README.md

The Tea/Horse Proving System

by Weiji Guo from Lightec Labs

Note: by the time of writing, this document has NOT been reviewed. It may contain mistakes of various kinds. Also most claims are NOT backed by benchmarks. This is so far a live document rather than a paper draft intended to be published. We will address reviewing and benchmarking as we move on.

Note: there are some issues rendering latex in GitHub/Browser. You may download the document to preview in VS Code. We will also provide a pdf version later.

We are developing the Tea/Horse proving system as the underlying scheme for the proposed OP_ZKP soft-fork upgrade to Bitcoin.

Introduction

The Tea/Horse proving system consists of three major parts:

- Inner-Product Argument (IPA), thus the (EC)DLP hardness assumption only, and the secp256k1 curve only.
- Aggregated Proving to effectively reduce the symptotical Verifier complexity to sub-linear.
- Single circuit for a recursive verifier, to address the linear verification key issue.

The name Tea/Horse comes from an ancient confidential transaction style, in which people traded tea for horse or vice versa, (perhaps sometimes) keeping the price/bidding information within sleeves, exchanged during hand-shaking. Below picture was taken in Songpan where the author attended a trail running event early July. He came up with the idea of aggregated proving on his flight going there.

Technical Requirements and Key Design

As had been explained in an email to the bitcoin developer mailing list, we are considering **Inner-Product Argument**. We could achieve succinct (sub-linear) verifier with **aggregated proving**, but are left with one open issue. That is, the verification key is too large for on-chain deployment.

Searching for a commitment scheme to address this open issue does not land a definitive answer. Instead, it seems quite the opposite, that we might not be able to find such a scheme. Circuit constants are essentially matrixes of size O(N^2), however sparse. And as pointed out before, these matrixes may not be sparse any more after adopting aggregated proving.

Luckily there is a third option besides (1)committing to the circuit constants, or (2)turning to Dory. That is, **recursive verification**. The idea is straightforward. Instead of verifying proofs from individual circuits directly, the OP_ZKP implementation has a verifier to verify proofs from one special circuit (the recursive circuit), which in turn recursively verifies proofs for business logic related circuits (application circuits).

The advantages are two fold:

(1) There is no need to deploy verification keys on-chain. The constants (and scheme-related public parameters) of the recursive verifier could be hard-coded in the Bitcoin client, taking up no block space. As for all application circuits, their circuit constants could be provided as witness to the recursive prover, thus hidden from the recursive verifier.

(2) It is possible to upgrade the recursive prover, the recursive verifier or the circuit without interrupting the applications based on OP_ZKP, in case of critical security patches. Still, it is each application circuit's responsibility to take care of their own security and efficiency.

Preliminary

We use bold lower case letters to denote vectors, for example, $\mathbf{v} = (v_0, v_1, v_2, \ldots, v_{d-1})$ v = (v0, v1, v2, ..., vd−1), with explicit or implicit length dd; bold number or its value representation to denote the vector of that number in increasing exponents, for example, $\mathbf{2}$2 = $(1, 2, 4, 8, \ldots, 2^{d-1})$(1, 2, 4, 8, ..., 2d−1) and $\mathbf{x_0}$x0 = $(1, x_0, x_0^2, \ldots, x_0^{d-1})$(1, x0, x02 , ..., x0d−1); bold upper case letters to denote matrixes, for example, $\mathbf{W}$W.

We use $[m]$[m] to denote close range $[1, m]$[1, m]. $[\cdot] \cdot [\cdot]$ denotes scalar multiplication for an implicit EC group $\mathbb{G}$G of order pp. $\langle \cdot, \cdot \rangle$⟨·, ·⟩ denotes the inner product operator, regardless of the types of the parameters: they could be either $\mathbb{Z}_p \times \mathbb{Z}_p \rightarrow \mathbb{Z}_p$Zp × Zp → Zp, or $\mathbb{Z}_p \times \mathbb{G} \rightarrow \mathbb{G}$Zp × G → G.

Throughout this document, the commitment scheme is Pedersen vector commitment. We also modify the Inner-Product Argument from Bulletproofs a bit following the Halo paper, as one of the two vectors are usually known.

Pedersen Vector Commitment

For EC group $\mathbb{G}$G of order pp, let $\mathbf{G} \in \mathbb{G}^d, H \in \mathbb{G}$G ∈ Gd, H ∈ G be random generators. We commit to a vector $\mathbf{v} \in \mathbb{Z}_p^d$v ∈ Zpd as

$$\text{Commit}(\mathbf{G}, H, \mathbf{v}, r): C = \langle \mathbf{v}, \mathbf{G} \rangle + [r]H$$

Commit(G, H, v, r) : C = ⟨v, G⟩ + [r]H

where $r \in \mathbb{Z}_p$r ∈ Zp is a random blinder providing hiding.

We may ignore the public parameters $\mathbf{G}$G and HH in the parameter list, and simply put

$$\text{Commit}(\mathbf{v}, r): C = \langle \mathbf{v}, \mathbf{G} \rangle + [r]H \quad (1)$$

Commit(v, r) : C = ⟨v, G⟩ + [r]H(1)

$\mathbf{G}$G and HH should be fixed for Bitcoin use.

To open the commitment (C, r)(C, r), check if the identity holds

$$\text{Open}(C, \mathbf{v}, r): C =? \langle \mathbf{v}, \mathbf{G} \rangle + [r]H$$

Open(C, v, r) : C =? ⟨v, G⟩ + [r]H

Opening Commitments with Modified IPA

IPA reduces the communication cost of opening a Pedersen vector commitment from linear to log(n). We adopt a modified version of IPA based on Halo, as in the case of opening a commitment CC of a polynomial $p(X)$p(X), the opening point is usually provided by Verifier, thus known.

For polynomial $p(X)$p(X) of degree less than dd, let $\mathbf{v}$v represents the coefficiencies of pp:

$$p(X) = \sum_{i=0}^{d-1} v_i \cdot X^i$$

p(X) = i=0∑d−1 vi · Xi

Let its commitment value CC be defined according to Formula 1. Suppose pp evaluates to aa at x_0x0:

$$a = p(x_0) = \sum_{i=0}^{d-1} v_i \cdot x_0^i = \langle \mathbf{v}, \mathbf{x_0} \rangle$$

a = p(x0) = i=0∑d−1 vi · x0i = ⟨v, x0⟩

To open CC at x_0x0 to value aa, we use another random generator $U \in \mathbb{G}$U ∈ G, such that:

$$P = C + [a]U = \langle \mathbf{v}, \mathbf{G} \rangle + [r]H + [\langle \mathbf{v}, \mathbf{x_0} \rangle]U \quad (2)$$

P = C + [a]U = ⟨v, G⟩ + [r]H + [⟨v, x0⟩]U (2)

Instead of providing $\mathbf{v}$v (and rr) to open the above commitment, we reduce it to another commitment with half the size (assuming $d = 2^k$d = 2k for some $k > 0$k > 0). To do so, let $\mathbf{v}_{odd}$vodd and $\mathbf{v}_{even}$veven denotes the odd-index part and even-index part of $\mathbf{v}$v, and similarly $\mathbf{G}_{odd}$Godd and $\mathbf{G}_{even}$Geven, $\mathbf{x}_{0_{odd}}$x0odd and $\mathbf{x}_{0_{even}}$x0even:

$$\mathbf{v}_{odd} = (v_1, v_3, \ldots, v_{d-1}) \quad \mathbf{v}_{even} = (v0, v2, \ldots, v_{d-2})$$

vodd = (v1, v3, ..., vd−1) veven = (v0, v2, ..., vd−2)

$$\mathbf{G}_{odd} = (G_1, G_3, \ldots, G_{d-1}) \quad \mathbf{G}_{even} = (G_0, G_2, \ldots, G_{d-2})$$

Godd = (G1, G3, ..., Gd−1) Geven = (G0, G2, ..., Gd−2)

$$\mathbf{x}_{0_{odd}} = (x_0, x_0^3, \ldots, x_0^{d-1}) \quad \mathbf{x}_{0_{even}} = (1, x_0^2, \ldots, x_0^{d-2})$$

x0odd = (x0, x03, ..., x0d−1) x0even = (1, x02, ..., x0d−2)

Given a challenge value $\gamma \in \mathbb{Z}_p$γ ∈ Zp, define new vectors of half the size:

$$\mathbf{v}^{'} = \mathbf{v}_{odd} + \gamma \cdot \mathbf{v}_{even}$$

v′ = vodd + γ · veven

$$\mathbf{G}^{'} = \mathbf{G}_{odd} + [\gamma^{-1}]\mathbf{G}_{even}$$

G′ = Godd + [γ−1]Geven

$$\mathbf{x_0^{'}} = \mathbf{x}_{0_{odd}} + \gamma^{-1} \cdot \mathbf{x}_{0_{even}}$$

x0′ = x0odd + γ−1 · x0even

The new commitment value becomes

$$P^{'} = \langle \mathbf{v}^{'}, \mathbf{G}^{'} \rangle + [r]H + [\langle \mathbf{v}^{'}, \mathbf{x_0^{'}} \rangle]U$$

P′ = ⟨v′, G′⟩ + [r]H + [⟨v′, x0′⟩]U

$$= P + [\gamma](\langle \mathbf{v}_{even}, \mathbf{G}_{odd} \rangle + [\langle \mathbf{v}_{even}, \mathbf{x}_{0_{odd}} \rangle]U) + [\gamma^{-1}](\langle \mathbf{v}_{odd}, \mathbf{G}_{even} \rangle + [\langle \mathbf{v}_{odd}, \mathbf{x}_{0_{even}} \rangle]U)$$

= P + [γ](⟨veven, Godd⟩ + [⟨veven, x0odd⟩]U) + [γ−1](⟨vodd, Geven⟩ + [⟨vodd, x0even⟩]U)

$$= P + [\gamma]L + [\gamma^{-1}]R$$

= P + [γ]L + [γ−1]R

where LL and RR are provided from Prover to Verifier before the latter generating the challenge value $\gamma\gamma$:

$$L = \langle \mathbf{v}_{even}, \mathbf{G}_{odd} \rangle + [\langle \mathbf{v}_{even}, \mathbf{x}_{0_{odd}} \rangle]U$$

L = ⟨veven, Godd⟩ + [⟨veven, x0odd⟩]U

$$R = \langle \mathbf{v}_{odd}, \mathbf{G}_{even} \rangle + [\langle \mathbf{v}_{odd}, \mathbf{x}_{0_{even}} \rangle]U$$

R = ⟨vodd, Geven⟩ + [⟨vodd, x0even⟩]U

Based on the values P, L, R, γP, L, R, γ, Verifier could calculate $P^{'}$P′ on its own.

Rename $(P^{'}, \mathbf{G}^{'}, \mathbf{v}^{'}, \mathbf{x_0^{'}})$(P′, G′, v′, x0′) to $(P, \mathbf{G}, \mathbf{v}, \mathbf{x_0})$(P, G, v, x0). Repeat this process till $\mathbf{v}^{'}$v′ contains only one element, than Prover can directly provide its value to Verifier. Verifier checks if the commitment opens correctly.

Remark Chosing odd/even rather than lower/higher halves is based on the need to align many sub-circuits with different size.

Batched Opening

Same Polynomial, Multiple Points

First we describe how to open a commitment value CC to a polynomial $p(X)$p(X) at multiple points $(x_1, x_2, \ldots, x_m)$(x1, x2, ..., xm) with IPA. Suppose we have $a_i = p(x_i), i \in [m]$ai = p(xi), i ∈ [m], we need to open CC at each $x_i$xi to $a_i$ai.

Let $U_i, i \in [m]$Ui, i ∈ [m] be mm random generators. At the begining we shall have

$$P = C + \sum_{i=1}^{m} [a_i]U_i = \langle \mathbf{v}, \mathbf{G} \rangle + [r]H + \sum_{i=1}^{m} [\langle \mathbf{v}, \mathbf{x_i} \rangle]U_i$$

P = C + i=1∑m[ai]Ui = ⟨v, G⟩ + [r]H + i=1∑m[⟨v, xi⟩]Ui

For each round, in a similar way we may have:

$$\mathbf{x_i}' = \mathbf{x}_{i_{odd}} + \gamma^{-1} \cdot \mathbf{x}_{i_{even}}$$

xi′ = xiodd + γ−1 · xieven

and

$$P' = P + [\gamma]L + [\gamma^{-1}]R$$

P′ = P + [γ]L + [γ−1]R

$$L = \langle \mathbf{v}_{even}, \mathbf{G}_{odd} \rangle + \sum_{i=1}^{m} [\langle \mathbf{v}_{even}, \mathbf{x}_{i_{odd}} \rangle]U_i$$

L = ⟨veven, Godd⟩ + i=1∑m[⟨veven, xiodd⟩]Ui

$$R = \langle \mathbf{v}_{odd}, \mathbf{G}_{even} \rangle + \sum_{i=1}^{m} [\langle \mathbf{v}_{odd}, \mathbf{x}_{i_{even}} \rangle]U_i$$

R = ⟨vodd, Geven⟩ + i=1∑m[⟨vodd, xieven⟩]Ui

Besides necessary values of $a_i$ai, there are no extra communication costs.