Bulletproofs *

Dmitry Khovratovich

Sikoba Research

15th May 2019

1 Introduction

Bulletproofs [BBB+18] is a recent technique for verifiable computation that is particularly efficient for range proofs (they take only 600 bytes). Bulletproofs has been recently implemented for a few privacy-oriented cryptocurrencies, including Monero [mon18], to reduce the range proof sizes.

Bulletproofs has the following features:

- It does not require a trusted setup as compared to ZK-SNARKs [BCG+13];
- It does not use pairings and works with any elliptic curve with a reasonably large subgroup size; the fastest elliptic curves such as Ristretto [ris18] are supported.
- It uses its own format for computation, which is easily convertible to R1CS [PHGR13] and back using linear algebra.
- The verifier cost scales linearly with the computation size.

2 Performance

The exact performance of Bulletproofs depends on the chosen elliptic curve and undertaken optimization. With the optimization proposed in the original paper, we have the following complexity for proving a statement about a circuit with n multiplication gates:

- Prover makes 6 group multi-exponentiations of length $2n$, each taking $O(n/\log n)$ time using the Pippenger algorithm [Pip80], and makes $O(n)$ scalar multiplications in $\mathbb{F}_p$.
- The proof size is $8 + 2\log n$ group elements and 5 scalars.
- Verifier makes 1 group multi-exponentiation of length $2n$, taking $O(n/\log n)$ time, and also makes $O(n)$ scalar multiplications. Benchmarks demonstrate that the Verifier running time is roughly $1/20$ of the Prover running time.

*Research supported by Fantom Foundation

3 Technical Details

3.1 Overview

3.1.1 Computation

Prover and Verifier agree on the computation to be executed. This computation must be converted to an arithmetic circuit which operates on a certain prime field $\mathbb{F}_p$. The circuit should contain multiplication and addition gates, as well as scalar multiplication. Let circuit C take input variables I and produce output variables O, then the Bulletproofs protocol allows proving statements of the form

$$C(I) = O, \tag{1}$$

where each variable of I and O is either a public constant or a private variable (known only to Prover). We denote these external private variables by $\mathbf{v}$, while (private) variables that are local to the circuit are denoted by $\mathbf{a}$.

The circuit with n multiplication gates is described as a set of three vectors of field elements $\mathbf{a_L}, \mathbf{a_R}, \mathbf{a_O}$, which satisfy n multiplication constraints $\mathbf{a_L}_i \cdot \mathbf{a_R}_i = \mathbf{a_O}_i$, collectively denoted as

$$\mathbf{a_L} \circ \mathbf{a_R} = \mathbf{a_O}, \tag{2}$$

As this does not completely describe the circuit, we need to introduce q additional affine constraints

$$L_j(\mathbf{a_L}, \mathbf{a_R}, \mathbf{a_O}, \mathbf{v}) = 0. \tag{3}$$

which can be expressed as a matrix equation:

$$W_L\mathbf{a_L} + W_R\mathbf{a_R} + W_O\mathbf{a_O} + W_V\mathbf{v} = \mathbf{c}. \tag{4}$$

This is done so that (1) becomes equivalent to (2) and (4) taken together.

It is a simple linear algebra exercise to show that a set of *R1CS constraints* [PHGR13] of form $L_1(\mathbf{a}) \cdot L_2(\mathbf{a}) = L_3(\mathbf{a})$ can be converted to such representation by introducing new variables and performing Gaussian elimination.

3.1.2 Protocol

In short, the Bulletproofs protocol for arithmetic circuits works as follows.

1. Prover and Verifier agree on the circuit C in the format of equations (2) and (4).

2. Prover commits to internal circuit variables $\mathbf{a}$.

3. The equations apparently hold if for certain vector polynomials $\mathbf{l}, \mathbf{r}$, whose coefficients linearly depend on $\mathbf{a}$, the product t is a polynomial with coefficient t_2 being equal to the circuit-determined affine function of external variables: $t_2 = \mathcal{L}(\mathbf{v})$.

4. For random x Prover commits to $\mathbf{l}(x), \mathbf{r}(x), t(x)$, and other coefficients of t.

5. Using a clever inner-product argument, Prover proves that in the commitments above, $t(x)$ is a product of the former two committed evaluations $\mathbf{l}(x), \mathbf{r}(x)$.

6. Prover proves that $t(x)$ and other committed coefficients of t correspond to a polynomial with $t_2 = \mathcal{L}(\mathbf{v})$.

7. Prover proves that the committed evaluations $\mathbf{l}(x), \mathbf{r}(x)$ match the committed variables $\mathbf{a}$ according to the definition of $\mathbf{l}, \mathbf{r}$.

3.2 Details

1. (a) Parties agree on the circuit and its arithmetic representation. They select group $\mathbb{G}$ where DLP is hard (commonly – a group of elliptic curve points).

   (b) Parties agree on generators: $g, h, \mathbf{g}, \mathbf{h}$ (the last two are vectors of length n) in $\mathbb{G}$.

   (c) The external variables $\mathbf{v}$ are given as commitments $V_j = g^{v_j} h^{\gamma_j},\ j \in [1, m]$ where Prover knows all γ_j, v_j .

2. (a) Prover commits to $\mathbf{a_L}, \mathbf{a_R}, \mathbf{a_O}$ and blinding vectors $\mathbf{s_L}, \mathbf{s_R}$ with random α, β, ρ:

$$A_I = h^\alpha \mathbf{g}^{\mathbf{a_L}} \mathbf{h}^{\mathbf{a_R}}; \quad A_O = h^\beta \mathbf{g}^{\mathbf{a_O}}; \quad S = h^\rho \mathbf{g}^{\mathbf{s_L}} \mathbf{h}^{\mathbf{s_R}};$$

   (b) From the commitments Prover generates challenge values y, z.

3. (a) Equation (2) is equivalent to

$$\langle \mathbf{a_L} \circ \mathbf{a_R} - \mathbf{a_O}, \mathbf{y}^n \rangle = 0. \tag{5}$$

   where $\mathbf{y}^n = (1, y, y^2, \ldots, y^{n-1})$.

   (b) Equation (5) and (4) hold together if

$$\mathbf{z}^q W_L \mathbf{a_L} + \mathbf{z}^q W_R \mathbf{a_R} + \mathbf{z}^q W_O \mathbf{a_O} + \langle \mathbf{a_L} \circ \mathbf{a_R} - \mathbf{a_O}, \mathbf{y}^n \rangle = \langle \mathbf{z}^q, \mathbf{c} \rangle - \mathbf{z}^q W_V \mathbf{v}. \tag{6}$$

   where $\mathbf{z}^q = (z, z^2, \ldots, z^q)$.

   (c) Polynomials are defined as

$$\mathbf{l}(X) = \mathbf{a_L} X + \mathbf{a_O} X^2 + \mathbf{s_L} X^3 + \mathbf{y}^{-n} \mathbf{z}^q W_R X;$$
$$\mathbf{r}(X) = (\mathbf{y}^n \circ \mathbf{a_R}) X - \mathbf{y}^n + \mathbf{z}^q W_O + \mathbf{z}^q W_L X + (\mathbf{y}^n \circ \mathbf{s_R}) X^3.$$

   (d) Let $t(X) = \langle \mathbf{l}(X), \mathbf{r}(X) \rangle = \sum_{i \in [1,6]} t_i X^i$. Then (6) holds if

$$t_2 = \mathbf{y}^{-n} \mathbf{z}^q W_R \mathbf{z}^q W_L + \langle \mathbf{z}^q, c \rangle - \mathbf{z}^q W_v \mathbf{v}$$

4. (a) Prover commits to coefficients of t:

$$T_i = g^{t_i} h^{\tau_i}$$

   for randomly selected τ_i.

   (b) Prover computes x as as hash of all previous commitments.

   (c) Prover commits to $\mathbf{l}(x), \mathbf{r}(x), t(x)$:

$$C_1' = \mathbf{g}^{\mathbf{l}(x)} \mathbf{h}^{\mathbf{y}^{-n} \circ \mathbf{r}(x)}; \quad C_2' = g^{t(x)}. \tag{7}$$

5. Prover proves that C_2' is a commitment to the inner product of what is committed in C_1' using a special subroutine that produces a logarithmic-size proof. Concretely, we prove that for given P we know $\mathbf{a}, \mathbf{b}$ such that

$$P = \mathbf{g}^{\mathbf{a}} \mathbf{h}^{\mathbf{b}} g^{\langle \mathbf{a}, \mathbf{b} \rangle}. \tag{8}$$

   This is done as follows:

   (a) Partition all vectors in the left and right parts and denote them $_{left}$ and $_{right}$, respectively.

(b) Compute

$$L = \mathbf{g}_{right}^{\mathbf{a}_{left}} \mathbf{h}_{left}^{\mathbf{b}_{right}} g^{\langle \mathbf{a}_{left}, \mathbf{b}_{right} \rangle}; \quad R = \mathbf{g}_{left}^{\mathbf{a}_{right}} \mathbf{h}_{right}^{\mathbf{b}_{left}} g^{\langle \mathbf{a}_{right}, \mathbf{b}_{left} \rangle};$$

and send to Verifier.

(c) Hash L, R, P and get x.

(d) Compute $\mathbf{a}' = x\mathbf{a}_{left} + \mathbf{a}_{right}/x$ and $\mathbf{b}' = \mathbf{b}_{left}/x + x\mathbf{b}_{right}$.

(e) Prove that

$$L^{x^2} \cdot P \cdot R^{1/x^2} = \mathbf{g}_{left}^{\mathbf{a}'/x} \mathbf{g}_{right}^{x\mathbf{a}'} \mathbf{h}_{left}^{x\mathbf{b}'} \mathbf{h}_{right}^{\mathbf{b}'/x} g^{\langle \mathbf{a}', \mathbf{b}' \rangle}. \tag{9}$$

This is done by taking vectors $\mathbf{a}', \mathbf{b}'$ out of the exponent and formulate (9) as (8). Then we do this recursively starting with step (a). For vectors of length 1 send them as is.

6. Prover proves that $\mathbf{l}(x)$ and $\mathbf{r}(x)$ in C_1' are formed according to the definition:

$$C_1' \stackrel{?}{=} A_I^x A_O^{x^2} S^{x^3} \mathbf{g}^{\mathbf{y}^{-n}\mathbf{z}^q W_R} \mathbf{h}^{-1^n + \mathbf{y}^{-n} \circ \mathbf{z}^q W_O + \mathbf{y}^{-n} \circ \mathbf{z}^q W_L x} h^{\alpha x + \beta x^2 + \rho x^3}$$

where $\mu = \alpha x + \beta x^2 + \rho x^3$ is given to Verifier as a single value.

7. Prover proves that $t(x)$ is the evaluation of a polynomial where coefficients t_i are determined by the commitments T_i, commitments $\mathbf{V}$ and $\delta(y, z, c) = \mathbf{y}^{-n}\mathbf{z}^q W_R \mathbf{z}^q W_L + \langle \mathbf{z}^q, c \rangle$. As in the previous proof, this is done in exponent using T_i as bases:

$$C_2' h^{\tau_x} \stackrel{?}{=} g^{x^2 \delta(y,z,\mathbf{c})} \mathbf{V}^{x^2 \mathbf{z}^q \mathbf{W}_v} T_1^x T_3^{x^3} T_4^{x^4} T_5^{x^5} T_6^{x^6}.$$

where $\tau_x = \tau_1 x + \tau_3 x^3 + \tau_4 x^4 + \tau_5 x^5 + \tau_6 x^6$ is sent to Verifier as a single value..

4 Implementation

Bulletproofs can use any group of prime order where the discrete logarithm problem is hard. The fastest such groups with 128-bit security level are brought by elliptic curves such as Ed25519 [BDL+12]. Currently, two implementations are available: reference one that uses the secp256k1 curve, and *dalek* [dal18] that uses the Ristretto group [ris18] – a compressed group of Ed25519 points.

In the following text we consider the *dalek* implementation. *Dalek* supports both the natural Bulletproofs format for circuits and the R1CS format [BCG+13]. We show how to create proofs using the R1CS wrapper.

The procedure is as follows:

1. Prover and Verifier agree on the R1CS generation code (see below). Both Prover and Verifier will run a copy of this code at a later step.

2. Verifier creates a set of generators by calling `PedersenGens` and `BulletproofGens` and provides them to Prover.

3. Prover initializes the proof procedure by initializing the `Prover` class with the generators above.

4. Prover commits to variables $\mathbf{v}$ (see Section 2) by calling `commit`. The commitments are added to Prover's transcript. The transcript is used to create challenges.

5. Prover runs the R1CS generation code:

   - The set S of the variables that can be used in constraints is $\mathbf{v}$ in the beginning. Note that the values of S are known to the Prover class.
   - To create a constraint of form

   $$L_1(S) \times L_2(S) = L_3(S),$$

     Prover calls multiply with two arguments x and y, where x and y are created as LinearCombination of variables from S with the coefficients determined by L_1, L_2. The call returns three new variables x', y', z' with new additional constraints $x' = L_1(S)$, $y' = L_2(S)$, and $x' \times y' = z'$. Prover then calls constrain with $z' - z$ where z is created as LinearCombination of variables from S with the coefficients determined by L_3.

     Note that this procedure also evaluates all new variables and adds their values to the Prover object.
   - If Prover needs to create a variable not defined by a previous multiply call, he has to create a Variable object and provide the actual value for it to be included into S.

6. Prover calls prove to finalize the proof and passes it to Verifier.

7. Verifier creates the Verifier object and imports commitments.

8. Verifier runs the same R1CS generation code but he does not know the values.

9. Verifier checks the proof by calling verify.

5 Definitions

The vector exponentiation $\mathbf{h}^\mathbf{x}$ for vectors $\mathbf{h} = (h_1, h_2, \ldots, h_l), \mathbf{x} = (x_1, x_2, \ldots, x_l)$ of dimension l is defined as

$$\mathbf{h}^\mathbf{x} = h_1^{x_1} h_2^{x_2} \cdots h_l^{x_l},$$

References

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *IEEE Symposium on Security and Privacy*, pages 315–334. IEEE, 2018.

[BCG+13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. Snarks for C: verifying program executions succinctly and in zero knowledge. In *CRYPTO (2)*, volume 8043 of *Lecture Notes in Computer Science*, pages 90–108. Springer, 2013.

[BDL+12] Daniel J Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2(2):77–89, 2012.

[dal18] dalek-cryptography: Bulletproofs, 2018. `https://github.com/dalek-cryptography/bulletproofs`.

[mon18] Monero becomes bulletproof, 2018. `https://medium.com/digitalassetresearch/monero-becomes-bulletproof-f98c6408babf`.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *IEEE Symposium on Security and Privacy*, pages 238–252. IEEE Computer Society, 2013.

[Pip80] Nicholas Pippenger. On the evaluation of powers and monomials. *SIAM J. Comput.*, 9(2):230–250, 1980.

[ris18] The ristretto group, 2018. `https://ristretto.group/ristretto.html`.