Bulletproof Protocol

for Set (Non)membership Proofs

Security and Implementation Considerations

Doris Benda, Ph.D., Senior Software Engineer at Concordium

Background

Bulletproof Protocol	Set (Non)membership Proofs	

Example

Background

Bulletproof Protocol	Set (Non)membership Proofs	
Goal: Range proofs Age check example: Prove that you are older than 18 years.	Goal: Prove that a public value v is in a public set S $v \in S$ Prove that a value v is NOT in a public set S $v \notin S$	

Example

Background

Bulletproof Protocol	Set (Non)membership Proofs	Zero-Knowledge Set/ Not-Set Membership Proofs
Goal: Range proofs Age check example: Prove that you are older than 18 years.	Goal: Prove that a public value v is in a public set S $v \in S$ Prove that a value v is NOT in a public set S $v \notin S$	Goal: Prove that a secret value v is or is NOT in a public set S European election example: Prove that you are European citizen. v = “DK” and S = {”DE”, “DK”, “UK”, ... }

Example

When to Use Which Protocol: A 3-Category Guide

The efficiency of the proof depends on |S|.

Small S	Medium S	Large S
Sigma Protocol + OR adapter (v = 1) OR (v = 2)	Bulletproof Protocol Example: Citizenship/Nationality proofs (with a set of about 200 possible countries).	- Curve trees - Incremental merkle trees (nullifiers) e.g. Tornado cash - Sparse merkle trees (for (non)membership proofs)

Example

Set (Non)membership Proof

Everything is a number going forward

(“DK” becomes “0x444b”)

$$11 \in \{1, 2, \ldots, 20\}$$

$$5 \notin \{1, 42, 9999\}$$

Hadamard Product : Math Notation I

Vectors

$$\mathbf{a} \circ \mathbf{b} = \begin{pmatrix} a_1 b_1 \\ a_2 b_2 \\ \vdots \\ a_n b_n \end{pmatrix} \in \mathbb{F}_q^n$$

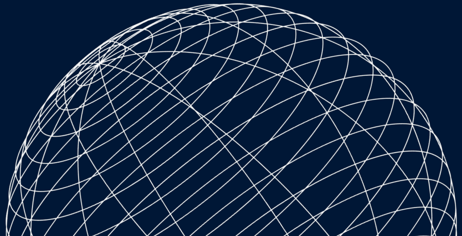
$$\begin{pmatrix} 2 \\ 5 \\ 7 \end{pmatrix} \circ \begin{pmatrix} 3 \\ 4 \\ 6 \end{pmatrix} = \begin{pmatrix} 2 \cdot 3 \\ 5 \cdot 4 \\ 7 \cdot 6 \end{pmatrix} = \begin{pmatrix} 6 \\ 20 \\ 42 \end{pmatrix}$$

Inner Product : Math Notation II

Vectors Sum Number

$$\langle \mathbf{a}, \mathbf{b} \rangle = \sum_{i=1}^{n} a_i b_i = c \in \mathbb{F}_q^n$$

$$\left\langle \begin{pmatrix} 2 \\ 5 \\ 7 \end{pmatrix}, \begin{pmatrix} 3 \\ 4 \\ 6 \end{pmatrix} \right\rangle = 2 \cdot 3 + 5 \cdot 4 + 7 \cdot 6 = 68 \pmod{q}$$



Core Bulletproof Protocol

maybe secrets

Vectors

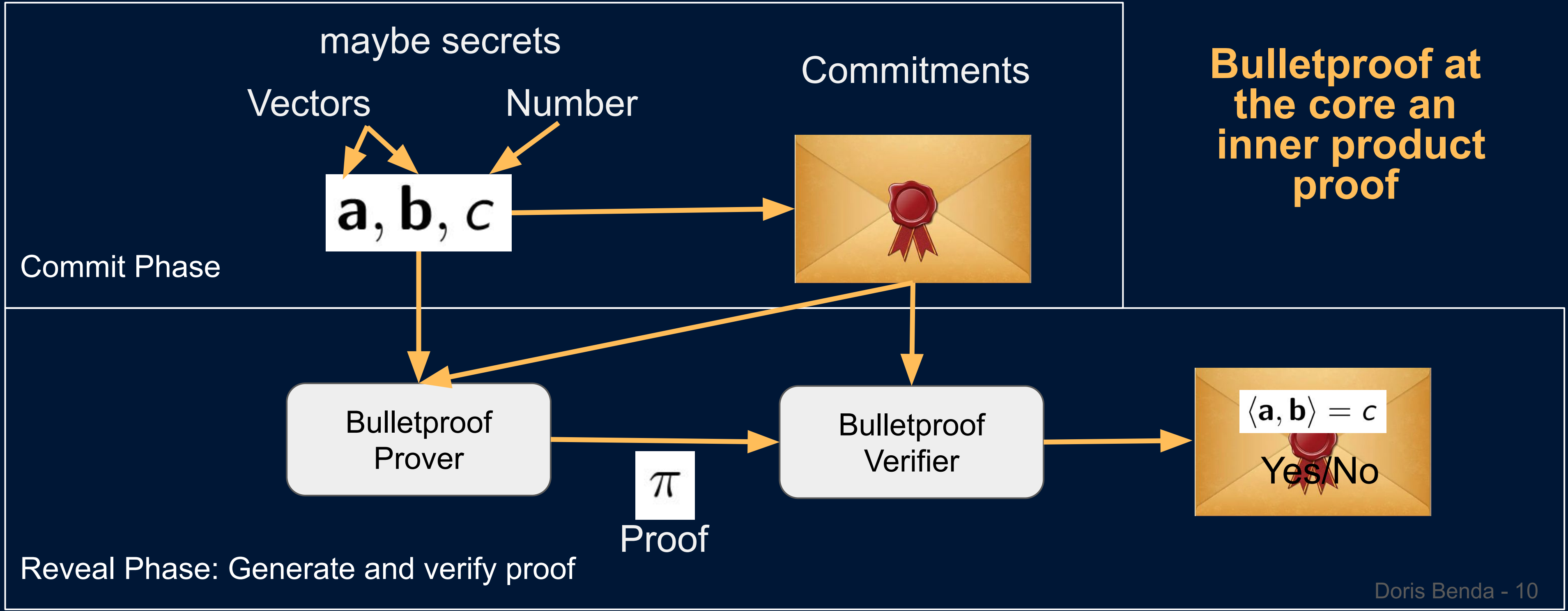
Number

$\mathbf{a}, \mathbf{b}, c$

Commitments

Commit Phase

Core Bulletproof Protocol



Bulletproof at the core an inner product proof

Plan

1) Express the set (non)membership constraints with the inner product equation

2) Keep the input vectors secret (adding blinding factors to the commitments)

3) Plug into the Bulletproof protocol

4) Implement non-interactive version of protocol (applying Fiat-Shamir transformation)

Set Membership Converter

Goal:

$$v \in S \quad \iff \quad \langle \mathbf{a}, \mathbf{b} \rangle = c$$

Express the set (non)membership problem using two vectors and

a scalar number such that they satisfy a inner-product relation.

Set Membership Converter

Start:

$$v \in S = \{s_1, s_2, \ldots, s_n\} \qquad v = s_i$$

I know that $v \in S$

Vectorize:

$$\mathbf{s} = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{pmatrix} \qquad \mathbf{a}_L = \begin{pmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix}$$

at position i

secret

public

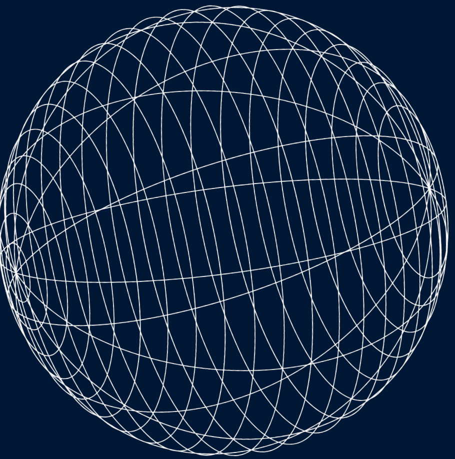
$$\langle \mathbf{s}, \mathbf{a}_L \rangle = v$$

I know the index i such that $v = s_i$

Set Membership Converter

$$\mathbf{a}_L = \begin{pmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix}$$

Further reading: Chapter 9.5.4 Set Membership
https://docs.concordium.com/governance/bluepaper/concordium-bluepaper.pdf



Set Membership Converter

For the consistency checks of a_L,

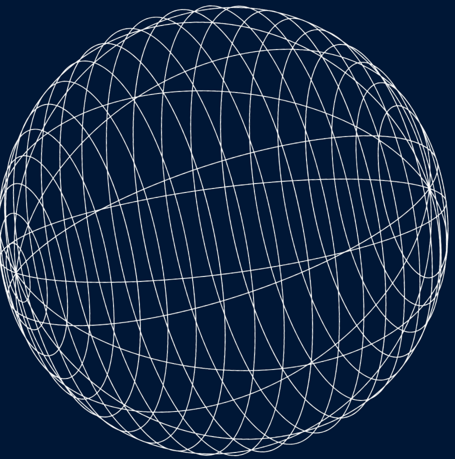
define a_R (a_L shifted by 1)

$$\mathbf{a}_R := \mathbf{a}_L - 1$$

$$\mathbf{a}_L = \begin{pmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix}$$

$$\mathbf{a}_R = \begin{pmatrix} -1 \\ \vdots \\ 0 \\ \vdots \\ -1 \end{pmatrix}$$

Further reading: Chapter 9.5.4 Set Membership
https://docs.concordium.com/governance/bluepaper/concordium-bluepaper.pdf



Set Membership Converter

For the consistency checks of a_L,

define a_R (a_L shifted by 1)

$$\mathbf{a}_R := \mathbf{a}_L - 1$$

$$\mathbf{a}_L = \begin{pmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix}$$

Constrain: a_L (Zero vector with exactly one 1)

$(\mathbf{a}_L - \mathbf{1}) - \mathbf{a}_R = \mathbf{0}$ (a_R is really $a_L - 1$)

$\mathbf{a}_R \circ \mathbf{a}_L = \mathbf{0}$ (one of the vector elements must be zero for each position)

$\langle \mathbf{a}_L, \mathbf{1} \rangle = 1$ (a_L has exactly one 1)

$$\mathbf{a}_R = \begin{pmatrix} -1 \\ \vdots \\ 0 \\ \vdots \\ -1 \end{pmatrix}$$

Further reading: Chapter 9.5.4 Set Membership
https://docs.concordium.com/governance/bluepaper/concordium-bluepaper.pdf

Nonmembership Converter

$$v \notin S = \{s_1, \ldots, s_n\} \qquad \text{(I know } v \text{ is not in } S\text{)}$$

$$\Longleftrightarrow$$

$$\forall i \quad v \neq s_i \qquad \text{(I know } v \text{ is not equal to any } s_i\text{)}$$

$$\Longleftrightarrow$$

$$\forall i \quad v - s_i \neq 0 \qquad \text{(I know each difference } v - s_i \text{ is not zero)}$$

$$\Longleftrightarrow$$

property of a finite field

$$\forall i \, \exists \, a_i \quad a_i(v - s_i) = 1 \qquad \text{(I know the multiplicative inverse of } v - s_i\text{)}$$

Further reading: Chapter 9.5.4 Set Membership
https://docs.concordium.com/governance/bluepaper/concordium-bluepaper.pdf

Nonmembership Converter

Public Set

Secret v

Multiplicative inverse elements

$$\mathbf{s} = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{pmatrix} \qquad \mathbf{a}_L = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} \qquad \mathbf{a}_R = \begin{pmatrix} v \\ \vdots \\ v \\ \vdots \\ v \end{pmatrix}$$

Further reading: Chapter 9.5.4 Set Membership
https://docs.concordium.com/governance/bluepaper/concordium-bluepaper.pdf

Public Set

Nonmembership Converter

Secret v

Multiplicative inverse elements

$$\mathbf{s} = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{pmatrix} \qquad \mathbf{a}_L = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} \qquad \mathbf{a}_R = \begin{pmatrix} v \\ \vdots \\ v \\ \vdots \\ v \end{pmatrix}$$

$$\mathbf{a}_R = v \cdot \mathbf{1} \qquad (\mathbf{a}_R \text{ encodes the value } v)$$

$$\mathbf{a}_L \circ (\mathbf{a}_R - \mathbf{s}) = \mathbf{1} \qquad (\text{a multiplicative inverse exists}$$

$$\iff v - s_i \neq 0$$

$$\iff v \notin S)$$

Further reading: Chapter 9.5.4 Set Membership
https://docs.concordium.com/governance/bluepaper/concordium-bluepaper.pdf

Must-Have Security Checks

- Vector constraints (especially checking `special` zero-one vectors)
- All “secret” vectors are blinded.
- All serializable types (commitments, proofs, context, audit records, …) include explicit version fields (e.g., V1, V2).
  - Avoids replay across protocol upgrades
- Randomness/ blinding factors:
  - Generate fresh, uniformly random values (Merlin transcript randomness)
  - No-reuse of blinding factors
- Fiat-Shamir Transformation (Merlin transcript):
  - Add all values that until this point have been part of the transcript
  - Add public/contextual inputs (generators, keys, …)
  - Add variable-length data with its length prefixed
  - Add domain separation

$$\mathbf{a}_L = \begin{pmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix}$$

Security Considerations & Pitfalls in ZK Proof Systems

Most academic descriptions define the ZK protocol interactively.

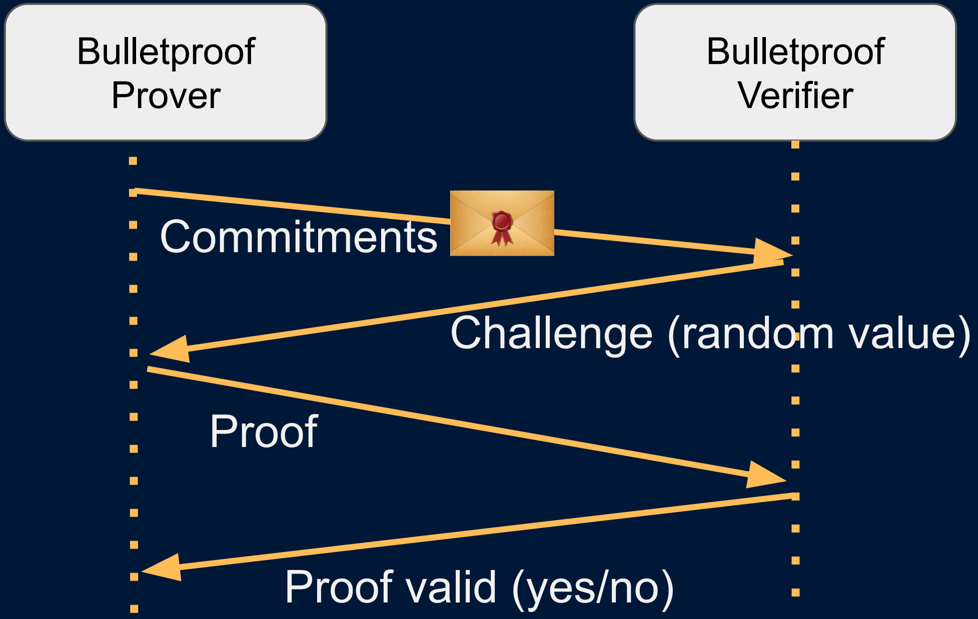
Fiat-Shamir Transformation/Heuristic

Engineers must transform it into a secure non-interactive proof, typically via Fiat–Shamir.

This transformation is subtle and frequently a source of mistakes.

Fiat-Shamir Transformation/Heuristic

interactive



non-interactive

```
// Initialize hash function/transcript
// with a domain separator
T = Transcript("Domain")

// All public/contextual information
// (keys, curve generator points, …) and
// all messages in the proof transcript
// until this point.
T.append("Label1", exchanged_value1)
T.append("Label2", exchanged_value2)

// Fiat–Shamir challenge =
// hash of the entire transcript
Challenge = Hash(T)
```

Further reading: Merlin transcript can be studied here: <https://merlin.cool/index.html> (implemented at <https://github.com/dalek-cryptography/merlin>).

Fiat-Shamir Transformation/Heuristic

What can go wrong:

“Failed to include all values exchanged between the prover and verifier up to this point.”

non-interactive

```
// Initialize hash function/transcript
// with a domain separator
T = Transcript("Domain")

// All public/contextual information
// (keys, curve generator points, …) and
// all messages in the proof transcript
// until this point.
T.append("Label1", exchanged_value1)
T.append("Label2", exchanged_value2)

// Fiat–Shamir challenge =
// hash of the entire transcript
Challenge = Hash(T)
```

Further reading: Merlin transcript can be studied here: <https://merlin.cool/index.html> (implemented at <https://github.com/dalek-cryptography/merlin>).

Fiat-Shamir Transformation/Heuristic

What can go wrong:

“Appending variable-length types such as `String`, `Vectors`, `Sets`, `Maps`, or other collections. Naively appending the bytes (without including the length of the collection) can produce collisions

Example:

Hash(“AA”,”BB”) = Hash(“AABB”) = Hash(“A”,”ABB”)

Better:

Hash(“2”,“AA”,”2”,”BB”) ≠ Hash(“1”,“A”,”3”,”ABB”)

non-interactive

```
// Initialize hash function/transcript
// with a domain separator
T = Transcript("Domain")

// All public/contextual information
// (keys, curve generator points, …) and
// all messages in the proof transcript
// until this point.
T.append("Label1", exchanged_value1)
T.append("Label2", exchanged_value2)

// Fiat–Shamir challenge =
// hash of the entire transcript
Challenge = Hash(T)
```

Further reading: Merlin transcript can be studied here: <https://merlin.cool/index.html> (implemented at <https://github.com/dalek-cryptography/merlin>).

Fiat-Shamir Transformation/Heuristic

What can go wrong:

“Missing domains or labels”

non-interactive

```
// Initialize hash function/transcript
// with a domain separator
T = Transcript("Domain")

// All public/contextual information
// (keys, curve generator points, …) and
// all messages in the proof transcript
// until this point.
T.append("Label1", exchanged_value1)
T.append("Label2", exchanged_value2)

// Fiat–Shamir challenge =
// hash of the entire transcript
Challenge = Hash(T)
```

Further reading: Merlin transcript can be studied here: <https://merlin.cool/index.html> (implemented at <https://github.com/dalek-cryptography/merlin>).

Thank you for listening

NAME: Doris Benda, Ph.D.

WORK: Senior Software Engineer

at Concordium

GITHUB: https://github.com/DOBEN

LinkedIn: https://www.linkedin.com/in/dorisbenda/

Fiat-Shamir Transformation/Heuristic

What can go wrong:

“Appending variable-length types such as `String`, `Vectors`, `Sets`, `Maps`, or other collections. Naively appending the bytes (without including the length of the collection) can produce collisions

```
let items = vec!["AA", "BB", "CC"];

T.append_u64("items_len", items.len() as u64);

for (i, item) in items.iter().enumerate() {
        T.append(format!("{i}"), item.as_bytes());
}
```

non-interactive

```
// Initialize hash function/transcript
// with a domain separator
T = Transcript("Domain")

// Add all prover→verifier and
// verifier→prover messages
T.append("Label1", exchanged_value1)
T.append("Label2", exchanged_value2)

// Fiat–Shamir challenge =
// hash of the entire transcript
Challenge = Hash(T)
```

Further reading: Merlin transcript can be studied here: <https://merlin.cool/index.html> (implemented at <https://github.com/dalek-cryptography/merlin>).