice wishes to send some of her hard-earned Monero to Bob. Bob's master wallet address, which he has not published anywhere, is $(A, B) = (aG, bG)$, where a and b are secret scalars and G a common elliptic curve basepoint. Bob wishes to use his master address to generate a subaddress, which he can give to Alice or otherwise publish.

Bob is assumed to maintain a list on the computer tasked with managing his wallets; this list consists of scalars used in previous subaddresses. To create a new subaddress, Bob chooses a scalar i, not necessarily at random, that is not already in the list. He computes the following:

$$\begin{aligned} D_i &\equiv B + H_s(a,i)G \\ C_i &\equiv aD_i \end{aligned}$$

Here H_s is a cryptographic scalar hash function. The subaddress is defined as the pair of points (C_i, D_i). Bob is assumed to also have a hash table stored on his computer that maps $D_i \mapsto i$.

To send Monero to Bob's subaddress (C_i, D_i), Alice chooses a random transaction scalar s and computes the transaction public key:

$$R \equiv sD_i$$

Alice then computes the output public key:

$$P \equiv H_s(sC_i)G + D_i$$

The use of separate transaction keys for each output allows Alice to include multiple outputs directed to subaddresses.

If Alice wants to send change to her own master wallet address $(X, Y) = (xG, yG)$, she can do so by constructing a change public key:

$$P_{\text{change}} \equiv H_s(xR)G + Y$$

If instead Alice has a subaddress (Z_j, W_j) of her own, she can direct her change output there:

$$P_{\text{change}} \equiv H_s(xR)G + W_j$$

When Bob scans incoming transactions, he checks each output public key P (with associated transaction public key R) by computing the following:

$$D' \equiv P - H_s(aR)G$$

If Bob sees that this value D' maps to a scalar i in his local hash table, he is assured that the output was sent to the subaddress (C_i, D_i). This is because

$$\begin{aligned} P - H_s(aR)G &= H_s(sC_i)G + D_i - H_s(a(sD_i))G \\ &= H_s(sC_i)G + D_i - H_s(s(aD_i))G \\ &= H_s(sC_i)G + D_i - H_s(sC_i)G \\ &= D_i. \end{aligned}$$

In order to use his funds as the input to a later transaction, Bob needs to be able to determine the private key associated to P. He can do this easily using the index returned from the hash table lookup:

$$p \equiv H_s(aR) + b + H_s(a,i)$$

This succeeds since

$$\begin{aligned} pG &= (H_s(aR) + b + H_s(a,i))G \\ &= (H_s(sC_i) + b + H_s(a,i))G \\ &= H_s(sC_i)G + D_i \\ &= P. \end{aligned}$$

3 Integration with transactions

The present subaddress scheme introduces a necessary change in the handling of transactions, and cannot be considered a "drop-in" replacement for standard wallets. When Alice wishes to spend to Bob's standard wallet, she constructs a transaction public key $R = rG$ using the common basepoint. One naive way that Alice could later prove she was the author of the transaction is to prove knowledge of r (or, even more naively, reveal r directly) to a third party; notably, she can do this without revealing that the funds go to Bob's wallet address.

Using subaddresses, Alice must instead construct the transaction public key as $R = sD$, using Bob's subaddress public key as the basepoint. Since the secret key s is chosen uniformly at random, a subaddress transaction public key is still uniformly distributed; however, Alice can no longer naively prove she knows the common basepoint secret key without also revealing Bob's subaddress as the recipient of the funds, since it is tied in with Bob's subaddress in the Diffie-Hellman exchange. She would need to reveal D in addition to s. While other methods exist to prove authorship of a transaction using commitments, it is worth noting this change.

With standard transactions, Bob can ask Charlie to watch incoming transactions for him, or otherwise wish for Charlie to audit his wallet. To do this, Bob would reveal to Charlie the secret view key a and wallet address component B. In the subaddress scheme, Bob can similarly reveal a to Charlie. Charlie must then construct the hash table using either a list of known indices from Bob or a set range that he precomputes.

4 Subaddress accounts

Since a change output can be directed to either the sender's master wallet or a subaddress, it is possible to logically group subaddresses in a natural way that parallels the function of separate wallet balances. One way to do this is to replace the subaddress index i with an ordered pair (i,j). For any fixed i, the wallet holder defines the set of subaddresses $\{i,j\}_j$ as an *account*, where each subaddress has *major index* i and *minor index* j.

When Bob receives funds at a subaddress (i,j) within account i, his wallet software sums the funds held by all subaddresses in the account as a single balance. Change can then be redirected to the subaddress $(i,0)$ when spent. We stress that this is a protocol convenience only, and has no bearing on the cryptography of the subaddress scheme.

5 Analysis

5.1 Linking and wallet determination

Because the cryptographic scalar hash function H_s has uniformly distributed output, the set of Bob's possible subaddress components

$$\{D_i\}_i = \{B + H_s(a, i)G\}_i$$

is also uniformly distributed [2]. This means an adversary in possession of an arbitrary collection of subaddresses $\{(C_i, D_i)\}_i = \{aD_i, D_i\}$ can neither determine the discrete logarithm $a = \log_{C_i} D_i$ nor invert the hash function (which would also require knowledge of B). This means Bob's master wallet address is protected, even if the adversary convinces him to generate new subaddresses with chosen indices.

5.2 Reconstructed wallet

If Bob loses access to his local wallet software and restores from the seed, he will not immediately be able to identify transactions destined for his subaddresses, as he would need access to the hash table. To reconstruct the table without missing any likely subaddresses, Bob chooses two lookahead values, L_M and L_m. He generates the hash table using major indices $i \leq L_M$ and minor indices $j \leq L_m$ (for each major index). After scanning transactions using the initial table, Bob repeats this process, ensuring he has generated hash entries L_M past the highest major index of any transaction, and L_m past the highest minor index within each major index. Provided used subaddresses do not fall beyond these lookahead values in any transactions, Bob will recover the necessary hash entries.

5.3 Mixed-type transactions

Suppose that Alice's wallet software had one too many glasses of cryptocognac, and formats a transaction incorrectly. In particular, suppose Alice produces the transaction public key $R = sG$ (instead of $R = sD$) but continues to compute the output public key as $P = H_s(sC_i) + D_i$. That is, the transaction is now of "mixed type" and is not a correct subaddress transaction.

In this case, the transaction public key no longer contains information about the subaddress destination, and Bob cannot use a single scan to detect this transaction. Indeed, his wallet software will fail to recognize that the mixed-type transaction is directed at him. However, if Bob suspects that this situation has arisen, he can iterate through his subaddress indices and compute

$$P - H_s(a[b + H_s(a, j)]R)$$

for each j that he has used to generate a subaddress. For $j = i$, the result is guaranteed to be D_i. This means that Alice's error can be detected if Bob performs such a linear scan, and Bob can still recover the private key needed to spend his money. The tradeoff is that Bob loses the ability to perform single transaction scans, so he may choose to have his wallet software complete a linear scan every so often if he is concerned about mixed-type transactions.

5.4 Secret view key

It is not possible to provide view access to a third party selectively to only particular subaddresses under the assumption of uniformly distributed secret keys. If Charlie receives Bob's view key a and one hash table entry $D_i \mapsto i$, he computes the master wallet address component $B = D_i - H_s(a, i)G$. If Bob does not choose his subaddress indices at random, Charlie can easily reconstruct other hash table entries using low indices that are likely also subaddresses Bob has used:

$$D_j = B + H_s(a, j)G$$

In order to keep other transactions private so Charlie cannot view them, Bob must create a new master wallet and ensure Charlie does not have access to its secret view key.

5.5 Secret spend key

Similarly, it is not possible to provide a third party with the ability to selectively spend funds from a particular subaddress if the secret keys are uniformly distributed. To spend funds from any subaddress, Bob needs both master wallet secret keys a and b, as well as the subaddress index. Revealing this information to Charlie would allow him to spend funds sent to any subaddress to which he knows (or can otherwise determine) the index. Of course, the correct way to do this is for Bob to send funds to Charlie's wallet or subaddress, just as with a standard wallet setup.

5.6 Efficiency

Construction of a new subaddress requires a trivial number of elliptic curve operations and is negligible since this operation is performed only as needed.

Sending to a subaddress requires the same number of operations as an equivalent traditional transaction to a standard wallet address. However, instead of computing the transaction key $R = rG$ using the common basepoint in a traditional transaction, Alice instead computes the key as $R = sD$ using the subaddress basepoint; indeed, Alice does not even know the transaction secret key in this case! This means she cannot precompute the transaction key without knowing the desired recipient's subaddress. In practice, this is not an issue. Computation of the output public key is precisely analogous to the traditional case.

Using a standard wallet, Bob would check a transaction against his master wallet secret view key by applying one hash-to-scalar operation, two elliptic curve scalar multiplications, and one elliptic curve point addition (where the hash time is negligible). In the present scheme, Bob must apply the same number of equivalent operations, with the point addition replaced with point subtraction. There is an additional hash lookup for each transaction, but this scales in constant time with the number of generated subaddresses. This is in stark contrast to the use of multiple wallet addresses, where each transaction must be separately checked against each secret view key.

We therefore conclude that scanning incoming (correctly formatted) transactions for ownership by an arbitrary number of subaddresses is no greater than scanning for ownership by a single standard wallet address. This reduces the otherwise linear scaling of scanning for multiple standard wallet addresses to constant time in the subaddress case.

6 Conclusions

We have presented brief documentation of a subaddress scheme, an efficient alternative to maintenance of multiple wallet addresses. Under the standard CryptoNote wallet implementation in Monero, Bob must publish separate wallet addresses if he wishes to keep them unlinkable from the perspective of a third party or adversary. However, this approach requires Bob (or another holder of his secret view key) to check every incoming transaction against each of his addresses.

The proposed scheme allows Bob to use a single master wallet address to efficiently generate as many subaddresses as he wishes. An adversary can neither determine if two subaddresses are generated from the same master wallet address, nor identify the parent address. Bob need only maintain a local private hash table that links subaddresses to the scalar index used to generate them.

Transactions to a subaddress require only a minor change to the existing transaction protocol. Multiple outputs are supported, and the change output of a transaction from Alice to Bob can be directed to either Alice's master wallet address or to a subaddress of her choice. When examining incoming transactions, Bob performs only a single verification for each transaction regardless of the number of subaddresses he maintains, using his local hash table to recover the corresponding transaction private key.

Unlinkable subaddresses represent an efficient and secure solution to the problem of multiple wallet addresses, allowing users to determine how their wallets are presented to others and the level of privacy they desire.

References

[1] Monero project. Subaddresses. `https://github.com/monero-project/monero/pull/2056`. GitHub pull request #2056.

[2] Paola Scozzafava. Uniform distribution and sum modulo m of independent random variables. *Statistics & Probability Letters*, 18(4):313–314, 1993.

[3] Nicolas van Saberhagen. Cryptonote v 2. 0, 2013.