Transaction efficiency then and now

Monero Konferenco

Sarang Noether, Ph.D.

23 June 2019

Disclaimer

The views expressed in this presentation are solely those of the author, and do not necessarily reflect those of any other person, organization, or community.

The author receives funding support from the Monero community.

What is transaction efficiency?

Space
Bandwidth/storage

Generation time
Low-capacity devices

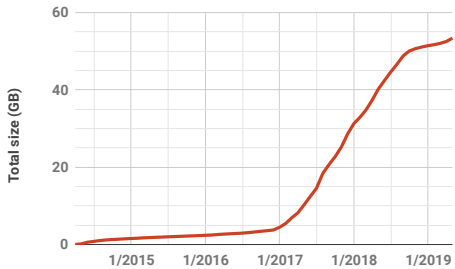
Verification time
Chain sync

Levels of optimization

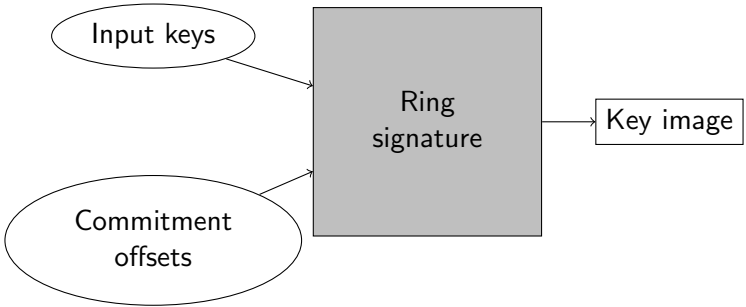
We can optimize at difference levels:

- low-level cryptographic operations
- higher-level constructions
- protocol parameters

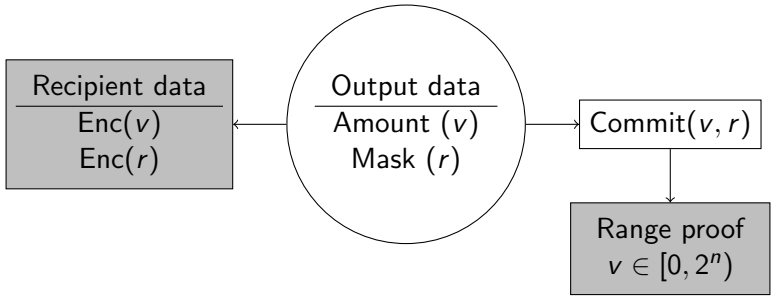
We will focus on *constructions* today, but all of these have effects.



Chain growth over time



The ring signature proves key ownership, helps demonstrate transaction balance, and asserts correct key image construction.



The output constructions communicate output data to the recipient, are used (with auxiliary commitments) to show balance, and prevent amount overflow.

Construction: ring signature

We use **MLSAG** signatures to prove ownership of spent outputs without revealing the specific output in question. This type of ring signature shows more, though.

- The signer controls one output among a given set
- The key image (used to detect double-spend attempts) was generated correctly
- The (hidden) transaction amounts balance

These signatures scale *linearly* with the ring size in space, signing time, and verify time.

CLSAG: the new hotness

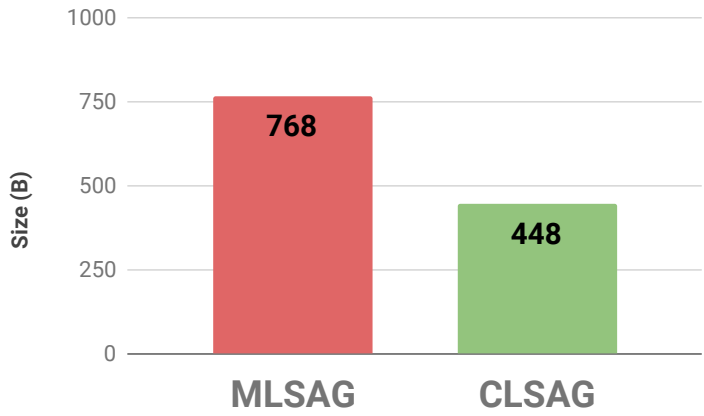
We propose **CLSAG** signatures to do the same thing, but more efficiently.

CLSAG signatures use a linear combination of output signing keys and commitment keys to provide a drop-in replacement to MLSAG signatures.

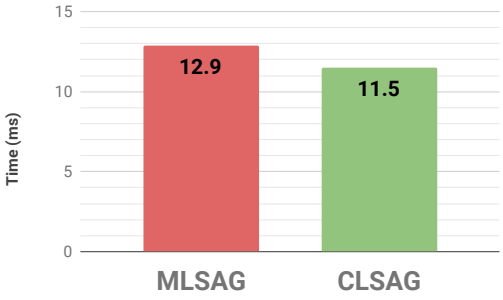
They still scale linearly, but take up less space and are faster to sign and verify.

Security is more formally proven.

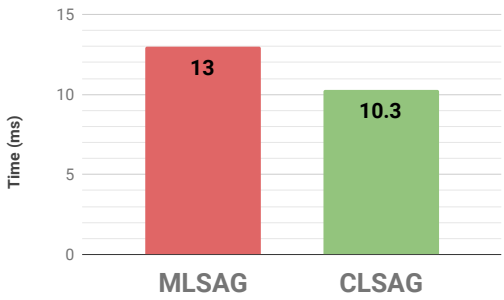
Signature size (11-ring)



Timing (11-ring)



Signing



Verification

Construction: commitments

All amounts are represented as Pedersen commitments with globally fixed base:

$$\mathsf{Commit}(v, r) \equiv \underbrace{vH}_{\text{value}} + \underbrace{rG}_{\text{mask}}$$

The recipient needs to know both v and r to reconstruct the commitment and later use it in a ring signature to spend the output.

Previously, both values were encrypted using the transaction shared secret $\mathsf{H}(aR, i)$:

$$\begin{aligned} \mathsf{Enc}(v) &\equiv v + \mathsf{H}(\mathsf{H}(\mathsf{H}(aR, i))) \\ \mathsf{Enc}(r) &\equiv r + \mathsf{H}(\mathsf{H}(aR, i)) \end{aligned}$$

New hotness: being clever

This is wasteful. If generated randomly, both r and $\mathrm{Enc}(r)$ are uniformly distributed and known only to the sender and recipient. So why include $\mathrm{Enc}(r)$ in the transaction at all?

Solution: Do not include $\mathrm{Enc}(r)$. The sender and recipient both compute

$$r \equiv \mathrm{H}(\text{“commitment\_mask”}, \mathrm{H}(aR, i))$$

instead, and save 32 bytes per output.

New hotness: being clever

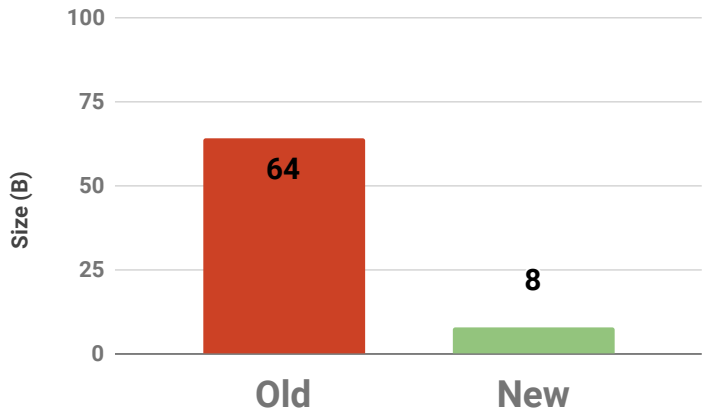
This is still wasteful. The amount cannot be larger than 8 bytes, but is represented as a 32-byte scalar.

Solution: Restrict the amount representation to 8 bytes and encrypt with a length-restricted XOR operation to save 24 bytes per output:

$$\mathrm{Enc}(v) \equiv \Big(v \oplus \mathrm{H}(\text{“amount”}, \mathrm{H}(aR, i))\Big)_{0-7}$$

The full 32-byte scalar can be recovered in the same way.

Commitment data size



Construction: range proofs

Since amounts are represented as commitments and balance is proven using commitment sums and differences, amounts must be restricted (relative to scalar field size) to avoid overflow.

We must show that for $\text{Commit}(v, r)$, the value v is in the range $[0, 2^{64})$.

Strategy: Show that v can be decomposed to 64 bits, each of which is 0 or 1, without revealing the value of the bits.

Example: $13_{10} = 1101_2 = (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)$

Old way: bitwise ring signatures

The previous method, bitwise Borromean range proofs, used ring signatures.

- Construct 64 commitments, each to a bit of v, with masks that sum appropriately.
- Build a ring signature over each bit commitment to show the value is either 0 or 1.

To verify:

- Take a weighted sum of the bit commitments (using powers of 2) and show it recovers the full amount commitment.
- Verify the ring signature to assert that each commitment is to 0 or 1.

This scales in space and time linearly with the number of bits!

Bulletproofs: the new hotness

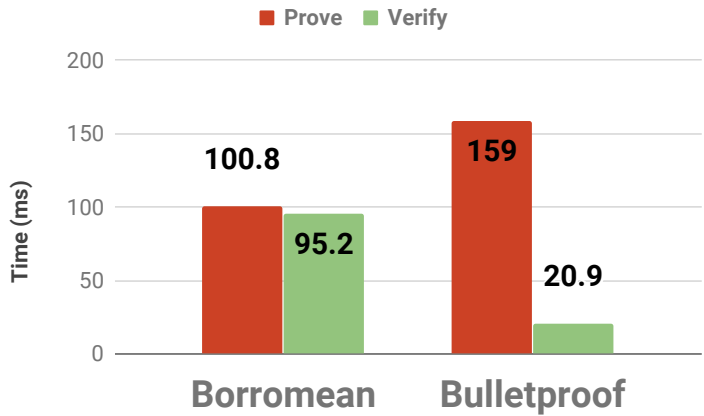
Bulletproofs take a different approach.

- Reduce the range assertion to a set of vector inner product equations.
- Prove knowledge of vector components in zero knowledge.
- Compress this proof using a clever folding technique.

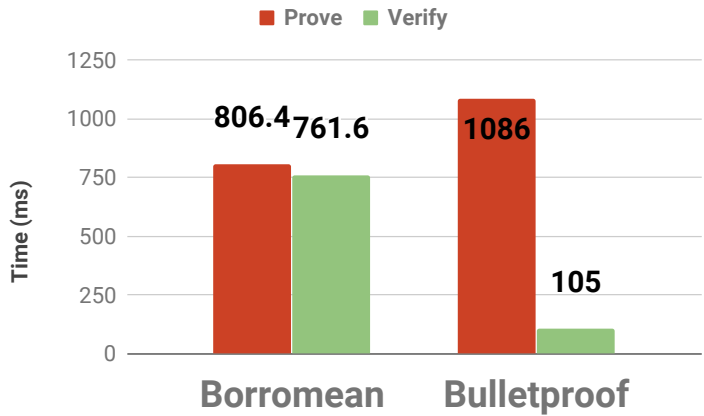
This achieves precisely the same goal as bitwise Borromean proofs, but scales logarithmically with the number of bits. Plus:

- A prover can include multiple values in the same proof, to extend the logarithmic space scaling.
- Generation and verification are still linear in the bit length, but we can use tricks.
- Verification of a set of proofs adds only marginal time cost! (This is *batching*.)

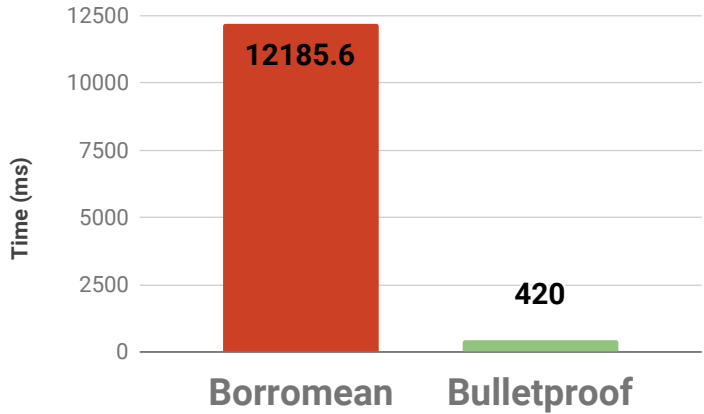
2-output verification



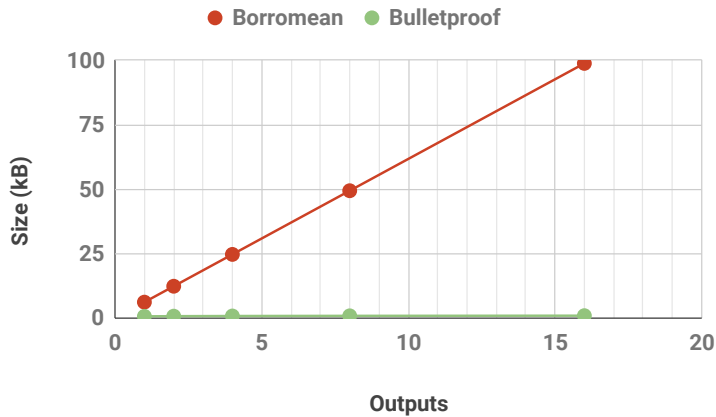
16-output verification



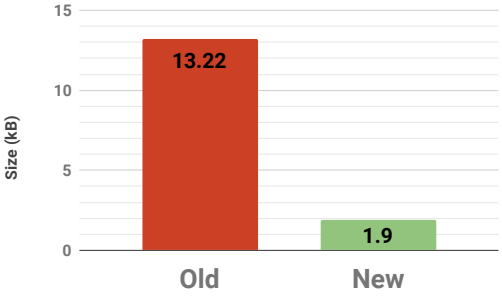
2-output 128-batch verification



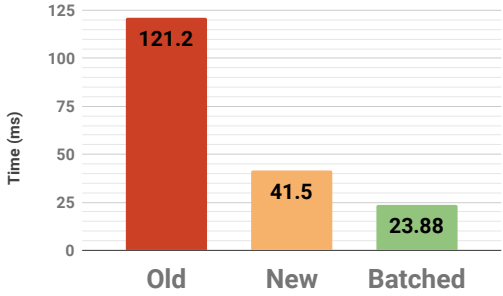
Proof size


Borromean
Bulletproof
100
75
50
25
0
Size (kB)
0
5
10
15
20
Outputs

Typical transaction



Transaction size



Verification time

We save 85% on space and up to 81% on verification time!

These are not complete scaling solutions!

These are incremental (but useful) improvements to slow blockchain growth and speed up new node sync time.

- Sublinear fixed-decoy transaction protocols
  - Lelantus
  - Omniring
- Efficient full-anonymity transaction protocols
  - Bulletproofs (poor verification scaling for circuits)
  - Older-style SNARKs (toxic; excellent space/time scaling)
  - Newer-style SNARKs (nontoxic; better scaling, but not quite there)
- Off-chain transaction layers (requires new plumbing)
  - Atomic swaps
  - Payment channels/networks

Thank you!

GitHub: `https://github.com/SarangNoether`

- `research-lab`: older research
- `skunkworks`: newer research (work in branches)

GitLab: `https://repo.getmonero.org/SarangNoether`