MONERO RESEARCH LAB

TECHNICAL NOTE MRL-XXXX

Application of Bulletproofs in Monero Transactions

Sarang Noether*
Monero Research Lab
March 5, 2018

Abstract

This technical note briefly describes the proposed application of Bulletproofs [**?**] in Monero. The proofs are used as a drop-in replacement of the existing Borromean bitwise non-interactive zero-knowledge range proofs used to show that a committed amount is in a specified range. Bulletproofs reduce both proof size and verification time, as well as provide a straightforward method for batch verification of proofs from multiple transactions. We describe our implementation, noting specific areas of optimization from the original paper.

1 Introduction

The implementation of confidential transaction amounts in Monero is accomplished using homomorphic commitments. Each input and output amount, including fees, is represented by a commitment of the form $vG+\mu H$, where G and H are elliptic curve generators, v is the amount, and μ is a mask. Without knowledge of the commitment opening, a third party cannot determine the amount; however, it is trivial for the third party to convince itself that a transaction balances (that is, that the difference between inputs and output amounts is zero). The homomorphic property of the commitments is such that the difference in commitments must itself be a commitment to zero.

However, this is not sufficient to ensure a correct and safe transaction model. An adversary could easily construct a combination of positive and negative outputs such that the transaction amounts balance. A third party would still verify that the transaction balances, though the adversary has effectively printed free money in an undetected fashion. To combat this, we require that each amount commitment come equipped with a *range proof* that convinces a verifier that the corresponding output is both positive and does not risk an overflow by being too large. The range proof scheme must be non-interactive and zero-knowledge; that is, the verifier does not need to communicate with the prover once the proof is generated, and the proof itself reveals no information about the amount except that it is within the stated range.

The current range proof style used in Monero confidential transactions is a *Borromean bitwise* range proof. To generate a proof that a commitment $C \equiv vG + \mu H$ represents an amount $v \in [0, 2^n - 1]$ for some bit length $n > 0$ (in Monero $n = 64$), the prover generates separate commitments for each bit. The prover then generates a Borromean ring signature showing that each commitment is to either 0 or 2^i for appropriate i. Any third-party verifier can then convince itself that the bit commitments reconstruct the committed amount, that each commitment is to either 0 or 2^i, and therefore that the committed amount lies in the correct range.

However, this comes at a cost. Borromean bitwise proofs scale linearly in size with the number of bits in the range. Further, if multiple outputs are used in a transaction, a separate proof is required for each. Each proof is large, taking up 6.2 kB of space.

*sarang.noether@protonmail.com

2 Bulletproofs

Bulletproofs are a recent general non-interactive zero-knowledge proof construction [**?**]. Using a novel inner product argument, they can be used in a variety of applications ranging from range proofs (pun intended) to verifiable shuffles and even proofs of general arithmetic circuit evaluation. For our purposes, they can accomplish the same goal as Borromean bitwise range proofs: convincing a verifier that a committed amount is within a claimed range.

The details of Bulletproof construction, both for prover and verifier, are discussed in the paper [**?**], so we will not duplicate them here. However, several definitions are useful when discussing the scaling. A standard Bulletproof that shows an amount is within the n-bit range $[0, 2^n - 1]$ is called a *single-output proof* or a *1-proof*. However, it is possible for a prover to construct a single proof showing that m separate amounts (with separate random masks) each lie within the range $[0, 2^n - 1]$, where m is a power of two. Such a proof is called an *aggregate proof* or, more precisely, an *m-proof*. The scheme is constructed in such a way that a single-output proof is trivially an m-proof with $m = 1$ (which simplifies the code). It is important to note that the construction of an aggregate proof requires that the prover know each amount and mask; this means that while it is useful for all outputs in a transaction to be contained within a single aggregate proof for space savings, it is not possible for a third party to take existing proofs and construct an aggregate proof, either within a single transaction or between different transactions.

The size scaling benefits of Bulletproofs occur at two levels:

1. **Bit length of range**. The size of a Bulletproof increases logarithmically with the number of bits in the range. In bitwise range proofs, the proof size increased linearly with the number of bits.

2. **Number of amounts in aggregate proof**. The size of a Bulletproof increases logarithmically with the number of amounts included in a single aggregate proof. In bitwise range proofs, the proof size increased linearly with the number of bits (since a separate proof was needed for each amount).

We discuss efficiency in more detail below.

There is a separate scaling argument that is useful. A new node that comes online will receive many m-proofs, at least one per post-Bulletproof transaction in the blockchain. Instead of verifying each of the proofs separately, the node can perform a *batch verification* of as many proofs at a time as it wishes. As described below, this process requires that certain portions of each proof be verified separately, but allows for the remaining parts of the proofs to be batched and verified together. The resulting verification time is linear in the number of proofs, but with a significantly lower time per proof. An existing node that has already verified the transactions in the blockchain can still use batch verification on new transactions it receives, but the benefits are not as great due to the lower number of transactions that must be verified in a short time.

3 Optimizations

For the most part, the proposed implementation of Bulletproofs in Monero follows the Bulletproofs paper in scope and notation wherever possible. However, we include several optimizations that have also been discussed for other projects. These optimizations are algebraically equivalent to those in the paper, but reduce the time required for verification. The author understands that some or all of the optimizations may be included in an update to the Bulletproofs paper sometime in the future. However, we document them here for completeness and ease of code review. The reader is encouraged to refer to the paper for the complete context of our changes.

3.1 Curve group notation

The paper is written with a general group structure in mind, so scalar-group operations are written multiplicatively (*e.g.* $x = a^b c^d$). In the case of elliptic curve groups, we use additive notation instead (*e.g.* $X = bA + dC$) and use case to differentiate between curve points and scalars for clarity. This is purely a notational convenience.

3.2 Basepoint notation

Throughout the paper, amount commitments are expressed as $V \equiv vG + \mu H$, where G and H are distinct (but arbitrary) fixed elliptic curve group generators. We interchange the roles of G and H throughout our implementation to match the use of existing base points used in commitments elsewhere in the Monero codebase. Note that the indexed $\{G_i\}$ and $\{H_i\}$ curve points are not modified in this way.

3.3 Fiat-Shamir challenges

To make the Bulletproof scheme non-interactive, we follow the paper by introducing Fiat-Shamir challenges computed by hashing the proof transcript up to the point that a new challenge is needed. This is done by introducing a rolling hash that uses as input the previous challenge and any new proof elements introduced. The prover and verifier compute these challenges identically.

3.4 Inner product argment

The inner product argument in Protocol 1 of the Bulletproofs paper uses recursion to shrink the size of its input vectors down to single elements. These inputs include distinct curve group generators $\{G_i\}$ and $\{H_i\}$, which we compute using an indexed hash function. We make several optimizations to this protocol for the verifier.

First, we observe that the curve points in Equation (10) are in fact linear combinations of $\{G_i\}$ and $\{H_i\}$ that use the scalar challenges in Equations (24)-(25). Next, we note that the point P in Equation (62) is passed into Protocol 1 as described in Section 4.2 of the paper. Since this curve point contains a linear combination of the same group generators as Protocol 1, we can take advantage of this and compute a single linear combination, rather than separately compute Equations (62) and (10).

In practice, we replace Equations (62) and (10) with the following check, where $M \equiv |\{L_j\}| = |\{R_j\}|$:

$$A + xS - \mu G + \sum_{j=0}^{M-1} (w_j^2 L_j + w_j^{-2} R_j) + (t - ab)xH - \sum_{i=0}^{mn-1} (g_i G_i + h_i H_i) = 0$$

The symbols are mostly those used in the paper. However, we use w_j to represent the round challenges in Lines (21)-(22), and x to represent the challenge in Lines (32)-(33) to avoid reuse of symbols. The scalars g_i and h_i are computed in the following way. Express the index $i = b_0 b_1 \cdots b_{M-1}$ bitwise, where b_{M-1} is the least-significant bit. Then

$$g_i = a \prod_{j=0}^{M-1} w_j^{2b_j - 1} + z$$

and

$$h_i = \left(by^{-i} \prod_{j=0}^{M-1} w_j^{-2b_j + 1} - zy^i + z^{2 + \lfloor i/N \rfloor} 2^{i \bmod N} \right) y^{-i}$$

This optimization is applied only to the verifier.

3.5 Batch verification

Our implementation permits the verifier to take many aggregate proofs and verify them together as a batch. We do not assume that the proofs each have the same number of outputs, nor make any restrictions on the maximum size of a batch. The batch verification we describe will only succeed if each proof is valid, and will fail if one or more proofs are invalid.

Batch verification is split into two checks, performed after iterating over each proof in the batch. During the iteration, the verifier keeps ongoing sums of components from each proof and then performs the first-stage check for Equation (61):

$$\sum_l (\beta_l \tau_{xl}) G + \sum_l \beta_l \left[t_l - (k_l + z_l \langle \overline{1}^{mn}, \overline{y_l}^{mn} \rangle) \right] H - \sum_l \beta_l \left(\sum_j z_l^{j+2} V_{lj} - x_l T_{1l} - x_l^2 T_{2l} \right) = 0$$

The second-phase check proceeds similarly:

$$\sum_l \beta_l (A_l + x_l S_l) - \sum_l (\beta_l \mu_l) G + \sum_l \left[\beta_l \sum_j (w_{lj}^2 L_{lj} + w_{lj}^{-2} R_{lj}) \right] + \sum_l \beta_l x_l (t_l - a_l b_l) H$$
$$- \sum_i \left[\sum_l (\beta_l g_{li}) G_i + \sum_l (\beta_l h_{li}) H_i \right] = 0$$

Here each l-indexed sum is over each proof in the batch, and β_l is a weighting factor chosen at random (not deterministically) by the verifier. This ensures that, except with negligible probability, the checks will only succeed if each proof is separately valid; an adversary cannot selectively provide a batch containing invalid proofs in an attempt to fool the verifier. The benefit to this approach is that the sums can be computed as large multi-exponentiation operations after the scalars from all proofs have been assembled.

If the batch fails either check, at least one proof in the batch is invalid. To identify which proofs are at fault, the verifier can either iterate through each proof and perform the checks separately (in linear time), or perform a binary search by successively performing the checks on half-batches until it identifies all faulty proofs (in logarithmic time).

4 Proof size

Including the amount commitment V, a single Borromean bitwise range proof occupies 6.2 kB of space; a transaction with m outputs therefore requires $6.2m$ kB of space. An m-proof (with a 64-bit range) requires $2 \lg m + 17$ group elements and 5 scalars, each of which takes up 32 bytes. Table 1 shows the space savings from Bulletproofs for several values of m.

m	Bulletproof	Borromean	Relative size
1	704	6200	0.114
2	768	12400	0.062
8	896	49600	0.018
16	960	99200	0.010
128	1152	793600	0.001

Table 1: Size (bytes) of m Borromean proofs versus m-proof

Using data from the Monero blockchain[†] on the distribution of the number of outputs in transactions, the use of Bulletproofs would reduce the total size of range proofs by 94%.

References

[†]Data was taken from blocks 1400000 through 1500000