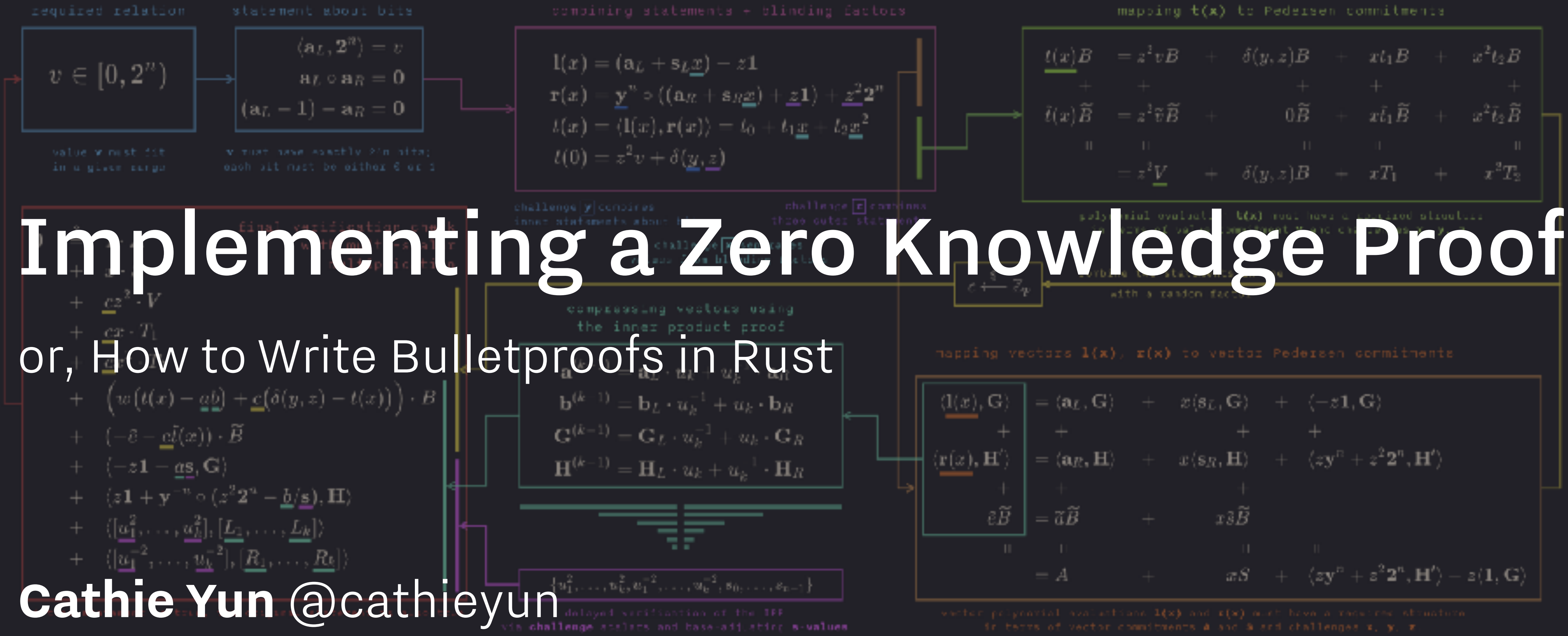


Implementing a Zero Knowledge Proof

or, How to Write Bulletproofs in Rust

Cathie Yun @cathieyun

Crypto Village - DEFCON 2019

The story

1. Motivation
2. Understanding the paper
3. Tools of the trade

The code

4. Range proof
5. Rust tricks
6. Above and beyond

The story

The code

Why do we care about zero knowledge proofs?

Non-confidential transaction

INPUTS	OUTPUTS
5	3
4	6

$$5 + 4 = 3 + 6$$

Confidential transaction (broken)

INPUTS	OUTPUTS
A = Com(5)	C = Com(3)
B = Com(4)	D = Com(6)

Additively homomorphic commitment

$$A + B = C + D$$

Confidential transaction (broken)

INPUTS	OUTPUTS
A = Com(5)	C = Com(-1)
B = Com(4)	D = Com(10)

OH NO!!!
integer underflow!

$$A + B = C + D$$

Confidential transaction

INPUTS	OUTPUTS
A = Com(5)	C = Com(3) proof$_{(C)}$
B = Com(4)	D = Com(6) proof$_{(D)}$

ZK proof that amount is in range

$$A + B == C + D$$

Why do we care about Bulletproofs?

Blockchains & Bulletproofs

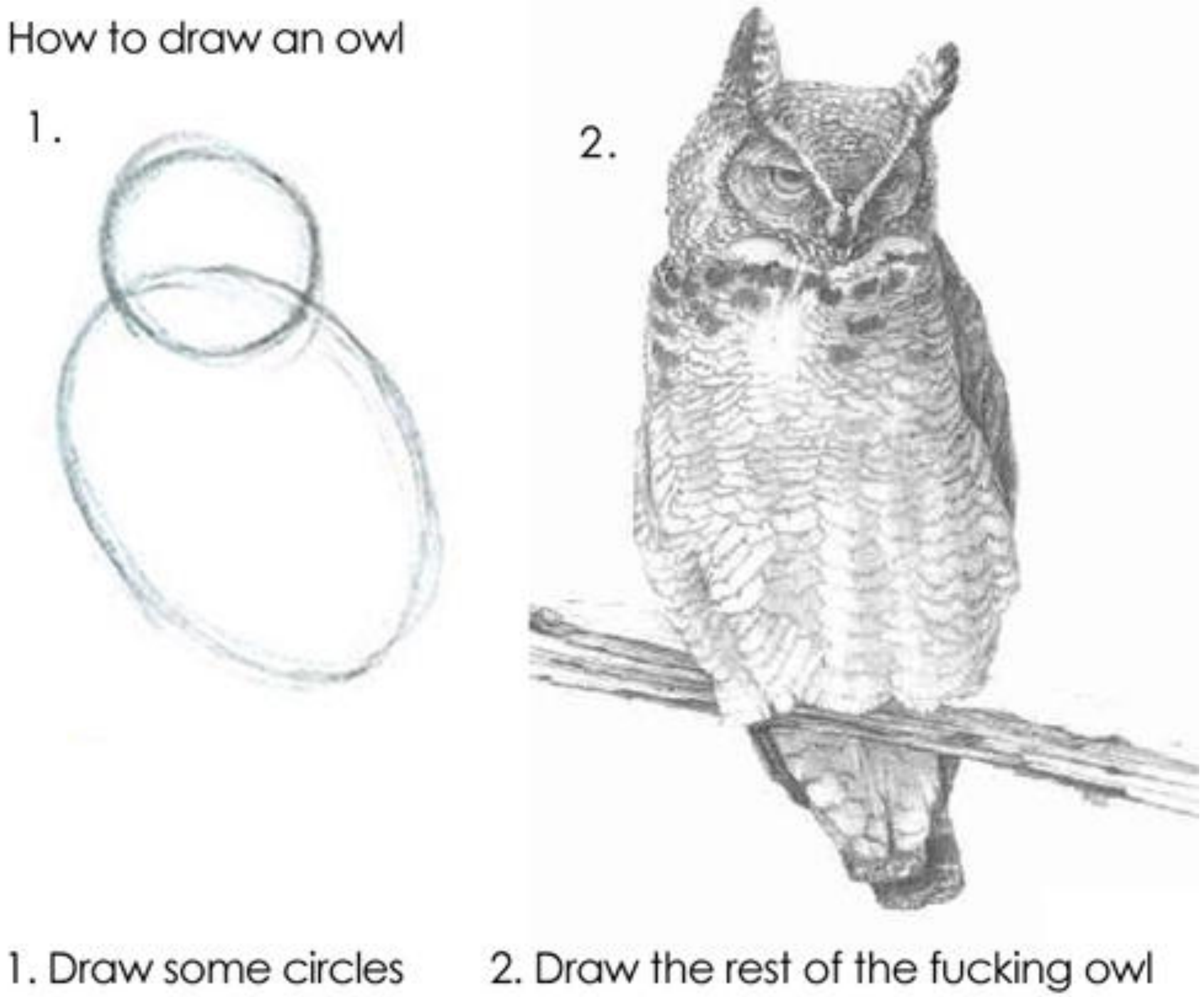
Blockchain requirement	Bulletproofs provide
Constrained proof size (all nodes must receive and verify proofs)	**O(log(N))** proof sizes, less than **1 Kb** for most cases.
Fast verification (low latency - all verifiers must sync quickly)	Fast verification with **Ristretto** and **AVX2**; scales well via **batching** and **aggregation**.
Ad hoc logic (different value flows, custom smart contracts)	**No trusted setup -** cheap on-the-fly initialization of verification circuit

The story

1. Motivation
2. **Understanding the paper**
3. Tools of the trade

The code

4. Range proof
5. Rust tricks
6. Above and beyond


How to draw an owl
1.
2.
1. Draw some circles
2. Draw the rest of the fucking owl

range change (calc.)

$v \in [0, 2^n)$ (1)

let $\vec{a}$ be the bits of v

(2) $\langle \vec{a}, \vec{2}^n \rangle = v$ $\Big\} \Leftrightarrow (1)$

(3) $\vec{a} \circ (\vec{a} - \vec{1}) = \vec{0}^n$

Since $\vec{a} \circ (\vec{a} - \vec{1}^n) = \vec{0} \iff \forall y \in \mathbb{Z}_p, \langle \vec{a}, (\vec{a} - \vec{1}) \circ \vec{y}^n \rangle = 0$

we have (4) $\langle \vec{a}, \vec{y}^n \circ (\vec{a} - \vec{1}^n) \rangle = 0 \xrightarrow{\text{prob}} (3)$

for verifier's choice of y.

(2)+(4) $\Uparrow$

(5) $z \langle \vec{a}, \vec{2}^n \rangle + \langle \vec{a}, \vec{y}^n \circ (\vec{a} - \vec{1}^n) \rangle = zv$

Paper: rearrange to put component dot a in

(6) $\langle \vec{a}, \vec{y}^n \circ (\vec{a} - \vec{1}^n) + z \vec{2}^n \rangle = zv$

As in the paper, want to blind the a values.

Prover selects $\vec{s} \xleftarrow{\$} \mathbb{Z}_p^n$

$$\vec{l}(x) = \vec{a} + x\vec{s}$$

$$\vec{r}(x) = \vec{y}^n \circ (\vec{l}(x) - \vec{1}^n) + z\vec{2}^n$$

Set $t(x) = \langle \vec{l}(x), \vec{r}(x) \rangle = t_0 + t_1 x + t_2 x^2$

$$= \langle \vec{a}, \vec{y}^n \circ (\vec{a} - \vec{1}^n) + z\vec{2}^n \rangle + \left(\langle \vec{s}, \vec{y}^n \circ (\vec{a} - \vec{1}^n) + z\vec{2}^n \rangle + \langle \vec{a}, \vec{y}^n \circ \vec{s} \rangle \right) x + \langle \vec{s}, \vec{y}^n \circ \vec{s} \rangle \cdot x^2$$

$$\langle \vec{l}(x), \vec{G} \rangle = \langle \vec{a}, \vec{G} \rangle + x \langle \vec{s}, \vec{G} \rangle$$

Set $\vec{H}' = \vec{y}^n \circ \vec{H}$ (lazy eval)

$$\langle \vec{r}(x), \vec{H} \rangle = \langle \vec{y}^n \circ (\vec{a} + x\vec{s} - \vec{1}^n) + z\vec{2}^n, \vec{H} \circ \vec{y}^{-n} \rangle$$

$$= \langle \vec{a} + x\vec{s} - \vec{1}^n + z\left(\tfrac{2}{y}\right)^n, \vec{H} \rangle$$

$$= \langle \vec{a}, \vec{H} \rangle + x \langle \vec{s}, \vec{H} \rangle + \langle z\left(\tfrac{2}{y}\right)^n - \vec{1}^n, \vec{H} \rangle$$

$$t(x)B = zvB + xt_1B + x^2t_2B$$

$$+ \quad + \quad + \quad + \quad +$$

$$\tilde{t}(x)\tilde{B} = z\tilde{v}\tilde{B} + x\tilde{t}_1\tilde{B} + x^2\tilde{t}_2\tilde{B}$$

$$t(x)B + \tilde{t}(x)\tilde{B} = zV + xT_1 + x^2T_2$$

$$= z\,\mathrm{Com}(v) + x\,\mathrm{Com}(t_1) + x^2\,\mathrm{Com}(t_2)$$

$$\langle \vec{l}(x), \vec{G} \rangle = \langle \vec{a}, \vec{G} \rangle + x \langle \vec{s}, \vec{G} \rangle$$

$$+$$

$$\langle \vec{r}(x), \vec{H}' \rangle = \langle \vec{a}, \vec{H} \rangle + x \langle \vec{s}, \vec{H} \rangle + \langle z\left(\tfrac{2}{y}\right)^n - \vec{1}, \vec{H} \rangle$$

$$+ \quad + \quad +$$

$$\tilde{e}\tilde{B} = \tilde{a}\tilde{B} + x\tilde{s}\tilde{B}$$

$$P = A + xS + \langle z\left(\tfrac{2}{y}\right)^n - \vec{1}, \vec{H} \rangle$$

$\mathrm{Com}(\vec{l}(x), \vec{r}(x), \tilde{e})$ $\quad \mathrm{Com}(\vec{a}, \vec{a} - \vec{1})$ $\quad x\,\mathrm{Com}(\vec{s}, \vec{s})$

Prover sends $\tilde{e}, t(x), \tilde{t}(x), \vec{l}(x)$ to verifier.

Verif: $\vec{r}(x) \leftarrow \cdots, \vec{H} \leftarrow \cdots$ (lazy), $P \leftarrow \langle \vec{l}(x), \vec{G} \rangle + \langle \vec{r}(x), \vec{H} \rangle + \tilde{e}\tilde{B}$

checks t poly: $t(x)B + \tilde{t}(x)\tilde{B} = zV + xT_1 + x^2T_2$

checks l, r: $P = A + xS + \langle z\left(\tfrac{2}{y}\right)^n - \vec{1}^n, \vec{H} \rangle$

$t(x) = \langle \vec{l}(x), \vec{r}(x) \rangle$

$\cdot) = 0$

$\underline{\underline{W_L}} \cdot \vec{a}_L + \underline{\underline{W_R}} \cdot \vec{a}_R + \underline{\underline{W_O}} \cdot \vec{a}_O$

$\langle \vec{z}_{[1:]}^{\,Q+1}, \vec{w} \rangle$

$\cdot \underline{\underline{W_O}}) \cdot X + \vec{a}_O \cdot X^2$

$\underline{\underline{W_L}} \cdot X - \vec{y}^n + z_{[1:]}^{Q+1} (\underline{\underline{W_O}}) \rangle$

$^{-n} \circ (z_{[1:]}^{Q+1} \cdot \underline{\underline{W_R}}), z_{[1:]}^{Q+1} \cdot \underline{\underline{W_L}} \rangle)$

$\in \mathbb{Z}_p^n[X]$

$\in \mathbb{Z}_p^n[X]$

$\delta(y,z)$

$\mathbb{Z}_p^n[X]$

← multiply by $\langle \vec{y}^{-n}, \vec{y}^{-n} \rangle$

$\langle \vec{a}_O, -\vec{y}^n \rangle + \langle \vec{a}_O, \vec{z}^Q \cdot \underline{\underline{W_O}} \rangle$ ← merge

$= \langle \vec{a}_L, \vec{y}^n \circ \vec{a}_R \rangle + \langle \vec{a}_L, \vec{z}^Q \cdot \underline{\underline{W_L}} \rangle +$

$\langle \vec{y}^n \circ \vec{a}_R, \vec{y}^{-n} \circ (\vec{z}^Q \cdot \underline{\underline{W_R}}) \rangle +$ } merge

$\langle \vec{a}_O, -\vec{y}^n + \vec{z}^Q \cdot \underline{\underline{W_O}} \rangle$ ← $+\delta(y,z) - \delta(y,z)$

$= \langle \vec{a}_L + \vec{y}^{-n} \circ (\vec{z}^Q \cdot \underline{\underline{W_R}}), \vec{y}^n \circ \vec{a}_R \rangle +$

$\langle \vec{a}_L, \vec{z}^Q \underline{\underline{W_L}} \rangle +$

$\langle \vec{y}^{-n} \circ (\vec{z}^Q \cdot \underline{\underline{W_R}}), \vec{z}^Q \cdot \underline{\underline{W_L}} \rangle +$ } merge

$- \delta(y,z)$

$+ \langle \vec{a}_O, -\vec{y}^n + \vec{z}^Q \cdot \underline{\underline{W_O}} \rangle$

$= \langle \vec{a}_L + \vec{y}^{-n} \circ (\vec{z}^Q \cdot \underline{\underline{W_R}}), \vec{y}^n \circ \vec{a}_R \rangle +$ } merge

$\langle \vec{a}_L + \vec{y}^{-n} \circ (\vec{z}^Q \cdot \underline{\underline{W_R}}), \vec{z}^Q \cdot \underline{\underline{W_L}} \rangle +$

$- \delta(y,z) + \langle \vec{a}_O, -\vec{y}^n + \vec{z}^Q \cdot \underline{\underline{W_O}} \rangle$

$\left[\langle \vec{z}^Q, \vec{c} + \underline{\underline{W_V}} \cdot \vec{v} \rangle X^2 = \langle \vec{a}_L + \vec{y}^{-n} \circ (\vec{z}^Q \cdot \underline{\underline{W_R}}), \vec{y}^n \circ \vec{a}_R + \vec{z}^Q \cdot \underline{\underline{W_L}} \rangle X^2 + - \delta(y,z) X^2 + \langle \vec{a}_O, -\vec{y}^n + \vec{z}^Q \cdot \underline{\underline{W_O}} \rangle \cdot X^2 \right]$

BLINDING inner product:

$\vec{a}_L \to \vec{a}_L \cdot X + \vec{s}_L \cdot X^3$ $\quad \vec{a}_O \to \vec{a}_O \cdot X^2$ $\quad \underline{\underline{W_R}} \to \underline{\underline{W_R}} \cdot X$

$\vec{a}_R \to \vec{a}_R \cdot X + \vec{s}_R \cdot X^3$ $\quad \underline{\underline{W_O}} \to \underline{\underline{W_O}} \cdot X^0$ $\quad \underline{\underline{W_L}} \to \underline{\underline{W_L}} \cdot X$

$\langle \vec{z}^Q, \vec{c} + \underline{\underline{W_V}} \cdot \vec{v} \rangle = \langle (\vec{a}_L \cdot X + \vec{s}_L \cdot X^3) + \vec{y}^{-n} \circ (\vec{z}^Q \cdot \underline{\underline{W_R}} \cdot X),$

$\vec{y}^n \circ (\vec{a}_R \cdot X + \vec{s}_R \cdot X^3) + z^Q \cdot \underline{\underline{W_L}} \cdot X \rangle +$

$\langle \vec{a}_O \cdot X^2, -\vec{y}^{-n} + \vec{z}^Q \cdot \underline{\underline{W_O}} \rangle +$

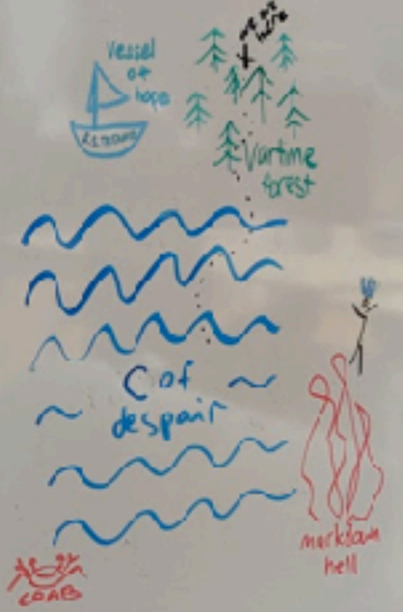
$- \delta(y,z)$ ← how come variable definitions in $\delta(y,z)$ don't change?

↑ maybe they do! $\delta(X^2)$

$\overset{X \;\; X}{\langle a, b \rangle} + \overset{X^2 \;\; X^0}{\langle c, d \rangle} \neq \langle a+c, b+d \rangle$

$=$ 2nd degree: $\langle a \cdot X + c \cdot X^2, b \cdot X + d \cdot X^0 \rangle$ ←

$\langle a, b \rangle X^2 + \langle c, d \rangle X^2 +$



Proving range statements with bit vectors

Let $\mathbf{a}$ be the vector of bits of v. Then v can be represented as an inner product of bits $\mathbf{a}$ and powers of two $\mathbf{2}^n = (1, 2, 4, \ldots, 2^{n-1})$:

$$\begin{aligned} v &= \langle \mathbf{a}, \mathbf{2}^n \rangle \\ &= a_0 \cdot 2^0 + \cdots + a_{n-1} \cdot 2^{n-1}. \end{aligned}$$

We need $\mathbf{a}$ to be a vector of integers $\{0, 1\}$. This can be expressed with an additional condition

$$\mathbf{a} \circ (\mathbf{a} - \mathbf{1}) = \mathbf{0},$$

where $\mathbf{x} \circ \mathbf{y}$ denotes the entry-wise multiplication of two vectors. The result of multiplication can be all-zero if and only if every bit is actually 0 or[1] 1.

As a result of representing value in binary, the range condition $v \in [0, 2^n)$ is equivalent to the pair of conditions

$$\begin{aligned} \langle \mathbf{a}, \mathbf{2}^n \rangle &= v, \\ \mathbf{a} \circ (\mathbf{a} - \mathbf{1}) &= \mathbf{0}. \end{aligned}$$

We will eventually need to make separate commitments to the vectors $\mathbf{a}$ and $\mathbf{a} - \mathbf{1}$, so we set $\mathbf{a}_L = \mathbf{a}$, $\mathbf{a}_R = \mathbf{a} - \mathbf{1}$ to obtain

$$\begin{aligned} \langle \mathbf{a}_L, \mathbf{2}^n \rangle &= v, \\ \mathbf{a}_L \circ \mathbf{a}_R &= \mathbf{0}, \\ (\mathbf{a}_L - \mathbf{1}) - \mathbf{a}_R &= \mathbf{0}. \end{aligned}$$

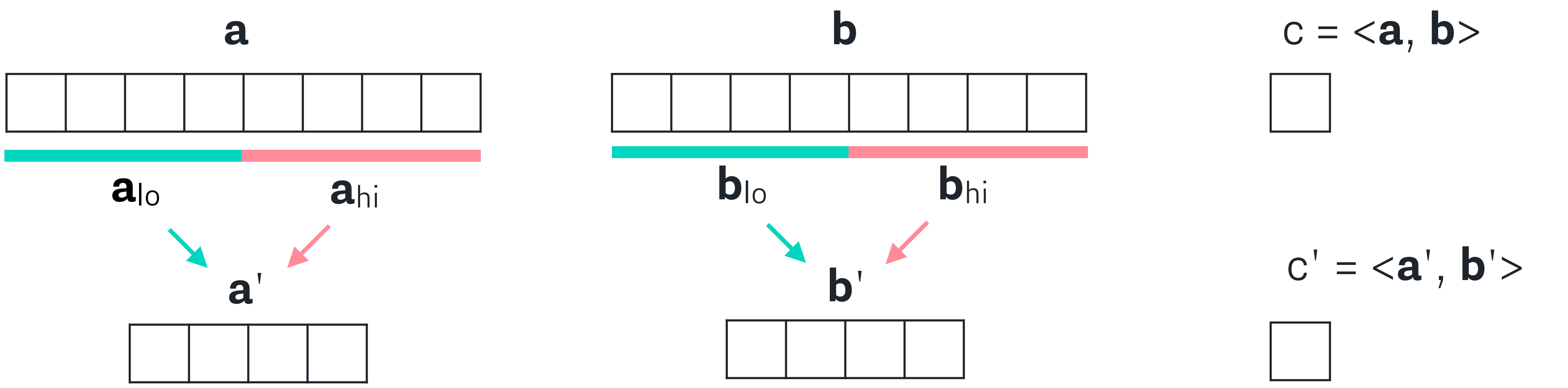
Bulletproof building block: inner products

$$c = \langle \mathbf{a}, \mathbf{b} \rangle$$

$$\boxed{c} \quad = \quad \sum \begin{array}{c|c|c|c|c|c|} \cline{2-6} \mathbf{a} & a_0 & a_1 & a_2 & \dots & a_{n-1} \\ \cline{2-6} \multicolumn{1}{c}{} & \multicolumn{1}{c}{\times} & \multicolumn{1}{c}{\times} & \multicolumn{1}{c}{\times} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{\times} \\ \cline{2-6} \mathbf{b} & b_0 & b_1 & b_2 & \dots & b_{n-1} \\ \cline{2-6} \end{array}$$

We can make a proof that $c = \langle \mathbf{a}, \mathbf{b} \rangle$
in size and $O(\log(n))$ instead of $O(n)$.

Prover gets random challenge scalar x from verifier



$\mathbf{a}' = \mathbf{a}_{lo} \cdot x + \mathbf{a}_{hi} \cdot x^{-1}$

$\mathbf{b}' = \mathbf{b}_{lo} \cdot x^{-1} + \mathbf{b}_{hi} \cdot x$

$c' = \langle \mathbf{a}_{lo} \cdot x + \mathbf{a}_{hi} \cdot x^{-1}, \mathbf{b}_{lo} \cdot x^{-1} + \mathbf{b}_{hi} \cdot x \rangle$

...

...

The proof size is O(log(n)) instead of O(n).

Bulletproof range proofs

How do we express a range as an inner product?

$$0 \leq v < 2^n \longrightarrow \text{apply math \& crypto} \longrightarrow c = \langle \mathbf{a}, \mathbf{b} \rangle$$

Range statement → inner product

We want to prove:

$$0 \leq v < 2^n$$

If this is true, then v must be a binary number of length n.

$$v = \sum \begin{array}{c} \boxed{0}\boxed{1}\boxed{1}\boxed{1} \\ \text{x} \\ \boxed{2^3}\boxed{2^2}\boxed{2^1}\boxed{2^0} \end{array}$$

If v=7, n=4

Range statement → inner product

We want to prove:

$$0 \le v < 2^n$$

If this is true, then v must be a binary number of length n.

$$v = \sum \begin{array}{c} \boxed{0 \mid 1 \mid 1 \mid 1} \\ \times \\ \boxed{2^3 \mid 2^2 \mid 2^1 \mid 2^0} \end{array}$$

Let's call this $\mathbf{a}_L$

$$v = \langle \mathbf{a}_L, \mathbf{2}^n \rangle$$

Range statement → inner product

We want to prove:

$$0 \leq v < 2^n$$

We can do this by proving:

1) $v = \langle \mathbf{a}_L, \mathbf{2}^n \rangle$ ← binary structure of v

2) $\mathbf{a}_L \circ (\mathbf{a}_L - \mathbf{1}^n) = \mathbf{0}^n$ ← bits are actually bits (0s or 1s)

Range statement → inner product

$$0 \le v < 2^n$$

$\longrightarrow$ $v = \langle \mathbf{a}_L, \mathbf{2}^n \rangle$

$\mathbf{a}_L \circ (\mathbf{a}_L - \mathbf{1}^n) = \mathbf{0}^n$

$\longrightarrow$ Add blinding factors

Combine statements

$\longrightarrow$ $c = \langle \mathbf{a}, \mathbf{b} \rangle$

Want more details?

I’m giving a “Bulletproofs deep dive” talk at DEFCON too!
Sunday 8/11 at **11:45am** at **Monero Village**

The Bulletproofs paper:
`https://eprint.iacr.org/2017/1066.pdf`

Our notes on the Bulletproofs math:
`https://doc-internal.dalek.rs/bulletproofs/notes/index.html`

The story

1. Motivation
2. Understanding the paper
3. **Tools of the trade**

The code

4. Range proof
5. Rust tricks
6. Above and beyond

We need to build:

- Prime order group
- Fiat-Shamir heuristic

Bulletproofs requires a prime-order group

2.4 Notation

Let $\mathbb{G}$ denote a cyclic group of prime order p,

Sounds good, but how do you actually implement this?

What kind of elliptic curve should we use?

	Edwards e.g. Curve25519, FourQ	Weierstrass e.g. secp256k1
fastest formulas	✔	✘
complete formulas	✔	✘
easy in constant time	✔	✘
prime-order group	✘	✔

Examples of cofactor problems

Ed25519 signature verification **differs between single and batch** verification

As specified in the RFC, the set of valid signatures is not defined!

Onion Service addresses in Tor had to add **extra validation.**

Cofactor problem: 8 addresses for the same server.

Monero had a **critical vulnerability** due to cofactors.

Cofactor problem: allowed **spending the same amount** 8 times.

Decaf & Ristretto: the best of both worlds

- **Decaf** - Mike Hamburg '15
  - Cofactor 4 reduction
- **Ristretto** - Mike Hamburg, Henry de Valence
  - Cofactor 8 reduction
  - Curve25519 has **cofactor 8**

Decaf: https://eprint.iacr.org/2015/673.pdf

Ristretto: https://ristretto.group

Curve25519 is fast!

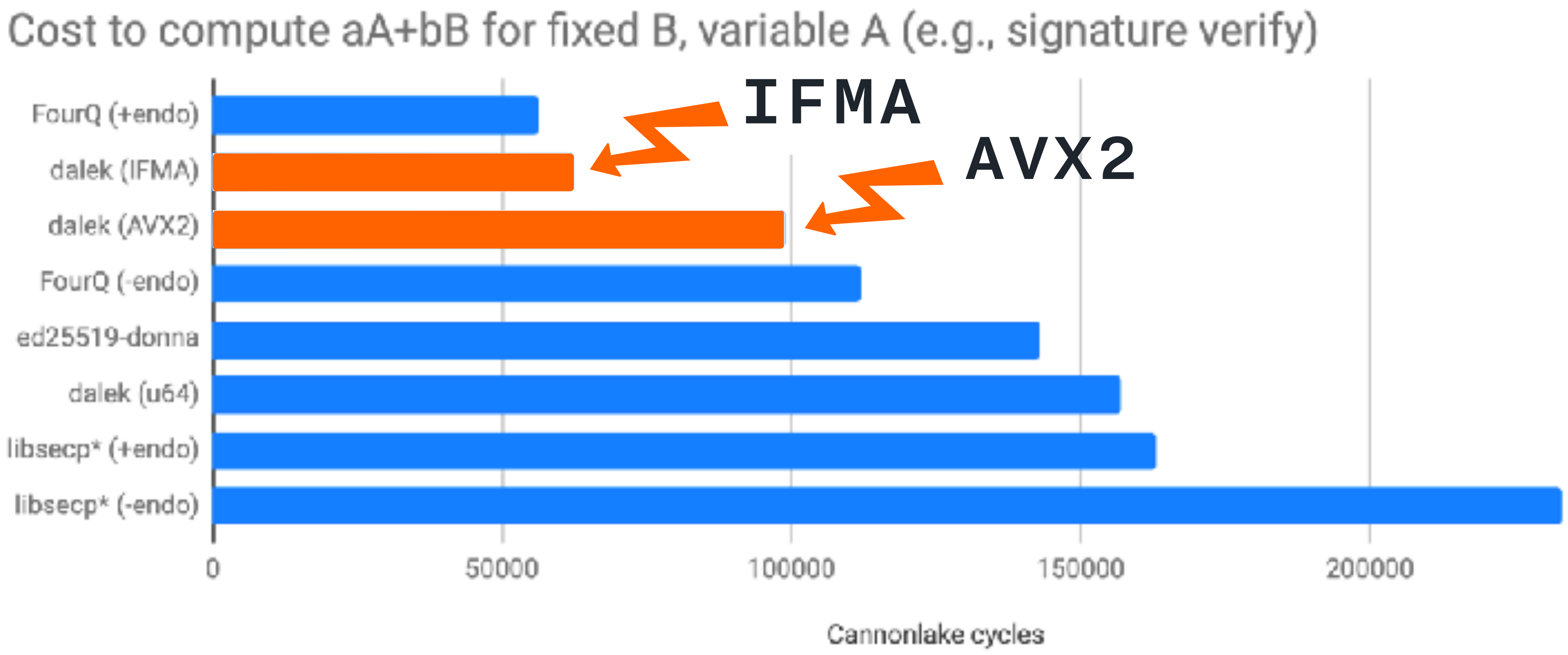
- Curve25519 has **cofactor 8**
- Hisil, Wong, Carter, Dawson ’08 introduced fast **parallel formulas** for Curve25519
- **Curve25519-dalek** is a fast, pure-Rust AVX2 implementation of those formulas

curve25519-dalek: https://doc-internal.dalek.rs/curve25519_dalek/backend/avx2/index.html

HWCD: https://www.iacr.org/archive/asiacrypt2008/53500329/53500329.pdf

Blog post: https://medium.com/@hdevalence/accelerating-edwards-curve-arithmetic-with-parallel-formulas-ac12cf5015be

Is this strategy fast? Yes!



ristretto255: a prime-order group up to **4x faster** than secp256k1.

The Fiat-Shamir Heuristic

Converts an **interactive** argument into a **non-interactive** one.

Idea: replace a verifier's **random challenges** with a **hash** of the prover's messages.

$$\mathcal{P}_{\mathsf{IP}} \to \mathcal{V}_{\mathsf{IP}} : L, R$$

$$\mathcal{V}_{\mathsf{IP}} : x \xleftarrow{\$} \mathbb{Z}_p^\star$$

$$\mathcal{V}_{\mathsf{IP}} \to \mathcal{P}_{\mathsf{IP}} : x$$

Sounds good, but how do you actually implement this?

Hashing data is kind of complicated!

What if you **forget** to feed data into the hash?

What if your data is **ambiguously encoded** in the hash?

How do you handle **multi-round protocols**?

Where do you put **domain separators**?

… and many more edge cases.

What if there was a first-class transcript object?

Paper

$$\mathcal{P}_{\mathsf{IP}} \rightarrow \mathcal{V}_{\mathsf{IP}} : L, R$$

$$\mathcal{V}_{\mathsf{IP}} : x \xleftarrow{\$} \mathbb{Z}_p^\star$$
$$\mathcal{V}_{\mathsf{IP}} \rightarrow \mathcal{P}_{\mathsf{IP}} : x$$

Implementation

```
transcript.commit_point(b"L", L);
transcript.commit_point(b"R", R);
```

```
let x = transcript.challenge_scalar(b"x");
```

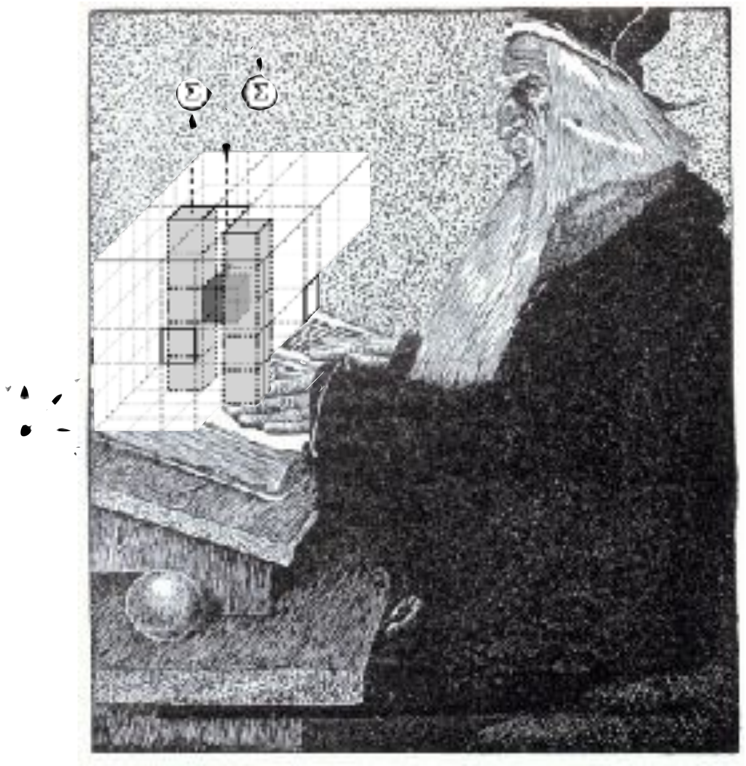
Merlin: STROBE-based transcripts for ZKPs

Implement protocols **as if they were interactive**, passing a **transcript** parameter.
Transformation is done in software, not by hand.

Byte-oriented API, automatic **message framing**.

Easy **domain separation**.

Automatic sequential **composition of proofs**.



https://merlin.cool

The story

1. Motivation
2. Understanding the paper
3. Tools of the trade

The code

4. Range proof
5. Rust tricks
6. Above and beyond

https://github.com/dalek-cryptography/bulletproofs

The story

1 Motivation

2 Understanding the paper

3 Tools of the trade

The code

4 **Range proof**

5 Rust tricks

6 Above and beyond

Proving

$$\mathbf{a}_L \in \{0,1\}^n \text{ s.t. } \langle \mathbf{a}_L, \mathbf{2}^n \rangle = v$$

$$\mathbf{a}_R - \mathbf{a}_L - \mathbf{1}^n$$

$$\alpha \xleftarrow{\$} \mathbb{Z}_p$$

$$A = h^{\alpha} \mathbf{g}^{\mathbf{a}_L} \mathbf{h}^{\mathbf{a}_R}$$

page 17, lines 36-45 of the Bulletproofs paper

Proving: paper & code

```
let alpha = Scalar::random(rng);
let A = h * alpha + msm(g_vec, a_L) + msm(h_vec, a_R);
```

Pseudocode of implementation: `src/range_proof/party.rs` lines 84-110 and `src/range_proof/dealer.rs` lines 100-108

$$\mathbf{a}_L \in \{0,1\}^n \text{ s.t. } \langle \mathbf{a}_L, \mathbf{2}^n \rangle = v$$
$$\mathbf{a}_R - \mathbf{a}_L - \mathbf{1}^n$$
$$\alpha \xleftarrow{\$} \mathbb{Z}_p$$
$$A = h^\alpha \mathbf{g}^{\mathbf{a}_L} \mathbf{h}^{\mathbf{a}_R}$$

$$\mathbf{s}_L, \mathbf{s}_R \xleftarrow{\$} \mathbb{Z}_p^n$$
$$\rho \xleftarrow{\$} \mathbb{Z}_p$$
$$S = h^\rho \mathbf{g}^{\mathbf{s}_L} \mathbf{h}^{\mathbf{s}_R}$$

page 17, lines 36-45 of the Bulletproofs paper

Proving: paper & code

```
let alpha = Scalar::random(rng);
let A = h * alpha + msm(g_vec, a_L) + msm(h_vec, a_R);


let s_L = (0..n).map(|_| Scalar::random(rng).collect());
let s_R = (0..n).map(|_| Scalar::random(rng).collect());
let rho = Scalar::random(rng);
let S = h * rho + msm(g_vec, s_L) + msm(h_vec, s_R);
```

Pseudocode of implementation: `src/range_proof/party.rs` lines 84-110 and `src/range_proof/dealer.rs` lines 100-108

$$\mathbf{a}_L \in \{0,1\}^n \text{ s.t. } \langle \mathbf{a}_L, \mathbf{2}^n \rangle = v$$

$$\mathbf{a}_R - \mathbf{a}_L - \mathbf{1}^n$$

$$\alpha \xleftarrow{\$} \mathbb{Z}_p$$

$$A = h^{\alpha} \mathbf{g}^{\mathbf{a}_L} \mathbf{h}^{\mathbf{a}_R}$$

$$\mathbf{s}_L, \mathbf{s}_R \xleftarrow{\$} \mathbb{Z}_p^n$$

$$\rho \xleftarrow{\$} \mathbb{Z}_p$$

$$S = h^{\rho} \mathbf{g}^{\mathbf{s}_L} \mathbf{h}^{\mathbf{s}_R}$$

$$\mathcal{P} \rightarrow \mathcal{V} : A, S$$

$$\mathcal{V} : y, z \xleftarrow{\$} \mathbb{Z}_p^{\star}$$

$$\mathcal{V} \rightarrow \mathcal{P} : y, z$$

page 17, lines 36-45 of the Bulletproofs paper

Proving: paper & code

```
let alpha = Scalar::random(rng);
let A = h * alpha + msm(g_vec, a_L) + msm(h_vec, a_R);


let s_L = (0..n).map(|_| Scalar::random(rng).collect());
let s_R = (0..n).map(|_| Scalar::random(rng).collect());
let rho = Scalar::random(rng);
let S = h * rho + msm(g_vec, s_L) + msm(h_vec, s_R);


transcript.commit_point(b"A", A);
transcript.commit_point(b"S", S);
let y = transcript.challenge_scalar(b"y");
let z = transcript.challenge_scalar(b"z");
```

Pseudocode of implementation: `src/range_proof/party.rs` lines 84-110 and `src/range_proof/dealer.rs` lines 100-108

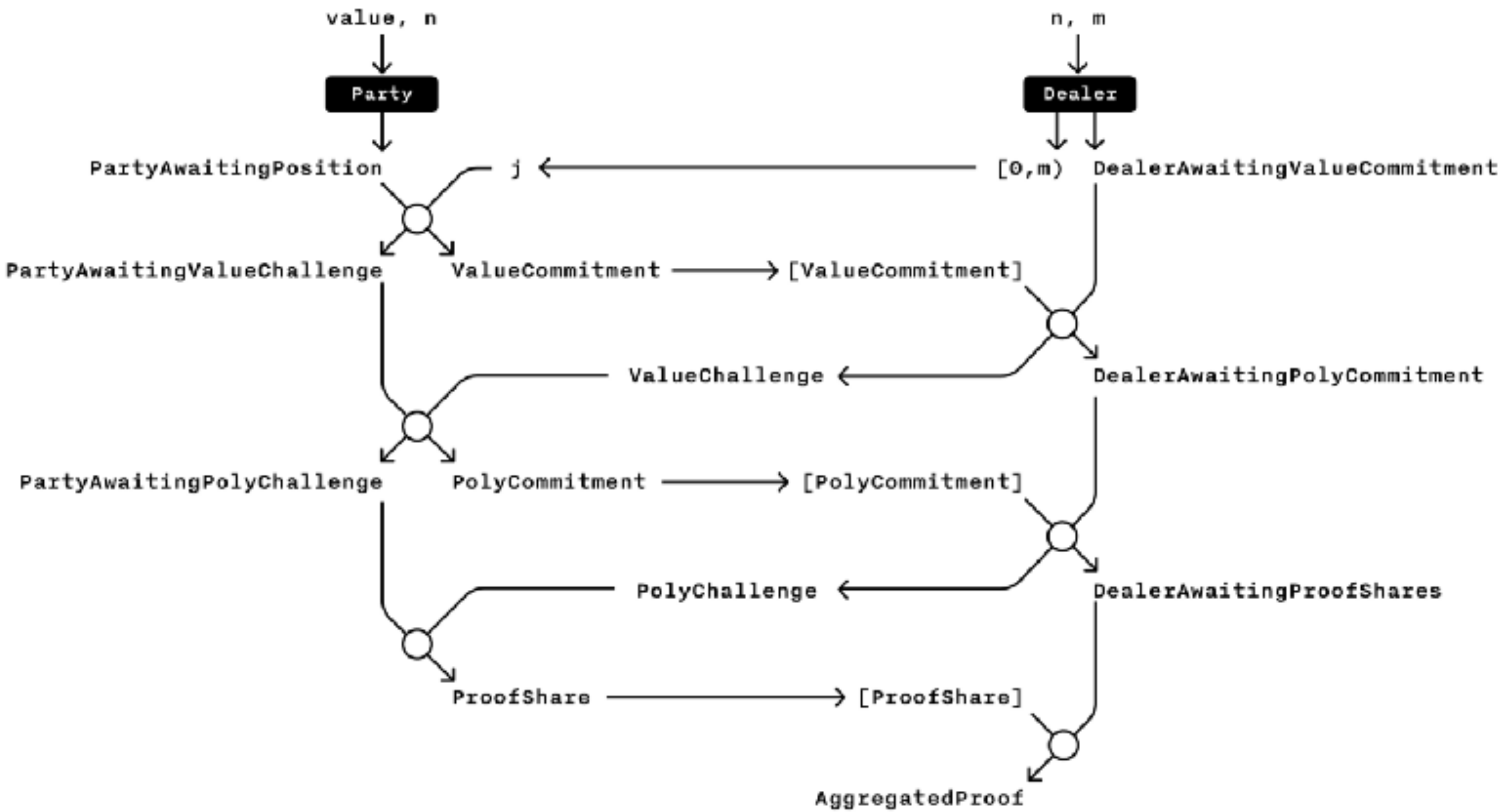
The story

1. Motivation
2. Understanding the paper
3. Tools of the trade

The code

4. Range proof
5. **Rust tricks**
6. Above and beyond

Session types for MPC



Blog post: https://blog.chain.com/bulletproof-multi-party-computation-in-rust-with-session-types-b3da6e928d5d

Optimizations

Rust iterators

Lazy and zero-cost*

- Can build up points & scalars using Rust iterators & pass them into the multiscalar API to inline computation
- Don’t have to do extra allocations or manage temporaries

$$Q = c_1 P_1 + \cdots + c_n P_n$$

* except for build times

Performance of 64-bit rangeproof verification

<1 millisecond, with SIMD backends in curve25519-dalek

IFMA

3x faster than libsecp256k1, **7x** faster than Monero.

AVX2

2x faster than libsecp256k1, **4.6x** faster than Monero.

The story

1. Motivation
2. Understanding the paper
3. Tools of the trade

The code

4. Range proof
5. Rust tricks
6. **Above and beyond**

Constraint System API

for fully programmable proofs

Constraints

Multiplicative constraint (secret-secret multiplication):

$$x \cdot y = z$$

Linear constraint (secret variables with cleartext weights):

$$a \cdot x + b \cdot y + c \cdot z + \ldots = 0$$

Why constraint systems?

A constraint system can represent
any efficiently verifiable program.

A **CS proof** is proof that all the constraints
are **satisfied** by certain **secret** inputs.

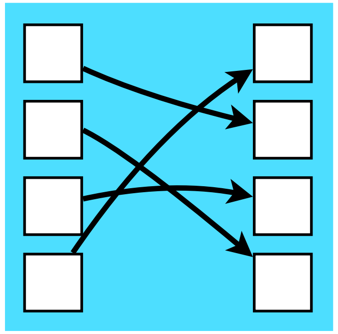
FURTHER READING

https://medium.com/interstellar/programmable-constraint-systems-for-bulletproofs-365b9feb92f7

Cloak:
a confidential assets protocol

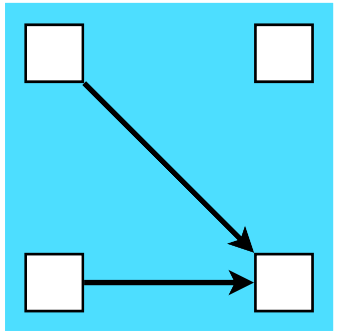
Composition of gadgets in Cloak

Cloak transaction is a combination of smaller gadgets with different roles.



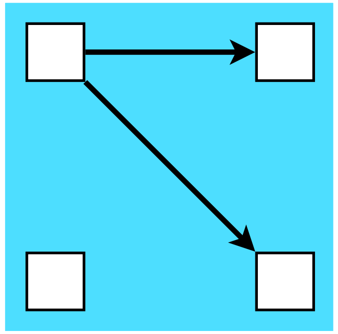
SHUFFLE

Secretly **reorder** N values.



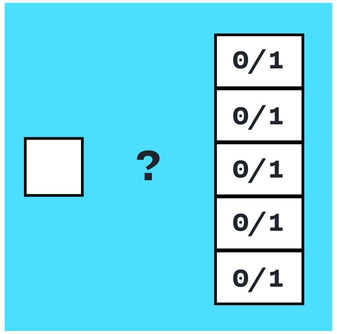
MERGE

Secretly **merge or move** two values.



SPLIT

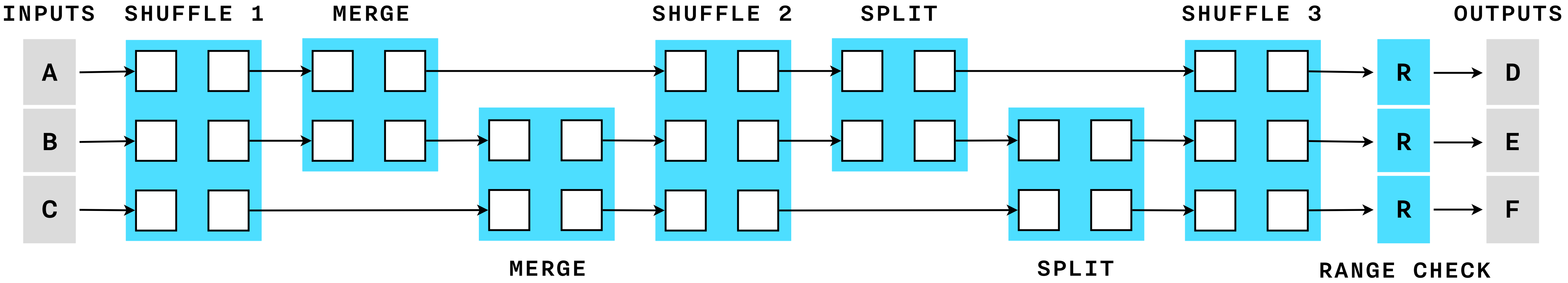
Secretly **split or move** two values.



RANGE

Check that value is **not negative**.

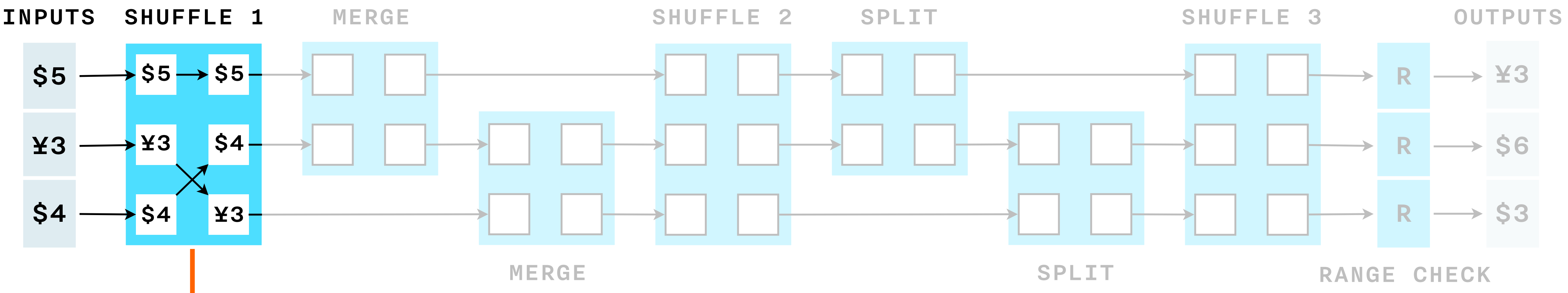
Cloak transaction



Observers cannot tell where values are actually **split**, **merged** or **moved** without modification.

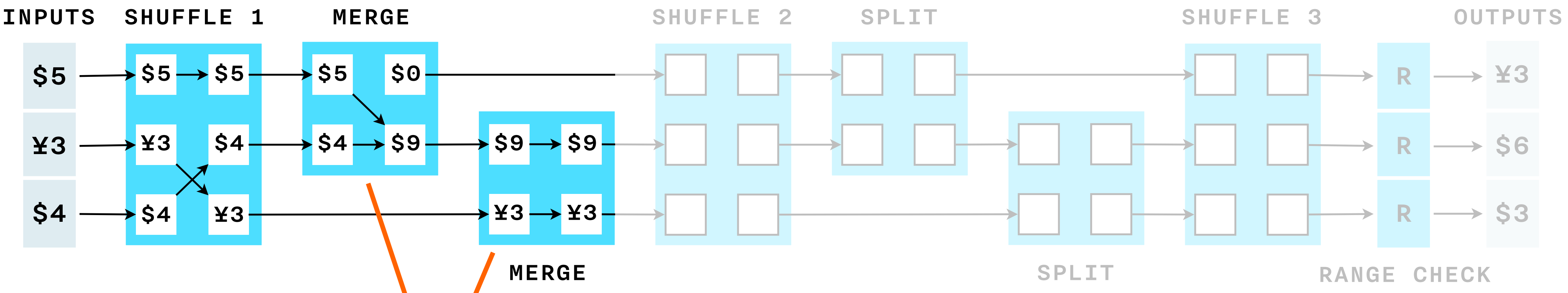
Only the prover knows where values are **modified** or **moved**.

Cloak walkthrough



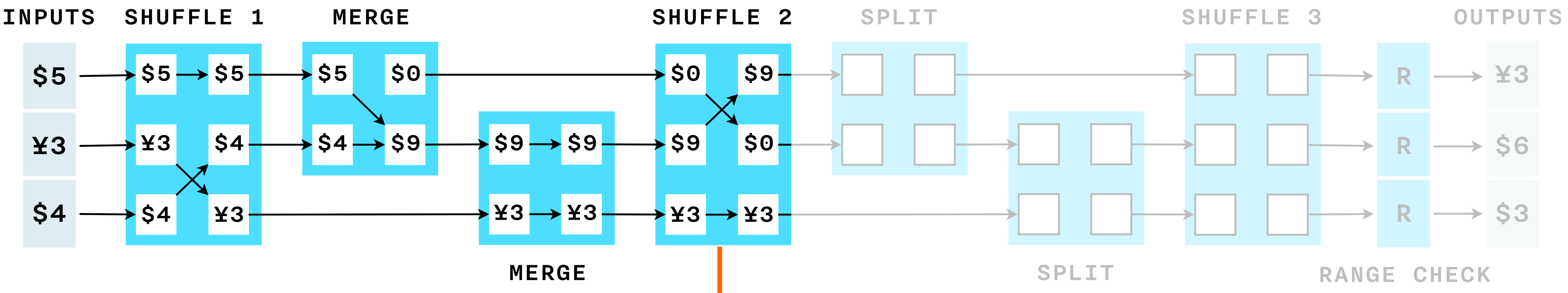
Randomly ordered input values are grouped by asset type.

Cloak walkthrough



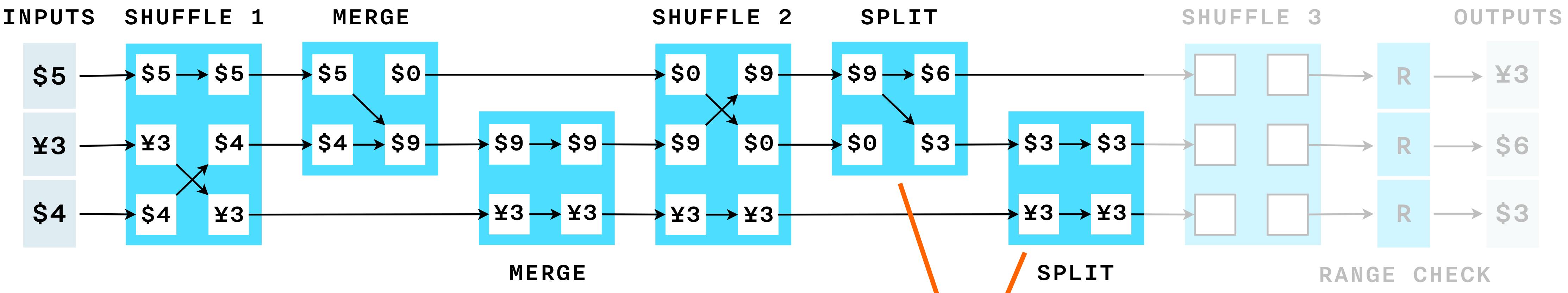
Values of the same asset type
are fully merged together.

Cloak walkthrough



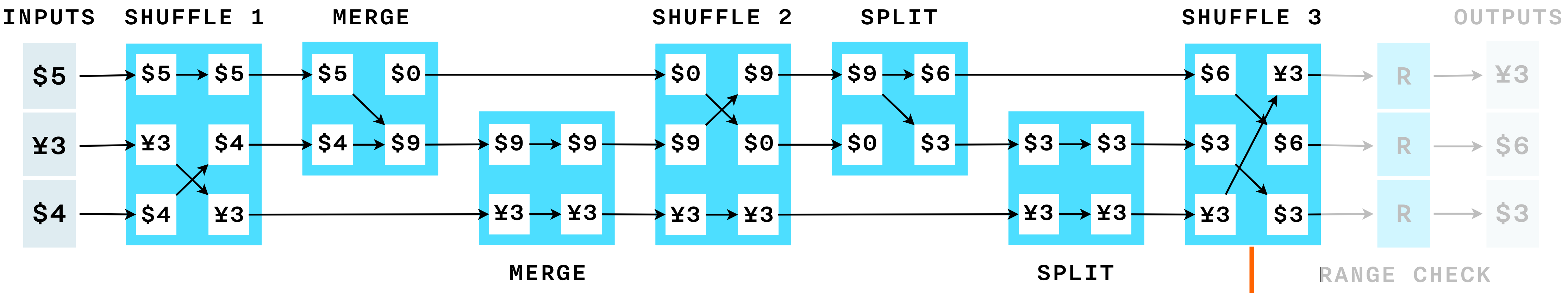
Non-zero values are reordered to the top, still grouped by asset type.

Cloak walkthrough



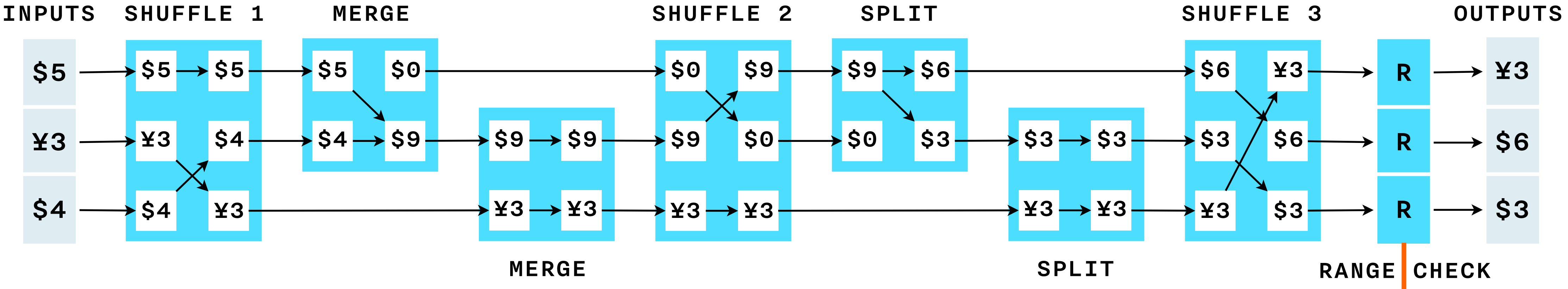
Values are split into target payment amounts.

Cloak walkthrough



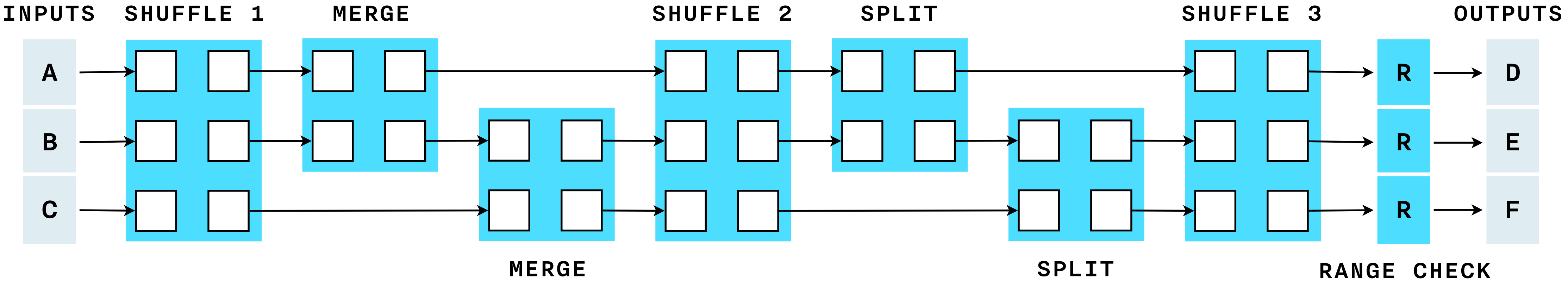
Values that were grouped by asset type are shuffled into a random order.

Cloak walkthrough



All values are checked to be non-negative.

Complete 3:3 Cloak transaction



Transactions of the same size are **indistinguishable.**

SPEC & CODE

https://github.com/stellar/slingshot/spacesuit

ZkVM:

a zero-knowledge

smart contract language

https://github.com/stellar/slingshot/ZkVM

Thanks!

Henry de Valence

@hdevalence

Oleg Andreev

@oleganza

George Tankersley

@gtank__

Deirdre Connolly

@durumcrustulum

Further Reading

Cathie Yun

@cathieyun

Bulletproofs paper:
https://eprint.iacr.org/2017/1066.pdf

Open-source GitHub repo for Bulletproofs in Rust:
https://github.com/dalek-cryptography/bulletproofs

Notes on the Bulletproofs math & implementation docs:
https://doc.dalek.rs/bulletproofs/index.html

Slide deck:
https://speakerdeck.com/cathieyun