Thanks for your reviews! @levs57, @curryrasul, @AtHeartEngineer!

TL;DR

- Uses KZG polynomial commitment per epoch
- Uses KZG opening per message
- Should be less than 1 ms to generate a proof per message, which is almost 1000x improvment
- Finally, we can use RLN, the spam protection layer, for Tor network and the Ethereum validator network!

Intro

The PSE team is delivering a project RLN - Rate-Limiting Nullifier which is originated from Barry's post. Briefly, RLN is a spam protection layer which can be used in the anonymous network where we have the network level privacy. More technically, transmitter chooses a polynomial where its f(0)

is its private key and share a point on that polynomial when they send a message. Therefore, if f

is a n

-degree polynomial, te message limit becomes n

since the transmitter cannot share more than n

points not to get exploited the private key.

To make everything works well, we're using zkSNARK to prove the membership proof to ensure that the transmitter commited a proof of stake and also the message proof is containinng a point on the polynomial. But the problem is that it takes about 1 second to generate a proof per message and that is not affordable for many cases, for example Tor, Validator network, mobile environment, and etc.

By the way, we can use KZG polynomial commitment scheme and its opening which fits pretty perfect for the RLN scheme. Here's the detail technical setup about how to use KZG commitment & opening to achieve a 1ms proving time instead of 1 sec.

Preparation

- For a given epoch e

and a message limit n

, a user creates a polynomial f(x)

of degree n

, which satisfies the condition that f(0)

equals the user's private key pk

.

- Trusted setup:
- We need a shared common reference: g, g^\alpha, g^{\alpha^2}, ..., g^{\alpha^n}

for message limit n

- In the non-anonymous version, we should execute a trusted setup for each message limit,
- In the anonymous version, we can use an existing reference.
- In the non-anonymous version, we should execute a trusted setup for each message limit,
- In the anonymous version, we can use an existing reference.
- We need a shared common reference: g, g^\alpha, g^{\alpha^2}, ..., g^{\alpha^n}

for message limit n

- In the non-anonymous version, we should execute a trusted setup for each message limit,

- In the anonymous version, we can use an existing reference.
- In the non-anonymous version, we should execute a trusted setup for each message limit,
- In the anonymous version, we can use an existing reference.

Commitment

- User selects a n

-degree polynomial f

for an epoch to send maximum n

messages which f(0)

is the private key.

- User computes the KZG polynomial commitment C = g^{f(\alpha)}

using the reference string.

- User shares the polynomial commitment C

for the given epoch

- To send a message, user shares (f(m), g^{\psi_m(\alpha)})

where m

is the hash of the messgae value and g^{\psi_m(\alpha)}, \psi_m(x) = {f(x) - f(m) \over x-m}

is the opening proof

Evaluation of a message

- Calculate the hash of the message: m
- RLN message: m, f(m),g^{\psi_{m}(\alpha)}
- Verifier has the polynomial commitment of the given epoch g^{f(\alpha)}
- Verifier evaluates the message:

e(g^{f(\alpha)},g) \stackrel{?}{=} e(g^{\psi_m(\alpha)}, g^{\alpha}\cdot g^{-m})\cdot e(g,g)^{f(m)}

Various Commitment Schemes(for each epoch)

Version A: the simplest approach with a non-anonymous setting

- User's public key: g^{f(0)}
- User submits the commitment of the polynomial, its public key, and the opening proof. g^{f(\alpha)}, g^{\psi_{0}(\alpha)}, g^{f(0)}
- Verifier checks if the user's public key is on the committed polynomial by verifying the opening of the kzg commitment.

e(g^{\psi_{0}(\alpha)},g^\alpha)\cdot e(g,g^{f(0)}) \stackrel{?}{=} e(g, g^{f(\alpha)})

How KZG opening works?

To prove the opening of (\beta, f(\beta))

on f

The quotient \psi_\beta(x)

is

\psi_{\beta}(x) = {f(x) - f(\beta)\over x-\beta }

And then we can simply prove that opening by checking the pairings like below

$$e(g^{\psi_\beta(\alpha)}, g^\alpha \cdot g^{-\beta}) \cdot e(g, g^{f(\beta)}) \stackrel{?}{=} e(g^{f(\alpha)}, g)$$

because the left side of the equation should be

$$e(g^{\psi_\beta(\alpha)}, g^\alpha \cdot g^{-\beta}) \cdot e(g, g^{f(\beta)}) = e(g^{\frac{f(\alpha) - f(\beta)}{\alpha-\beta}}, g^{\alpha - \beta}) \cdot e(g, g^{f(\beta)}) = e(g^{f(\alpha) - f(\beta)}, g) \cdot e(g, g^{f(\beta)}) = e(g^{f(\alpha)}, g) \cdot e(g, g) = e(g^{f(\alpha)}, g)$$

Version B: Using zkp for the anonymous setting

- User creates a zkp to prove that
- public: $g^{f(\alpha)}$, n
- private: f(x), pk
- constraints
- f(0) = pk
- Membership proof of $g^{f(0)}$
- $c_i = 0$

when i>n

and $f(x) = \Sigma_{i=0}^{k} c_i x^i$

- f(0) = pk
- Membership proof of $g^{f(0)}$
- $c_i = 0$

when i>n

and $f(x) = \Sigma_{i=0}^{k} c_i x^i$

- public: $g^{f(\alpha)}$, n
- private: f(x), pk
- constraints
- f(0) = pk
- Membership proof of $g^{f(0)}$
- $c_i = 0$

when i>n

and $f(x) = \Sigma_{i=0}^{k} c_i x^i$

- f(0) = pk
- Membership proof of $g^{f(0)}$
- $c_i = 0$

when i>n

and $f(x) = \Sigma_{i=0}^{k} c_i x^i$

- And submits the proof π

and the polynomial commitment $g^{f(\alpha)}$

- Verifier checks the proof:

$$\text{verify}(\pi, g^{f(\alpha)}, \text{root}, n) \rightarrow true$$

Version C: Using multiple polynomial commitments for multi-epoch commitment

Commitment

- User creates m

polynomials of degree n

for m

epochs, then the user commit whole polynomials at once.

- For a given epoch e

, let’s say f_e(x)

is a polynomial of degree n

for that epoch.

- Then we can rewrite f_e(x) = \Sigma_{i=0}^{n} c_i(e) \cdot x^i
- For example,f_1(x) = c_0(1) + c_1(1) x + c_2(1) x^2 + ... + c_n(1) x^n\ f_2(x) = c_0(2) + c_1(2) x + c_2(2) x^2 + ... + c_n(2) x^n\ f_3(x) = c_0(3) + c_1(3) x + c_2(3) x^2 + ... + c_n(3) x^n\ ...\ f_m(x) = c_0(m) + c_1(m) x + c_2(m) x^2 + ... + c_n(m) x^n
- Then the user creates polynomial commitments g^{c_i(\alpha)}

for each c_i(e)

- And creates a zkp to prove that
- public: g^{c_i(\alpha)}, n, m
- private: c_i(e), pk
- constraints
- c_0(e) = pk

for every e \leq m

- c_i(e) = 0

when i>n

and f(x) = \Sigma_{i=0}^{k} c_i(e) x^i

- Membership proof of g^{f(0)}
- The polynomial commitment g^{c_i(\alpha)}

is correct.

- c_0(e) = pk

for every e \leq m

- c_i(e) = 0

when i>n

and f(x) = \Sigma_{i=0}^{k} c_i(e) x^i

- Membership proof of g^{f(0)}
- The polynomial commitment g^{c_i(\alpha)}

is correct.

- public: g^{c_i(\alpha)}, n, m
- private: c_i(e), pk

- constraints
- c_0(e) = pk

for every e \leq m

- c_i(e) = 0

when i>n

and f(x) = \Sigma_{i=0}^{k} c_i(e) x^i

- Membership proof of g^{f(0)}
- The polynomial commitment g^{c_i(\alpha)}

is correct.

- c_0(e) = pk

for every e \leq m

- c_i(e) = 0

when i>n

and f(x) = \Sigma_{i=0}^{k} c_i(e) x^i

- Membership proof of g^{f(0)}
- The polynomial commitment g^{c_i(\alpha)}

is correct.

Verification

- For the registration,
- Verifier checks the proof:
- Verifier checks the proof:

\text{verify}(\pi, g^{c_0(\alpha)}, ..., g^{c_n(\alpha)}, \text{root}, n, m) \rightarrow true

- For each epoch e

, * User submits * {(g^{c_0(e)}, g^{\phi_{0, e}(\alpha)}), (g^{c_1(e)}, g^{\phi_{1, e}(\alpha)}), ..., (g^{c_n(e)}), g^{\phi_{n, e}(\alpha)})}

where \phi_{(i, e)}(x) = {c_i(x) - c_i(e)\over x-e }

- g^{f_e(\alpha)}
- {(g^{c_0(e)}, g^{\phi_{0, e}(\alpha)}), (g^{c_1(e)}, g^{\phi_{1, e}(\alpha)}), ..., (g^{c_n(e)}), g^{\phi_{n, e}(\alpha)})}

where \phi_{(i, e)}(x) = {c_i(x) - c_i(e)\over x-e }

- g^{f_e(\alpha)}

- Then the verifier checks
- User submits
- {(g^{c_0(e)}, g^{\phi_{0, e}(\alpha)}), (g^{c_1(e)}, g^{\phi_{1, e}(\alpha)}), ..., (g^{c_n(e)}), g^{\phi_{n, e}(\alpha)})}

where \phi_{(i, e)}(x) = {c_i(x) - c_i(e)\over x-e }

- g^{f_e(\alpha)}
- {(g^{c_0(e)}, g^{\phi_{0, e}(\alpha)}), (g^{c_1(e)}, g^{\phi_{1, e}(\alpha)}), ..., (g^{c_n(e)}), g^{\phi_{n, e}(\alpha)})}

where \phi_{(i, e)}(x) = {c_i(x) - c_i(e)\over x-e }

- g^{f_e(\alpha)}

- Then the verifier checks

$g^{f_e(\alpha)} = \Pi_{i=0}^{n} g^{c_i(e)\alpha^i}$

by

$e(g, g^{f_e(\alpha)}) = e(g, \Pi_{i=0}^{n} g^{c_i(e)\alpha^i}) \stackrel{?}{=} \Pi_{i=0}^{n}e(g^{c_i(e)}, g^{\alpha^i})$

- For i = 0...n

, the verifier checks the submitted homomorphically hidden coefficients is in the polynomial commitments by

$e(g^{c_i(\alpha)}, g) \stackrel{?}{=} e(g^{\phi_{(i, e)}(\alpha)}, g^{\alpha} \cdot g^{-e}) \cdot e(g,g^{c_i(e)})$

where

$\phi_{(i, e)}(x) = {c_i(x) - c_i(e)\over x-e }$

- After then,
- Calculate the hash of the message: m
- RLN message: m, f(m),$g^{\psi_{m}(\alpha)}$
- Verifier evaluates the message:
- Calculate the hash of the message: m
- RLN message: m, f(m),$g^{\psi_{m}(\alpha)}$
- Verifier evaluates the message:

$e(g^{f(\alpha)},g) \stackrel{?}{=} e(g^{\psi_m(\alpha)} ,g^{\alpha}\cdot g^{-m})\cdot e(g,g)^{f(m)}$

Conclusion

Version A

Version B

Version C

Description

Pairing check using the public key

Use zkp for each epoch to prove the membership proof and polynomial commitment for each epoch’s secret sharing polynomial

Use zkp only once to prove the membership proof and the polynomial commitments of the coefficients of the secret sharing polynomials for multiple epochs

preparation

trusted setup for each message limit number

can use a universal trusted setup

can use a universal trusted setup

zkp

N/A

1 zkp per epoch

1 zkp per m

epochs

anonymity

no

yes

yes

proof size per epoch

1 group element

depends on zkp scheme

2n + 1 group elements

verification for each epoch

3 pairings

1 zkp

4n + 1 pairing (n: message limit)

proof size per message

1 group element

1 group element

1 group element

verification for each message

3 pairings

3 pairings

3 pairings