

The background of the cover is a complex geometric design. It features several overlapping triangular and polygonal shapes in various shades of blue and purple. A subtle pattern of white hexagons is visible in the lower half of the image. A white rectangular frame is centered on the page, containing the title and subtitle.

# **offshift** PROTOCOL

Yellowpaper

Draft, v.0.1

# CONTENT

<b>CONTENT</b>	<b>2</b>
<b>1. INTRODUCTION</b>	<b>4</b>
<b>2. WHY BULLETPROOFS</b>	<b>5</b>
2.1 Three major ZKP techniques	5
2.1.1 zk-SNARKs	5
2.1.2 zk-STARKs	6
2.1.3 Bulletproofs	6
2.2 High level approaches comparison	7
<b>3. UTXO-BASED IMPLEMENTATIONS</b>	<b>8</b>
<b>4. ACCOUNT-BASED MODEL FEATURES</b>	<b>8</b>
<b>5. HOW BULLETPROOFS WORK</b>	<b>9</b>
5.1 EC Pedersen commitment	9
5.1.1 Basis	9
5.1.2 Pedersen commitment for the vector of values	11
5.1.3 Using Pedersen commitments in interactive schemes	12
5.2 Range proofs	15
5.2.1 Applied math	16
5.2.2 How Karatsuba's method works	17
5.2.3 Basis of range proofs	18
5.2.4 From $a = \langle a_{\text{binary}}, 2^n \rangle$ to $t(x) = \langle l(x), r(x) \rangle$	19
5.3 Inner product proof	24
5.4 Inner product compression	26
5.4.1 How inner product compression works	26
5.4.2 Scheme of interaction between prover and verifier with inner product proof creating and verifying	29

5.5 Summary .....	30
<b>6. Algorithms.....</b>	<b>30</b>
6.1 Prover algorithm .....	31
6.1.1 Single proof.....	31
6.1.2 Aggregated proof .....	31
6.2 Verifier algorithm .....	34
<b>7. THE OFFSHIFT PROTOCOL.....</b>	<b>35</b>
7.1 Commitment creation (Deposit) .....	35
<b>8. FUNCTIONALITY OF A.....</b>	<b>36</b>
<b>SMART-CONTRACT AND ITS METHODS.....</b>	<b>36</b>
7.2 Payments .....	36
7.3 Withdrawal .....	36
<b>9. FUTURE WORK &amp; IMPLEMENTATIONS .....</b>	<b>37</b>
<b>APPENDIX A. NUMS .....</b>	<b>38</b>
<b>APPENDIX B. FIAT-SHAMIR HEURISTIC.....</b>	<b>38</b>
<b>REFERENCES.....</b>	<b>39</b>

# 1. INTRODUCTION

At the moment, it is very difficult to ensure privacy when managing tokenized assets (especially in the conditions of openness and transparency of accounting systems in which these assets are tokenized). Privacy approaches have long been used in cryptocurrencies but have not been applied in a regulated environment yet, mostly due to the fact that regulators require the tracking of financial flows (at least inbound and outbound).

For this reason, there can be a protocol that allows transparency of the deposit and withdrawal processes, but with privacy in the middle (transfers). This protocol turns virtually any asset into a form of digital cache. You deposit the tokenized asset (this process is open and subject to regulation) and receive a corresponding hidden ownership obligation. When you withdraw, you prove that such an obligation exists, it is valid, and you own it (the process of converting obligations into an asset is also transparent and amenable to regulation). But everything that happens between these two actions is hidden. You can pay with obligations to others, exchange obligations with other ones, etc. At the same time:

1. there is confidence that new obligations do not appear out of thin air;
2. the actual amounts are available only to the involved counterparties.

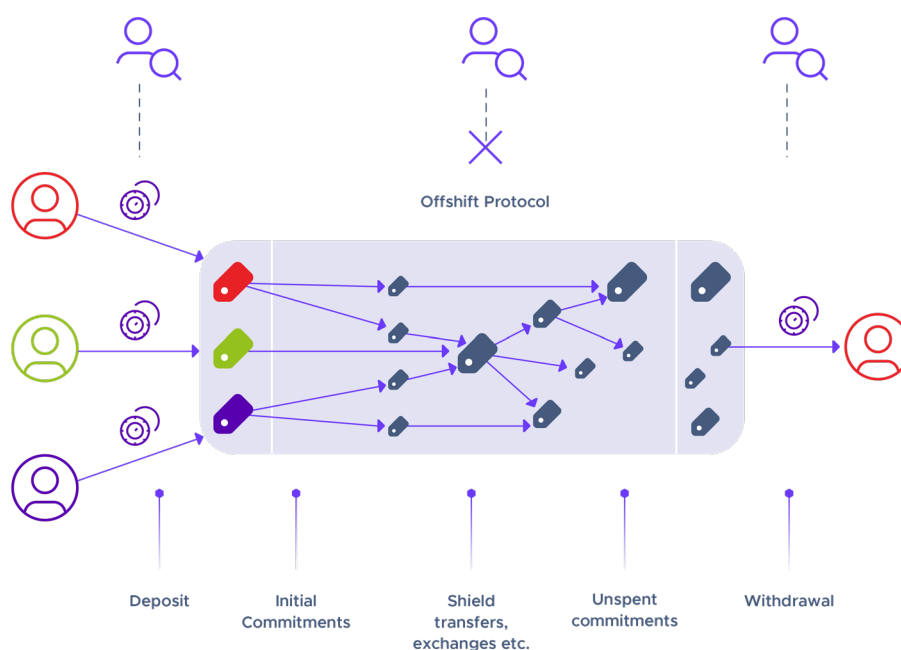


FIGURE 1 - OFFSHIFT PROTOCOL PROPERTIES



This paper describes the structure of the Offshift protocol, which implements the functionality of confidential transactions with ERC-20 tokens. The core of the protocol is zero knowledge proofs. Bulletproofs are a reference implementation (we'll describe why later).

Also, a feature of the protocol is getting the price of an asset when converting it (depositing and withdrawing assets) through oracles (integration with an oracle solution). We will consider in more detail the principles of operation in section 7.

## 2. WHY BULLETPROOFS

### 2.1 Three major ZKP techniques

#### 2.1.1 zk-SNARKs

The idea for the creation of a Zero-Knowledge Succinct Non-Interactive Argument of Knowledge was first described in 1985 in the article “The Knowledge Complexity of Interactive Proof-Systems” [\[1, 2\]](#).

When were zk-SNARKs first used? It was thought at the time that Bitcoin transactions were quite confidential and untraceable, because they were not associated with users' IDs. As it turned out, re-identifications in incomplete datasets using generative models are possible. After discovering this fact, developers began to work on cryptocurrencies that have a reasonable level of confidentiality of transactions. This is how Zcash was born. The main goal of its creation is to make all transactions completely anonymous [\[3, 4\]](#). For this purpose zk-SNARKs were used.

Zk-SNARKs are used in:

- Cryptocurrencies [\[5\]](#);
- Sidechains [\[6\]](#);
- Rollups;
- Identity infrastructures.

Zk-SNARKs have the following components [\[7\]](#):

1. Transforming the initial task to the polynomial problem.

2. Simplifying the whole task to check equality:  $t(s)h(s) = w(s)v(s)$ .
3. Homomorphic encryption  $E(t(s))$ ,  $E(h(s))$ ,  $E(w(s))$ ,  $E(v(s))$ .
4. Zero knowledge - operating with hidden values.

### 2.1.2 zk-STARKs

The first work about STARKs was created in 1990, but practical applications did not yet exist [8]. That was until 2018, when Eli Ben-Sasson (a professor at the Technion-Israel Institute of Technology who has been researching zero-knowledge proofs for approximately 15 years), Iddo Bentov, Yinon Horesh, and Michael Riabzev presented their work “Scalable, transparent, and post-quantum secure computational integrity” [9, 10]. In this paper, a case was also described in which police prove that the offender database doesn’t contain DNA of a presidential candidate without revealing any information about the database or the DNA.

Projects that use zk-STARKs:

- Decentralized exchanges like OpenZKP [11];
- Rollups;
- Identity management.

Main components of STARKs construction [12]:

- Homomorphic operations;
- Secure multiparty computing (MPC);
- Hash functions for quantum resistance [12, 13].

### 2.1.3 Bulletproofs

The Bulletproofs whitepaper was published in late 2017 by Jonathan Bootle from University College London, England, and by Benedikt Bunz from Stanford University, United States [14]. It was originally designed to be implemented on the Bitcoin blockchain. All existing ZKP implementations at the time required a trusted setup, and their worst aspect is that they must be initiated by some trusted authority. As is well known, the security properties of the Bitcoin system don’t apply to that authority. So the bulletproofs were designed to solve this problem [14].

Now Bulletproofs are being used in:

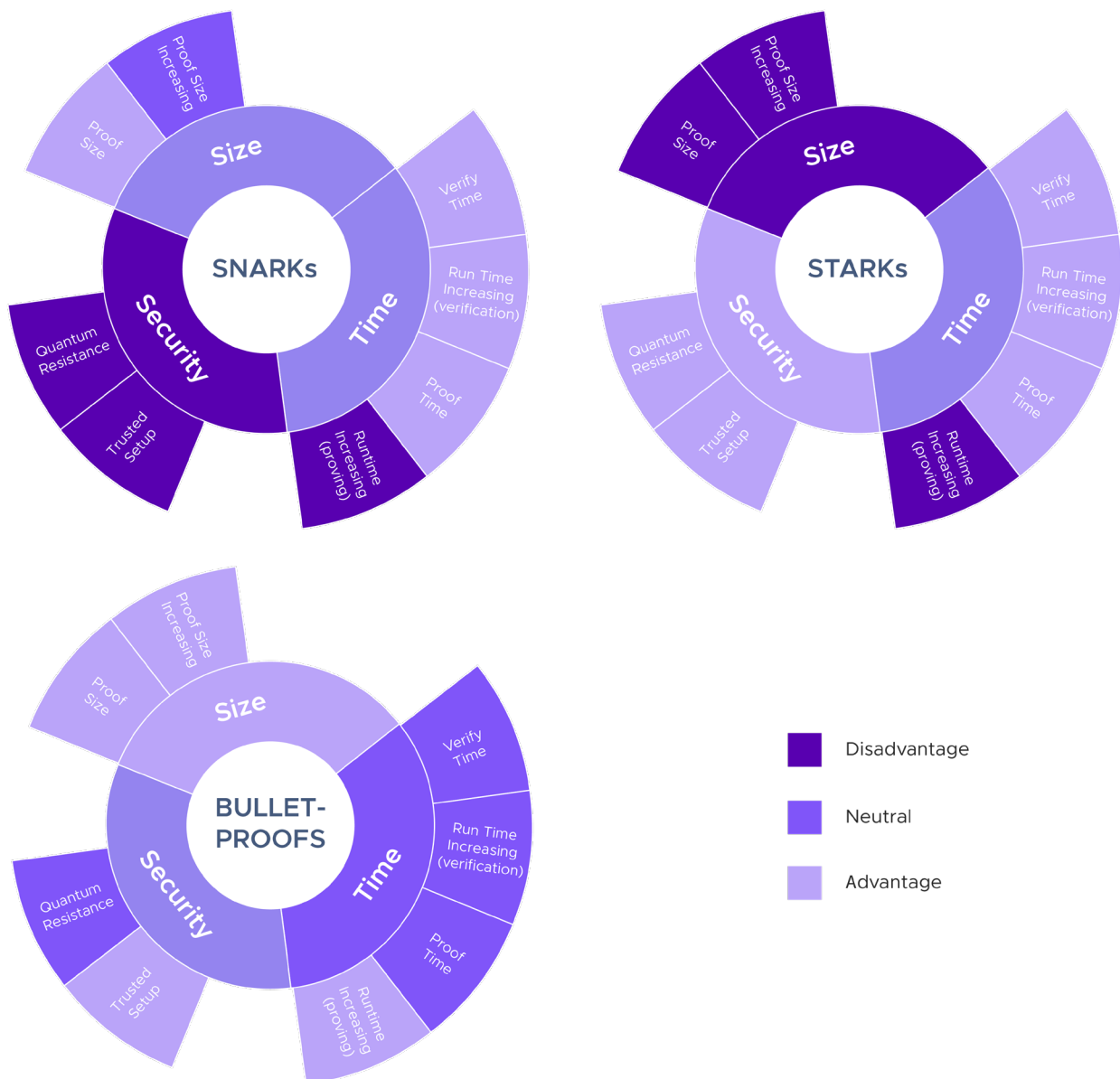
- UTXO-based cryptocurrencies.

At the heart of the Bulletproofs:

- Pedersen commitment;
- Improved inner product proof.

## 2.2 High level approaches comparison

You can find the following diagram that represents the difference between approaches to zero-knowledge proofs [\[15, 16\]](#).



Based on the considered comparison, it was decided to use Bulletproofs as zero knowledge proofs for the Offshift protocol.

### 3. UTXO-BASED IMPLEMENTATIONS

There are several successful applications of Bulletproofs in UTXO-based accounting systems. The most striking examples are Monero and Mimblewimble.

In 2018, Bulletproofs began to be used in Monero as a replacement for Borromean's ring signatures. The main reason for this was the size of the range proofs (bulletproofs significantly reduced this size), as well as compactness - the ability to place the proof not for a separate transaction output but for all of the outputs that are contained within the transaction (in this case, the proof will be larger than a single one, but much less than the total size of all outputs) [\[17\]](#).

On July 19th, 2016, a paper titled "MIMBLEWIMBLE" was published. It was a protocol for accounting system construction with a high level of user privacy and system scalability [\[18\]](#). A little later, Andrew Poelstra, one of the most famous people in the Bitcoin community, published a more detailed document, which described the details and technical features of the protocol [\[18\]](#).

### 4. ACCOUNT-BASED MODEL FEATURES

The UTXO model makes it harder to link transactions, whereas the account model provides better fungibility [\[19\]](#).

Changing the address for each input/output payment is making the transaction history harder in the linkability context (UTXO). A newly generated address doesn't have a known owner and requires chain analysis to be linked to a single user (which sometimes is potentially impossible) [\[20\]](#).

In the account-based model, all transactions are associated with a particular account. But after funds come to the account, it's impossible to define which coins will be spent next time.

This, in turn, leads to a model where it makes sense to transfer part of the transactions to the UTXO model (UTXOs will be tied to one account but hidden), while the output of these UTXOs to the general account balance allows using the built-in account-based model mixer.

## 5. HOW BULLETPROOFS WORK

To understand how bulletproofs work, we will first look at the basic elements of the mathematics that are used: EC Pedersen commitment, range proofs and inner product proof [\[21 - 24\]](#).

### 5.1 EC Pedersen commitment

*Cryptographic commitments are what all zero-knowledge proofs are based on.*

© Yoda

*Cryptographic commitments are used for setting some value without revealing it. After a certain period of time, you can publish the knowledge and prove that you had it earlier.*

© Obi-Wan Kenobi

#### 5.1.1 Basis

Pedersen commitment can provide:

- irreversibility (the impossibility of obtaining a secret value having only a commitment);
- completeness (having a commitment, the verifier can check the knowledge with a high level of certainty).

In a Pedersen commitment scheme - irreversibility and completeness are ensured by the complexity of the discrete logarithm problem.

$$C = r \cdot H + a \cdot G$$

**C** - commitment, it is just a point on the EC;

**a** - committed (integer) value, for which zero-knowledge has to be completed (has to be hidden);

**r** - random (integer) value, is used for:

- preventing the situation when the commitments for the same values will be equal;
- preventing brute force attacks if the committed value is not large (like the amount of the transfer);

**G** - the base point (EC generator);

**H** - another EC point:

- the discrete logarithm of H no one has to know (**H = q\*G**). Attackers who know q can produce valid proofs without knowledge of a.

**Note 1.**  $C = r \cdot q \cdot G + a \cdot G = (rq+a) \cdot G$ . If the attacker knows q, they can provide proof with  $a_1 \neq a$  and  $r_1 \neq r$  in such a way that  $C = (r_1 q_1 + a_1) \cdot G$ .

**Note 2.** To prevent this attack and prove "not knowing" q methods, NUMS "nothing-up-my-sleeve" is used.

**Note 3.** Pedersen commitments support homomorphism. Homomorphism is an important property of this approach since it allows further calculations over the hidden values:

$$\begin{aligned} C(r_1, a_1) + C(r_2, a_2) &= r_1 H + a_1 G + r_2 H + a_2 G \\ &= (r_1 + r_2) \cdot H + (a_1 + a_2) \cdot G = C(r_1 + r_2, a_1 + a_2) \end{aligned}$$

#### Algorithm 1. (Pedersen commitment scheme)

1. Alice (prover) generates a random secret **r**.
2. Alice calculates a commitment **C(r,a) = rH + aG**.
3. Alice publishes commitment **C(r,a)**. Value is stamped by Bob (verifier).
4. Then Alice reveals **r'** and **a'**.
5. Bob computes **C(r',a') = r'H + a'G**, and verifies that **C(r',a') = C(r,a)**.

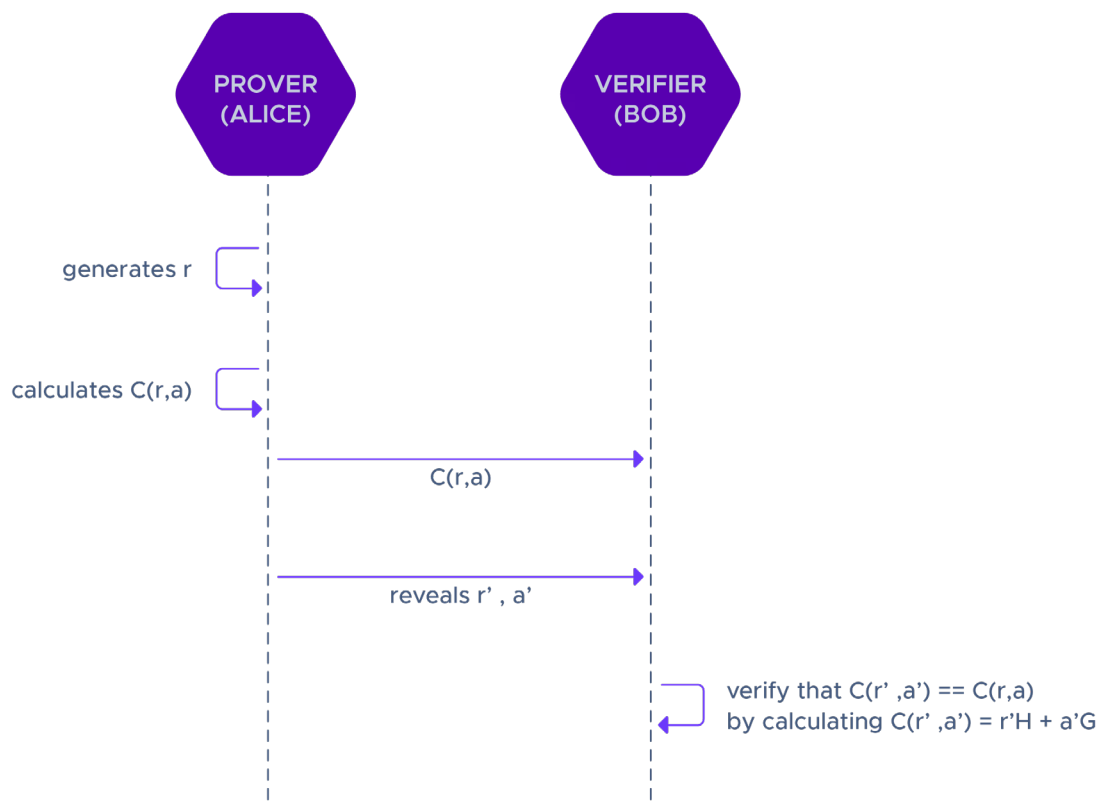


FIGURE 1 - PEDERSEN COMMITMENT SCHEME

### 5.1.2 Pedersen commitment for the vector of values

Pedersen commitments can be used for hiding several values in the following way:

$$C(r, \bar{a}) = rH + a_1G_1 + a_2G_2 + \dots + a_nG_n$$

Each  $G_n$ , in this case, has to be formed using the NUMS approach.

**Note 5.** If the same group generator is being used to hide each  $a_n$  value, the scheme could be broken. Initially the attacker can provide commitment  $C(r, \bar{a})$ , where  $\bar{a} = [a_1, a_2]$ . Then they can provide a witness  $(r, \bar{a}')$ , where  $\bar{a}' = [a_1', (a_2+a_1)-a_1']$  and it will be the correct witness for the verifier.

**Note 6.** Commitments for vectors can be used for more complex proofs (for example, for operations with vectors without their divulgence).



This scheme can be extended to create the commitment for several vectors via:

$$C(r, \bar{a}, \bar{e}) = rH + \bar{a}\bar{G} + \bar{e}\bar{P}$$

### 5.1.3 Using Pedersen commitments in interactive schemes

The Schnorr Identification Scheme is one of the simplest interactive zero-knowledge proof schemes. Consideration of this scheme will help in the understanding of further approaches.

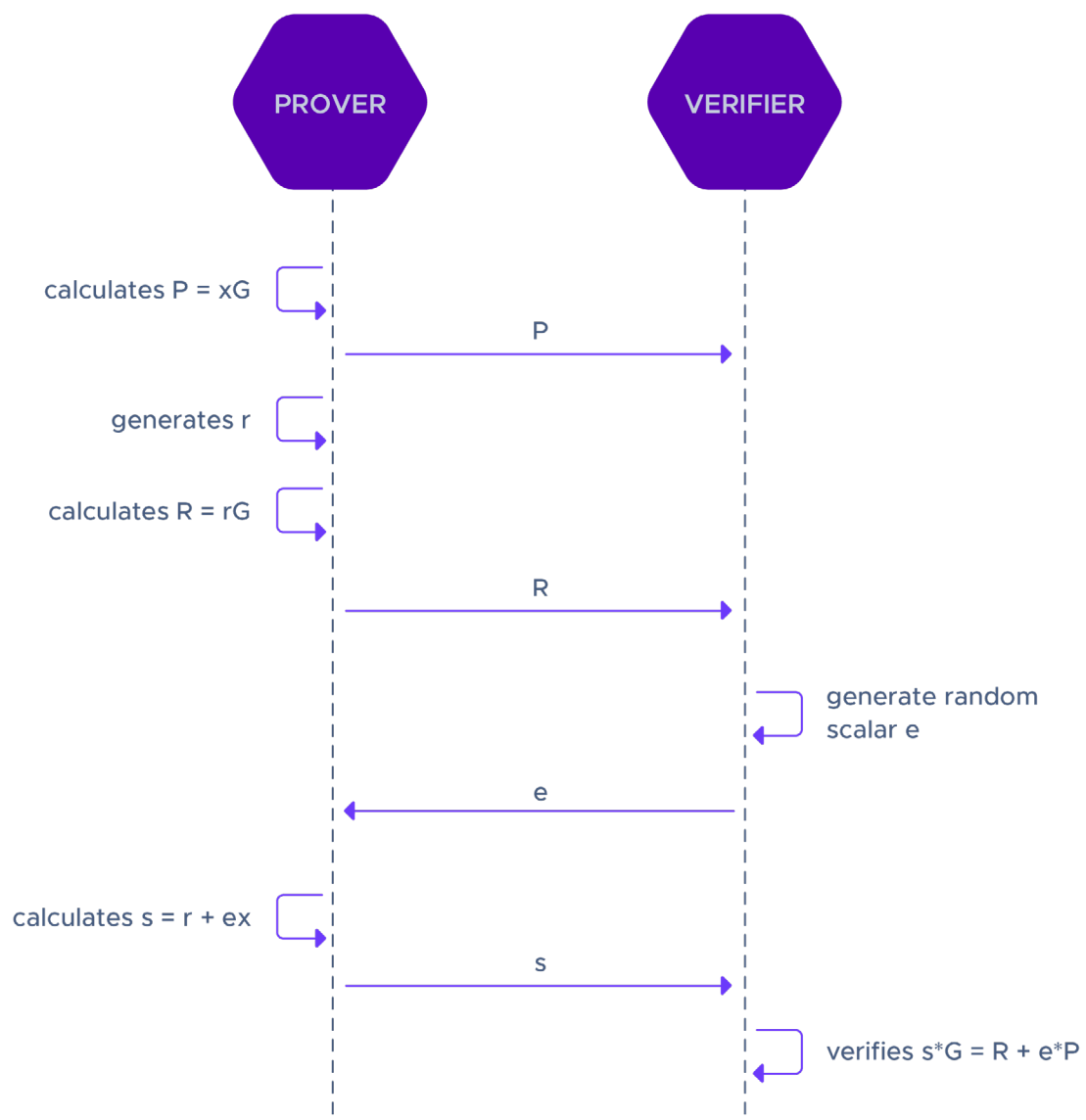


FIGURE 2 - SCHNORR IDENTIFICATION PROTOCOL

1. The prover has a secret  $\mathbf{x}$ . To ensure that they know the secret, the prover has to form a commitment  $\mathbf{P} = \mathbf{xG}$ .
2.  $\mathbf{P}$  is sent to the verifier (like a public key).
3. Next, the prover generates a random value  $r$  and forms a new commitment:  
 $\mathbf{R} = r\mathbf{G}$ .
4.  $\mathbf{R}$  is sent to the verifier.
5. The verifier in turn generates a random scalar  $e$  and sends it to the prover.
6. Now the prover can compute  $\mathbf{s} = r + e\mathbf{x}$ .
7.  $\mathbf{s}$  is being sent to the verifier.
8. Verifier can check that  $\mathbf{s}*\mathbf{G} = \mathbf{R} + e*\mathbf{P}$ .

The Pedersen commitment allows a large number of values to be committed (vector). In the context of the applicability of this property to the Schnorr identification scheme, the scheme will look like this:

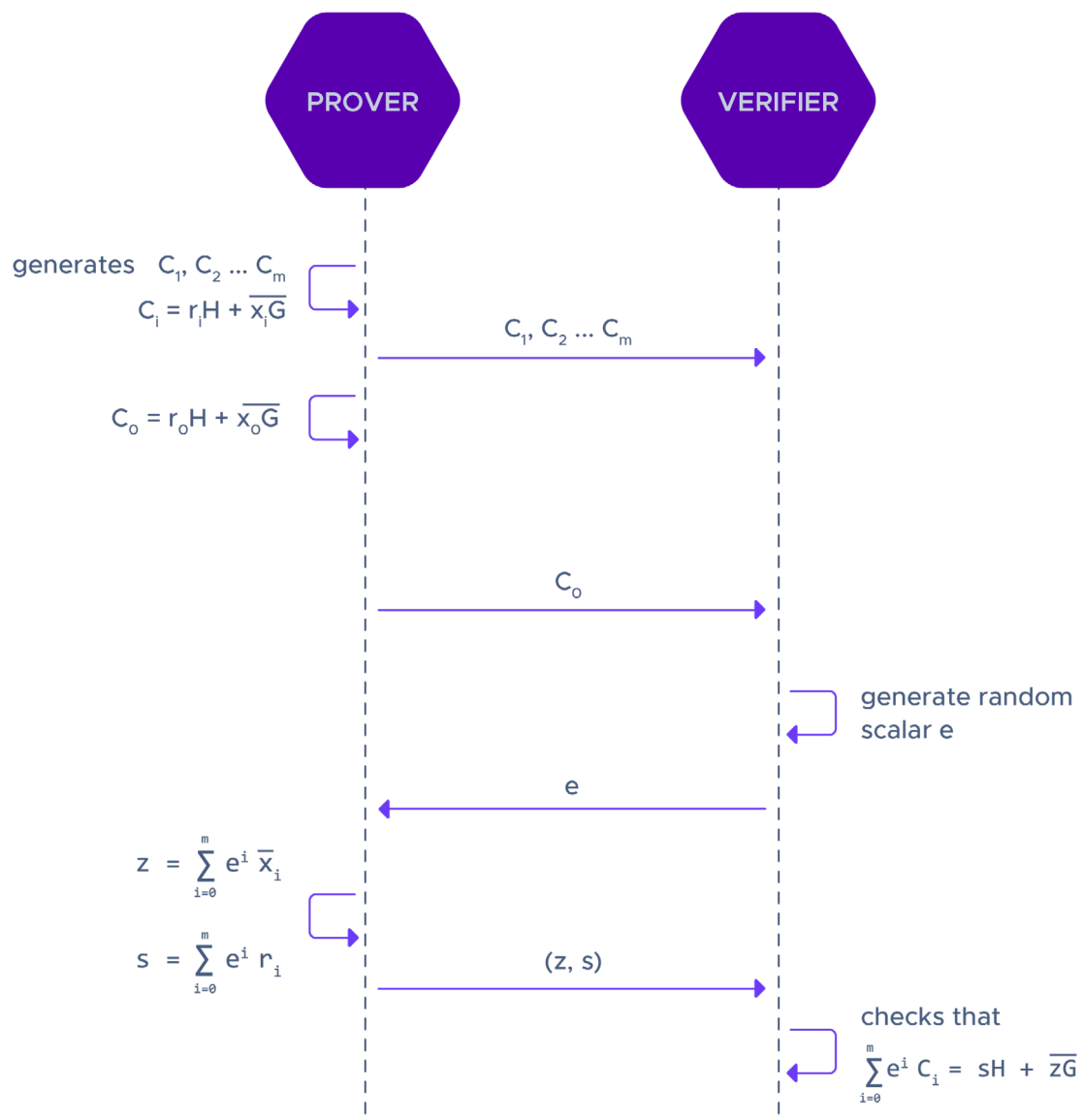


FIGURE 3 - SCHNORR IDENTIFICATION PROTOCOL WITH PEDERSEN COMMITMENTS

How this scheme works with the presence of interactivity [25]:

1. Prover generates a “knowledge argument”  $C_1, C_2 \dots C_m$  using secret vectors  $\overline{x}_m$  of the same dimension  $N$ .

$$C_1 = r_1 H + \overline{x}_1 G$$

$$C_2 = r_2 H + \overline{x}_2 G$$

...

$$C_m = r_m H + \overline{x}_m G$$

2. Prover sends  $C_1, C_2 \dots C_m$  to the verifier. In the future, they may prove that they know the set of secrets behind each  $C_m$ .
3. Prover sends  $C_0$  to the verifier.

$$C_0 = r_0 H + \overline{x}_0 G$$

- a new commitment to random vector  $\mathbf{x}$  of  $N$  dimension.

4. Verifier sends  $e$  to prover;  $e$  - random scalar.
5. Prover sends to verifier  $z$  and  $s$ .

$$z = \sum_{i=0}^m e^i \overline{x}_i$$

- vector of dimension  $N$ ,

$$s = \sum_{i=0}^m e^i r_i$$

- scalar.

6. Verifier checks that

$$\sum_{i=0}^m e^i C_i = sH + \overline{z}G$$

**Note 7.** *The scheme described above is interactive. It requires the generation of the random value  $e$ , and in this case, it is generated by and received from a particular verifier. Therefore the proof for one verifier will not be true for another (no one can guarantee that the verifier is not in cahoots with the prover). A non-interactive scheme does not require direct communication between the verifier and the prover, so the verification of the commitment can be valid proof for everyone.*

**Note 8.** *A “knowledge argument” is a technical term distinguished from “proof of knowledge” by the idea that the proof is only computational – an adversary with enough computing power may be able to convince you that they know the secret value(s) even if they don’t.*

## 5.2 Range proofs

Using Pedersen commitments, it is possible to achieve the following feature: we can prove that the sum of inputs and the sum of outputs are equal (in the case of the UTXO model).

But with using it in a raw form to make transactions confidential, this problem arises: the amounts (a values) can be negative, so a check for non-negativity is also needed. What is the best way to prove this without disclosure?

Inputs	Outputs
User A signature $E = r_1G + 10H$	Address C $I = r_3G + 112H$
User A signature $F = r_2G + 2H$	Address D $J = r_4G + (-100)H$

So there is a need for a method that allows us to prove that each  $a > 0$ .

To solve this, range proofs can be used. A Zero-knowledge range proof is a proof that allows us to verify that the secret is in a certain range without revealing it.

### 5.2.1 Applied math

Any number can be represented as a polynomial if  $x$  is replaced with the base of a numeral system:

$$A(x) = a_0 + a_1x^1 + a_2x^2 + \dots + a_nx^n$$

Then the reduction of the number to the original form will be:

$$\sum_{i=0}^n a_i x^i$$

And the multiplication of two numbers represented in this form will be:

$$\left( \sum_{i=0}^n a_i x^i \right) \cdot \left( \sum_{j=0}^m b_j x^j \right) = \sum_{k=0}^{n+m} x^k \sum_{i+j=k}^{n+m} a_i b_j$$

The base of the numeral system can be chosen arbitrarily.

The fastest known multiplication method was long multiplication, until 1960, when Andrei Kolmogorov and others proposed the "**hypothesis of  $n^2$** ". It states: It's impossible to multiply two  $n$ -digit numbers faster than in  **$O(n^2)$** .

Later, Anatoly Karatsuba proposed a multiplication method with a time estimate  **$O(n^{\log 3})$**  and disproved the hypothesis. Now this multiplication method can help us to make proof verification faster [\[26, 27\]](#).

**Note 9.** Nowadays, this problem is solved for  **$O(n * \log n)$** , using the fast Fourier transform algorithm, but it is only used "for its intended purpose": to process signals, not to multiply numbers.

## 5.2.2 How Karatsuba's method works.

With example

1. We have two polynomials have to be multiplied:

$$\begin{aligned} a(x) &= 2 + 3x + 9x^2 + 6x^3 \\ b(x) &= 4 + 2x + 4x^2 + 1x^3 \end{aligned}$$

$$\begin{aligned} a(x) &= a_0 + a_1x^1 + a_2x^2 + \dots + a_nx^n \\ b(x) &= b_0 + b_1x^1 + b_2x^2 + \dots + b_nx^n \end{aligned}$$

2. Polynomials must be represented in the following form ( $n = 2k$ ):

$$\begin{aligned} a(x) &= a_1(x) + x^ka_2(x) \\ b(x) &= b_1(x) + x^kb_2(x) \end{aligned}$$

$$\begin{aligned} a(x) &= (2 + 3x) + x^2 \cdot (9 + 6x) \\ b(x) &= (4 + 2x) + x^2 \cdot (4 + 1x) \end{aligned}$$

3. This way polynomials multiplication is represented like:

$$a(x)b(x) = a_1(x)b_1(x) + x^ka_1(x)b_2(x) + x^ka_2(x)b_1(x) + x^{2k}a_2(x)b_2(x)$$

4. Then we introduce the following notation:

$$\begin{aligned} p_1(x) &= a_1(x) \cdot b_1(x) \\ p_2(x) &= a_2(x) \cdot b_2(x) \end{aligned}$$

$$\begin{aligned} p_1(x) &= (2 + 3x) \cdot (4 + 2x) = 6x^2 + 16x + 8 \\ p_2(x) &= (9 + 6x) \cdot (4 + 1x) = 6x^2 + 33x + 36 \end{aligned}$$

$$t(x) = (a_1(x) + a_2(x)) \cdot (b_1(x) + b_2(x))$$

$$\begin{aligned} t(x) &= ((2 + 3x) + (9 + 6x)) \cdot ((4 + 2x) + (4 + 1x)) \\ &= (11 + 9x) \cdot (8 + 3x) \\ &= 27x^2 + 105x + 88 \end{aligned}$$

5. Final multiplication will be represented like

$$\begin{aligned} c(x) &= a(x)b(x) = a_1(x)b_1(x) + x^ka_1(x)b_2(x) + x^ka_2(x)b_1(x) + x^{2k}a_2(x)b_2(x) = p_1(x) \\ &+ x^ka_1(x)b_2(x) + x^ka_2(x)b_1(x) + x^{2k}p_2(x) = p_1(x) + x^k(a_1(x)b_2(x) + a_2(x)b_1(x)) + x^{2k}p_2(x) \end{aligned}$$

If:

$$\begin{aligned} t(x) - p_1(x) - p_2(x) &= (a_1(x) + a_2(x)) \cdot (b_1(x) + b_2(x)) - a_1(x) \cdot b_1(x) - a_2(x) \cdot b_2(x) = \\ &= a_1(x) \cdot b_1(x) + a_1(x)b_2(x) + a_2(x)b_1(x) + a_2(x) \cdot b_2(x) - a_1(x) \cdot b_1(x) - a_2(x) \cdot b_2(x) = \\ &= a_1(x)b_2(x) + a_2(x)b_1(x) \end{aligned}$$

Then:

$$\begin{aligned} c(x) &= p_1(x) + x^k(a_1(x)b_2(x) + a_2(x)b_1(x)) + x^{2k}p_2(x) = \\ &= p_1(x) + x^k(t(x) - p_1(x) - p_2(x)) + x^{2k}p_2(x) \end{aligned}$$

Example

$$\begin{aligned} c(x) &= (6x^2 + 16x + 8) + x^2 \cdot (27x^2 + 105x + 88 - (6x^2 + 16x + 8) - (6x^2 + 33x + 36)) + \\ &+ x^4 \cdot (6x^2 + 33x + 36) = 8 + 16x + 50x^2 + 56x^3 + 51x^4 + 33x^5 + 6x^6 \end{aligned}$$

### 5.2.3 Basis of range proofs

Let’s imagine that we need to prove that  $0 \leq a < 2^n$ .

Why we use  $2^n$ : This gives us the knowledge that if the statement is true, then the length of the binary form of  $a$  will be  $n$  bits [28].

We can also represent the initial secret in the form of two vectors:

- The first vector is responsible for the binary representation of  $a$ .
- The second vector is for degrees of 2.

For example, we have  $a = 5$ .

**Note 10.** *The simple commitment of this value will look like  $A = Com(a) = Com(a, a') = a*B + a'B'$ , where  $a'$  is a blinding factor for the value  $a$ ;  $B$  and  $B'$  are the generators used for the values and blinding factors.*

Using the above math, we get:

With example

<p>Every number can be represented as two vectors, so we represent <math>a</math> as:</p> <p><math>a = \langle \overline{a_{\text{binary}}}, \overline{2^n} \rangle</math>, where:</p> <p><math>\overline{a_{\text{binary}}}</math> - binary form of <math>a</math></p> <p><math>\overline{2^n}</math> - a vector consisting of powers of 2</p>	$\overline{a_{\text{binary}}} = (\overline{0,1,0,1})$ $\overline{2^n} = (\overline{2^3, 2^2, 2^1, 2^0})$ $a = \langle (\overline{0,1,0,1}), (\overline{2^3, 2^2, 2^1, 2^0}) \rangle = 5$
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------



### 5.2.4 From $a = \langle \overline{a_{\text{binary}}}, \overline{2^n} \rangle$ to $t(x) = \langle l(x), r(x) \rangle$ .

The whole scheme of mathematical manipulation ultimately looks like this:

$\emptyset < a < 2^n$	
$a = \langle \overline{a_{\text{binary}}}, \overline{2^n} \rangle$ $\overline{a_{\text{binary}}} \cdot (\overline{a_{\text{binary}}} - \overline{1}) = \emptyset$	
$a = \langle \overline{a_L}, \overline{2^n} \rangle$ $\overline{a_L} \cdot \overline{a_R} = \overline{\emptyset}$ $(\overline{a_L} - \overline{1}) - \overline{a_R} = \overline{\emptyset}$	$\overline{a_{\text{binary}}} = \overline{a_L}$ $\overline{a_{\text{binary}}} - \overline{1} = \overline{a_R}$
$a = \langle \overline{a_L}, \overline{2^n} \rangle$ $\langle (\overline{a_L} - \overline{1}) - \overline{a_R}, \overline{y^n} \rangle = \overline{\emptyset}$ $\langle \overline{a_L}, \overline{a_R} \cdot \overline{y^n} \rangle = \overline{\emptyset}$	combining each of the two vector-statements into a single statement vector $\overline{y}$ generated by verifier
$z^2 a = z^2 \langle \overline{a_L}, \overline{2^n} \rangle$ $+ z \langle (\overline{a_L} - \overline{1}) - \overline{a_R}, \overline{y^n} \rangle$ $+ \langle \overline{a_L}, \overline{a_R} \cdot \overline{y^n} \rangle$	combining three statements into a single statement $z$ obtained from the verifier then transformation to $a$ from in with $a_L$ , $a_R$ , and the non-secret values are located separately
$z^2 a + \delta(y, z)$ $= \langle (\overline{a_L} - z\overline{1}, \overline{y^n} \cdot (\overline{a_R} - z\overline{1}) + z^2 \overline{2^n}) \rangle$	$\delta(y, z) = (z - z^2) \langle \overline{1}, \overline{y^n} \rangle - z^3 \langle \overline{1}, \overline{2^n} \rangle$
$z^2 a + \delta(y, z) =$ $\langle \text{unblinded } \overline{l(x)}, \text{unblinded } \overline{r(x)} \rangle$	unblinded $l(x) = \overline{a_L} - z\overline{1}$ unblinded $r(x) = \overline{y^n} \cdot (\overline{a_R} - z\overline{1}) + z^2 \overline{2^n}$
$z^2 a + \delta(y, z) = \langle \overline{l_\emptyset}, \overline{r_\emptyset} \rangle$	$l_\emptyset = \text{unblinded } l(x) \quad l_1 x = \overline{s_L} x$ $r_\emptyset = \text{unblinded } r(x) \quad r_1 x = \overline{s_R} x$
$t(x) = \langle \overline{l(x)}, \overline{r(x)} \rangle$	$\overline{l(x)} = \overline{l_\emptyset} + \overline{l_1} x = \overline{a_L} + \overline{s_L} x - z\overline{1}$ $\overline{r(x)} = \overline{r_\emptyset} + \overline{r_1} x = \overline{y^n} \cdot (\overline{a_R} + \overline{s_R} x) + z\overline{1} + z^2 \overline{2^n}$

**Note 11.** Below, the designation “\*” will be used in the context of vectors. It means element-wise multiplication of vectors.

To prove that our secret is in some interval, we need to prove the following two expressions:

$$\begin{aligned} a &= \langle \overline{a_{\text{binary}}}, \overline{2^n} \rangle \\ \overline{a_{\text{binary}}} \cdot (\overline{a_{\text{binary}}} - \overline{1}) &= \overline{0} \end{aligned}$$

**Note 12.** It is very important to be sure that the values of the bit representation vector  $\bar{a}$  are either 1 or 0. The second expression is responsible for this, since the multiplication of inverted vectors from 0 and 1 in result provides a vector of zero values.

We can define the following 2 vectors:

$$\begin{aligned} \overline{a_{\text{binary}}} &= \bar{a}_L \\ \overline{a_{\text{binary}}} - \overline{1} &= \bar{a}_R \end{aligned}$$

**Note 13.** The commitment of these vectors will look like

$$\text{Com}(\bar{a}_L, \bar{a}_R) = \langle \bar{a}_L, \bar{G} \rangle + \langle \bar{a}_R, \bar{H} \rangle + a'B$$

where  $G$  and  $H$  are vectors of generators,  $B'$  is the generator used for blinding factors.  $a'$  is a blinding factor for the value

Then we move on to the following expressions:

$$\begin{aligned} a &= \langle \bar{a}_L, \overline{2^n} \rangle \\ \bar{a}_L \cdot \bar{a}_R &= \overline{0} \\ (\bar{a}_L - \overline{1}) - \bar{a}_R &= \overline{0} \end{aligned}$$

Then the challenge must be added to combine each of the two vector statements into a single statement. It will be added as a vector  $y$  generated by the verifier. So it is possible to modify the above expressions to this form:

$$\begin{aligned} a &= \langle \bar{a}_L, \bar{2}^n \rangle \\ \langle (\bar{a}_L - \bar{1}) - \bar{a}_R, \bar{y}^n \rangle &= \bar{0} \\ \langle \bar{a}_L, \bar{a}_R \cdot \bar{y}^n \rangle &= \bar{0} \end{aligned}$$

**Note 14.** Since  $\bar{b} = 0$  if and only if  $\langle \bar{b}, \bar{y} \rangle = 0$  (For example:  $\langle (1, 0, 1, 1, 1, 0), (0, -1, 0, 0, 0, -1) \rangle = (0, 0, 0, 0, 0, 0)$ ).

Then it is possible to combine these three statements into one, using the new value  $z$  obtained from the verifier:

$$z^2 a = z^2 \langle \bar{a}_L, \bar{2}^n \rangle + z \langle \bar{a}_L - \bar{1} - \bar{a}_R, \bar{y}^n \rangle + \langle \bar{a}_L, \bar{a}_R \cdot \bar{y}^n \rangle$$

Next, we need to make some transformations to bring our expression to a form in which  $a_L$ ,  $a_R$ , and the non-secret values are located separately:

$$\begin{aligned} z^2 a &= z^2 \langle \bar{a}_L, \bar{2}^n \rangle + z \langle \bar{a}_L, \bar{y}^n \rangle - z \langle \bar{a}_R, \bar{y}^n \rangle - z \langle \bar{1}, \bar{y}^n \rangle + \langle \bar{a}_L, \bar{a}_R \cdot \bar{y}^n \rangle \\ z^2 a + z \langle \bar{1}, \bar{y}^n \rangle &= z^2 \langle \bar{a}_L, \bar{2}^n \rangle + z \langle \bar{a}_L, \bar{y}^n \rangle - z \langle \bar{a}_R, \bar{y}^n \rangle + \langle \bar{a}_L, \bar{a}_R \cdot \bar{y}^n \rangle \\ z^2 a + z \langle \bar{1}, \bar{y}^n \rangle &= z^2 \langle \bar{a}_L, \bar{2}^n \rangle + z \langle \bar{a}_L, \bar{y}^n \rangle - z \langle \bar{1}, \bar{a}_R \cdot \bar{y}^n \rangle + \langle \bar{a}_L, \bar{a}_R \cdot \bar{y}^n \rangle \\ z^2 a + z \langle \bar{1}, \bar{y}^n \rangle &= \langle \bar{a}_L, z^2 \bar{2}^n \rangle + \langle \bar{a}_L, z \bar{y}^n \rangle + \langle -z \bar{1}, \bar{a}_R \cdot \bar{y}^n \rangle + \langle \bar{a}_L, \bar{a}_R \cdot \bar{y}^n \rangle \\ z^2 a + z \langle \bar{1}, \bar{y}^n \rangle &= \langle \bar{a}_L, z^2 \bar{2}^n + z \bar{y}^n + \bar{a}_R \cdot \bar{y}^n \rangle + \langle -z \bar{1}, \bar{a}_R \cdot \bar{y}^n \rangle \end{aligned}$$

And then we just add

$$\langle -z \bar{1}, z^2 \bar{2}^n + z \bar{y}^n \rangle$$

to each part of the equation:

$$\begin{aligned} z^2 a + z \langle \bar{1}, \bar{y}^n \rangle - \langle z \bar{1}, z^2 \bar{2}^n + z \bar{y}^n \rangle &= \langle \bar{a}_L, z^2 \bar{2}^n + z \bar{y}^n + \bar{a}_R \cdot \bar{y}^n \rangle + \\ &+ \langle -z \bar{1}, \bar{a}_R \cdot \bar{y}^n \rangle - \langle z \bar{1}, z^2 \bar{2}^n + z \bar{y}^n \rangle \\ z^2 a + z \langle \bar{1}, \bar{y}^n \rangle - \langle z \bar{1}, z^2 \bar{2}^n + z \bar{y}^n \rangle &= \langle \bar{a}_L, z^2 \bar{2}^n + z \bar{y}^n + \bar{a}_R \cdot \bar{y}^n \rangle + \\ &+ \langle -z \bar{1}, z^2 \bar{2}^n + z \bar{y}^n + \bar{a}_R \cdot \bar{y}^n \rangle \\ z^2 a + z \langle \bar{1}, \bar{y}^n \rangle - \langle z \bar{1}, z^2 \bar{2}^n \rangle - \langle z \bar{1}, z \bar{y}^n \rangle &= \langle \bar{a}_L - z \bar{1}, z^2 \bar{2}^n + z \bar{y}^n + \bar{a}_R \cdot \bar{y}^n \rangle \\ z^2 a + z \langle \bar{1}, \bar{y}^n \rangle - z^3 \langle \bar{1}, \bar{2}^n \rangle - z^2 \langle \bar{1}, \bar{y}^n \rangle &= \langle \bar{a}_L - z \bar{1}, z^2 \bar{2}^n + z \bar{y}^n + \bar{a}_R \cdot \bar{y}^n \rangle \\ z^2 a + (z - z^2) \langle \bar{1}, \bar{y}^n \rangle - z^3 \langle \bar{1}, \bar{2}^n \rangle &= \langle \bar{a}_L - z \bar{1}, z^2 \bar{2}^n + z \bar{y}^n + \bar{a}_R \cdot \bar{y}^n \rangle \end{aligned}$$

Eventually, we can replace all the non-secret parts with:

$$\delta(\bar{y}, z) = (z - z^2) \langle \bar{1}, \bar{y}^n \rangle - z^3 \langle \bar{1}, \bar{2}^n \rangle$$

The expression will take the form:

$$z^2 a + \delta(y, z) = \langle \bar{a}_L - z\bar{1}, \bar{y}^n \cdot (\bar{a}_R + z\bar{1}) + z^2 \bar{2}^n \rangle$$

Then we can designate:

$$\text{unblinded } \overline{l(x)} = \bar{a}_L - z\bar{1}$$

$$\text{unblinded } \overline{r(x)} = \bar{y}^n \cdot (\bar{a}_R + z\bar{1}) + z^2 \bar{2}^n$$

$$z^2 a + \delta(y, z) = \langle \text{unblinded } \overline{l(x)}, \text{unblinded } \overline{r(x)} \rangle$$

Then we need to hide these unblinded values because this form does not allow us to send this data to the verifier without revealing  $a$ .

The prover selects vectors  $\bar{s}_L$  and  $\bar{s}_R$ , and then uses them to blind unblinded  $\overline{l(x)}$  and unblinded  $\overline{r(x)}$ .

Let's denote the unblinded parts:

$$\bar{l}_\theta = \text{unblinded } l(x) = \bar{a}_L - z\bar{1}$$

$$\bar{r}_\theta = \text{unblinded } r(x) = \bar{y}^n \cdot (\bar{a}_R + z\bar{1}) + z^2 \bar{2}^n$$

So the blinded  $\overline{l(x)}$  and  $\overline{r(x)}$  will look like this:

$$\overline{l(x)} = \bar{l}_\theta + \bar{l}_1 x = (\bar{a}_L + \bar{s}_L x) - z\bar{1}$$

$$\quad \quad \quad || \quad \quad ||$$

blinding factor

$$\overline{r(x)} = \bar{r}_\theta + \bar{r}_1 x = \bar{y}^n \cdot ((\bar{a}_R + \bar{s}_R x) + z\bar{1}) + z^2 \bar{2}^n$$

$$\quad \quad \quad || \quad \quad ||$$

blinding factor

So we have

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

- this gives the understanding that  $a$  is in the interval from 0 to  $2^n$ .

Next, we need to get the product of our two blinded vectors. Vectors are multiplied, using Karatsuba's method:

$$t(x) = \langle \overline{l(x)}, \overline{r(x)} \rangle = t_0 + t_1x + t_2x^2$$

where:

$$t_0 = \langle \overline{l_0}, \overline{r_0} \rangle = z^2a + \delta(y, z)$$

$$t_2 = \langle \overline{l_1}, \overline{r_1} \rangle$$

$$t_1 = \langle \overline{l_0} + \overline{l_1}, \overline{r_0} + \overline{r_1} \rangle - t_0 - t_2$$

$$t_0 = \langle \overline{a_L} - z\overline{1}, \overline{y^n} \cdot (\overline{a_R} + z\overline{1}) + z^2\overline{2^n} \rangle$$

Next, we need to prove that

$$t_0 = z^2a + \delta(y, z)$$

and that  $t(x)$  is the correct polynomial.

To do this, the prover forms a commitment to the coefficients of  $t(x)$  and then convinces the verifier that he knows  $t(x)$  by evaluating the polynomial at a challenge point  $x$ .

Then the prover calculates

$$T_1 = \text{Com}(t_1) = \text{Com}(t_1, t'_1) = t_1B + t'_1B'$$

and

$$T_2 = \text{Com}(t_2) = \text{Com}(t_2, t'_2) = t_2B + t'_2B'$$

and sends  $A, T_1, T_2$  to the verifier. The commitments  $A, T_1, T_2$  are related to each other and to  $t(x)$  in this way:

$$t(x)B = (z^2a + \delta(y,z) + t_1x + t_2x^2)B$$

$$t'(x)B' = (z^2a' + 0 + t_1'x + t_2'x^2)B'$$

$$z^2a + z^2a' = z^2A$$

$$t_1x + t_1'x = T_1x$$

$$t_2x^2 + t_2'x^2 = T_2x^2$$

So, to convince the verifier that

$$t(x) = z^2a + \delta(y,z) + xt_1 + x^2t_2$$

the prover sends  $t(x)$  and  $t(x)'$  to him. The verifier, in order to verify the validity of this equation, verifies:

$$t(x)B + t'(x)B' = z^2A + \delta(y,z) + xT_1 + x^2T_2$$

$$t(x) = z^2a + \delta(y,z) + xt_1 + x^2t_2$$

The next step is to prove that  $\overline{l(x)}$  and  $\overline{r(x)}$  are correct [\[29\]](#).

## 5.3 Inner product proof

After we understand how Pedersen commitments and range proofs work, we need to understand how it all fits together.

So we have an expression

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

and we can merge it with the Pedersen commitment this way:

$$P = \langle \overline{l(x)}, \overline{G} \rangle + \langle \overline{r(x)}, \overline{H'} \rangle$$

Let's change this expression a little:

$$\overline{l(x)} \Rightarrow \bar{a}$$

$$\overline{r(x)} \Rightarrow \bar{b}$$

$$\overline{t(x)} \Rightarrow c = \langle \bar{a}, \bar{b} \rangle$$

We have:

$$P = \langle \bar{a}, \bar{G} \rangle + \langle \bar{b}, \bar{H} \rangle$$

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

Now we need to combine these equations, so to do this, we determine a new indeterminate variable  $w$ , orthogonal generator  $B$ , and multiply  $\langle \bar{a}, \bar{b} \rangle$  by these values:

$$P = \langle \bar{a}, \bar{G} \rangle + \langle \bar{b}, \bar{H} \rangle$$

+

$$cwB = \langle \bar{a}, \bar{b} \rangle wB$$

Then denote:

$$k = \lg n$$

$$P' = P + cwB$$

$$Q = wB$$

As a result:

$$P' = \langle \bar{a}, \bar{G} \rangle + \langle \bar{b}, \bar{H} \rangle + \langle \bar{a}, \bar{b} \rangle Q$$



## 5.4 Inner product compression

We can benefit from the fact that the equation

$$P' = \langle \bar{a}, \bar{G} \rangle + \langle \bar{b}, \bar{H} \rangle + \langle \bar{a}, \bar{b} \rangle Q$$

is combined. We can use compression  $\lg n$  times and make it look like we have just the scalar product of two vectors.

Let's consider how scalar product compression can work on two vectors.

$O(n)$  time and space would take a "non-compact proof," while a compact would take  $O(\log(n))$ .

### 5.4.1 How inner product compression works.

With example

1. We have two polynomials have to be multiplied:

$$\bar{a} = \begin{array}{|c|c|c|c|c|c|c|c|} \hline \square & \square & \square & \square & \square & \square & \square & \square \\ \hline \end{array}$$

$\bar{a}$                        $\bar{a}$

$$\bar{b} = \begin{array}{|c|c|c|c|c|c|c|c|} \hline \square & \square & \square & \square & \square & \square & \square & \square \\ \hline \end{array}$$

$\bar{b}_{Lo}$                        $\bar{b}_{hi}$

The simplest case when we have length  $a$  and  $b$  equal to 2.

$$\bar{a} = \begin{array}{|c|c|} \hline 1 & 0 \\ \hline \end{array}$$

$\bar{a}$        $\bar{a}$

$$\bar{b} = \begin{array}{|c|c|} \hline 0 & 1 \\ \hline \end{array}$$

$\bar{b}_{Lo}$        $\bar{b}_{hi}$

- Then the prover gets a random scalar  $x$  from the verifier. It is needed to combine the hi and lo parts of  $a$  and  $b$ . It allows for the creation of  $\bar{a}'$  and  $\bar{b}'$  - more compact versions of  $\bar{a}$  and  $\bar{b}$ . And it also allows calculating  $C' = \langle \bar{a}', \bar{b}' \rangle$ .

$$\begin{aligned}\bar{a}' &= \bar{a}_{lo}x + \bar{a}_{hi}x^{-1} \\ \bar{b}' &= \bar{b}_{lo}x^{-1} + \bar{b}_{hi}x\end{aligned}$$

$$\begin{aligned}C &= \langle \bar{a}, \bar{b} \rangle \\ C' &= \langle \bar{a}', \bar{b}' \rangle\end{aligned}$$

$$x = 3$$

$$\begin{aligned}\bar{a}' &= 1 \cdot 3 + 0 \cdot \frac{1}{3} = 3 \\ \bar{b}' &= 0 \cdot \frac{1}{3} + 1 \cdot 3 = 3\end{aligned}$$

$$\begin{aligned}C &= \langle (1, 0), (0, 1) \rangle = 0 \\ C' &= \langle (3), (3) \rangle = 9\end{aligned}$$

- We can also represent  $C'$  in another form, which is convenient for verification by the verifier.

$$\begin{aligned}C' &= \langle \bar{a}_{lo}x + \bar{b}_{lo}x, \bar{a}_{hi}x^{-1} + \bar{b}_{hi}x^{-1} \rangle = \\ &= \langle \bar{a}_{lo}x, \bar{b}_{lo}x \rangle + \langle \bar{a}_{hi}x, \bar{b}_{hi}x \rangle + \\ &+ x^2 \langle \bar{a}_{lo}, \bar{b}_{hi} \rangle + x^{-2} \langle \bar{a}_{hi}, \bar{b}_{lo} \rangle\end{aligned}$$

Denote:

$$\begin{aligned}C &= \langle \bar{a}_{lo}x, \bar{b}_{lo}x \rangle + \langle \bar{a}_{hi}x, \bar{b}_{hi}x \rangle \\ L &= \langle \bar{a}_{lo}, \bar{b}_{hi} \rangle \\ R &= \langle \bar{a}_{hi}, \bar{b}_{lo} \rangle\end{aligned}$$

$$\begin{aligned}L &= \langle (1), (1) \rangle = 1 \\ R &= \langle (0), (0) \rangle = 0\end{aligned}$$

- The prover sends  $L$  and  $R$  to the verifier.

$$C' = C + x^2L + x^{-2}R$$

$$C' = 0 + 9 \cdot 1 + 1/9 \cdot 0 = 9$$

- Then the same process repeats  $k = \log(n)$  times to achieve maximum compactness of  $\bar{a}'$  and  $\bar{b}'$ , when its length is 1. Every next round prover uses  $a', b', c'$  as  $a, b, c$ .
- At the end, the prover just sends  $C', a'$  and  $b'$  to the verifier.

**Note 15.** *The described scheme is a simple example of how inner product compression works. In a real protocol we have the equation:*

$$\begin{aligned} L_i &= \langle \overline{a_{lo}}, \overline{G_{hi}} \rangle + \langle \overline{b_{hi}}, \overline{H_{lo}} \rangle + \langle \overline{a_{lo}}, \overline{b_{hi}} \rangle Q \\ R_i &= \langle \overline{a_{hi}}, \overline{G_{lo}} \rangle + \langle \overline{b_{lo}}, \overline{H_{hi}} \rangle + \langle \overline{a_{hi}}, \overline{b_{lo}} \rangle Q \end{aligned}$$

*It is sent to the verifier, who responds with challenge  $x$ . So:*

$$\begin{aligned} a' &= \overline{a_{lo}}x + \overline{a_{hi}}x^{-1} \\ b' &= \overline{b_{lo}}x^{-1} + \overline{b_{hi}}x \end{aligned}$$

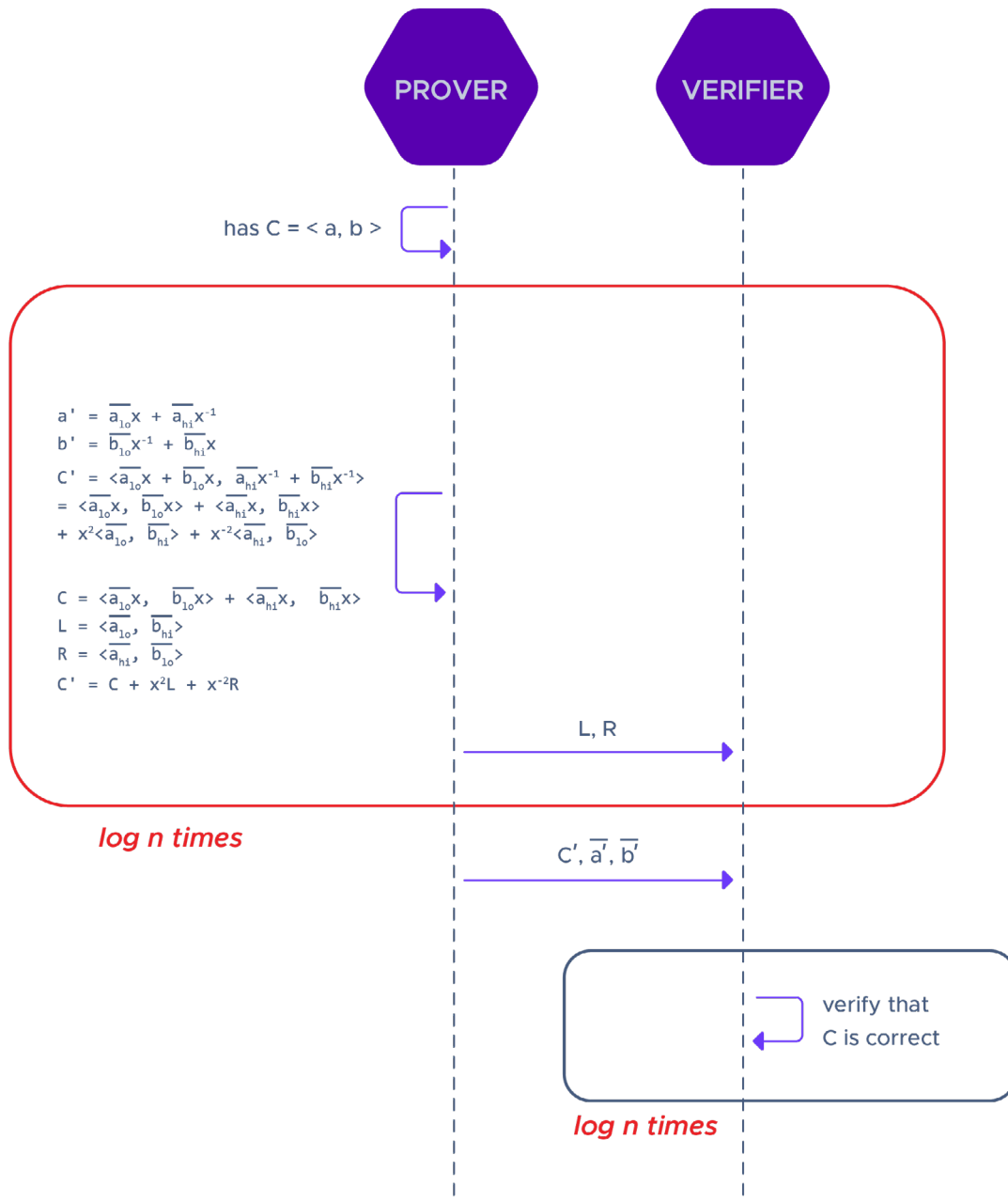
$$\begin{aligned} \overline{G'} &= \overline{G_{lo}}x^{-1} + \overline{G_{hi}}x \\ \overline{H'} &= \overline{H_{lo}}x + \overline{H_{hi}}x^{-1} \end{aligned}$$

*At the end, the prover sends values  $a$  and  $b$  to the verifier, and they verify that:*

$$L_1 u_1^{-2} + \dots + L_k u_k^{-2} + P' + R_k u_k^{-2} + \dots + R_1 u_1^{-2} = aG + bH + abQ$$

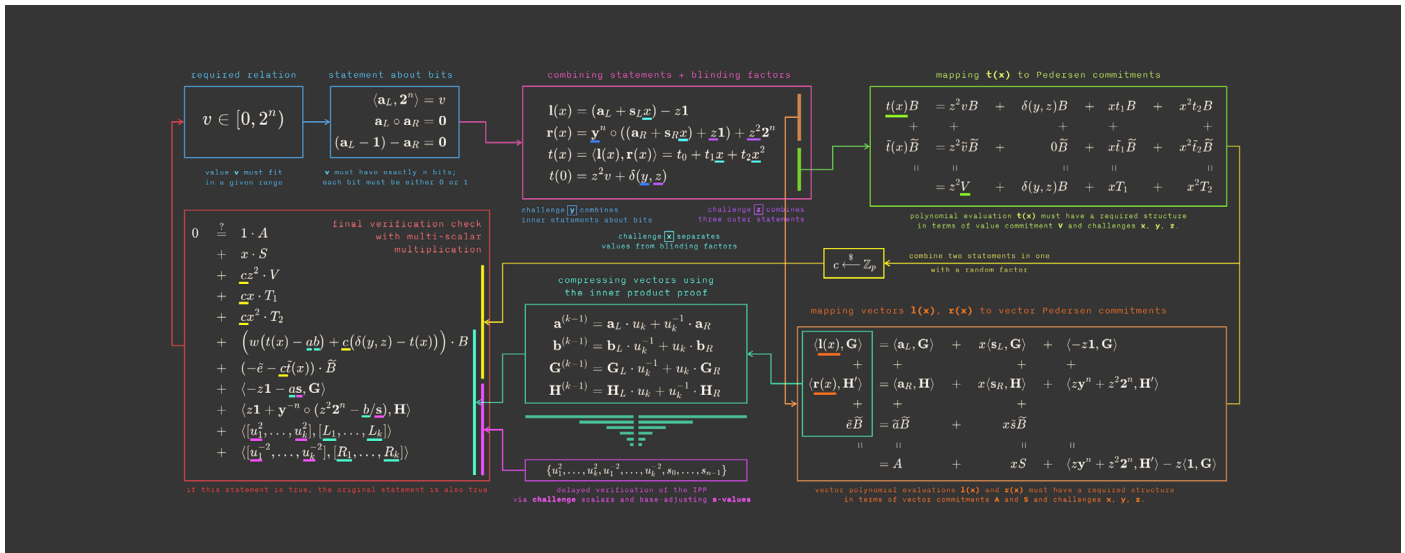
*To make such a protocol non-interactive, the obtaining of the challenge  $x$  should be replaced with Fiat-Shamir challenge. It's replaced by a transcript of  $L$  and  $R$  values. At the end of all computations  $a, b, L_k, R_k, \dots, L_1, R_1$  are sent to the verifier.*

## 5.4.2 Scheme of interaction between prover and verifier with inner product proof creating and verifying



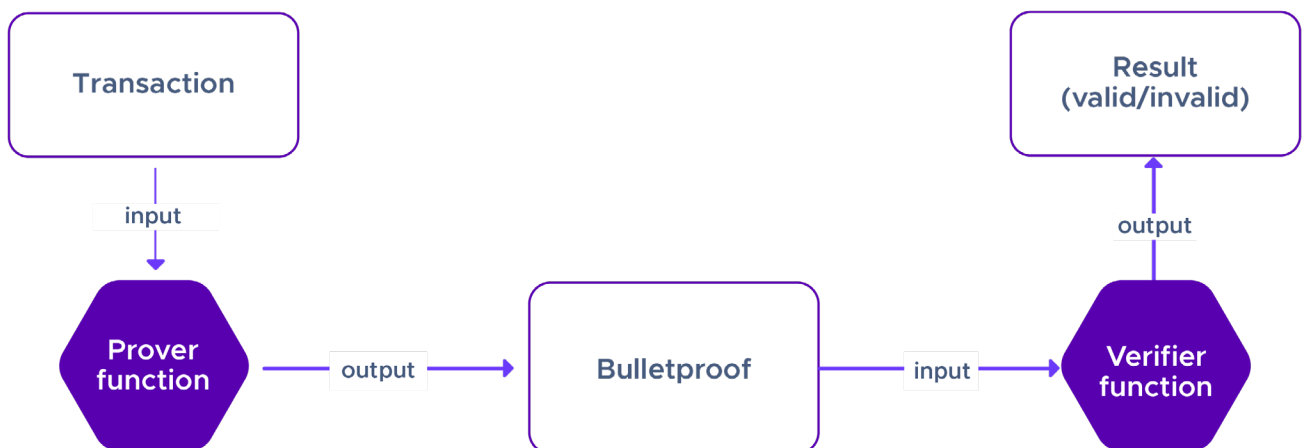
## 5.5 Summary

The most compact way of explaining the circuit was made by Cathie Yun, Henry de Valence, and Oleg Andreev. Many thanks to them for this scheme [31, 32]:



## 6. ALGORITHMS

The main idea of the relation between prover and verifier functions:



## 6.1 Prover algorithm

Different types of proofs can be implemented:

- Single proof
- Aggregated proof

### 6.1.1 Single proof

The single proof generation we described above in section [5](#).

### 6.1.2 Aggregated proof

An aggregated proof implies that a proof is generated by more than one participant. It has the same size as a single proof, but its generation and verification is a bit more complicated. The essence of proof aggregation is to aggregate range proofs that take up a lot of space. The easiest way to create an aggregated proof is to use a mediator.

Mediator's algorithm

Let's look at the generation of an aggregate proof with  $m$  participants:

Each of the participants generates 3 commitments:

$$\begin{aligned} V_i &= \text{Com} (v_i, v_i') = v_i B + v_i' B' \\ A_i &= \text{Com} (\overline{a_{L,i}}, \overline{a_{R,i}}) = \langle \overline{a_{L,i}}, \overline{G_i} \rangle + \langle \overline{a_{R,i}}, \overline{H_i} \rangle + a_i B' \\ S_i &= \text{Com} (\overline{s_{L,i}}, \overline{s_{R,i}}) = \langle \overline{s_{L,i}}, \overline{G_i} \rangle + \langle \overline{s_{R,i}}, \overline{H_i} \rangle + s_i B' \end{aligned}$$

Then provers send  $V_i, A_i, S_i$  to the mediator, who calculates:

$$\begin{aligned} A &= \sum_{i=0}^{m-1} A_i \\ S &= \sum_{i=0}^{m-1} S_i \end{aligned}$$

The mediator adds  $A$  and  $S$  to the protocol transcript and obtains challenge scalars  $y, z$  from the transcript. Then the mediator sends  $y, z$  values to all participants.

Each participant uses  $y, z$  as challenge and calculates:

$$\begin{aligned}\overline{l_{0,i}} &= \overline{a_{L,i}} - z\overline{1} \\ \overline{l_{1,i}} &= \overline{s_{L,i}} \\ \overline{r_{0,i}} &= \overline{y_i^n} \cdot (\overline{a_{R,i}} + z\overline{1}) + z^2\overline{2^n} \\ \overline{r_{1,i}} &= \overline{y_i^n} \cdot \overline{s_{R,i}}\end{aligned}$$

That is, all participants themselves act as a source of “provable randomness”.

Then everything is calculated as in the [above algorithm](#). As a result, each participant has two commitments -  $T_1$  and  $T_2$ , which they send to the mediator, who calculates:

$$\begin{aligned}T_1 &= \sum_{i=0}^{m-1} (T_1)_i \\ T_2 &= \sum_{i=0}^{m-1} (T_2)_i\end{aligned}$$

The mediator adds  $T_1$  and  $T_2$  to the protocol transcript, obtains a challenge  $x$ , and sends it to every participant.

Participants calculate the  $\overline{l(x)}$  and  $\overline{r(x)}$  using an obtained challenge  $x$ :

$$\begin{aligned}\overline{l_i(x)} &= \overline{l_{0,i}} + \overline{l_{1,i}}x = (\overline{a_{L,i}} + \overline{s_{L,i}}x) - z\overline{1} \\ \overline{r_i(x)} &= \overline{r_{0,i}} + \overline{r_{1,i}}x = \overline{y_i^n} \cdot ((\overline{a_{R,i}} + \overline{s_{R,i}}x) + z\overline{1}) + z^2\overline{2^n}\end{aligned}$$

$$\overline{t_i(x)} = \langle \overline{l(x)}, \overline{r(x)} \rangle = \overline{t_{0,i}} + \overline{t_{1,i}}x + \overline{t_{2,i}}x^2$$

Then each participant calculates synthetic blinding factors this way:

$$\begin{aligned}t_i'(x) &= z^2v_i' + t_{1,i}'x + t_{2,i}'x^2 \\ e_i' &= a_i' + s_i'x\end{aligned}$$



Participants send  $t_i(x)$ ,  $t'_i(x)$ ,  $e'_i(x)$ ,  $l_i(x)$  and  $r_i(x)$  to the mediator.

The mediator calculates:

$$t(x) = \sum_{i=0}^{m-1} t_{2,i}(x)$$

$$t'(x) = \sum_{i=0}^{m-1} t'_{2,i}(x)$$

$$e' = \sum_{i=0}^{m-1} e'_i$$

Then mediator adds it to the protocol transcript and obtains challenge scalar  $w$  and uses it to calculate

$$Q = w \cdot B$$

The mediator concatenates  $l_i(x)$  and  $r_i(x)$  and gets aggregated  $l(x)$  and  $r(x)$ .

$$l(x) = l_0(x) || l_1(x) || \dots || l_{m-1}(x)$$

$$r(x) = r_0(x) || r_1(x) || \dots || r_{m-1}(x)$$

The mediator performs the inner product argument protocol to prove such relation:

$$P' = \langle l, G \rangle + \langle r, H' \rangle + \langle l, r \rangle Q,$$

where  $\bar{H}' = y^{-n \cdot m} \cdot \bar{H}$

The result of the inner product proof is a list of  $2k$  points and 2 scalars, where

$$k = \log_2(n \cdot m)$$

The complete range proof consists of  $9+2k$  32 byte elements:

$$\{A, S, T_1, T_2, t(x), t'(x), e', L_k, R_k, \dots, L_1, R_1, a, b\}$$

## 6.2 Verifier algorithm

Consider verification of aggregated proof:

On the input verifier has

$$\{A, S, T_1, T_2, t(x), t'(x), e', L_k, R_k, \dots, L_1, R_1, a, b\}$$

To verify such a proof, the verifier uses a Fiat-Shamir transform to obtain challenges:

Verifier obtains challenges  $y, z$ , then obtains  $x$ , and then obtains  $w$ .

This way there is no need to transmit challenges, because the verifier itself can calculate them [33, 34].

Then the verifier calculates  $x$  values that were used during inner product proof compression and calculates  $s$  values.

Then the verifier checks two equations:

$$\theta = \sum_{i=0}^{m-1} z^{i+2} V_i + \delta(y, z) B + x T_1 + x^2 T_2 - t(x) B - t'(x) B'$$

$$\theta = P' + t(x) w B - \langle a \cdot s, \bar{G} \rangle - \langle \overline{y^{-nm}} \cdot (b/s, \bar{H}) \rangle - abwB - \sum_{i=1}^k (L_i x_i^2 + R_i x_i^{-2})$$

where

$$P' = e' B' + A + x S - z \langle \bar{1}, \bar{G} \rangle + \\ + z \langle \bar{1}, \bar{H} \rangle + \langle \overline{y^{-nm}} \cdot z^2 (z^0 \bar{2}^n || z^1 \bar{2}^n || \dots || z^{m-1} \bar{2}^n), \bar{H} \rangle$$

To simplify calculation, the verifier combines the two equations into one:

$$\begin{aligned}
\emptyset &= 1 \cdot A \\
&+ x \cdot S \\
&+ cz^2 \cdot V_0 + cz^3 \cdot V_1 + \dots + cz^{m+1} \cdot V_{m-1} \\
&+ cx \cdot t_1 \\
&+ (w(t(x) - ab) + c(\delta(y, z) - t(x))) \cdot B \\
&+ (-e' - ct'(x)) \cdot B \\
&+ \langle -z\bar{1} - as, \bar{G} \rangle \\
&+ \langle z\bar{1} + \overline{y^{-nm}} \cdot z^2(z^{\emptyset}\bar{2}^n || z^1\bar{2}^n || \dots || z^{m-1}\bar{2}^n) - b/s, \bar{H} \rangle \\
&+ \langle [u_1^2, \dots, u_k^2], [L_1, \dots, L_k] \rangle \\
&+ \langle [u_1^{-2}, \dots, u_k^{-2}], [R_1, \dots, R_k] \rangle
\end{aligned}$$


## 7. THE OFFSHIFT PROTOCOL

The Offshift protocol enables coins (or assets) to be turned into hidden inputs (through a special smart contract), users to transfer commitments (without disclosing the amount), and inputs to be withdrawn through a contract (with a corresponding payment to the owner's account).

### 7.1 Commitment creation (Deposit)

In order to create shielded tokens (turn them into cryptographic commitments), the user needs to perform the following steps:

1. Exchange token X for XFT (wherever they want);
2. Deposit XFT in the contract and define that they want to receive zkX. After funds are deposited they will be burned.
3. The contract receives a rate from the oracle and issues the corresponding amount of zkX to the user's account.



In this model, users can own the list of commitments, whether received from deposit or incoming payment.

An additional feature is the ability to merge commitments among themselves - that is, the user can create a transaction for aggregating his own funds (regardless of their number) into one. After merging commitments, all payments to other accounts will be mixed - no one will be able to determine the actual amount of transfers and/or change.

## 7.2 Payments

Also, users can form a kind of payment transaction. Such transactions contain: a set of unspent commitments, the account identifier of the recipient, and newly created commitments.

## 7.3 Withdrawal

When a user wants to receive the unblinded tokens, they can withdraw them. In this case, the user “pays” to the contract (opens the commitment values) and the contract mints (issues) the appropriate amount of XFT tokens:

1. User locks zkX on a contract.
2. The contract receives data from the oracle and provides the corresponding amount of XFT (by current price).
3. Users can exchange XFT for token X.

# 8. FUNCTIONALITY OF A SMART-CONTRACT AND ITS METHODS

When a user wants to receive the unblinded tokens, they can withdraw them. In this case, the user “pays” to the contract (opens the commitment values) and the contract mints (issues) the appropriate amount of XFT tokens:

## 9. FUTURE WORK & IMPLEMENTATIONS

We expect that future work will be related to the creation of decentralized applications on top of zero knowledge assets: exchanges, borrowing and lending, etc.

Another direction of improving the protocol will be an even greater level of privacy, which is associated not only with hiding the amount of transfers, but also with the sender and recipient of assets.

The transfer of the Offshift protocol to other infrastructures (for example, Moonbeam) for cross-platform, private asset transfers is upcoming as well.

To learn more and stay connected, join our [Discord](#), [GitLab](#), and check out our website at [offshift.io](https://offshift.io).

## APPENDIX A. NUMS

This method enables the generation of verifiable random values. NUMS allows a prover to pick values in a way that demonstrates the values were not selected for a "nefarious purpose" - for example, to create a "backdoor" to the algorithm [\[35, 36\]](#).

Examples of using such algorithms are:

- Ron Rivest used the trigonometric sine function to generate constants for the widely-used MD5 hash.
- RFC 3526 describes prime numbers for internet key exchange that are also generated from  $\pi$ .
- The key schedule of the RC5 cipher uses binary digits from both  $e$  and the golden ratio.
- To generate values, elliptic curves also can be used.

One easy way is to use a hash function:

Use encoding  $G$  in binary, perhaps in compressed or uncompressed form, take the SHA256 of that binary string and check if it is the coordinate of an elliptic curve point. Not all the hashes will be the points on the curve, but half of them. So it may be a simple iterative algorithm (e.g. concat encoded  $G$  with 1, 2, 3...).

To generate random values, any of the described methods can be used.

## APPENDIX B. FIAT-SHAMIR HEURISTIC

Application of this technique involves moving away from interactivity.

How it works:

To make the proof non-interactive, it is necessary to replace the step of receiving a challenge from the verifier with another algorithm of proven random value generation. A hash of the prover's messages can be a good replacement. The hash function is modeled as a random oracle (Random Oracle Model is a black box that outputs a random value even when the input is the same), which models the unpredictability of the verifier [\[37\]](#).

What makes bulletproofs so compact?

The length of vectors with each step is halved and then, in the base case, the length of  $a$  and  $b$  is 1.

# REFERENCES

1. <https://drive.google.com/file/d/1TMVuXPgiJ8ibzJehGxIZXksHPAXoywy/view>
2. <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.419.8132&rep=rep1&type=pdf>
3. <https://eprint.iacr.org/2019/191.pdf>
4. <https://blockgeeks.com/guides/ethereum-metropolis/>
5. <http://zerocash-project.org/media/pdf/zerocash-extended-20140518.pdf>
6. <https://iohk.io/en/research/library/papers/zendooa-zk-snark-verifiable-cross-chain-transfer-protocol-enabling-decoupled-and-decentralized-sidechains/>
7. <https://blog.ethereum.org/2016/12/05/zksnarks-in-a-nutshell/>
8. <https://academy.bit2me.com/en/what-are-zk-stark/>
9. <https://eprint.iacr.org/2018/046>
10. [http://cryptowiki.net/index.php?title=Zero-knowledge\\_Scalable\\_Transparent\\_Arguments\\_of\\_Knowledge\\_\(zk-STARKs\)](http://cryptowiki.net/index.php?title=Zero-knowledge_Scalable_Transparent_Arguments_of_Knowledge_(zk-STARKs))
11. <https://github.com/OxProject/OpenZKP>
12. <https://eprint.iacr.org/2018/046.pdf>
13. <https://medium.com/coinmonks/zk-starks-create-verifiable-trust-even-against-quantum-computers-dd9c6a2bb13d>
14. <https://crypto.stanford.edu/bulletproofs/>
15. <https://ethereum.stackexchange.com/questions/59145/zk-snarks-vs-zk-starks-vs-bulletproofs-updated/63778>
16. <https://github.com/matter-labs/awesome-zero-knowledge-proofs>
17. <https://www.getmonero.org/library/Zero-to-Monero-1-0-0.pdf>
18. <https://scalingbitcoin.org/papers/mimblewimble.txt>
19. <https://medium.com/@exantech/about-anonymity-in-account-based-blockchains-22a1ce5b0c7b>
20. <https://academy.glassnode.com/concepts/utxo>
21. [https://moodle.unige.ch/pluginfile.php/309148/mod\\_folder/content/0/From%20zero%20knowledge%20to%20bulletproofs%20-%20Gibson.pdf?forcedownload=1](https://moodle.unige.ch/pluginfile.php/309148/mod_folder/content/0/From%20zero%20knowledge%20to%20bulletproofs%20-%20Gibson.pdf?forcedownload=1)
22. <https://eprint.iacr.org/2017/1066.pdf>
23. <https://tlu.tarilabs.com/cryptography/bulletproofs-and-mimblewimble/MainReport.html>
24. [https://sikoba.com/docs/SKOR\\_DK\\_Bulletproofs\\_201905.pdf](https://sikoba.com/docs/SKOR_DK_Bulletproofs_201905.pdf)
25. [https://link.springer.com/content/pdf/10.1007%2F978-3-642-03356-8\\_12.pdf](https://link.springer.com/content/pdf/10.1007%2F978-3-642-03356-8_12.pdf)
26. <https://algorithmica.org/ru/karatsuba>

27. <https://otus.ru/nest/post/1795/>
28. <http://www0.cs.ucl.ac.uk/staff/J.Groth/MatrixZK.pdf>
29. <https://github.com/ElementsProject/secp256k1-zkp>
30. <https://devblogs.microsoft.com/oldnewthing/20180404-00/?p=98435>
31. <https://cathieyun.medium.com/building-on-bulletproofs-2faa58af0ba8>
32. <https://doc-internal.dalek.rs/bulletproofs/index.html>
33. <https://doc-internal.dalek.rs/bulletproofs/notes/index.html>
34. <https://github.com/apoelstra/secp256k1-zkp/tree/bulletproofs>
35. <https://datatracker.ietf.org/doc/html/draft-black-numscurves-02>
36. [https://www.researchgate.net/publication/272352484\\_NUMS\\_Elliptic\\_Curves\\_and\\_Their\\_Implementation\\_Invited\\_Talk\\_at\\_the\\_University\\_of\\_Washington-Tacoma\\_2014](https://www.researchgate.net/publication/272352484_NUMS_Elliptic_Curves_and_Their_Implementation_Invited_Talk_at_the_University_of_Washington-Tacoma_2014)
37. <https://merlin.cool/print.html>