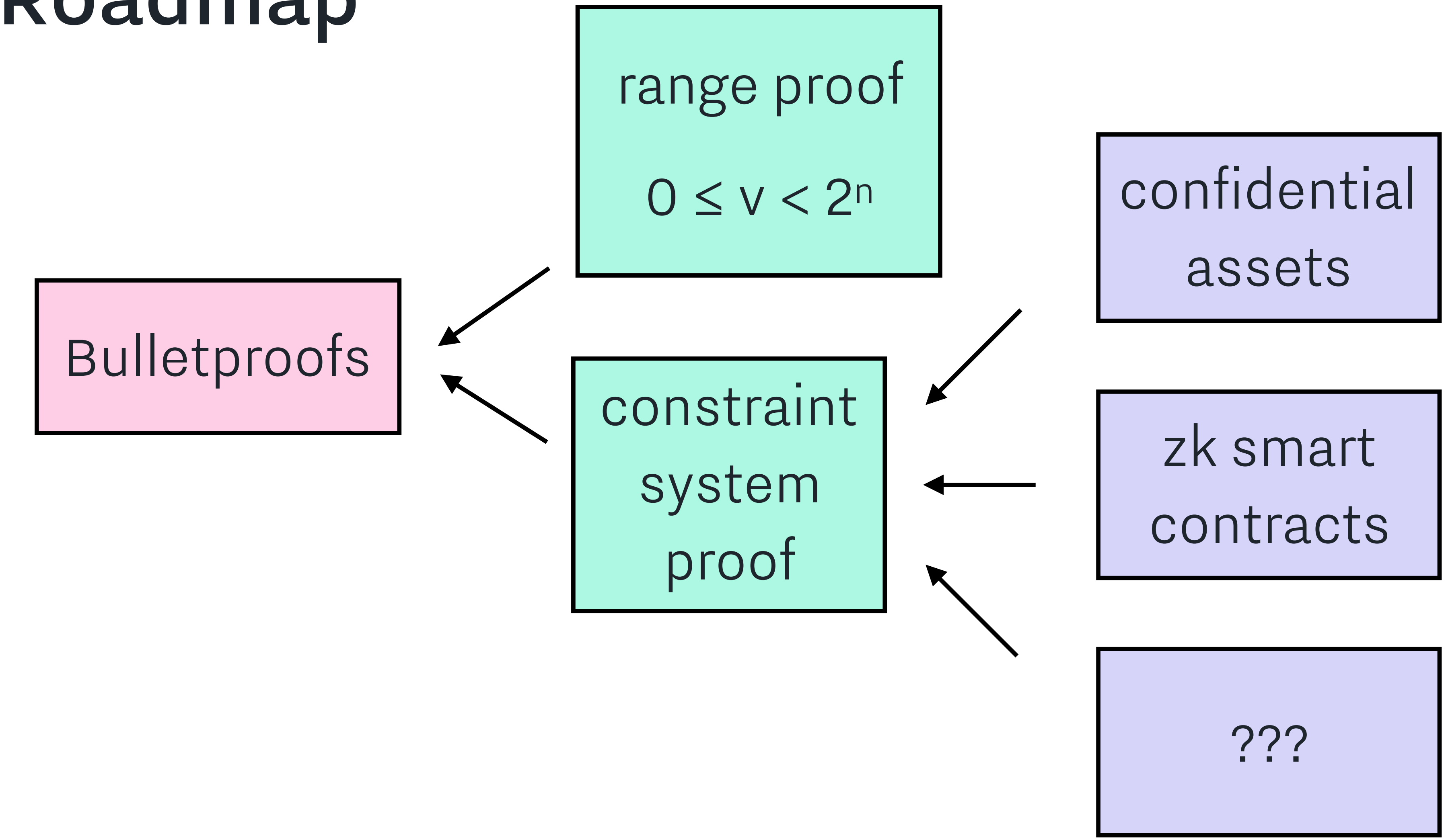
R1CS and smart contracts with Bulletproofs

Cathie Yun, Interstellar

April 10, 2019 | ZKProof Workshop

Roadmap



Bulletproofs

range proof

$0 \leq v < 2^n$

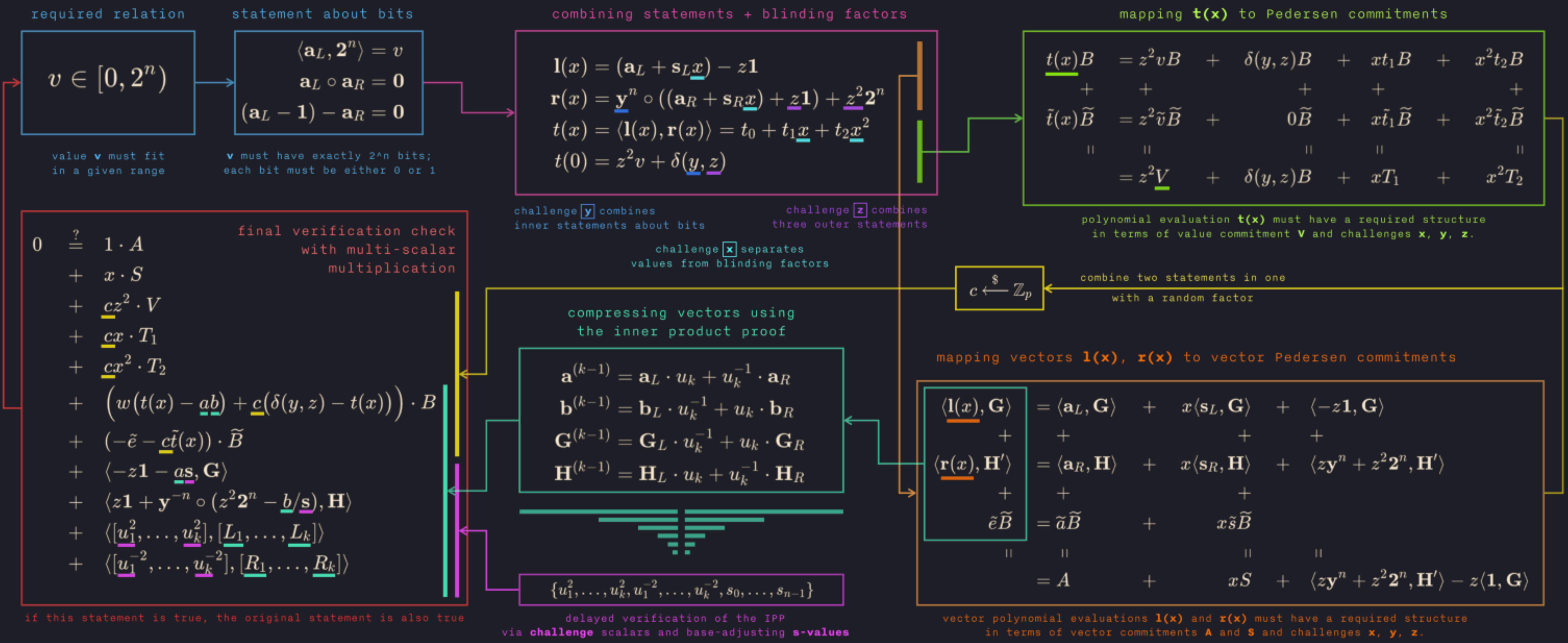
constraint system proof

confidential assets

zk smart contracts

???

Range proof



required relation

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

value **v** must fit in a given range

statement about bits

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

v must have exactly 2^n bits; each bit must be either 0 or 1

combining statements + blinding factors

$$\begin{aligned} \mathbf{l}(x) &= (\mathbf{a}_L + \mathbf{s}_L x) - z\mathbf{1} \\ \mathbf{r}(x) &= \mathbf{y}^n \circ ((\mathbf{a}_R + \mathbf{s}_R x) + z\mathbf{1}) + z^2\mathbf{2}^n \\ t(x) &= \langle \mathbf{l}(x), \mathbf{r}(x) \rangle = t_0 + t_1 x + t_2 x^2 \\ t(0) &= z^2 v + \delta(y, z) \end{aligned}$$

challenge $\boxed{y}$ combines inner statements about bits

challenge $\boxed{z}$ combines three outer statements

challenge $\boxed{x}$ separates values from blinding factors

mapping **t(x)** to Pedersen commitments

$$\begin{array}{rcccccccc} t(x)B & = z^2 vB & + & \delta(y,z)B & + & xt_1 B & + & x^2 t_2 B \\ + & + & & + & & + & & + \\ \tilde{t}(x)\widetilde{B} & = z^2 \tilde{v}\widetilde{B} & + & 0\widetilde{B} & + & x\tilde{t}_1\widetilde{B} & + & x^2\tilde{t}_2\widetilde{B} \\ \shortparallel & \shortparallel & & \shortparallel & & \shortparallel & & \shortparallel \\ & = z^2 V & + & \delta(y,z)B & + & xT_1 & + & x^2 T_2 \end{array}$$

polynomial evaluation **t(x)** must have a required structure in terms of value commitment **V** and challenges **x**, **y**, **z**.

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

combine two statements in one with a random factor

final verification check with multi-scalar multiplication

$$\begin{array}{lll} 0 & \overset{?}{=} & 1 \cdot A \\ & + & x \cdot S \\ & + & cz^2 \cdot V \\ & + & cx \cdot T_1 \\ & + & cx^2 \cdot T_2 \\ & + & \Big(w\big(t(x) - ab\big) + c\big(\delta(y,z) - t(x)\big)\Big) \cdot B \\ & + & (-\tilde{e} - c\tilde{t}(x)) \cdot \widetilde{B} \\ & + & \langle -z\mathbf{1} - a\mathbf{s}, \mathbf{G} \rangle \\ & + & \langle z\mathbf{1} + \mathbf{y}^{-n} \circ (z^2\mathbf{2}^n - b/\mathbf{s}), \mathbf{H} \rangle \\ & + & \langle [u_1^2, \dots, u_k^2], [L_1, \dots, L_k] \rangle \\ & + & \langle [u_1^{-2}, \dots, u_k^{-2}], [R_1, \dots, R_k] \rangle \end{array}$$

if this statement is true, the original statement is also true

compressing vectors using the inner product proof

$$\begin{aligned} \mathbf{a}^{(k-1)} &= \mathbf{a}_L \cdot u_k + u_k^{-1} \cdot \mathbf{a}_R \\ \mathbf{b}^{(k-1)} &= \mathbf{b}_L \cdot u_k^{-1} + u_k \cdot \mathbf{b}_R \\ \mathbf{G}^{(k-1)} &= \mathbf{G}_L \cdot u_k^{-1} + u_k \cdot \mathbf{G}_R \\ \mathbf{H}^{(k-1)} &= \mathbf{H}_L \cdot u_k + u_k^{-1} \cdot \mathbf{H}_R \end{aligned}$$

$$\{u_1^2, \dots, u_k^2, u_1^{-2}, \dots, u_k^{-2}, s_0, \dots, s_{n-1}\}$$

delayed verification of the IPP via **challenge** scalars and base-adjusting **s-values**

mapping vectors **l(x)**, **r(x)** to vector Pedersen commitments

$$\begin{array}{rcccccc} \langle \mathbf{l}(x), \mathbf{G} \rangle & = \langle \mathbf{a}_L, \mathbf{G} \rangle & + & x\langle \mathbf{s}_L, \mathbf{G} \rangle & + & \langle -z\mathbf{1}, \mathbf{G} \rangle \\ + & + & & + & & + \\ \langle \mathbf{r}(x), \mathbf{H}' \rangle & = \langle \mathbf{a}_R, \mathbf{H} \rangle & + & x\langle \mathbf{s}_R, \mathbf{H} \rangle & + & \langle z\mathbf{y}^n + z^2\mathbf{2}^n, \mathbf{H}' \rangle \\ + & + & & + & & \\ \tilde{e}\widetilde{B} & = \tilde{a}\widetilde{B} & + & x\tilde{s}\widetilde{B} & & \\ \shortparallel & \shortparallel & & \shortparallel & & \shortparallel \\ & = A & + & xS & + & \langle z\mathbf{y}^n + z^2\mathbf{2}^n, \mathbf{H}' \rangle - z\langle \mathbf{1}, \mathbf{G} \rangle \end{array}$$

vector polynomial evaluations **l(x)** and **r(x)** must have a required structure in terms of vector commitments **A** and **S** and challenges **x**, **y**, **z**.

Performance of 64-bit range proof verification

with SIMD backends in curve25519-dalek

IFMA = 0.7 milliseconds

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

AVX2 = 1.04 milliseconds

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

Constraint system (R1CS) proofs

What they are, and an example of how to use them

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

Extension: using challenges

Bulletproofs allows for constraint system construction with **no setup.**

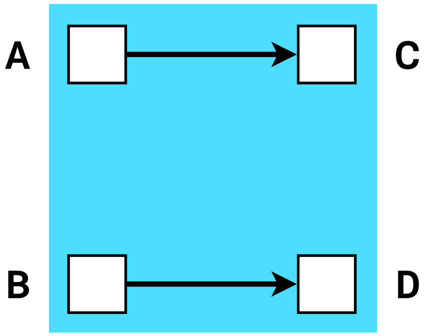
This allows us to select a circuit from a family parameterized by **challenges.**

- Get & use **random challenge scalars** from commitments to variables**.**
- Make **smaller & more efficient** constraint systems (e.g. shuffle)

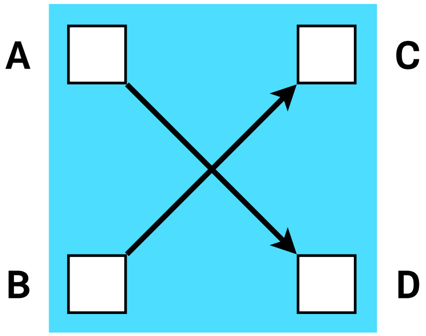
Currently **under research**.

Shuffle gadget

Permutation is secret and values are preserved.



OR

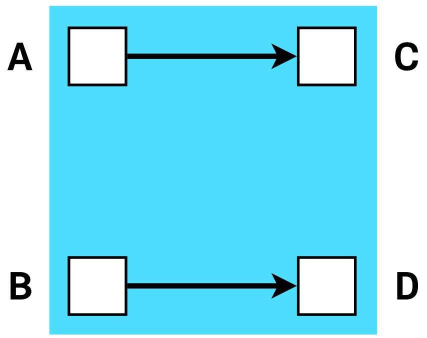


A = C

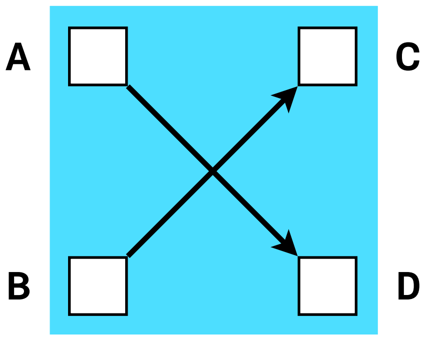
B = D

A = D

B = C



OR



$$(A - x)\cdot(B - x) = (C - x)\cdot(D - x)$$

Uses **equality of polynomials** when roots are permuted.

If the equation holds for random **x** then
{A,B} must equal **{C,D}** in any order.

```
pub fn two_shuffle<CS: ConstraintSystem>(
    cs: &mut CS, A: Variable, B: Variable, C: Variable, D: Variable,
) -> Result<(), R1CSError> {
    cs.specify_randomized_constraints(move |cs| {
        // Get challenge scalar x
        let x = cs.challenge_scalar(b"shuffle challenge");
        // (A - x)*(B - x) = input_mul
        let (_, _, input_mul) = cs.multiply(A - x, B - x);
        // (C - x)*(D - x) = output_mul
        let (_, _, output_mul) = cs.multiply(C - x, D - x);
        // input_mul - output_mul = 0
        cs.constrain(input_mul - output_mul);
        Ok(())
    })
}
```

```rust
// Make a prover instance
let mut prover = Prover::new(&bpgens, &pcgens, &mut transcript);

// Create commitments and allocate high-level variables for A, B, C, D
let mut rng = rand::thread_rng();
let (A_com, A_var) = prover.commit(A, Scalar::random(&mut rng));
let (B_com, B_var) = prover.commit(B, Scalar::random(&mut rng));
let (C_com, C_var) = prover.commit(C, Scalar::random(&mut rng));
let (D_com, D_var) = prover.commit(D, Scalar::random(&mut rng));

// Add 2-shuffle gadget constraints to the prover's constraint system
two_shuffle(&mut prover, A_var, B_var, C_var, D_var)?;

// Create a proof
let proof = prover.prove()?;
```

```rust
// Make a verifier instance
let mut verifier = Verifier::new(&bpgens, &pcgens, &mut transcript);

// Allocate high-level variables for A, B, C, D from commitments
let A_var = verifier.commit(A_com);
let B_var = verifier.commit(B_com);
let C_var = verifier.commit(C_com);
let D_var = verifier.commit(D_com);

// Add 2-shuffle gadget constraints to the verifier's constraint system
two_shuffle(&mut verifier, A_var, B_var, C_var, D_var)?;

// Verify the proof
verifier.verify(&proof)
```

FULL SAMPLE CODE
https://github.com/interstellar/spacesuit/blob/2-shuffle/src/gadgets/two_shuffle.rs

Constraint System API

commit: makes high-level variables. (not in Constraint System trait).

allocate: makes low-level variables using a multiplication gate

input: scalar assignments; **output**: left, right, output variables

constrain: enforces that a linear combination equals zero

input: linear combination

multiply: makes low-level variables using a multiplication gate

input: linear combinations; **output**: left, right, output variables

specify_randomized_constraints: allow the use of challenges

input: closure in which user can generate one or more challenges

Recap: constraint system proofs

API for **multiplicative** and **linear** constraints

Protocol extension for making **challenges**

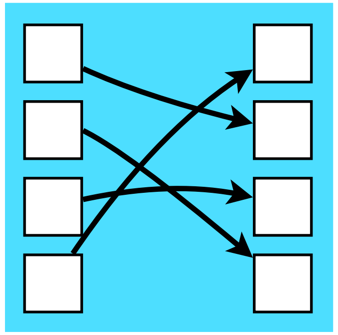
Shuffle gadget using constraint API and challenges

Cloak

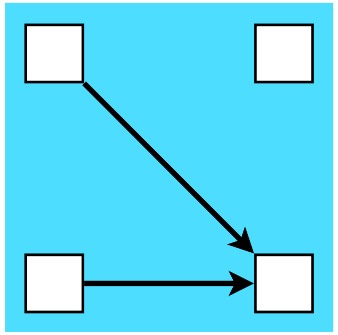
Confidential assets with Bulletproofs

Composition of gadgets in Cloak

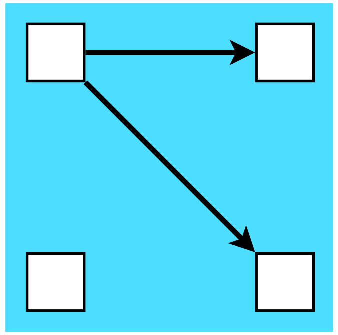
Cloak transaction is a combination of smaller gadgets with different roles.



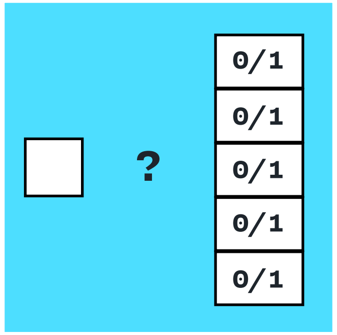
SHUFFLE



MERGE



SPLIT



RANGE

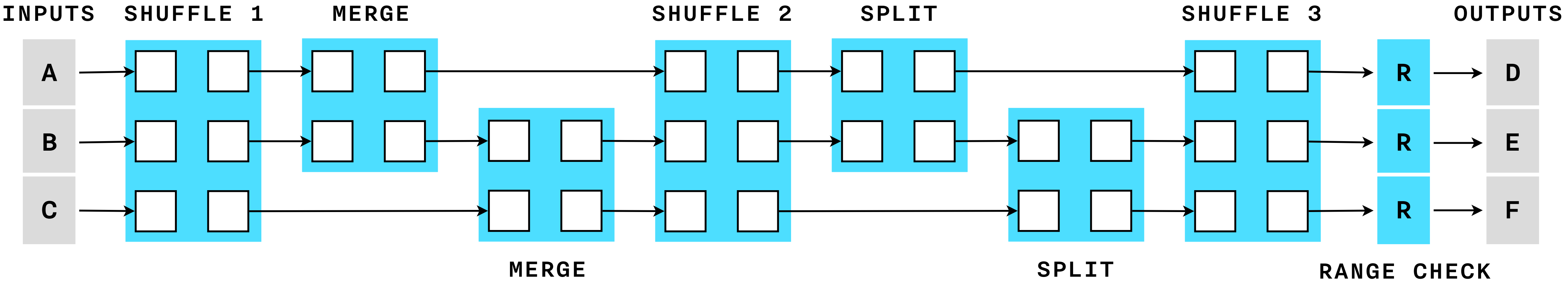
Secretly **reorder** N values.

Secretly **merge or move** two values.

Secretly **split or move** two values.

Check that value is **not negative**.

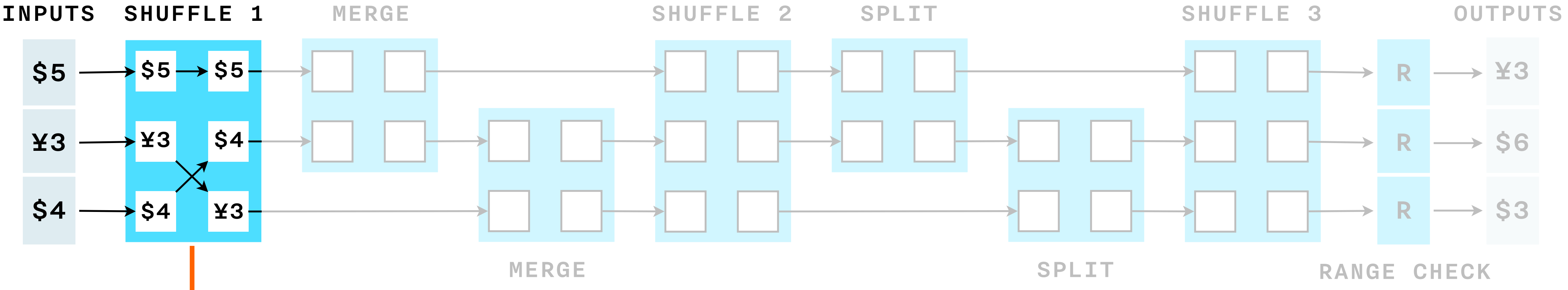
Cloak transaction



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

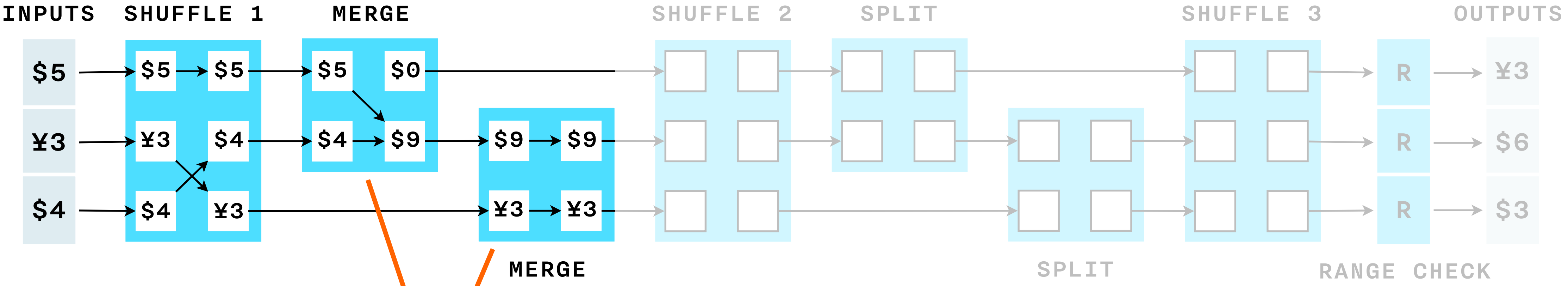
Only the prover knows where values are **modified** or **moved**.

Cloak walkthrough



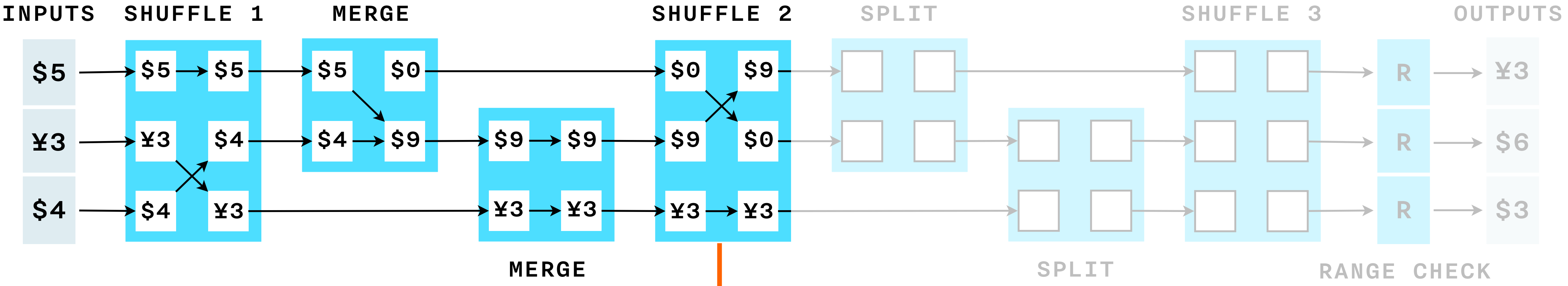
Randomly ordered input values are grouped by asset type.

Cloak walkthrough



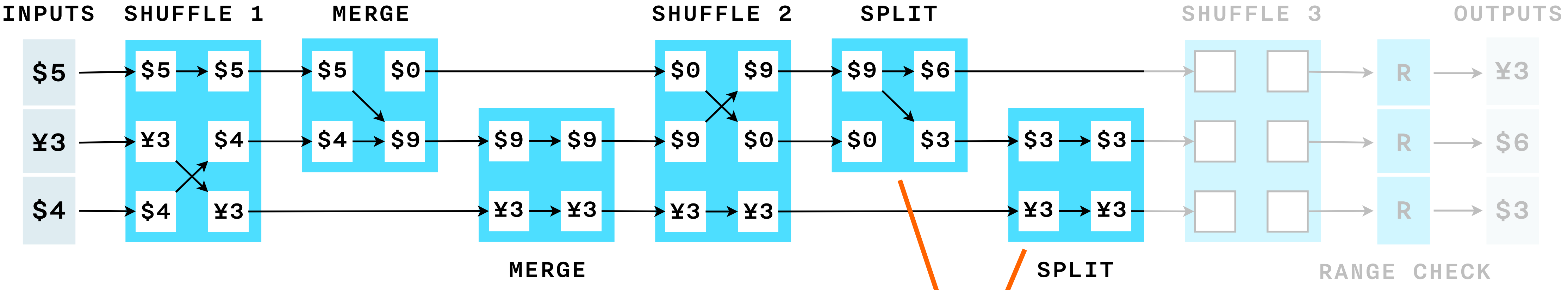
Values of the same asset type are fully merged together.

Cloak walkthrough



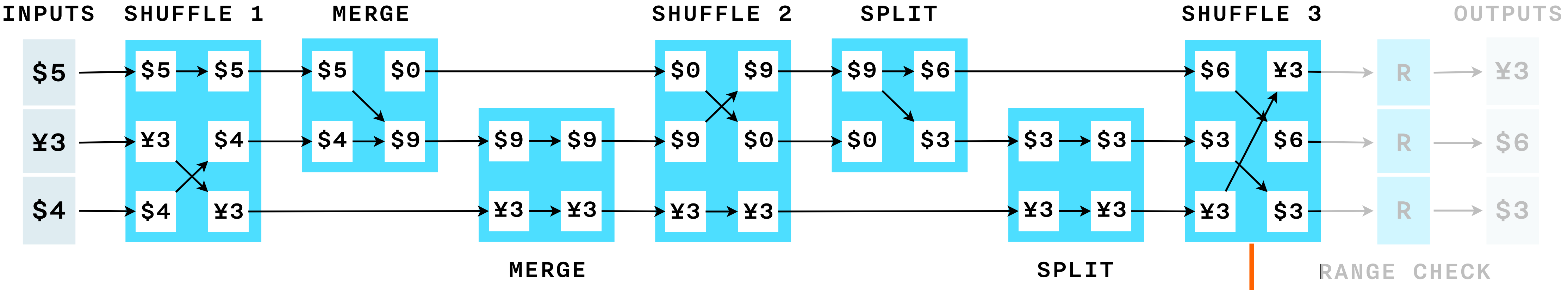
Non-zero values are reordered to the top, still grouped by asset type.

Cloak walkthrough



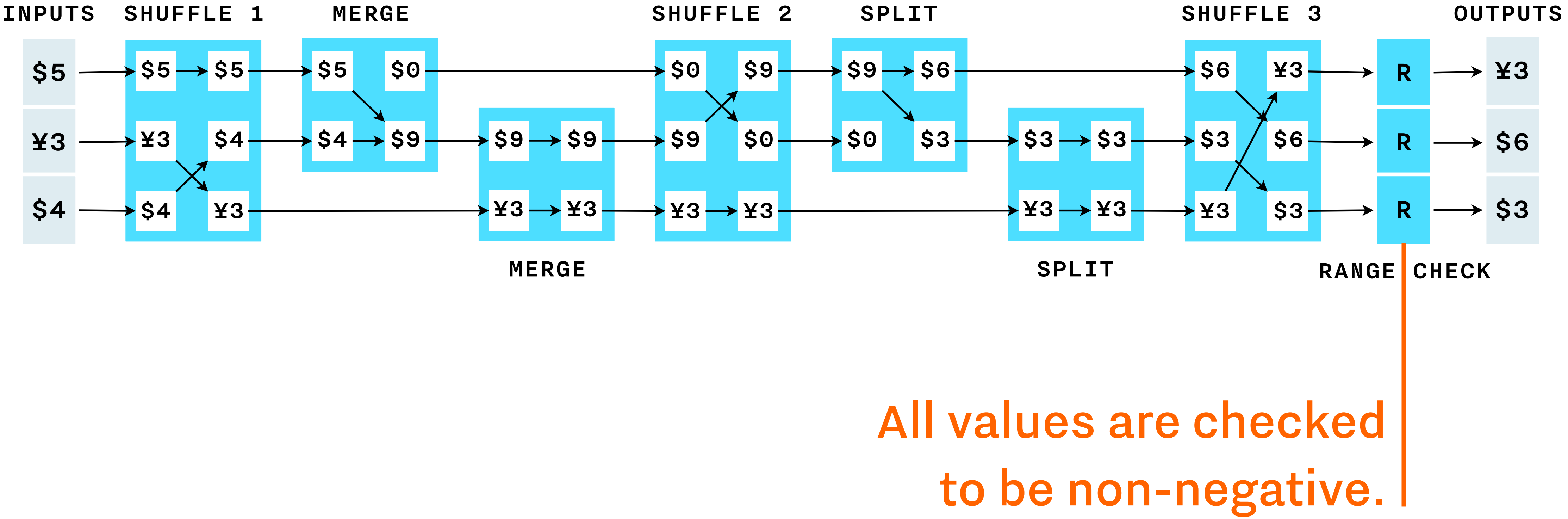
Values are split into target payment amounts.

Cloak walkthrough



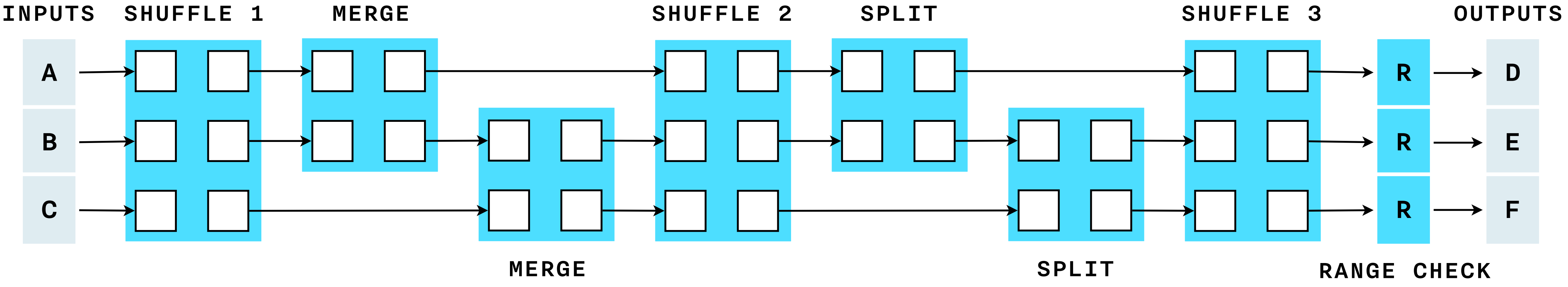
Values that were grouped by asset type are shuffled into a random order.

Cloak walkthrough



All values are checked to be non-negative.

Complete 3:3 Cloak transaction



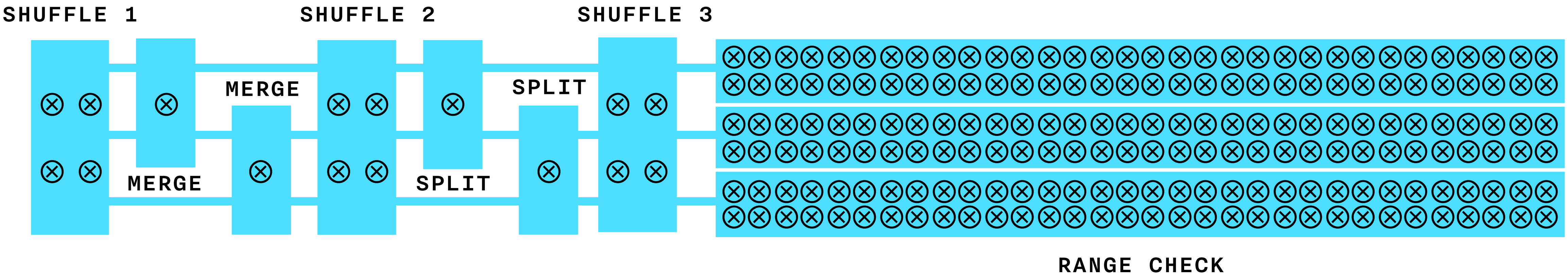
Transactions of the same size are **indistinguishable.**

SPEC & CODE

https://github.com/interstellar/spacesuit

Cloak performance

Most of the cost is **concentrated in range proofs**, the rest is relatively cheap.



⊗ — one multiplication gate

SPEC & CODE

https://github.com/interstellar/spacesuit

ZkVM

Zero-knowledge smart contracts

Introducing ZkVM

	BTC	EVM	TxVM	ZkVM
deterministic results	✓	✗	✓	✓
expressive language	✗	✓	✓	✓
safe environment	✓	✗	✓	✓
confidentiality	✗	✗	✗	✓

ZkVM = TxVM + Bulletproofs

TxVM

Linear types **Value** and **Contract** with the guaranteed “law of conservation”.

Contracts implement **“object capabilities”** pattern.

State updates via **deterministic tx log**.

Bulletproofs

Encrypted **values** and contract **parameters**.

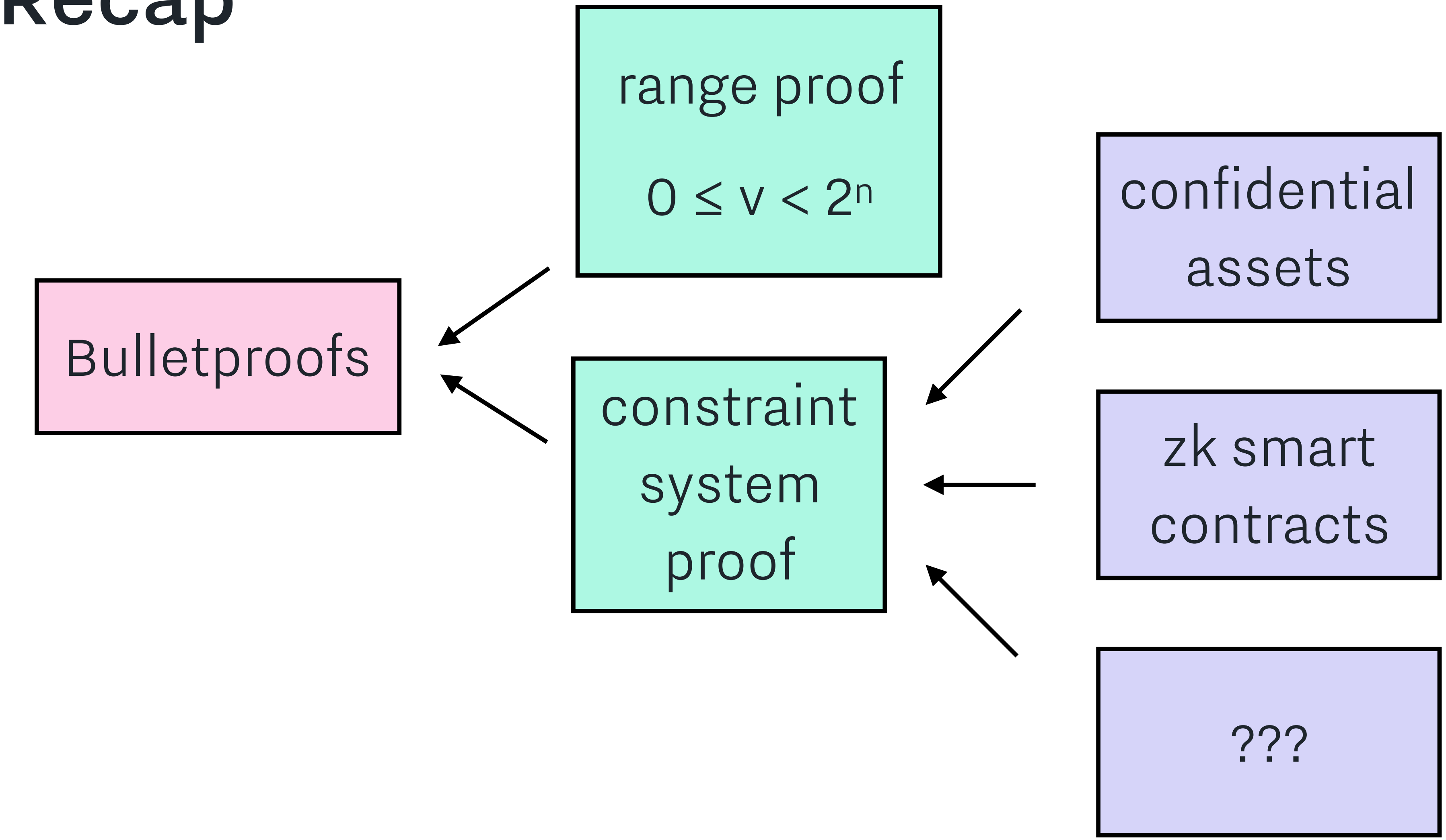
Contracts built with arbitrary **custom constraints**.

Asset flow protected with **Cloak**.

UNDER DEVELOPMENT

https://github.com/interstellar/zkvm

Recap



Bulletproofs

range proof

$0 \leq v < 2^n$

constraint system proof

confidential assets

zk smart contracts

???

Further reading

Bulletproofs paper

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

Interstellar research projects

https://interstellar.com/protocol

Cryptography libraries' API & protocol documentation

https://doc.dalek.rs

https://doc-internal.dalek.rs

@cathieyun