Aggregatable Subvector Commitments for Stateless Cryptocurrencies

Alin Tomescu[1]
@alinush407

Ittai Abraham[1]
@ittaia

Vitalik Buterin[2]
@VitalikButerin

Justin Drake[2]
@drakefjustin

Dankrad Feist[2]
@dankrad

Dmitry Khovratovich[2]
@Khovr

[1]VMware Research, [2]Ethereum Foundation

May 13th, 2020

Motivation

Stateless Cryptocurrencies

Stateless Cryptocurrencies

Stateful transaction validation (for account-based cryptocurrencies)

Stateless Cryptocurrencies

Stateful transaction validation (for account-based cryptocurrencies):

Check $tx = [\text{TXFER}, PK_i \rightarrow PK_j, v]$ against state on disk

Stateless Cryptocurrencies

Stateful transaction validation (for account-based cryptocurrencies):

$$\text{Check } tx = [\mathsf{TXFER}, PK_i \to PK_j, v] \text{ against state on disk}$$

State is just a dictionary $D(PK_i) \to bal_i$ (+ counters for replay attacks)

Stateless Cryptocurrencies

Stateful transaction validation (for account-based cryptocurrencies):

$$\text{Check } tx = [\mathsf{TXFER}, PK_i \to PK_j, v] \text{ against state on disk}$$

State is just a dictionary $D(PK_i) \to bal_i$ (+ counters for replay attacks)

Observations

- Could “authenticate” state and verify transaction against *digest* [Mil12, Tod16, But17, RMCI17]

Stateless Cryptocurrencies

Stateful transaction validation (for account-based cryptocurrencies):

$$\text{Check } tx = [\text{TXFER}, PK_i \rightarrow PK_j, v] \text{ against state on disk}$$

State is just a dictionary $D(PK_i) \rightarrow bal_i$ (+ counters for replay attacks)

Observations

- Could "authenticate" state and verify transaction against *digest* [Mil12, Tod16, But17, RMCI17]
- Each block t now stores digest d_t of the latest state

Stateless Cryptocurrencies

Stateful transaction validation (for account-based cryptocurrencies):

$$\text{Check } tx = [\text{TXFER}, PK_i \rightarrow PK_j, v] \text{ against state on disk}$$

State is just a dictionary $D(PK_i) \rightarrow bal_i$ (+ counters for replay attacks)

Observations

- Could "authenticate" state and verify transaction against *digest* [Mil12, Tod16, But17, RMCI17]
- Each block t now stores digest d_t of the latest state
- Each user has a proof π_i, which is perpetually updated

Stateless Cryptocurrencies

Stateful transaction validation (for account-based cryptocurrencies):

Check $tx = [\text{TXFER}, PK_i \rightarrow PK_j, v]$ against state on disk

State is just a dictionary $D(PK_i) \rightarrow bal_i$ (+ counters for replay attacks)

Observations

- Could "authenticate" state and verify transaction against *digest* [Mil12, Tod16, But17, RMCI17]
- Each block t now stores digest d_t of the latest state
- Each user has a proof π_i, which is perpetually updated

Stateles transaction validation

Stateless Cryptocurrencies

Stateful transaction validation (for account-based cryptocurrencies):

$$\text{Check } tx = [\textsf{TXFER}, PK_i \rightarrow PK_j, v] \text{ against state on disk}$$

State is just a dictionary $D(PK_i) \rightarrow bal_i$ (+ counters for replay attacks)

Observations

- Could "authenticate" state and verify transaction against *digest* [Mil12, Tod16, But17, RMCI17]
- Each block t now stores digest d_t of the latest state
- Each user has a proof π_i, which is perpetually updated

Stateles transaction validation:

$$\text{Check } tx = [\textsf{TXFER}, PK_i \rightarrow PK_j, v, t, \pi_i, bal_i \geq v] \text{ against digest } d_t \text{ in block } t$$

Stateless Cryptocurrencies

Stateful transaction validation (for account-based cryptocurrencies):

$$\text{Check } tx = [\textbf{TXFER}, PK_i \to PK_j, v] \text{ against state on disk}$$

State is just a dictionary $D(PK_i) \to bal_i$ (+ counters for replay attacks)

Observations

- Could "authenticate" state and verify transaction against *digest* [Mil12, Tod16, But17, RMCI17]
- Each block t now stores digest d_t of the latest state
- Each user has a proof π_i, which is perpetually updated

Stateles transaction validation:

$$\text{Check } tx = [\textbf{TXFER}, PK_i \to PK_j, v, t, \pi_i, bal_i \geq v] \text{ against digest } d_t \text{ in block } t$$

Why go stateless?

Stateless Cryptocurrencies

Stateful transaction validation (for account-based cryptocurrencies):

$$\text{Check } tx = [\text{TXFER}, PK_i \to PK_j, v] \text{ against state on disk}$$

State is just a dictionary $D(PK_i) \to bal_i$ (+ counters for replay attacks)

Observations

- Could "authenticate" state and verify transaction against *digest* [Mil12, Tod16, But17, RMCI17]
- Each block t now stores digest d_t of the latest state
- Each user has a proof π_i, which is perpetually updated

Stateles transaction validation:

$$\text{Check } tx = [\text{TXFER}, PK_i \to PK_j, v, t, \pi_i, bal_i \geq v] \text{ against digest } d_t \text{ in block } t$$

Why go stateless?

(1) Faster validation.

Stateless Cryptocurrencies

Stateful transaction validation (for account-based cryptocurrencies):

$$\text{Check } tx = [\textbf{TXFER}, PK_i \rightarrow PK_j, v] \text{ against state on disk}$$

State is just a dictionary $D(PK_i) \rightarrow bal_i$ (+ counters for replay attacks)

Observations

- Could "authenticate" state and verify transaction against *digest* [Mil12, Tod16, But17, RMCI17]
- Each block t now stores digest d_t of the latest state
- Each user has a proof π_i, which is perpetually updated

Stateles transaction validation:

$$\text{Check } tx = [\textbf{TXFER}, PK_i \rightarrow PK_j, v, t, \pi_i, bal_i \geq v] \text{ against digest } d_t \text{ in block } t$$

Why go stateless?

(1) Faster validation. (2) Less storage $\Rightarrow$ lower barrier to entry.

Stateless Cryptocurrencies

Stateful transaction validation (for account-based cryptocurrencies):

$$\text{Check } tx = [\mathbf{TXFER}, PK_i \rightarrow PK_j, v] \text{ against state on disk}$$

State is just a dictionary $D(PK_i) \rightarrow bal_i$ (+ counters for replay attacks)

Observations

- Could "authenticate" state and verify transaction against *digest* [Mil12, Tod16, But17, RMCI17]
- Each block t now stores digest d_t of the latest state
- Each user has a proof π_i, which is perpetually updated

Stateles transaction validation:

$$\text{Check } tx = [\mathbf{TXFER}, PK_i \rightarrow PK_j, v, t, \pi_i, bal_i \geq v] \text{ against digest } d_t \text{ in block } t$$

Why go stateless?

(1) Faster validation. (2) Less storage ⇒ lower barrier to entry. (3) Easy sharding.

Stateless Cryptocurrencies from Vector Commitments (VCs)

Stateless Cryptocurrencies from Vector Commitments (VCs)

An *authenticated dictionary (AD)* is ideal, but...

Stateless Cryptocurrencies from Vector Commitments (VCs)

An *authenticated dictionary (AD)* is ideal, but... a vector commitment (VC) is sufficient [CPZ18].

Stateless Cryptocurrencies from Vector Commitments (VCs)

An *authenticated dictionary (AD)* is ideal, but... a vector commitment (VC) is sufficient [CPZ18].

- $v_i = (H(PK_i)|bal_i)$
- $PK_i = (i, tpk_i, upk_i)$ and upk_i is a *user-specific* update key, which should be small

Stateless Cryptocurrencies from Vector Commitments (VCs)

An *authenticated dictionary (AD)* is ideal, but... a vector commitment (VC) is sufficient [CPZ18].

- $v_i = (H(PK_i)|bal_i)$
- $PK_i = (i, tpk_i, upk_i)$ and upk_i is a *user-specific* update key, which should be small

Consider miners validating and including $tx = [\mathsf{TXFER}, PK_i \rightarrow PK_j, v, t, \pi_i, bal_i]$

Stateless Cryptocurrencies from Vector Commitments (VCs)

An *authenticated dictionary (AD)* is ideal, but... a vector commitment (VC) is sufficient [CPZ18].

- $v_i = (H(PK_i)|bal_i)$
- $PK_i = (i, tpk_i, upk_i)$ and upk_i is a *user-specific* update key, which should be small

Consider miners validating and including $tx = [\mathrm{TXFER}, PK_i \rightarrow PK_j, v, t, \pi_i, bal_i]$ and users processing it.

Stateless Cryptocurrencies from Vector Commitments (VCs)

An *authenticated dictionary (AD)* is ideal, but... a vector commitment (VC) is sufficient [CPZ18].

- $v_i = (H(PK_i)|bal_i)$
- $PK_i = (i, tpk_i, upk_i)$ and upk_i is a *user-specific* update key, which should be small

Consider miners validating and including $tx = [\text{TXFER}, PK_i \to PK_j, v, t, \pi_i, bal_i]$ and users processing it.

1. Miners need $VC.VerifyPos(vrk, d_t, (H(PK_i)|bal_i), i, \pi_i)$

Stateless Cryptocurrencies from Vector Commitments (VCs)

An *authenticated dictionary (AD)* is ideal, but... a vector commitment (VC) is sufficient [CPZ18].

- $v_i = (H(PK_i)|bal_i)$
- $PK_i = (i, tpk_i, upk_i)$ and upk_i is a *user-specific* update key, which should be small

Consider miners validating and including $tx = [\text{TXFER}, PK_i \to PK_j, v, t, \pi_i, bal_i]$ and users processing it.

1. Miners need $VC.VerifyPos(vrk, d_t, (H(PK_i)|bal_i), i, \pi_i)$
   - vrk is a global verification key, which should be small

Stateless Cryptocurrencies from Vector Commitments (VCs)

An *authenticated dictionary (AD)* is ideal, but... a vector commitment (VC) is sufficient [CPZ18].

- $v_i = (H(PK_i)|bal_i)$
- $PK_i = (i, tpk_i, upk_i)$ and upk_i is a *user-specific* update key, which should be small

Consider miners validating and including $tx = [\text{TXFER}, PK_i \to PK_j, v, t, \pi_i, bal_i]$ and users processing it.

1. Miners need $VC.VerifyPos(vrk, d_t, (H(PK_i)|bal_i), i, \pi_i)$
   - vrk is a global verification key, which should be small
2. Miners need $d_{t+1} = VC.UpdateDig(d_{t+1}, \delta_i, i, upk_i)$

Stateless Cryptocurrencies from Vector Commitments (VCs)

An *authenticated dictionary (AD)* is ideal, but... a vector commitment (VC) is sufficient [CPZ18].

- $v_i = (H(PK_i)|bal_i)$
- $PK_i = (i, tpk_i, upk_i)$ and upk_i is a *user-specific* update key, which should be small

Consider miners validating and including $tx = [\text{TXFER}, PK_i \to PK_j, v, t, \pi_i, bal_i]$ and users processing it.

1. Miners need $VC.VerifyPos(vrk, d_t, (H(PK_i)|bal_i), i, \pi_i)$
   - vrk is a global verification key, which should be small
2. Miners need $d_{t+1} = VC.UpdateDig(d_{t+1}, \delta_i, i, upk_i)$
   - i.e., update digest with change in balance δ_i for each user i

Stateless Cryptocurrencies from Vector Commitments (VCs)

An *authenticated dictionary (AD)* is ideal, but... a vector commitment (VC) is sufficient [CPZ18].

- $v_i = (H(PK_i)|bal_i)$
- $PK_i = (i, tpk_i, upk_i)$ and upk_i is a *user-specific* update key, which should be small

Consider miners validating and including $tx = [\text{TXFER}, PK_i \rightarrow PK_j, v, t, \pi_i, bal_i]$ and users processing it.

1. Miners need $VC.VerifyPos(vrk, d_t, (H(PK_i)|bal_i), i, \pi_i)$
   - vrk is a global verification key, which should be small
2. Miners need $d_{t+1} = VC.UpdateDig(d_{t+1}, \delta_i, i, upk_i)$
   - i.e., update digest with change in balance δ_i for each user i
3. Users need $\pi_i' = VC.UpdateProof(\pi_i, \delta_j, j, upk_j)$

Stateless Cryptocurrencies from Vector Commitments (VCs)

An *authenticated dictionary (AD)* is ideal, but... a vector commitment (VC) is sufficient [CPZ18].

- $v_i = (H(PK_i)|bal_i)$
- $PK_i = (i, tpk_i, upk_i)$ and upk_i is a *user-specific* update key, which should be small

Consider miners validating and including $tx = [\text{TXFER}, PK_i \rightarrow PK_j, v, t, \pi_i, bal_i]$ and users processing it.

1. Miners need $VC.VerifyPos(vrk, d_t, (H(PK_i)|bal_i), i, \pi_i)$
   - vrk is a global verification key, which should be small
2. Miners need $d_{t+1} = VC.UpdateDig(d_{t+1}, \delta_i, i, upk_i)$
   - i.e., update digest with change in balance δ_i for each user i
3. Users need $\pi_i' = VC.UpdateProof(\pi_i, \delta_j, j, upk_j)$
   - i.e., update i's proof with change in balance δ_j for each user j

Stateless Cryptocurrencies from Vector Commitments (VCs)

An *authenticated dictionary (AD)* is ideal, but... a vector commitment (VC) is sufficient [CPZ18].

- $v_i = (H(PK_i)|bal_i)$
- $PK_i = (i, tpk_i, upk_i)$ and upk_i is a *user-specific* update key, which should be small

Consider miners validating and including $tx = [\text{TXFER}, PK_i \rightarrow PK_j, v, t, \pi_i, bal_i]$ and users processing it.

1. Miners need $VC.VerifyPos(vrk, d_t, (H(PK_i)|bal_i), i, \pi_i)$
   - vrk is a global verification key, which should be small
2. Miners need $d_{t+1} = VC.UpdateDig(d_{t+1}, \delta_i, i, upk_i)$
   - i.e., update digest with change in balance δ_i for each user i
3. Users need $\pi_i' = VC.UpdateProof(\pi_i, \delta_j, j, upk_j)$
   - i.e., update i's proof with change in balance δ_j for each user j
4. Miners want $\pi_I = VC.AggregateProofs(I, (\pi_i)_{i \in I})$, where I = set of users sending coins in a block

Stateless Cryptocurrencies from Vector Commitments (VCs)

An *authenticated dictionary (AD)* is ideal, but... a vector commitment (VC) is sufficient [CPZ18].

- $v_i = (H(PK_i)|bal_i)$
- $PK_i = (i, tpk_i, upk_i)$ and upk_i is a *user-specific* update key, which should be small

Consider miners validating and including $tx = [\text{TXFER}, PK_i \rightarrow PK_j, v, t, \pi_i, bal_i]$ and users processing it.

1. Miners need $VC.VerifyPos(vrk, d_t, (H(PK_i)|bal_i), i, \pi_i)$
   - vrk is a global verification key, which should be small
2. Miners need $d_{t+1} = VC.UpdateDig(d_{t+1}, \delta_i, i, upk_i)$
   - i.e., update digest with change in balance δ_i for each user i
3. Users need $\pi_i' = VC.UpdateProof(\pi_i, \delta_j, j, upk_j)$
   - i.e., update i's proof with change in balance δ_j for each user j
4. Miners want $\pi_I = VC.AggregateProofs(I, (\pi_i)_{i \in I})$, where I = set of users sending coins in a block
   - Would need corresponding $VC.VerifyPos(vrk, d_t, \left(H(PK_i)|bal_i\right)_{i \in I}, I, \pi_I)$

Stateless Cryptocurrencies from Vector Commitments (VCs)

An *authenticated dictionary (AD)* is ideal, but... a vector commitment (VC) is sufficient [CPZ18].

- $v_i = (H(PK_i)|bal_i)$
- $PK_i = (i, tpk_i, upk_i)$ and upk_i is a *user-specific* update key, which should be small

Consider miners validating and including $tx = [\text{TXFER}, PK_i \rightarrow PK_j, v, t, \pi_i, bal_i]$ and users processing it.

1. Miners need $VC.VerifyPos(vrk, d_t, (H(PK_i)|bal_i), i, \pi_i)$
    - vrk is a global verification key, which should be small
2. Miners need $d_{t+1} = VC.UpdateDig(d_{t+1}, \delta_i, i, upk_i)$
    - i.e., update digest with change in balance δ_i for each user i
3. Users need $\pi_i' = VC.UpdateProof(\pi_i, \delta_j, j, upk_j)$
    - i.e., update i's proof with change in balance δ_j for each user j
4. Miners want $\pi_I = VC.AggregateProofs(I, (\pi_i)_{i\in I})$, where I = set of users sending coins in a block
    - Would need corresponding $VC.VerifyPos(vrk, d_t, \left(H(PK_i)|bal_i\right)_{i\in I}, I, \pi_I)$
5. Proof serving nodes would like to compute all π_i's fast

Contributions

Our Contribution: Aggregatable Subvector Commitments with Scalable Updates

Table 1: Asymptotic comparison to previous (aS)VCs: n is the size of $\vec{v}$ and b is the # of proofs to aggregate.

Our Contribution: Aggregatable Subvector Commitments with Scalable Updates

Table 1: Asymptotic comparison to previous (aS)VCs: n is the size of $\vec{v}$ and b is the # of proofs to aggregate.

(aS)VC scheme	$\lvert vrk \rvert$	$\lvert upk_i \rvert$	$\lvert \pi_i \rvert$	Proof update time	Aggr. proofs time	Verify aggr. proof	Prove all

Our Contribution: Aggregatable Subvector Commitments with Scalable Updates

Table 1: Asymptotic comparison to previous (aS)VCs: n is the size of $\vec{v}$ and b is the # of proofs to aggregate.

(aS)VC scheme	$\|vrk\|$	$\|upk_i\|$	$\|\pi_i\|$	Proof update time	Aggr. proofs time	Verify aggr. proof	Prove all
CF/LM [CF13, LM19]	n	n	1	1	×	$b_{\mathbb{G}}$	n^2

Our Contribution: Aggregatable Subvector Commitments with Scalable Updates

Table 1: Asymptotic comparison to previous (aS)VCs: n is the size of $\vec{v}$ and b is the # of proofs to aggregate.

(aS)VC scheme	$\lvert vrk \rvert$	$\lvert upk_i \rvert$	$\lvert \pi_i \rvert$	Proof update time	Aggr. proofs time	Verify aggr. proof	Prove all
CF/LM [CF13, LM19]	n	n	1	1	×	$b_{\mathbb{G}}$	n^2
KZG [KZG10]	b	×	1	×	×	$b \lg^2 b_{\mathbb{F}} + b_{\mathbb{G}}$	n^2

Our Contribution: Aggregatable Subvector Commitments with Scalable Updates

Table 1: Asymptotic comparison to previous (aS)VCs: n is the size of $\vec{v}$ and b is the # of proofs to aggregate.

(aS)VC scheme	$\lvert vrk \rvert$	$\lvert upk_i \rvert$	$\lvert \pi_i \rvert$	Proof update time	Aggr. proofs time	Verify aggr. proof	Prove all
CF/LM [CF13, LM19]	n	n	1	1	×	$b_{\mathbb{G}}$	n^2
KZG [KZG10]	b	×	1	×	×	$b \lg^2 b_{\mathbb{F}} + b_{\mathbb{G}}$	n^2
CDHK [CDHK15]	n	n	1	1	×	$b \lg^2 b_{\mathbb{F}} + b_{\mathbb{G}}$	n^2

Our Contribution: Aggregatable Subvector Commitments with Scalable Updates

Table 1: Asymptotic comparison to previous (aS)VCs: n is the size of $\vec{v}$ and b is the # of proofs to aggregate.

(aS)VC scheme	$\lvert vrk \rvert$	$\lvert upk_i \rvert$	$\lvert \pi_i \rvert$	Proof update time	Aggr. proofs time	Verify aggr. proof	Prove all
CF/LM [CF13, LM19]	n	n	1	1	×	$b_{\mathbb{G}}$	n^2
KZG [KZG10]	b	×	1	×	×	$b \lg^2 b_{\mathbb{F}} + b_{\mathbb{G}}$	n^2
CDHK [CDHK15]	n	n	1	1	×	$b \lg^2 b_{\mathbb{F}} + b_{\mathbb{G}}$	n^2
CPZ [CPZ18]	$\log n$	$\log n$	$\log n$	$\log n$	×	×	$n \log n$

Our Contribution: Aggregatable Subvector Commitments with Scalable Updates

Table 1: Asymptotic comparison to previous (aS)VCs: n is the size of $\vec{v}$ and b is the # of proofs to aggregate.

(aS)VC scheme	$\|vrk\|$	$\|upk_i\|$	$\|\pi_i\|$	Proof update time	Aggr. proofs time	Verify aggr. proof	Prove all
CF/LM [CF13, LM19]	n	n	1	1	×	$b_{\mathbb{G}}$	n^2
KZG [KZG10]	b	×	1	×	×	$b \lg^2 b_{\mathbb{F}} + b_{\mathbb{G}}$	n^2
CDHK [CDHK15]	n	n	1	1	×	$b \lg^2 b_{\mathbb{F}} + b_{\mathbb{G}}$	n^2
CPZ [CPZ18]	$\log n$	$\log n$	$\log n$	$\log n$	×	×	$n \log n$
TCZ [TCZ+20, Tom20]	$\log n + b$	$\log n$	$\log n$	$\log n$	×	$b \lg^2 b_{\mathbb{F}} + b_{\mathbb{G}}$	$n \log n$

Our Contribution: Aggregatable Subvector Commitments with Scalable Updates

Table 1: Asymptotic comparison to previous (aS)VCs: n is the size of $\vec{v}$ and b is the # of proofs to aggregate.

(aS)VC scheme	$\lvert vrk \rvert$	$\lvert upk_i \rvert$	$\lvert \pi_i \rvert$	Proof update time	Aggr. proofs time	Verify aggr. proof	Prove all
CF/LM [CF13, LM19]	n	n	1	1	×	$b_{\mathbb{G}}$	n^2
KZG [KZG10]	b	×	1	×	×	$b \lg^2 b_{\mathbb{F}} + b_{\mathbb{G}}$	n^2
CDHK [CDHK15]	n	n	1	1	×	$b \lg^2 b_{\mathbb{F}} + b_{\mathbb{G}}$	n^2
CPZ [CPZ18]	$\log n$	$\log n$	$\log n$	$\log n$	×	×	$n \log n$
TCZ [TCZ+20, Tom20]	$\log n + b$	$\log n$	$\log n$	$\log n$	×	$b \lg^2 b_{\mathbb{F}} + b_{\mathbb{G}}$	$n \log n$
Pointproofs [GRWZ20]	n	n	1	1	$b_{\mathbb{G}}$	$b_{\mathbb{G}}$	n^2

Our Contribution: Aggregatable Subvector Commitments with Scalable Updates

Table 1: Asymptotic comparison to previous (aS)VCs: n is the size of $\vec{v}$ and b is the # of proofs to aggregate.

(aS)VC scheme	$\lvert vrk \rvert$	$\lvert upk_i \rvert$	$\lvert \pi_i \rvert$	Proof update time	Aggr. proofs time	Verify aggr. proof	Prove all
CF/LM [CF13, LM19]	n	n	1	1	×	$b_{\mathbb{G}}$	n^2
KZG [KZG10]	b	×	1	×	×	$b \lg^2 b_{\mathbb{F}} + b_{\mathbb{G}}$	n^2
CDHK [CDHK15]	n	n	1	1	×	$b \lg^2 b_{\mathbb{F}} + b_{\mathbb{G}}$	n^2
CPZ [CPZ18]	$\log n$	$\log n$	$\log n$	$\log n$	×	×	$n \log n$
TCZ [TCZ+20, Tom20]	$\log n + b$	$\log n$	$\log n$	$\log n$	×	$b \lg^2 b_{\mathbb{F}} + b_{\mathbb{G}}$	$n \log n$
Pointproofs [GRWZ20]	n	n	1	1	$b_{\mathbb{G}}$	$b_{\mathbb{G}}$	n^2
Our aSVC	b	1	1	1	$b \lg^2 b_{\mathbb{F}} + b_{\mathbb{G}}$		$n \log n$

Our Contribution: Aggregatable Subvector Commitments with Scalable Updates

Table 1: Asymptotic comparison to previous (aS)VCs: n is the size of $\vec{v}$ and b is the # of proofs to aggregate.

(aS)VC scheme	$\lvert vrk \rvert$	$\lvert upk_i \rvert$	$\lvert \pi_i \rvert$	Proof update time	Aggr. proofs time	Verify aggr. proof	Prove all
CF/LM [CF13, LM19]	n	n	1	1	×	$b_{\mathbb{G}}$	n^2
KZG [KZG10]	b	×	1	×	×	$b \lg^2 b_{\mathbb{F}} + b_{\mathbb{G}}$	n^2
CDHK [CDHK15]	n	n	1	1	×	$b \lg^2 b_{\mathbb{F}} + b_{\mathbb{G}}$	n^2
CPZ [CPZ18]	$\log n$	$\log n$	$\log n$	$\log n$	×	×	$n \log n$
TCZ [TCZ+20, Tom20]	$\log n + b$	$\log n$	$\log n$	$\log n$	×	$b \lg^2 b_{\mathbb{F}} + b_{\mathbb{G}}$	$n \log n$
Pointproofs [GRWZ20]	n	n	1	1	$b_{\mathbb{G}}$	$b_{\mathbb{G}}$	n^2
Our aSVC	b	1	1	1	$b \lg^2 b_{\mathbb{F}} + b_{\mathbb{G}}$		$n \log n$

All schemes can (1) verify a proof π_i in $O(\lvert\pi_i\rvert)$ time and (2) update digests in $O(1)$ time.

Techniques: VCs from Univariate Polynomial Commitments

Outline

KZG Constant-sized Polynomial Commitments [KZG10]

KZG Constant-sized Polynomial Commitments [KZG10]

Fix n-SDH public parameters $\left(g^{\tau^i}\right)_{0 \leq i \leq n}$ such that trapdoor τ is unknown.

KZG Constant-sized Polynomial Commitments [KZG10]

Fix n-SDH public parameters $\left(g^{\tau^i}\right)_{0 \le i \le n}$ such that trapdoor τ is unknown.

For any polynomial $\phi = \sum_{i=0}^{n} \phi_i X^i = \langle \phi_0, \phi_1, \dots, \phi_n \rangle$ of degree $\le n$:

KZG Constant-sized Polynomial Commitments [KZG10]

Fix n-SDH public parameters $\left(g^{\tau^i}\right)_{0\le i\le n}$ such that trapdoor τ is unknown.

For any polynomial $\phi = \sum_{i=0}^{n} \phi_i X^i = \langle \phi_0, \phi_1, \ldots, \phi_n \rangle$ of degree $\le n$:

$$c = g^{\phi(\tau)} \tag{1}$$

(2)

(3)

(4)

(5)

KZG Constant-sized Polynomial Commitments [KZG10]

Fix n-SDH public parameters $\left(g^{\tau^i}\right)_{0 \le i \le n}$ such that trapdoor τ is unknown.

For any polynomial $\phi = \sum_{i=0}^{n} \phi_i X^i = \langle \phi_0, \phi_1, \ldots, \phi_n \rangle$ of degree $\le n$:

$$
\begin{aligned}
c &= g^{\phi(\tau)} && (1)\\
&= \left(g^{\tau^n}\right)^{\phi_n} \left(g^{\tau^{n-1}}\right)^{\phi_{n-1}} \ldots (g^{\tau})^{\phi_1} (g)^{\phi_0} && (2)\\
& && (3)\\
& && (4)\\
& && (5)
\end{aligned}
$$

KZG Constant-sized Polynomial Commitments [KZG10]

Fix n-SDH public parameters $\left(g^{\tau^i}\right)_{0\le i\le n}$ such that trapdoor τ is unknown.

For any polynomial $\phi = \sum_{i=0}^{n} \phi_i X^i = \langle \phi_0, \phi_1, \ldots, \phi_n \rangle$ of degree $\le n$:

$$
\begin{align}
c &= g^{\phi(\tau)} \tag{1}\\
&= \left(g^{\tau^n}\right)^{\phi_n} \left(g^{\tau^{n-1}}\right)^{\phi_{n-1}} \ldots (g^{\tau})^{\phi_1} (g)^{\phi_0} \tag{2}\\
&= g^{\phi_n \tau^n} g^{\phi_{n-1}\tau^{n-1}} \ldots g^{\phi_1 \tau} g^{\phi_0} \tag{3}\\
& \tag{4}\\
& \tag{5}
\end{align}
$$

KZG Constant-sized Polynomial Commitments [KZG10]

Fix n-SDH public parameters $\left(g^{\tau^i}\right)_{0 \leq i \leq n}$ such that trapdoor τ is unknown.

For any polynomial $\phi = \sum_{i=0}^{n} \phi_i X^i = \langle \phi_0, \phi_1, \dots, \phi_n \rangle$ of degree $\leq n$:

$$\begin{align} c &= g^{\phi(\tau)} \tag{1} \\ &= \left(g^{\tau^n}\right)^{\phi_n} \left(g^{\tau^{n-1}}\right)^{\phi_{n-1}} \dots (g^{\tau})^{\phi_1} (g)^{\phi_0} \tag{2} \\ &= g^{\phi_n \tau^n} g^{\phi_{n-1} \tau^{n-1}} \dots g^{\phi_1 \tau} g^{\phi_0} \tag{3} \\ &= g^{\phi_n \tau^n + \phi_{n-1} \tau^{n-1} + \dots + \phi_1 \tau + \phi_0} \tag{4} \\ & \tag{5} \end{align}$$

KZG Constant-sized Polynomial Commitments [KZG10]

Fix n-SDH public parameters $\left(g^{\tau^i}\right)_{0\le i\le n}$ such that trapdoor τ is unknown.

For any polynomial $\phi = \sum_{i=0}^{n} \phi_i X^i = \langle \phi_0, \phi_1, \dots, \phi_n \rangle$ of degree $\le n$:

$$
\begin{aligned}
c &= g^{\phi(\tau)} && (1)\\
&= \left(g^{\tau^n}\right)^{\phi_n} \left(g^{\tau^{n-1}}\right)^{\phi_{n-1}} \dots (g^{\tau})^{\phi_1} (g)^{\phi_0} && (2)\\
&= g^{\phi_n \tau^n} g^{\phi_{n-1}\tau^{n-1}} \dots g^{\phi_1 \tau} g^{\phi_0} && (3)\\
&= g^{\phi_n \tau^n + \phi_{n-1}\tau^{n-1} + \dots + \phi_1 \tau + \phi_0} && (4)\\
&= g^{\phi(\tau)} && (5)
\end{aligned}
$$

KZG Constant-sized Polynomial Commitments [KZG10]

Fix n-SDH public parameters $\left(g^{\tau^i}\right)_{0\le i\le n}$ such that trapdoor τ is unknown.

For any polynomial $\phi = \sum_{i=0}^{n} \phi_i X^i = \langle \phi_0, \phi_1, \dots, \phi_n \rangle$ of degree $\le n$:

$$\begin{aligned}
c &= g^{\phi(\tau)} && (1)\\
&= \left(g^{\tau^n}\right)^{\phi_n} \left(g^{\tau^{n-1}}\right)^{\phi_{n-1}} \dots (g^{\tau})^{\phi_1} (g)^{\phi_0} && (2)\\
&= g^{\phi_n \tau^n} g^{\phi_{n-1}\tau^{n-1}} \dots g^{\phi_1 \tau} g^{\phi_0} && (3)\\
&= g^{\phi_n \tau^n + \phi_{n-1}\tau^{n-1} + \dots + \phi_1 \tau + \phi_0} && (4)\\
&= g^{\phi(\tau)} && (5)
\end{aligned}$$

Can interpolate polynomial from n points $(x_i, \phi(x_i))_{i\in[n]}$ in $O(n \log^2 n)$ field operations.

KZG Constant-sized Polynomial Commitments [KZG10]

Fix n-SDH public parameters $\left(g^{\tau^i}\right)_{0\le i\le n}$ such that trapdoor τ is unknown.

For any polynomial $\phi = \sum_{i=0}^{n} \phi_i X^i = \langle \phi_0, \phi_1, \dots, \phi_n \rangle$ of degree $\le n$:

$$
\begin{aligned}
c &= g^{\phi(\tau)} && (1)\\
&= \left(g^{\tau^n}\right)^{\phi_n} \left(g^{\tau^{n-1}}\right)^{\phi_{n-1}} \dots (g^{\tau})^{\phi_1} (g)^{\phi_0} && (2)\\
&= g^{\phi_n \tau^n} g^{\phi_{n-1}\tau^{n-1}} \dots g^{\phi_1 \tau} g^{\phi_0} && (3)\\
&= g^{\phi_n \tau^n + \phi_{n-1}\tau^{n-1} + \dots + \phi_1 \tau + \phi_0} && (4)\\
&= g^{\phi(\tau)} && (5)
\end{aligned}
$$

Can interpolate polynomial from n points $(x_i, \phi(x_i))_{i\in[n]}$ in $O(n \log^2 n)$ field operations.
Time to commit is an $O(n)$-sized multi-exponentiation.

VCs from Lagrange Basis Polynomials [CDHK15]

VCs from Lagrange Basis Polynomials [CDHK15]

Compute $\phi(X)$ s.t. $\phi(i) = v_i$:

VCs from Lagrange Basis Polynomials [CDHK15]

Compute $\phi(X)$ s.t. $\phi(i) = v_i$:

$$\phi(X) = \sum_{i=0}^{n-1} v_i \cdot L_i(X), \text{ where } L_i(X) = \prod_{\substack{j \in [0,n) \\ j \neq i}} \frac{X - j}{i - j} \tag{6}$$

VCs from Lagrange Basis Polynomials [CDHK15]

Compute $\phi(X)$ s.t. $\phi(i) = v_i$:

$$\phi(X) = \sum_{i=0}^{n-1} v_i \cdot L_i(X), \text{ where } L_i(X) = \prod_{\substack{j \in [0,n) \\ j \neq i}} \frac{X - j}{i - j} \tag{6}$$

Setup Lagrange basis polynomial commitments $\ell_i = g^{L_i(\tau)}, \forall i \in [0, n)$ (speed?)

VCs from Lagrange Basis Polynomials [CDHK15]

Compute $\phi(X)$ s.t. $\phi(i) = v_i$:

$$\phi(X) = \sum_{i=0}^{n-1} v_i \cdot L_i(X), \text{ where } L_i(X) = \prod_{\substack{j \in [0,n) \\ j \neq i}} \frac{X - j}{i - j} \tag{6}$$

Setup Lagrange basis polynomial commitments $\ell_i = g^{L_i(\tau)}, \forall i \in [0, n)$ (speed?). Then, commit to $\vec{v}$ as [CDHK15]:

VCs from Lagrange Basis Polynomials [CDHK15]

Compute $\phi(X)$ s.t. $\phi(i) = v_i$:

$$\phi(X) = \sum_{i=0}^{n-1} v_i \cdot L_i(X), \text{ where } L_i(X) = \prod_{\substack{j\in[0,n) \\ j\neq i}} \frac{X-j}{i-j} \tag{6}$$

Setup Lagrange basis polynomial commitments $\ell_i = g^{L_i(\tau)}, \forall i \in [0,n)$ (speed?). Then, commit to $\vec{v}$ as [CDHK15]:

$$c = \prod_{i\in[0,n)} \ell_i^{v_i} \tag{7}$$

$$\tag{8}$$

$$\tag{9}$$

VCs from Lagrange Basis Polynomials [CDHK15]

Compute $\phi(X)$ s.t. $\phi(i) = v_i$:

$$\phi(X) = \sum_{i=0}^{n-1} v_i \cdot L_i(X), \text{ where } L_i(X) = \prod_{\substack{j \in [0,n) \\ j \neq i}} \frac{X - j}{i - j} \tag{6}$$

Setup Lagrange basis polynomial commitments $\ell_i = g^{L_i(\tau)}, \forall i \in [0, n)$ (speed?). Then, commit to $\vec{v}$ as [CDHK15]:

$$c = \prod_{i \in [0,n)} \ell_i^{v_i} \tag{7}$$

$$= g^{\sum_{i \in [0,n)} L_i(\tau) v_i} \tag{8}$$

$$\tag{9}$$

VCs from Lagrange Basis Polynomials [CDHK15]

Compute $\phi(X)$ s.t. $\phi(i) = v_i$:

$$\phi(X) = \sum_{i=0}^{n-1} v_i \cdot L_i(X), \text{ where } L_i(X) = \prod_{\substack{j \in [0,n) \\ j \neq i}} \frac{X - j}{i - j} \tag{6}$$

Setup Lagrange basis polynomial commitments $\ell_i = g^{L_i(\tau)}, \forall i \in [0, n)$ (speed?). Then, commit to $\vec{v}$ as [CDHK15]:

$$\begin{aligned} c &= \prod_{i \in [0,n)} \ell_i^{v_i} \qquad (7) \\ &= g^{\sum_{i \in [0,n)} L_i(\tau) v_i} \qquad (8) \\ &= g^{\phi(\tau)} \qquad (9) \end{aligned}$$

Updating the Digest

Updating the Digest

Let $\phi'(X)$ be the updated polynomial after v_i changes to $v_i + \delta_i$:

Updating the Digest

Let $\phi'(X)$ be the updated polynomial after v_i changes to $v_i + \delta_i$:

$$\phi'(X) = \phi(X) + \delta_i L_i(X) \tag{10}$$

Updating the Digest

Let $\phi'(X)$ be the updated polynomial after v_i changes to $v_i + \delta_i$:

$$\phi'(X) = \phi(X) + \delta_i L_i(X) \tag{10}$$

To update c via $VC.UpdateDig(c, \delta_i, i, upk_i)$:

Updating the Digest

Let $\phi'(X)$ be the updated polynomial after v_i changes to $v_i + \delta_i$:

$$\phi'(X) = \phi(X) + \delta_i L_i(X) \tag{10}$$

To update c via $VC.UpdateDig(c, \delta_i, i, upk_i)$:

$$c' = c \cdot \ell_i^{\delta_i} \tag{11}$$

$$\tag{12}$$

$$\tag{13}$$

Updating the Digest

Let $\phi'(X)$ be the updated polynomial after v_i changes to $v_i + \delta_i$:

$$\phi'(X) = \phi(X) + \delta_i L_i(X) \tag{10}$$

To update c via $VC.UpdateDig(c, \delta_i, i, upk_i)$:

$$\begin{aligned} c' &= c \cdot \ell_i^{\delta_i} && (11) \\ &= g^{\phi(\tau)} \cdot g^{\delta_i L_i(\tau)} && (12) \\ & && (13) \end{aligned}$$

Updating the Digest

Let $\phi'(X)$ be the updated polynomial after v_i changes to $v_i + \delta_i$:

$$\phi'(X) = \phi(X) + \delta_i L_i(X) \tag{10}$$

To update c via $VC.UpdateDig(c, \delta_i, i, upk_i)$:

$$\begin{aligned} c' &= c \cdot \ell_i^{\delta_i} && (11) \\ &= g^{\phi(\tau)} \cdot g^{\delta_i L_i(\tau)} && (12) \\ &= g^{\phi'(\tau)} && (13) \end{aligned}$$

Updating the Digest

Let $\phi'(X)$ be the updated polynomial after v_i changes to $v_i + \delta_i$:

$$\phi'(X) = \phi(X) + \delta_i L_i(X) \tag{10}$$

To update c via $VC.UpdateDig(c, \delta_i, i, upk_i)$:

$$\begin{aligned} c' &= c \cdot \ell_i^{\delta_i} && (11)\\ &= g^{\phi(\tau)} \cdot g^{\delta_i L_i(\tau)} && (12)\\ &= g^{\phi'(\tau)} && (13) \end{aligned}$$

Jon done: Each upk_i must include ℓ_i.

Outline

Proof π_i for v_i Refresher

Proof π_i for v_i Refresher

A proof for v_i is just a KZG evaluation proof:

Proof π_i for v_i Refresher

A proof for v_i is just a KZG evaluation proof:
Step 1: Interpolate $\phi(X)$ in $O(n \log^2 n)$ time.

Proof π_i for v_i Refresher

A proof for v_i is just a KZG evaluation proof:

Step 1: Interpolate $\phi(X)$ in $O(n \log^2 n)$ time.

Step 2: Compute $q_i(X) = \frac{\phi(X)-v_i}{X-i}$ in $O(n)$ time.

Proof π_i for v_i Refresher

A proof for v_i is just a KZG evaluation proof:

Step 1: Interpolate $\phi(X)$ in $O(n \log^2 n)$ time.

Step 2: Compute $q_i(X) = \frac{\phi(X)-v_i}{X-i}$ in $O(n)$ time.

Step 3: Compute $\pi_i = g^{q_i(\tau)}$ using an $O(n)$-sized multiexp.

Proof π_i for v_i Refresher

A proof for v_i is just a KZG evaluation proof:

Step 1: Interpolate $\phi(X)$ in $O(n \log^2 n)$ time.

Step 2: Compute $q_i(X) = \frac{\phi(X) - v_i}{X - i}$ in $O(n)$ time.

Step 3: Compute $\pi_i = g^{q_i(\tau)}$ using an $O(n)$-sized multiexp.

To verify, use pairings and g^τ from *vrk* to check:

Proof π_i for v_i Refresher

A proof for v_i is just a KZG evaluation proof:

Step 1: Interpolate $\phi(X)$ in $O(n \log^2 n)$ time.

Step 2: Compute $q_i(X) = \frac{\phi(X) - v_i}{X - i}$ in $O(n)$ time.

Step 3: Compute $\pi_i = g^{q_i(\tau)}$ using an $O(n)$-sized multiexp.

To verify, use pairings and g^τ from *vrk* to check:

$$e(c/g^{v_i}, g) = e(\pi_i, g^\tau/g^i) \Leftrightarrow \tag{14}$$

(15)

(16)

(17)

Proof π_i for v_i Refresher

A proof for v_i is just a KZG evaluation proof:

Step 1: Interpolate $\phi(X)$ in $O(n \log^2 n)$ time.

Step 2: Compute $q_i(X) = \frac{\phi(X)-v_i}{X-i}$ in $O(n)$ time.

Step 3: Compute $\pi_i = g^{q_i(\tau)}$ using an $O(n)$-sized multiexp.

To verify, use pairings and g^τ from *vrk* to check:

$$e(c/g^{v_i}, g) = e(\pi_i, g^\tau/g^i) \Leftrightarrow \tag{14}$$

$$e(g^{\phi(\tau)}/g^{v_i}, g) = e(g^{q_i(\tau)}, g^{\tau-i}) \Leftrightarrow \tag{15}$$

(16)

(17)

Proof π_i for v_i Refresher

A proof for v_i is just a KZG evaluation proof:

Step 1: Interpolate $\phi(X)$ in $O(n \log^2 n)$ time.

Step 2: Compute $q_i(X) = \frac{\phi(X)-v_i}{X-i}$ in $O(n)$ time.

Step 3: Compute $\pi_i = g^{q_i(\tau)}$ using an $O(n)$-sized multiexp.

To verify, use pairings and g^τ from *vrk* to check:

$$e(c/g^{v_i}, g) = e(\pi_i, g^\tau/g^i) \Leftrightarrow \tag{14}$$

$$e(g^{\phi(\tau)}/g^{v_i}, g) = e(g^{q_i(\tau)}, g^{\tau-i}) \Leftrightarrow \tag{15}$$

$$e(g^{\phi(\tau)-v_i}, g) = e(g, g)^{q_i(\tau)(\tau-i)} \Leftrightarrow \tag{16}$$

$$\tag{17}$$

Proof π_i for v_i Refresher

A proof for v_i is just a KZG evaluation proof:

Step 1: Interpolate $\phi(X)$ in $O(n \log^2 n)$ time.

Step 2: Compute $q_i(X) = \frac{\phi(X) - v_i}{X - i}$ in $O(n)$ time.

Step 3: Compute $\pi_i = g^{q_i(\tau)}$ using an $O(n)$-sized multiexp.

To verify, use pairings and g^τ from *vrk* to check:

$$e(c / g^{v_i}, g) = e(\pi_i, g^\tau / g^i) \Leftrightarrow \tag{14}$$

$$e(g^{\phi(\tau)} / g^{v_i}, g) = e(g^{q_i(\tau)}, g^{\tau - i}) \Leftrightarrow \tag{15}$$

$$e(g^{\phi(\tau) - v_i}, g) = e(g, g)^{q_i(\tau)(\tau - i)} \Leftrightarrow \tag{16}$$

$$\phi(\tau) - v_i = q_i(\tau)(\tau - i) \tag{17}$$

Proof π_i for v_i Refresher

A proof for v_i is just a KZG evaluation proof:

Step 1: Interpolate $\phi(X)$ in $O(n \log^2 n)$ time.

Step 2: Compute $q_i(X) = \frac{\phi(X)-v_i}{X-i}$ in $O(n)$ time.

Step 3: Compute $\pi_i = g^{q_i(\tau)}$ using an $O(n)$-sized multiexp.

To verify, use pairings and g^τ from *vrk* to check:

$$e(c/g^{v_i}, g) = e(\pi_i, g^\tau/g^i) \Leftrightarrow \tag{14}$$

$$e(g^{\phi(\tau)}/g^{v_i}, g) = e(g^{q_i(\tau)}, g^{\tau-i}) \Leftrightarrow \tag{15}$$

$$e(g^{\phi(\tau)-v_i}, g) = e(g, g)^{q_i(\tau)(\tau-i)} \Leftrightarrow \tag{16}$$

$$\phi(\tau) - v_i = q_i(\tau)(\tau - i) \tag{17}$$

New technique: $O(n)$ time w/o interpolating $\phi(X)$ via upk_i's (see [TAB+20, Appendix D.7]).

New Technique: Updating Proofs π_i After a Change to v_i

New Technique: Updating Proofs π_i After a Change to v_i

Consider new $q_i'(X)$ when v_i changed to $v_i + \delta_i$:

New Technique: Updating Proofs π_i After a Change to v_i

Consider new $q_i'(X)$ when v_i changed to $v_i + \delta_i$:

$$q_i'(X) = \frac{\phi'(X) - (v_i + \delta_i)}{X - i} \tag{18}$$

(19)

(20)

(21)

New Technique: Updating Proofs π_i After a Change to v_i

Consider new $q_i'(X)$ when v_i changed to $v_i + \delta_i$:

$$q_i'(X) = \frac{\phi'(X) - (v_i + \delta_i)}{X - i} \tag{18}$$

$$= \frac{\big(\phi(X) + \delta_i L_i(X)\big) - v_i - \delta_i}{X - i} \tag{19}$$

(20)

(21)

New Technique: Updating Proofs π_i After a Change to v_i

Consider new $q_i'(X)$ when v_i changed to $v_i + \delta_i$:

$$q_i'(X) = \frac{\phi'(X) - (v_i + \delta_i)}{X - i} \tag{18}$$

$$= \frac{\big(\phi(X) + \delta_i L_i(X)\big) - v_i - \delta_i}{X - i} \tag{19}$$

$$= \frac{\phi(X) - v_i}{X - i} - \frac{\delta_i(L_i(X) - 1)}{X - i} \tag{20}$$

$$\tag{21}$$

New Technique: Updating Proofs π_i After a Change to v_i

Consider new $q_i'(X)$ when v_i changed to $v_i + \delta_i$:

$$
\begin{aligned}
q_i'(X) &= \frac{\phi'(X) - (v_i + \delta_i)}{X - i} && (18)\\
&= \frac{\big(\phi(X) + \delta_i L_i(X)\big) - v_i - \delta_i}{X - i} && (19)\\
&= \frac{\phi(X) - v_i}{X - i} - \frac{\delta_i (L_i(X) - 1)}{X - i} && (20)\\
&= q_i(X) + \delta_i \left(\frac{L_i(X) - 1}{X - i} \right) && (21)
\end{aligned}
$$

New Technique: Updating Proofs π_i After a Change to v_i

Consider new $q_i'(X)$ when v_i changed to $v_i + \delta_i$:

$$q_i'(X) = \frac{\phi'(X) - (v_i + \delta_i)}{X - i} \tag{18}$$

$$= \frac{\left(\phi(X) + \delta_i L_i(X)\right) - v_i - \delta_i}{X - i} \tag{19}$$

$$= \frac{\phi(X) - v_i}{X - i} - \frac{\delta_i (L_i(X) - 1)}{X - i} \tag{20}$$

$$= q_i(X) + \delta_i \left(\frac{L_i(X) - 1}{X - i}\right) \tag{21}$$

Let u_i be a KZG commitment to $\frac{L_i(X)-1}{X-i}$.

New Technique: Updating Proofs π_i After a Change to v_i

Consider new $q_i'(X)$ when v_i changed to $v_i + \delta_i$:

$$q_i'(X) = \frac{\phi'(X) - (v_i + \delta_i)}{X - i} \tag{18}$$

$$= \frac{\left(\phi(X) + \delta_i L_i(X)\right) - v_i - \delta_i}{X - i} \tag{19}$$

$$= \frac{\phi(X) - v_i}{X - i} - \frac{\delta_i(L_i(X) - 1)}{X - i} \tag{20}$$

$$= q_i(X) + \delta_i \left(\frac{L_i(X) - 1}{X - i}\right) \tag{21}$$

Let u_i be a KZG commitment to $\frac{L_i(X)-1}{X-i}$. Then, $\pi_i' = g^{q_i'(\tau)}$ can be computed as:

New Technique: Updating Proofs π_i After a Change to v_i

Consider new $q_i'(X)$ when v_i changed to $v_i + \delta_i$:

$$
\begin{aligned}
q_i'(X) &= \frac{\phi'(X) - (v_i + \delta_i)}{X - i} && (18)\\
&= \frac{\big(\phi(X) + \delta_i L_i(X)\big) - v_i - \delta_i}{X - i} && (19)\\
&= \frac{\phi(X) - v_i}{X - i} - \frac{\delta_i(L_i(X) - 1)}{X - i} && (20)\\
&= q_i(X) + \delta_i\left(\frac{L_i(X) - 1}{X - i}\right) && (21)
\end{aligned}
$$

Let u_i be a KZG commitment to $\frac{L_i(X)-1}{X-i}$. Then, $\pi_i' = g^{q_i'(\tau)}$ can be computed as:

$$\pi_i' = \pi_i \cdot (u_i)^{\delta_i} \tag{22}$$

New Technique: Updating Proofs π_i After a Change to v_i

Consider new $q_i'(X)$ when v_i changed to $v_i + \delta_i$:

$$q_i'(X) = \frac{\phi'(X) - (v_i + \delta_i)}{X - i} \tag{18}$$

$$= \frac{\big(\phi(X) + \delta_i L_i(X)\big) - v_i - \delta_i}{X - i} \tag{19}$$

$$= \frac{\phi(X) - v_i}{X - i} - \frac{\delta_i(L_i(X) - 1)}{X - i} \tag{20}$$

$$= q_i(X) + \delta_i \left(\frac{L_i(X) - 1}{X - i}\right) \tag{21}$$

Let u_i be a KZG commitment to $\frac{L_i(X)-1}{X-i}$. Then, $\pi_i' = g^{q_i'(\tau)}$ can be computed as:

$$\pi_i' = \pi_i \cdot (u_i)^{\delta_i} \tag{22}$$

Job done: Each upk_i must include u_i.

New Technique: Updating Proofs π_i After a Change to v_i

Consider new $q_i'(X)$ when v_i changed to $v_i + \delta_i$:

$$q_i'(X) = \frac{\phi'(X) - (v_i + \delta_i)}{X - i} \tag{18}$$

$$= \frac{\big(\phi(X) + \delta_i L_i(X)\big) - v_i - \delta_i}{X - i} \tag{19}$$

$$= \frac{\phi(X) - v_i}{X - i} - \frac{\delta_i(L_i(X) - 1)}{X - i} \tag{20}$$

$$= q_i(X) + \delta_i \left(\frac{L_i(X) - 1}{X - i}\right) \tag{21}$$

Let u_i be a KZG commitment to $\frac{L_i(X)-1}{X-i}$. Then, $\pi_i' = g^{q_i'(\tau)}$ can be computed as:

$$\pi_i' = \pi_i \cdot \left(u_i\right)^{\delta_i} \tag{22}$$

Job done: Each upk_i must include u_i. Wait, what about computing all u_i's?

New Technique: Updating Proofs π_i After a Change to v_i

Consider new $q_i'(X)$ when v_i changed to $v_i + \delta_i$:

$$\begin{aligned} q_i'(X) &= \frac{\phi'(X) - (v_i + \delta_i)}{X - i} && (18) \\ &= \frac{\big(\phi(X) + \delta_i L_i(X)\big) - v_i - \delta_i}{X - i} && (19) \\ &= \frac{\phi(X) - v_i}{X - i} - \frac{\delta_i(L_i(X) - 1)}{X - i} && (20) \\ &= q_i(X) + \delta_i \left(\frac{L_i(X) - 1}{X - i}\right) && (21) \end{aligned}$$

Let u_i be a KZG commitment to $\frac{L_i(X)-1}{X-i}$. Then, $\pi_i' = g^{q_i'(\tau)}$ can be computed as:

$$\pi_i' = \pi_i \cdot (u_i)^{\delta_i} \tag{22}$$

Job done: Each upk_i must include u_i. Wait, what about computing all u_i's? Later.

New Technique: Updating Proofs π_i After a Change to v_j, $j \neq i$

New Technique: Updating Proofs π_i After a Change to v_j, $j \neq i$

Consider new $q_i'(X)$ when v_j changed to $v_j + \delta_j$:

New Technique: Updating Proofs π_i After a Change to v_j, $j \neq i$

Consider new $q_i'(X)$ when v_j changed to $v_j + \delta_j$:

$$q_i'(X) = \frac{\phi'(X) - v_i}{X - i} \tag{23}$$

(24)

(25)

(26)

New Technique: Updating Proofs π_i After a Change to v_j, $j \neq i$

Consider new $q_i'(X)$ when v_j changed to $v_j + \delta_j$:

$$q_i'(X) = \frac{\phi'(X) - v_i}{X - i} \tag{23}$$

$$= \frac{\left(\phi(X) + \delta_j L_j(X)\right) - v_i}{X - i} \tag{24}$$

(25)

(26)

New Technique: Updating Proofs π_i After a Change to v_j, $j \neq i$

Consider new $q_i'(X)$ when v_j changed to $v_j + \delta_j$:

$$
\begin{aligned}
q_i'(X) &= \frac{\phi'(X) - v_i}{X - i} && (23)\\
&= \frac{\left(\phi(X) + \delta_j L_j(X)\right) - v_i}{X - i} && (24)\\
&= \frac{\phi(X) - v_i}{X - i} - \frac{\delta_j L_j(X)}{X - i} && (25)\\
& && (26)
\end{aligned}
$$

New Technique: Updating Proofs π_i After a Change to v_j, $j \neq i$

Consider new $q_i'(X)$ when v_j changed to $v_j + \delta_j$:

$$q_i'(X) = \frac{\phi'(X) - v_i}{X - i} \tag{23}$$

$$= \frac{\left(\phi(X) + \delta_j L_j(X)\right) - v_i}{X - i} \tag{24}$$

$$= \frac{\phi(X) - v_i}{X - i} - \frac{\delta_j L_j(X)}{X - i} \tag{25}$$

$$= q_i(X) + \delta_j \left(\frac{L_j(X)}{X - i}\right) \tag{26}$$

New Technique: Updating Proofs π_i After a Change to v_j, $j \neq i$

Consider new $q_i'(X)$ when v_j changed to $v_j + \delta_j$:

$$
\begin{aligned}
q_i'(X) &= \frac{\phi'(X) - v_i}{X - i} && (23)\\
&= \frac{\left(\phi(X) + \delta_j L_j(X)\right) - v_i}{X - i} && (24)\\
&= \frac{\phi(X) - v_i}{X - i} - \frac{\delta_j L_j(X)}{X - i} && (25)\\
&= q_i(X) + \delta_j \left(\frac{L_j(X)}{X - i}\right) && (26)
\end{aligned}
$$

Let $u_{i,j}$ be a KZG commitment to $U_{i,j}(X) = \frac{L_j(X)}{X-i}$.

New Technique: Updating Proofs π_i After a Change to v_j, $j \neq i$

Consider new $q_i'(X)$ when v_j changed to $v_j + \delta_j$:

$$q_i'(X) = \frac{\phi'(X) - v_i}{X - i} \tag{23}$$

$$= \frac{\left(\phi(X) + \delta_j L_j(X)\right) - v_i}{X - i} \tag{24}$$

$$= \frac{\phi(X) - v_i}{X - i} - \frac{\delta_j L_j(X)}{X - i} \tag{25}$$

$$= q_i(X) + \delta_j \left(\frac{L_j(X)}{X - i}\right) \tag{26}$$

Let $u_{i,j}$ be a KZG commitment to $U_{i,j}(X) = \frac{L_j(X)}{X-i}$. Then, $\pi_i' = g^{q_i'(\tau)}$ can be computed as:

New Technique: Updating Proofs π_i After a Change to v_j, $j \neq i$

Consider new $q_i'(X)$ when v_j changed to $v_j + \delta_j$:

$$\begin{aligned} q_i'(X) &= \frac{\phi'(X) - v_i}{X - i} && (23) \\ &= \frac{\left(\phi(X) + \delta_j L_j(X)\right) - v_i}{X - i} && (24) \\ &= \frac{\phi(X) - v_i}{X - i} - \frac{\delta_j L_j(X)}{X - i} && (25) \\ &= q_i(X) + \delta_j \left(\frac{L_j(X)}{X - i}\right) && (26) \end{aligned}$$

Let $u_{i,j}$ be a KZG commitment to $U_{i,j}(X) = \frac{L_j(X)}{X-i}$. Then, $\pi_i' = g^{q_i'(\tau)}$ can be computed as:

$$\pi_i' = \pi_i \cdot \left(u_{i,j}\right)^{\delta_j} \tag{27}$$

New Technique: Updating Proofs π_i After a Change to v_j, $j \neq i$

Consider new $q_i'(X)$ when v_j changed to $v_j + \delta_j$:

$$q_i'(X) = \frac{\phi'(X) - v_i}{X - i} \tag{23}$$

$$= \frac{\left(\phi(X) + \delta_j L_j(X)\right) - v_i}{X - i} \tag{24}$$

$$= \frac{\phi(X) - v_i}{X - i} - \frac{\delta_j L_j(X)}{X - i} \tag{25}$$

$$= q_i(X) + \delta_j \left(\frac{L_j(X)}{X - i}\right) \tag{26}$$

Let $u_{i,j}$ be a KZG commitment to $U_{i,j}(X) = \frac{L_j(X)}{X-i}$. Then, $\pi_i' = g^{q_i'(\tau)}$ can be computed as:

$$\pi_i' = \pi_i \cdot \left(u_{i,j}\right)^{\delta_j} \tag{27}$$

Big problem: To update π_i after a change to any j, need $upk_j = \{u_{i,j}, \forall i \neq j\}$

New Technique: Updating Proofs π_i After a Change to v_j, $j \neq i$

Consider new $q_i'(X)$ when v_j changed to $v_j + \delta_j$:

$$q_i'(X) = \frac{\phi'(X) - v_i}{X - i} \tag{23}$$

$$= \frac{\left(\phi(X) + \delta_j L_j(X)\right) - v_i}{X - i} \tag{24}$$

$$= \frac{\phi(X) - v_i}{X - i} - \frac{\delta_j L_j(X)}{X - i} \tag{25}$$

$$= q_i(X) + \delta_j \left(\frac{L_j(X)}{X - i}\right) \tag{26}$$

Let $u_{i,j}$ be a KZG commitment to $U_{i,j}(X) = \frac{L_j(X)}{X-i}$. Then, $\pi_i' = g^{q_i'(\tau)}$ can be computed as:

$$\pi_i' = \pi_i \cdot \left(u_{i,j}\right)^{\delta_j} \tag{27}$$

Big problem: To update π_i after a change to any j, need $upk_j = \{u_{i,j}, \forall i \neq j\} \Rightarrow |upk_j| = O(n)$.

New Technique: Updating Proofs π_i After a Change to v_j, $j \neq i$

Consider new $q_i'(X)$ when v_j changed to $v_j + \delta_j$:

$$q_i'(X) = \frac{\phi'(X) - v_i}{X - i} \tag{23}$$

$$= \frac{\left(\phi(X) + \delta_j L_j(X)\right) - v_i}{X - i} \tag{24}$$

$$= \frac{\phi(X) - v_i}{X - i} - \frac{\delta_j L_j(X)}{X - i} \tag{25}$$

$$= q_i(X) + \delta_j \left(\frac{L_j(X)}{X - i}\right) \tag{26}$$

Let $u_{i,j}$ be a KZG commitment to $U_{i,j}(X) = \frac{L_j(X)}{X-i}$. Then, $\pi_i' = g^{q_i'(\tau)}$ can be computed as:

$$\pi_i' = \pi_i \cdot \left(u_{i,j}\right)^{\delta_j} \tag{27}$$

Big problem: To update π_i after a change to any j, need $upk_j = \{u_{i,j}, \forall i \neq j\} \Rightarrow |upk_j| = O(n)$.

New technique: Compute $u_{i,j}$ in $O(1)$ time from upk_i and upk_j.

New Technique: Compute $u_{i,j}$ in $O(1)$ time

New Technique: Compute $u_{i,j}$ in $O(1)$ time

Let $A(X) = \prod_{i \in [0,n)}(X - i)$.

New Technique: Compute $u_{i,j}$ in $O(1)$ time

Let $A(X) = \prod_{i \in [0,n)}(X - i)$. Then, $L_j(X) = \frac{A(X)}{A'(j)(X-j)}$ (see Appendix 2),

New Technique: Compute $u_{i,j}$ in $O(1)$ time

Let $A(X) = \prod_{i \in [0,n)}(X - i)$. Then, $L_j(X) = \frac{A(X)}{A'(j)(X-j)}$ (see Appendix 2), and:

New Technique: Compute $u_{i,j}$ in $O(1)$ time

Let $A(X) = \prod_{i \in [0,n)}(X - i)$. Then, $L_j(X) = \frac{A(X)}{A'(j)(X-j)}$ (see Appendix 2), and:

$$U_{i,j}(X) = L_j(X)/(X - i) = \left(\frac{A(X)}{A'(j)(X - j)}\right)/(X - i) \tag{28}$$

(29)

(30)

New Technique: Compute $u_{i,j}$ in $O(1)$ time

Let $A(X) = \prod_{i\in[0,n)}(X - i)$. Then, $L_j(X) = \frac{A(X)}{A'(j)(X-j)}$ (see Appendix 2), and:

$$U_{i,j}(X) = L_j(X)/(X - i) = \left(\frac{A(X)}{A'(j)(X - j)}\right)/(X - i) \tag{28}$$

$$= \frac{1}{A'(j)} \cdot \frac{A(X)}{(X - j)(X - i)} \tag{29}$$

$$\tag{30}$$

New Technique: Compute $u_{i,j}$ in $O(1)$ time

Let $A(X) = \prod_{i \in [0,n)}(X - i)$. Then, $L_j(X) = \frac{A(X)}{A'(j)(X-j)}$ (see Appendix 2), and:

$$U_{i,j}(X) = L_j(X)/(X - i) = \left(\frac{A(X)}{A'(j)(X - j)}\right)/(X - i) \tag{28}$$

$$= \frac{1}{A'(j)} \cdot \frac{A(X)}{(X - j)(X - i)} \tag{29}$$

$$= \frac{1}{A'(j)} \cdot W_{i,j}(X) \tag{30}$$

New Technique: Compute $u_{i,j}$ in $O(1)$ time

Let $A(X) = \prod_{i \in [0,n)}(X - i)$. Then, $L_j(X) = \frac{A(X)}{A'(j)(X-j)}$ (see Appendix 2), and:

$$U_{i,j}(X) = L_j(X)/(X - i) = \left(\frac{A(X)}{A'(j)(X - j)}\right)/(X - i) \tag{28}$$

$$= \frac{1}{A'(j)} \cdot \frac{A(X)}{(X - j)(X - i)} \tag{29}$$

$$= \frac{1}{A'(j)} \cdot W_{i,j}(X) \tag{30}$$

Fortunately, it happens that (see Appendix 1):

New Technique: Compute $u_{i,j}$ in $O(1)$ time

Let $A(X) = \prod_{i \in [0,n)}(X - i)$. Then, $L_j(X) = \frac{A(X)}{A'(j)(X-j)}$ (see Appendix 2), and:

$$U_{i,j}(X) = L_j(X)/(X - i) = \left(\frac{A(X)}{A'(j)(X - j)}\right)/(X - i) \tag{28}$$

$$= \frac{1}{A'(j)} \cdot \frac{A(X)}{(X - j)(X - i)} \tag{29}$$

$$= \frac{1}{A'(j)} \cdot W_{i,j}(X) \tag{30}$$

Fortunately, it happens that (see Appendix 1):

$$\frac{1}{i - j} \cdot \frac{A(X)}{X - i} + \frac{1}{j - i} \cdot \frac{A(X)}{X - j} = \frac{A(X)}{(X - i)(X - j)} = W_{i,j}(X) \tag{31}$$

New Technique: Compute $u_{i,j}$ in $O(1)$ time

Let $A(X) = \prod_{i \in [0,n)} (X - i)$. Then, $L_j(X) = \frac{A(X)}{A'(j)(X-j)}$ (see Appendix 2), and:

$$U_{i,j}(X) = L_j(X)/(X - i) = \left(\frac{A(X)}{A'(j)(X - j)}\right)/(X - i) \tag{28}$$

$$= \frac{1}{A'(j)} \cdot \frac{A(X)}{(X - j)(X - i)} \tag{29}$$

$$= \frac{1}{A'(j)} \cdot W_{i,j}(X) \tag{30}$$

Fortunately, it happens that (see Appendix 1):

$$\frac{1}{i - j} \cdot \frac{A(X)}{X - i} + \frac{1}{j - i} \cdot \frac{A(X)}{X - j} = \frac{A(X)}{(X - i)(X - j)} = W_{i,j}(X) \tag{31}$$

New technique: Let $a_i = g^{A(\tau)/(\tau - i)}$ and compute $u_{i,j}$ as:

New Technique: Compute $u_{i,j}$ in $O(1)$ time

Let $A(X) = \prod_{i \in [0,n)} (X - i)$. Then, $L_j(X) = \frac{A(X)}{A'(j)(X-j)}$ (see Appendix 2), and:

$$U_{i,j}(X) = L_j(X)/(X - i) = \left(\frac{A(X)}{A'(j)(X - j)}\right)/(X - i) \tag{28}$$

$$= \frac{1}{A'(j)} \cdot \frac{A(X)}{(X - j)(X - i)} \tag{29}$$

$$= \frac{1}{A'(j)} \cdot W_{i,j}(X) \tag{30}$$

Fortunately, it happens that (see Appendix 1):

$$\frac{1}{i - j} \cdot \frac{A(X)}{X - i} + \frac{1}{j - i} \cdot \frac{A(X)}{X - j} = \frac{A(X)}{(X - i)(X - j)} = W_{i,j}(X) \tag{31}$$

New technique: Let $a_i = g^{A(\tau)/(\tau - i)}$ and compute $u_{i,j}$ as:

$$u_{i,j} = \left(a_i^{\frac{1}{i-j}} \cdot a_j^{\frac{1}{j-i}}\right)^{\frac{1}{A'(j)}} \tag{32}$$

New Technique: Compute $u_{i,j}$ in $O(1)$ time

Let $A(X) = \prod_{i\in[0,n)}(X - i)$. Then, $L_j(X) = \frac{A(X)}{A'(j)(X-j)}$ (see Appendix 2), and:

$$U_{i,j}(X) = L_j(X)/(X-i) = \left(\frac{A(X)}{A'(j)(X-j)}\right)/(X-i) \tag{28}$$

$$= \frac{1}{A'(j)} \cdot \frac{A(X)}{(X-j)(X-i)} \tag{29}$$

$$= \frac{1}{A'(j)} \cdot W_{i,j}(X) \tag{30}$$

Fortunately, it happens that (see Appendix 1):

$$\frac{1}{i-j} \cdot \frac{A(X)}{X-i} + \frac{1}{j-i} \cdot \frac{A(X)}{X-j} = \frac{A(X)}{(X-i)(X-j)} = W_{i,j}(X) \tag{31}$$

New technique: Let $a_i = g^{A(\tau)/(\tau-i)}$ and compute $u_{i,j}$ as:

$$u_{i,j} = \left(a_i^{\frac{1}{i-j}} \cdot a_j^{\frac{1}{j-i}}\right)^{\frac{1}{A'(j)}} \tag{32}$$

Job done: Each upk_i must include a_i and $A'(i)$.

New Technique: Compute $u_{i,j}$ in $O(1)$ time

Let $A(X) = \prod_{i \in [0,n)}(X - i)$. Then, $L_j(X) = \frac{A(X)}{A'(j)(X-j)}$ (see Appendix 2), and:

$$U_{i,j}(X) = L_j(X)/(X - i) = \left(\frac{A(X)}{A'(j)(X - j)}\right)/(X - i) \tag{28}$$

$$= \frac{1}{A'(j)} \cdot \frac{A(X)}{(X - j)(X - i)} \tag{29}$$

$$= \frac{1}{A'(j)} \cdot W_{i,j}(X) \tag{30}$$

Fortunately, it happens that (see Appendix 1):

$$\frac{1}{i - j} \cdot \frac{A(X)}{X - i} + \frac{1}{j - i} \cdot \frac{A(X)}{X - j} = \frac{A(X)}{(X - i)(X - j)} = W_{i,j}(X) \tag{31}$$

New technique: Let $a_i = g^{A(\tau)/(\tau - i)}$ and compute $u_{i,j}$ as:

$$u_{i,j} = \left(a_i^{\frac{1}{i-j}} \cdot a_j^{\frac{1}{j-i}}\right)^{\frac{1}{A'(j)}} \tag{32}$$

Job done: Each upk_i must include a_i and $A'(i)$. Wait, what about computing all a_i's?

New Technique: Compute $u_{i,j}$ in $O(1)$ time

Let $A(X) = \prod_{i \in [0,n)}(X - i)$. Then, $L_j(X) = \frac{A(X)}{A'(j)(X-j)}$ (see Appendix 2), and:

$$U_{i,j}(X) = L_j(X)/(X - i) = \left(\frac{A(X)}{A'(j)(X - j)}\right)/(X - i) \tag{28}$$

$$= \frac{1}{A'(j)} \cdot \frac{A(X)}{(X - j)(X - i)} \tag{29}$$

$$= \frac{1}{A'(j)} \cdot W_{i,j}(X) \tag{30}$$

Fortunately, it happens that (see Appendix 1):

$$\frac{1}{i - j} \cdot \frac{A(X)}{X - i} + \frac{1}{j - i} \cdot \frac{A(X)}{X - j} = \frac{A(X)}{(X - i)(X - j)} = W_{i,j}(X) \tag{31}$$

New technique: Let $a_i = g^{A(\tau)/(\tau - i)}$ and compute $u_{i,j}$ as:

$$u_{i,j} = \left(a_i^{\frac{1}{i-j}} \cdot a_j^{\frac{1}{j-i}}\right)^{\frac{1}{A'(j)}} \tag{32}$$

Job done: Each upk_i must include a_i and $A'(i)$. Wait, what about computing all a_i's? Later.

Outline

Computing I-subvector Proofs From Scratch

Computing I-subvector Proofs From Scratch

Can use KZG batch proofs to compute a constant-sized I-subvector proof π_I for all $(v_i)_{i \in I}$:

Computing I-subvector Proofs From Scratch

Can use KZG batch proofs to compute a constant-sized I-subvector proof π_I for all $(v_i)_{i \in I}$:

$$A_I(X) = \prod_{i \in I}(X - i) \tag{33}$$

(34)

(35)

Computing I-subvector Proofs From Scratch

Can use KZG batch proofs to compute a constant-sized I-subvector proof π_I for all $(v_i)_{i \in I}$:

$$A_I(X) = \prod_{i \in I}(X - i) \tag{33}$$

$$R_I(X) = \sum_{i \in I} v_i \cdot L_i^*(X), \text{ where } L_i^*(X) = \prod_{j \in I, j \neq i} \frac{X - j}{i - j} \tag{34}$$

$$\tag{35}$$

Computing I-subvector Proofs From Scratch

Can use KZG batch proofs to compute a constant-sized I-subvector proof π_I for all $(v_i)_{i \in I}$:

$$A_I(X) = \prod_{i \in I} (X - i) \tag{33}$$

$$R_I(X) = \sum_{i \in I} v_i \cdot L_i^*(X), \text{ where } L_i^*(X) = \prod_{j \in I, j \neq i} \frac{X - j}{i - j} \tag{34}$$

$$q_I(X) = \frac{\phi(X) - R_I(X)}{A_I(X)} \tag{35}$$

Computing I-subvector Proofs From Scratch

Can use KZG batch proofs to compute a constant-sized I-subvector proof π_I for all $(v_i)_{i \in I}$:

$$A_I(X) = \prod_{i \in I}(X - i) \tag{33}$$

$$R_I(X) = \sum_{i \in I} v_i \cdot L_i^*(X), \text{ where } L_i^*(X) = \prod_{j \in I, j \neq i} \frac{X - j}{i - j} \tag{34}$$

$$q_I(X) = \frac{\phi(X) - R_I(X)}{A_I(X)} \tag{35}$$

Note that

Computing I-subvector Proofs From Scratch

Can use KZG batch proofs to compute a constant-sized I-subvector proof π_I for all $(v_i)_{i\in I}$:

$$A_I(X) = \prod_{i\in I}(X - i) \tag{33}$$

$$R_I(X) = \sum_{i\in I} v_i \cdot L_i^*(X), \text{ where } L_i^*(X) = \prod_{j\in I, j\neq i} \frac{X - j}{i - j} \tag{34}$$

$$q_I(X) = \frac{\phi(X) - R_I(X)}{A_I(X)} \tag{35}$$

Note that

$A_I(i) = 0$ and $R_I(i) = v_i$, $\forall i \in I$.

Computing I-subvector Proofs From Scratch

Can use KZG batch proofs to compute a constant-sized I-subvector proof π_I for all $(v_i)_{i\in I}$:

$$A_I(X) = \prod_{i\in I}(X - i) \tag{33}$$

$$R_I(X) = \sum_{i\in I} v_i \cdot L_i^*(X), \text{ where } L_i^*(X) = \prod_{j\in I, j\neq i} \frac{X - j}{i - j} \tag{34}$$

$$q_I(X) = \frac{\phi(X) - R_I(X)}{A_I(X)} \tag{35}$$

Note that

$A_I(i) = 0$ and $R_I(i) = v_i, \forall i \in I$.
Can compute $A_I(X)$ and $R_I(X)$ in $O(|I| \log^2 |I|)$ field operations.

Computing I-subvector Proofs From Scratch

Can use KZG batch proofs to compute a constant-sized I-subvector proof π_I for all $(v_i)_{i\in I}$:

$$A_I(X) = \prod_{i\in I}(X - i) \tag{33}$$

$$R_I(X) = \sum_{i\in I} v_i \cdot L_i^*(X), \text{ where } L_i^*(X) = \prod_{j\in I, j\neq i} \frac{X - j}{i - j} \tag{34}$$

$$q_I(X) = \frac{\phi(X) - R_I(X)}{A_I(X)} \tag{35}$$

Note that

$A_I(i) = 0$ and $R_I(i) = v_i, \forall i \in I$.
Can compute $A_I(X)$ and $R_I(X)$ in $O(|I| \log^2 |I|)$ field operations.
Can compute $q_I(X)$ in $O(n \log n)$ field operations.

Computing I-subvector Proofs From Scratch

Can use KZG batch proofs to compute a constant-sized I-subvector proof π_I for all $(v_i)_{i \in I}$:

$$A_I(X) = \prod_{i \in I}(X - i) \tag{33}$$

$$R_I(X) = \sum_{i \in I} v_i \cdot L_i^*(X), \text{ where } L_i^*(X) = \prod_{j \in I, j \neq i} \frac{X - j}{i - j} \tag{34}$$

$$q_I(X) = \frac{\phi(X) - R_I(X)}{A_I(X)} \tag{35}$$

Note that

$A_I(i) = 0$ and $R_I(i) = v_i$, $\forall i \in I$.
Can compute $A_I(X)$ and $R_I(X)$ in $O(|I| \log^2 |I|)$ field operations.
Can compute $q_I(X)$ in $O(n \log n)$ field operations.

I-subvector proof is $\pi_I = g^{q_I(\tau)}$, computed with $O(n - |I|)$-sized multiexp.

Computing I-subvector Proofs From Scratch

Can use KZG batch proofs to compute a constant-sized I-subvector proof π_I for all $(v_i)_{i\in I}$:

$$A_I(X) = \prod_{i\in I}(X - i) \tag{33}$$

$$R_I(X) = \sum_{i\in I} v_i \cdot L_i^*(X), \text{ where } L_i^*(X) = \prod_{j\in I, j\neq i} \frac{X - j}{i - j} \tag{34}$$

$$q_I(X) = \frac{\phi(X) - R_I(X)}{A_I(X)} \tag{35}$$

Note that

$A_I(i) = 0$ and $R_I(i) = v_i, \forall i \in I$.
Can compute $A_I(X)$ and $R_I(X)$ in $O(|I| \log^2 |I|)$ field operations.
Can compute $q_I(X)$ in $O(n \log n)$ field operations.

I-subvector proof is $\pi_I = g^{q_I(\tau)}$, computed with $O(n - |I|)$-sized multiexp. To verify:

Computing I-subvector Proofs From Scratch

Can use KZG batch proofs to compute a constant-sized I-subvector proof π_I for all $(v_i)_{i\in I}$:

$$A_I(X) = \prod_{i\in I}(X - i) \tag{33}$$

$$R_I(X) = \sum_{i\in I} v_i \cdot L_i^*(X), \text{ where } L_i^*(X) = \prod_{j\in I, j\neq i} \frac{X - j}{i - j} \tag{34}$$

$$q_I(X) = \frac{\phi(X) - R_I(X)}{A_I(X)} \tag{35}$$

Note that

$A_I(i) = 0$ and $R_I(i) = v_i, \forall i \in I$.
Can compute $A_I(X)$ and $R_I(X)$ in $O(|I| \log^2 |I|)$ field operations.
Can compute $q_I(X)$ in $O(n \log n)$ field operations.

I-subvector proof is $\pi_I = g^{q_I(\tau)}$, computed with $O(n - |I|)$-sized multiexp. To verify:

$$e(c/g^{R_I(\tau)}, g) = e(\pi_I, g^{A_I(\tau)}) \tag{36}$$

Computing I-subvector Proofs From Scratch

Can use KZG batch proofs to compute a constant-sized I-subvector proof π_I for all $(v_i)_{i\in I}$:

$$A_I(X) = \prod_{i\in I}(X - i) \tag{33}$$

$$R_I(X) = \sum_{i\in I} v_i \cdot L_i^*(X), \text{ where } L_i^*(X) = \prod_{j\in I, j\neq i} \frac{X-j}{i-j} \tag{34}$$

$$q_I(X) = \frac{\phi(X) - R_I(X)}{A_I(X)} \tag{35}$$

Note that

$A_I(i) = 0$ and $R_I(i) = v_i, \forall i \in I$.
Can compute $A_I(X)$ and $R_I(X)$ in $O(|I| \log^2 |I|)$ field operations.
Can compute $q_I(X)$ in $O(n \log n)$ field operations.

I-subvector proof is $\pi_I = g^{q_I(\tau)}$, computed with $O(n - |I|)$-sized multiexp. To verify:

$$e(c/g^{R_I(\tau)}, g) = e(\pi_I, g^{A_I(\tau)}) \tag{36}$$

Note: *vrk* contains $(g^{\tau^i})_{i\in[0,|I|]}$

Aggregating I-subvector Proof π_I From $(\pi_i)_{i \in I}$

Aggregating I-subvector Proof π_I From $(\pi_i)_{i \in I}$

We use *partial fraction decomposition*, as proposed by Drake and Buterin [But20] (see Appendix 2):

Aggregating I-subvector Proof π_I From $(\pi_i)_{i\in I}$

We use *partial fraction decomposition*, as proposed by Drake and Buterin [But20] (see Appendix 2):

$$\frac{1}{A_I(X)} = \frac{1}{\prod_{i\in I}(X-i)} = \sum_{i\in I} \frac{1}{A_I'(i)} \cdot \frac{1}{X-i} \tag{37}$$

Aggregating I-subvector Proof π_I From $(\pi_i)_{i\in I}$

We use *partial fraction decomposition*, as proposed by Drake and Buterin [But20] (see Appendix 2):

$$\frac{1}{A_I(X)} = \frac{1}{\prod_{i\in I}(X-i)} = \sum_{i\in I} \frac{1}{A_I'(i)} \cdot \frac{1}{X-i} \tag{37}$$

We “decompose” the quotient $q_I(X) = \frac{\phi(X)-R_I(X)}{A_I(X)}$ in π_I as:

Aggregating I-subvector Proof π_I From $(\pi_i)_{i \in I}$

We use *partial fraction decomposition*, as proposed by Drake and Buterin [But20] (see Appendix 2):

$$\frac{1}{A_I(X)} = \frac{1}{\prod_{i \in I}(X - i)} = \sum_{i \in I} \frac{1}{A_I'(i)} \cdot \frac{1}{X - i} \tag{37}$$

We "decompose" the quotient $q_I(X) = \frac{\phi(X) - R_I(X)}{A_I(X)}$ in π_I as:

$$q_I(X) = \phi(X)\frac{1}{A_I(X)} - R_I(X)\frac{1}{A_I(X)} \tag{38}$$

(39)

(40)

(41)

(42)

(43)

Aggregating I-subvector Proof π_I From $(\pi_i)_{i \in I}$

We use *partial fraction decomposition*, as proposed by Drake and Buterin [But20] (see Appendix 2):

$$\frac{1}{A_I(X)} = \frac{1}{\prod_{i \in I}(X - i)} = \sum_{i \in I} \frac{1}{A_I'(i)} \cdot \frac{1}{X - i} \tag{37}$$

We "decompose" the quotient $q_I(X) = \frac{\phi(X) - R_I(X)}{A_I(X)}$ in π_I as:

$$q_I(X) = \phi(X)\frac{1}{A_I(X)} - R_I(X)\frac{1}{A_I(X)} \tag{38}$$

$$= \phi(X)\frac{1}{A_I(X)} - \left(\sum_{i \in I} v_i \cdot L_i^*(X)\right)\frac{1}{A_I(X)} \tag{39}$$

(40)

(41)

(42)

(43)

Aggregating I-subvector Proof π_I From $(\pi_i)_{i\in I}$

We use *partial fraction decomposition*, as proposed by Drake and Buterin [But20] (see Appendix 2):

$$\frac{1}{A_I(X)} = \frac{1}{\prod_{i\in I}(X-i)} = \sum_{i\in I} \frac{1}{A_I'(i)} \cdot \frac{1}{X-i} \tag{37}$$

We “decompose” the quotient $q_I(X) = \frac{\phi(X)-R_I(X)}{A_I(X)}$ in π_I as:

$$q_I(X) = \phi(X)\frac{1}{A_I(X)} - R_I(X)\frac{1}{A_I(X)} \tag{38}$$

$$= \phi(X)\frac{1}{A_I(X)} - \left(\sum_{i\in I} v_i \cdot L_i^*(X)\right)\frac{1}{A_I(X)} \tag{39}$$

$$= \phi(X)\sum_{i\in I}\frac{1}{A_I{}'(i)(X-i)} - \left(\sum_{i\in I} v_i \cdot \frac{A_I(X)}{A_I{}'(i)(X-i)}\right) \cdot \frac{1}{A_I(X)} \tag{40}$$

(41)

(42)

(43)

Aggregating I-subvector Proof π_I From $(\pi_i)_{i\in I}$

We use *partial fraction decomposition*, as proposed by Drake and Buterin [But20] (see Appendix 2):

$$\frac{1}{A_I(X)} = \frac{1}{\prod_{i\in I}(X-i)} = \sum_{i\in I} \frac{1}{A_I'(i)} \cdot \frac{1}{X-i} \tag{37}$$

We "decompose" the quotient $q_I(X) = \frac{\phi(X)-R_I(X)}{A_I(X)}$ in π_I as:

$$\begin{aligned}
q_I(X) &= \phi(X)\frac{1}{A_I(X)} - R_I(X)\frac{1}{A_I(X)} && (38)\\
&= \phi(X)\frac{1}{A_I(X)} - \left(\sum_{i\in I} v_i \cdot L_i^*(X)\right)\frac{1}{A_I(X)} && (39)\\
&= \phi(X)\sum_{i\in I}\frac{1}{A_I{'}(i)(X-i)} - \left(\sum_{i\in I} v_i \cdot \frac{A_I(X)}{A_I{'}(i)(X-i)}\right)\cdot\frac{1}{A_I(X)} && (40)\\
&= \sum_{i\in I}\frac{\phi(X)}{A_I{'}(i)(X-i)} - \sum_{i\in I}\frac{v_i}{A_I{'}(i)(X-i)} && (41)\\
& && (42)\\
& && (43)
\end{aligned}$$

Aggregating I-subvector Proof π_I From $(\pi_i)_{i \in I}$

We use *partial fraction decomposition*, as proposed by Drake and Buterin [But20] (see Appendix 2):

$$\frac{1}{A_I(X)} = \frac{1}{\prod_{i \in I}(X - i)} = \sum_{i \in I} \frac{1}{A_I'(i)} \cdot \frac{1}{X - i} \tag{37}$$

We "decompose" the quotient $q_I(X) = \frac{\phi(X) - R_I(X)}{A_I(X)}$ in π_I as:

$$\begin{aligned}
q_I(X) &= \phi(X)\frac{1}{A_I(X)} - R_I(X)\frac{1}{A_I(X)} && (38)\\
&= \phi(X)\frac{1}{A_I(X)} - \left(\sum_{i \in I} v_i \cdot L_i^*(X)\right)\frac{1}{A_I(X)} && (39)\\
&= \phi(X)\sum_{i \in I}\frac{1}{A_I{}'(i)(X - i)} - \left(\sum_{i \in I} v_i \cdot \frac{A_I(X)}{A_I{}'(i)(X - i)}\right) \cdot \frac{1}{A_I(X)} && (40)\\
&= \sum_{i \in I}\frac{\phi(X)}{A_I{}'(i)(X - i)} - \sum_{i \in I}\frac{v_i}{A_I{}'(i)(X - i)} && (41)\\
&= \sum_{i \in I}\frac{1}{A_I{}'(i)} \cdot \frac{\phi(X) - v_i}{X - i} && (42)\\
& && (43)
\end{aligned}$$

Aggregating I-subvector Proof π_I From $(\pi_i)_{i \in I}$

We use *partial fraction decomposition*, as proposed by Drake and Buterin [But20] (see Appendix 2):

$$\frac{1}{A_I(X)} = \frac{1}{\prod_{i \in I}(X - i)} = \sum_{i \in I} \frac{1}{A_I'(i)} \cdot \frac{1}{X - i} \tag{37}$$

We "decompose" the quotient $q_I(X) = \frac{\phi(X) - R_I(X)}{A_I(X)}$ in π_I as:

$$\begin{aligned}
q_I(X) &= \phi(X)\frac{1}{A_I(X)} - R_I(X)\frac{1}{A_I(X)} && (38)\\
&= \phi(X)\frac{1}{A_I(X)} - \left(\sum_{i \in I} v_i \cdot L_i^*(X)\right)\frac{1}{A_I(X)} && (39)\\
&= \phi(X)\sum_{i \in I}\frac{1}{A_I{'}(i)(X - i)} - \left(\sum_{i \in I} v_i \cdot \frac{A_I(X)}{A_I{'}(i)(X - i)}\right) \cdot \frac{1}{A_I(X)} && (40)\\
&= \sum_{i \in I}\frac{\phi(X)}{A_I{'}(i)(X - i)} - \sum_{i \in I}\frac{v_i}{A_I{'}(i)(X - i)} && (41)\\
&= \sum_{i \in I}\frac{1}{A_I{'}(i)} \cdot \frac{\phi(X) - v_i}{X - i} && (42)\\
&= \sum_{i \in I}\frac{1}{A_I{'}(i)} \cdot q_i(X) && (43)
\end{aligned}$$

Aggregating I-subvector Proof π_I From $(\pi_i)_{i \in I}$ (Continued)

Aggregating I-subvector Proof π_I From $(\pi_i)_{i \in I}$ (Continued)

To aggregate π_I:

Aggregating I-subvector Proof π_I From $(\pi_i)_{i \in I}$ (Continued)

To aggregate π_I:

Step 1: Interpolate $A_I(X) = \prod_{i \in I}(X - i)$ in $O(|I| \log^2 |I|)$ field operations.

Aggregating I-subvector Proof π_I From $(\pi_i)_{i \in I}$ (Continued)

To aggregate π_I:

Step 1: Interpolate $A_I(X) = \prod_{i \in I}(X - i)$ in $O(|I| \log^2 |I|)$ field operations.
Step 2: Compute its derivative $A_I'(X)$ in $O(|I|)$ field operations.

Aggregating I-subvector Proof π_I From $(\pi_i)_{i \in I}$ (Continued)

To aggregate π_I:

Step 1: Interpolate $A_I(X) = \prod_{i \in I}(X - i)$ in $O(|I| \log^2 |I|)$ field operations.
Step 2: Compute its derivative $A_I'(X)$ in $O(|I|)$ field operations.
Step 3: Compute all $A_I'(i)$ in $O(|I| \log^2 |I|)$ field operations via a *polynomial multipoint evaluation* [vzGG13].

Aggregating I-subvector Proof π_I From $(\pi_i)_{i \in I}$ (Continued)

To aggregate π_I:

Step 1: Interpolate $A_I(X) = \prod_{i \in I}(X - i)$ in $O(|I| \log^2 |I|)$ field operations.
Step 2: Compute its derivative $A_I'(X)$ in $O(|I|)$ field operations.
Step 3: Compute all $A_I'(i)$ in $O(|I| \log^2 |I|)$ field operations via a *polynomial multipoint evaluation* [vzGG13].
Step 4: Compute π_I using an $O(|I|)$-sized multiexp:

Aggregating I-subvector Proof π_I From $(\pi_i)_{i\in I}$ (Continued)

To aggregate π_I:

Step 1: Interpolate $A_I(X) = \prod_{i\in I}(X - i)$ in $O(|I| \log^2 |I|)$ field operations.
Step 2: Compute its derivative $A_I'(X)$ in $O(|I|)$ field operations.
Step 3: Compute all $A_I'(i)$ in $O(|I| \log^2 |I|)$ field operations via a *polynomial multipoint evaluation* [vzGG13].
Step 4: Compute π_I using an $O(|I|)$-sized multiexp:

$$\pi_I = \prod_{i\in I} \pi_i^{1/A_I'(i)} \tag{44}$$

Aggregating I-subvector Proof π_I From $(\pi_i)_{i \in I}$ (Continued)

To aggregate π_I:

Step 1: Interpolate $A_I(X) = \prod_{i \in I}(X - i)$ in $O(|I| \log^2 |I|)$ field operations.
Step 2: Compute its derivative $A_I'(X)$ in $O(|I|)$ field operations.
Step 3: Compute all $A_I'(i)$ in $O(|I| \log^2 |I|)$ field operations via a *polynomial multipoint evaluation* [vzGG13].
Step 4: Compute π_I using an $O(|I|)$-sized multiexp:

$$\pi_I = \prod_{i \in I} \pi_i^{1/A_I'(i)} \tag{44}$$

Job done: Can aggregate π_i's for $(v_i)_{i \in I}$ into constant-sized π_I.

Conclusion

Summary

Summary

Main contributions:

Summary

Main contributions:

- New aSVC with constant-sized (subvector) proofs and constant-sized update keys

Summary

Main contributions:

- New aSVC with constant-sized (subvector) proofs and constant-sized update keys
- Proofs can be aggregated efficiently into subvector proof

Summary

Main contributions:

- New aSVC with constant-sized (subvector) proofs and constant-sized update keys
- Proofs can be aggregated efficiently into subvector proof
- Proofs can be updated efficiently

Summary

Main contributions:

- New aSVC with constant-sized (subvector) proofs and constant-sized update keys
- Proofs can be aggregated efficiently into subvector proof
- Proofs can be updated efficiently
- Can be used to build highly-efficient, account-based stateless cryptocurrencies

Summary

Main contributions:

- New aSVC with constant-sized (subvector) proofs and constant-sized update keys
- Proofs can be aggregated efficiently into subvector proof
- Proofs can be updated efficiently
- Can be used to build highly-efficient, account-based stateless cryptocurrencies

Other goodies (not in this talk):

Summary

Main contributions:

- New aSVC with constant-sized (subvector) proofs and constant-sized update keys
- Proofs can be aggregated efficiently into subvector proof
- Proofs can be updated efficiently
- Can be used to build highly-efficient, account-based stateless cryptocurrencies

Other goodies (not in this talk):

- aSVC formalization that accounts for update keys

Summary

Main contributions:

- New aSVC with constant-sized (subvector) proofs and constant-sized update keys
- Proofs can be aggregated efficiently into subvector proof
- Proofs can be updated efficiently
- Can be used to build highly-efficient, account-based stateless cryptocurrencies

Other goodies (not in this talk):

- aSVC formalization that accounts for update keys
- Each update key upk_i is verifiable against vrk

Summary

Main contributions:

- New aSVC with constant-sized (subvector) proofs and constant-sized update keys
- Proofs can be aggregated efficiently into subvector proof
- Proofs can be updated efficiently
- Can be used to build highly-efficient, account-based stateless cryptocurrencies

Other goodies (not in this talk):

- aSVC formalization that accounts for update keys
- Each update key upk_i is verifiable against vrk
- New security definition for KZG batch proofs, which reduces to n-SBDH

Summary

Main contributions:

- New aSVC with constant-sized (subvector) proofs and constant-sized update keys
- Proofs can be aggregated efficiently into subvector proof
- Proofs can be updated efficiently
- Can be used to build highly-efficient, account-based stateless cryptocurrencies

Other goodies (not in this talk):

- aSVC formalization that accounts for update keys
- Each update key upk_i is verifiable against vrk
- New security definition for KZG batch proofs, which reduces to n-SBDH
- Subtleties of VC-based stateless cryptocurrencies

Summary

Main contributions:

- New aSVC with constant-sized (subvector) proofs and constant-sized update keys
- Proofs can be aggregated efficiently into subvector proof
- Proofs can be updated efficiently
- Can be used to build highly-efficient, account-based stateless cryptocurrencies

Other goodies (not in this talk):

- aSVC formalization that accounts for update keys
- Each update key upk_i is verifiable against vrk
- New security definition for KZG batch proofs, which reduces to n-SBDH
- Subtleties of VC-based stateless cryptocurrencies
- Aggregating multiple I-subvector proofs *across different commitments* [GRWZ20].

Roots of Unity Benefits

Our scheme actually uses $\phi(\omega_i) = v_i$, where ω is a primitive nth root of unity.

Roots of Unity Benefits

Our scheme actually uses $\phi(\omega_i) = v_i$, where ω is a primitive nth root of unity. This has several advantages:

Roots of Unity Benefits

Our scheme actually uses $\phi(\omega_i) = v_i$, where ω is a primitive nth root of unity. This has several advantages:

- Our public parameters can be *efficiently* derived from "powers-of-tau" g^{τ^i}'s:

Roots of Unity Benefits

Our scheme actually uses $\phi(\omega_i) = v_i$, where ω is a primitive nth root of unity. This has several advantages:

- Our public parameters can be *efficiently* derived from "powers-of-tau" g^{τ^i}'s:
  - ℓ_i's via an inverse FFT [Vir17]

Roots of Unity Benefits

Our scheme actually uses $\phi(\omega_i) = v_i$, where ω is a primitive nth root of unity. This has several advantages:

- Our public parameters can be *efficiently* derived from "powers-of-tau" g^{τ^i}'s:
    - ℓ_i's via an inverse FFT [Vir17]
    - a_i's via the Feist-Khovratovich (FK) technique [FK20]

Roots of Unity Benefits

Our scheme actually uses $\phi(\omega_i) = v_i$, where ω is a primitive nth root of unity. This has several advantages:

- Our public parameters can be *efficiently* derived from "powers-of-tau" g^{τ^i}'s:
  - ℓ_i's via an inverse FFT [Vir17]
  - a_i's via the Feist-Khovratovich (FK) technique [FK20]
  - u_i's via our new, FK-like, technique (see [TAB+20, Sec 3.4.5])

Roots of Unity Benefits

Our scheme actually uses $\phi(\omega_i) = v_i$, where ω is a primitive nth root of unity. This has several advantages:

- Our public parameters can be *efficiently* derived from "powers-of-tau" g^{τ^i}'s:
  - ℓ_i's via an inverse FFT [Vir17]
  - a_i's via the Feist-Khovratovich (FK) technique [FK20]
  - u_i's via our new, FK-like, technique (see [TAB+20, Sec 3.4.5])
- Since g^{τ^i}'s are updatable, our public parameters are *updatable*.

Roots of Unity Benefits

Our scheme actually uses $\phi(\omega_i) = v_i$, where ω is a primitive nth root of unity. This has several advantages:

- Our public parameters can be *efficiently* derived from “powers-of-tau” g^{τ^i}’s:
  - ℓ_i’s via an inverse FFT [Vir17]
  - a_i’s via the Feist-Khovratovich (FK) technique [FK20]
  - u_i’s via our new, FK-like, technique (see [TAB⁺20, Sec 3.4.5])
- Since g^{τ^i}’s are updatable, our public parameters are *updatable*.
- Can precompute all n proofs in $O(n \log n)$ time via FK technique [FK20].

Roots of Unity Benefits

Our scheme actually uses $\phi(\omega_i) = v_i$, where ω is a primitive nth root of unity. This has several advantages:

- Our public parameters can be *efficiently* derived from "powers-of-tau" g^{τ^i}'s:
  - ℓ_i's via an inverse FFT [Vir17]
  - a_i's via the Feist-Khovratovich (FK) technique [FK20]
  - u_i's via our new, FK-like, technique (see [TAB$^+$20, Sec 3.4.5])
- Since g^{τ^i}'s are updatable, our public parameters are *updatable*.
- Can precompute all n proofs in $O(n \log n)$ time via FK technique [FK20].
- Can remove $A'(i)$ from upk_i.

Questions?

Outline

Decomposition of $1/((X-i)(X-j))$

Decomposition of $1/A_I(X)$

Decomposition of $A(X)/\,((X-i)(X-j))$

Note that:

$$\frac{1}{i-j}\cdot\frac{A(X)}{X-i}+\frac{1}{j-i}\cdot\frac{A(X)}{X-j}=\frac{1}{i-j}\cdot\frac{A(X)(X-j)}{(X-i)(X-j)}+\frac{1}{j-i}\cdot\frac{A(X)(X-i)}{(X-j)(X-i)} \tag{45}$$

$$=\frac{\frac{1}{i-j}A(X)(X-j)-\frac{1}{i-j}A(X)(X-i)}{(X-i)(X-j)} \tag{46}$$

$$=\frac{\frac{1}{i-j}A(X)[(X-j)-(X-i)]}{(X-i)(X-j)} \tag{47}$$

$$=\frac{\frac{1}{i-j}A(X)(-j+i)}{(X-i)(X-j)} \tag{48}$$

$$=\frac{A(X)}{(X-i)(X-j)} \tag{49}$$

Outline

Decomposition of $1/((X-i)(X-j))$

Decomposition of $1/A_I(X)$

Partial Fraction Decomposition From Lagrange Interpolation

It is well-known that Lagrange coefficients can be *rewritten* as [BT04, vzGG13]:

$$L_i(X) = \prod_{j \in I, j \neq i} \frac{X - j}{i - j} = \frac{A_I(X)}{A_I'(i)(X - i)}, \text{ where } A_I(X) = \prod_{i \in I} (X - i) \tag{50}$$

Here, $A_I'(X)$ is the derivative of $A_I(X)$ and has the (non-obvious) property that $A_I'(i) = \prod_{j \in I, j \neq i}(i - j)$. Next, consider the Lagrange interpolation of $\phi(X) = 1$:

$$\phi(X) = \sum_{i \in I} v_i L_i(X) \Leftrightarrow \tag{51}$$

$$1 = A_I(X) \sum_{i \in [0,n)} \frac{v_i}{A_I'(i)(X - i)} \Leftrightarrow \tag{52}$$

$$\frac{1}{A_I(X)} = \sum_{i \in I} \frac{1}{A_I'(i)(X - i)} \Leftrightarrow \tag{53}$$

$$\frac{1}{A_I(X)} = \sum_{i \in I} \frac{1}{A_I'(i)} \cdot \frac{1}{(X - i)} \Rightarrow \tag{54}$$

$$c_i = \frac{1}{A_I'(i)} \tag{55}$$

J. Berrut and L. Trefethen.
Barycentric Lagrange Interpolation.
SIAM Review, 46(3):501–517, 2004.

Vitalik Buterin.
The stateless client concept.
ethresear.ch, 2017.
`https://ethresear.ch/t/the-stateless-client-concept/172`.

Vitalik Buterin.
Using polynomial commitments to replace state roots.
`https://ethresear.ch/t/using-polynomial-commitments-to-replace-state-roots/7095`, 2020.

References ii

Jan Camenisch, Maria Dubovitskaya, Kristiyan Haralambiev, and Markulf Kohlweiss.
Composable and Modular Anonymous Credentials: Definitions and Practical Constructions.
In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology – ASIACRYPT 2015*, pages 262–288, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

Dario Catalano and Dario Fiore.
Vector Commitments and Their Applications.
In Kaoru Kurosawa and Goichiro Hanaoka, editors, *Public-Key Cryptography – PKC 2013*, pages 55–72, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

Alexander Chepurnoy, Charalampos Papamanthou, and Yupeng Zhang.
Edrax: A Cryptocurrency with Stateless Transaction Validation.
Cryptology ePrint Archive, Report 2018/968, 2018.
`https://eprint.iacr.org/2018/968`.

Dankrad Feist and Dmitry Khovratovich.
Fast amortized Kate proofs, 2020.
https://github.com/khovratovich/Kate.

Sergey Gorbunov, Leonid Reyzin, Hoeteck Wee, and Zhenfei Zhang.
Pointproofs: Aggregating Proofs for Multiple Vector Commitments.
Cryptology ePrint Archive, Report 2020/419, 2020.
https://eprint.iacr.org/2020/419.

Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg.
Constant-Size Commitments to Polynomials and Their Applications.
In Masayuki Abe, editor, *ASIACRYPT '10*, pages 177–194, Berlin, Heidelberg, 2010.
Springer Berlin Heidelberg.

References iv

- Russell W. F. Lai and Giulio Malavolta.
  Subvector Commitments with Application to Succinct Arguments.
  In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019*, pages 530–560, Cham, 2019. Springer International Publishing.
- Andrew Miller.
  Storing UTXOs in a balanced Merkle tree (zero-trust nodes with O(1)-storage).
  BitcoinTalk Forums, 2012.
  `https://bitcointalk.org/index.php?topic=101734.msg1117428`.
- Leonid Reyzin, Dmitry Meshkov, Alexander Chepurnoy, and Sasha Ivanov.
  Improving Authenticated Dynamic Dictionaries, with Applications to Cryptocurrencies.
  In Aggelos Kiayias, editor, *Financial Cryptography and Data Security*, pages 376–392, Cham, 2017. Springer International Publishing.

- Alin Tomescu, Ittai Abraham, Vitalik Buterin, Justin Drake, Dankrad Feist, and Dmitry Khovratovich.
  Aggregatable Subvector Commitments for Stateless Cryptocurrencies.
  Cryptology ePrint Archive, Report 2020/527, 2020.
  `https://eprint.iacr.org/2020/527`.
- Alin Tomescu, Robert Chen, Yiming Zheng, Ittai Abraham, Benny Pinkas, Guy Golan Gueta, and Srinivas Devadas.
  Towards Scalable Threshold Cryptosystems.
  In *2020 IEEE Symposium on Security and Privacy (SP)*, May 2020.
- Peter Todd.
  Making utxo set growth irrelevant with low-latency delayed txo commitments, 2016.

  `https://petertodd.org/2016/delayed-txo-commitments`.

Alin Tomescu.
How to Keep a Secret and Share a Public Key (Using Polynomial Commitments).
PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2020.

Madars Virza.
On Deploying Succinct Zero-Knowledge Proofs.
PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2017.

Joachim von zur Gathen and Jurgen Gerhard.
Fast polynomial evaluation and interpolation.
In *Modern Computer Algebra*, chapter 10, pages 295–310. Cambridge University Press, New York, NY, USA, 3rd edition, 2013.