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

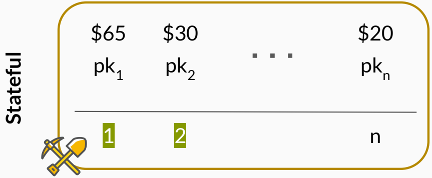
Dmitry Khovratovich[2]
@Khovr

[1]VMware Research, [2]Ethereum Foundation

September 14th, 2020

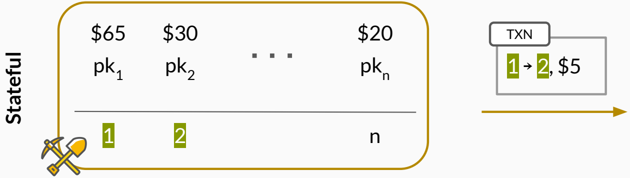
Motivation

Miners rely on state to validate transactions and blocks.



Motivation

Miners rely on state to validate transactions and blocks.



Motivation

Miners rely on state to validate transactions and blocks.

Motivation

Miners rely on state to validate transactions and blocks.

Validation state can be **very large**:

Motivation

Miners rely on state to validate transactions and blocks.

Validation state can be **very large**:

- Hundreds of GBs in Ethereum

Motivation

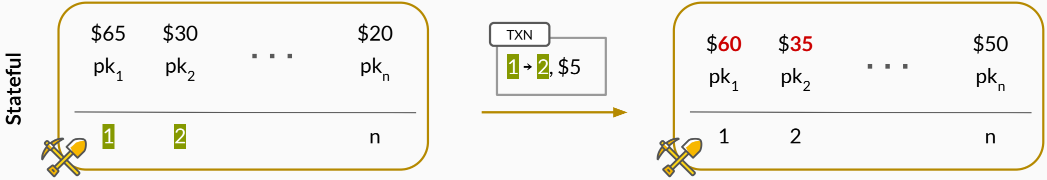
Miners rely on state to validate transactions and blocks.

Validation state can be **very large**:

- Hundreds of GBs in Ethereum
- GBs in Bitcoin

Motivation

Miners rely on state to validate transactions and blocks.



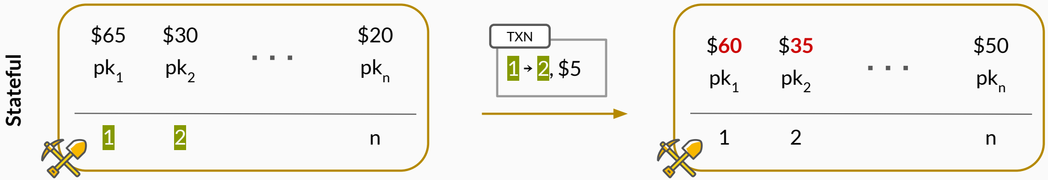
Validation state can be **very large**:

- Hundreds of GBs in Ethereum
- GBs in Bitcoin

This poses scalability challenges:

Motivation

Miners rely on state to validate transactions and blocks.



Validation state can be **very large**:

- Hundreds of GBs in Ethereum
- GBs in Bitcoin

This poses scalability challenges:

- Consensus via sharding

Motivation

Miners rely on state to validate transactions and blocks.

Validation state can be **very large**:

- Hundreds of GBs in Ethereum
- GBs in Bitcoin

This poses scalability challenges:

- Consensus via sharding
- Barrier to entry for P2P nodes

Motivation

Miners rely on state to validate transactions and blocks.

Validation state can be **very large**:

- Hundreds of GBs in Ethereum
- GBs in Bitcoin

This poses scalability challenges:

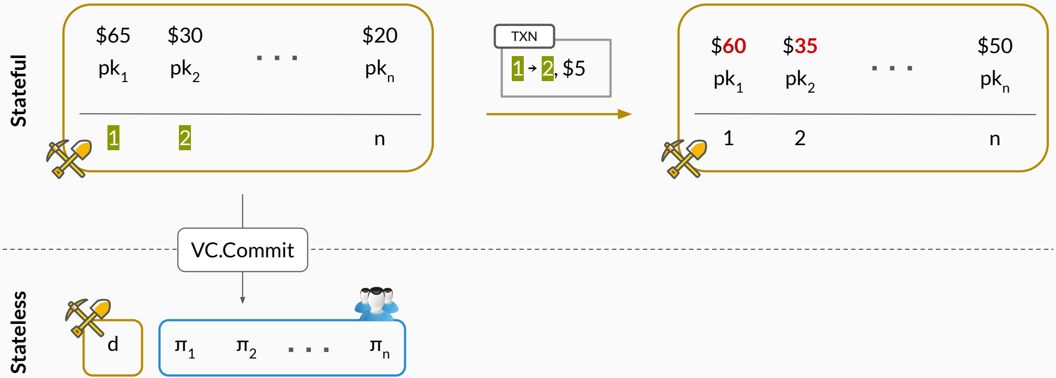
- Consensus via sharding
- Barrier to entry for P2P nodes
- DoS attacks

Previous Work Outsources State via *Authenticated Data Structures*

[CPZ18] proposed stateless validation using Vector Commitments (VCs)!

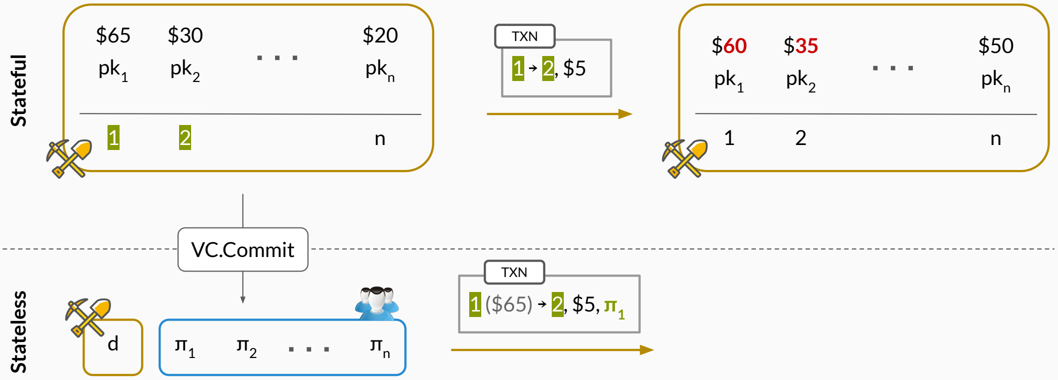
Previous Work Outsources State via *Authenticated Data Structures*

[CPZ18] proposed stateless validation using Vector Commitments (VCs)!



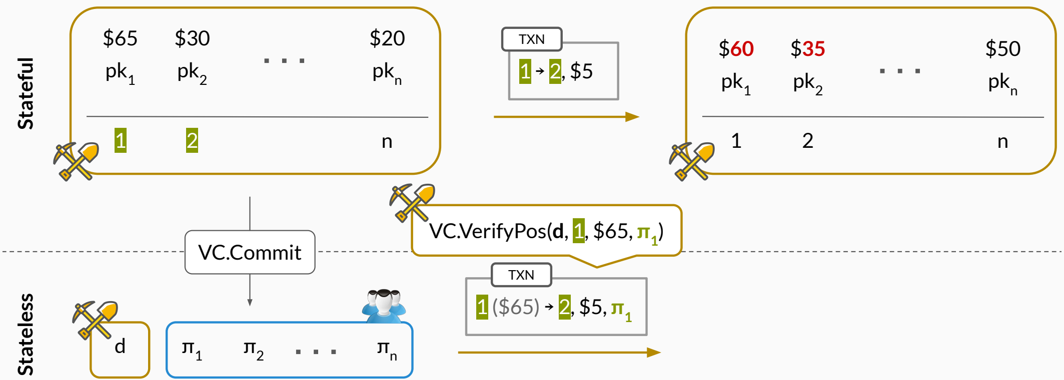
Previous Work Outsources State via *Authenticated Data Structures*

[CPZ18] proposed stateless validation using Vector Commitments (VCs)!



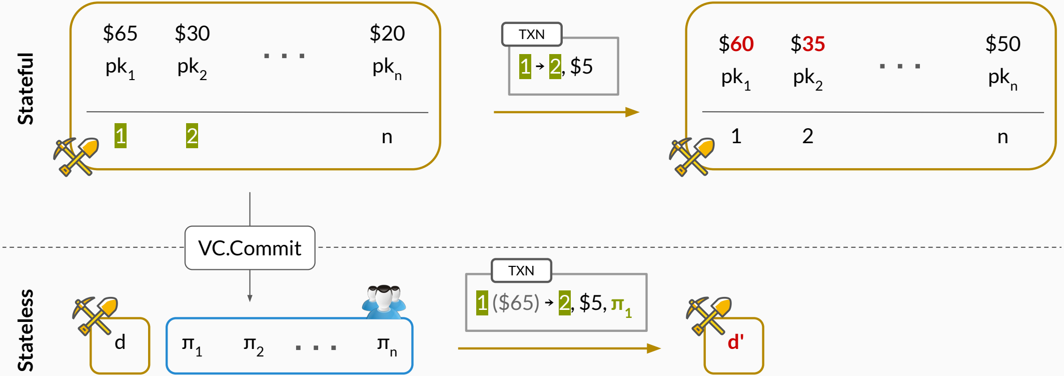
Previous Work Outsources State via *Authenticated Data Structures*

[CPZ18] proposed stateless validation using Vector Commitments (VCs)!



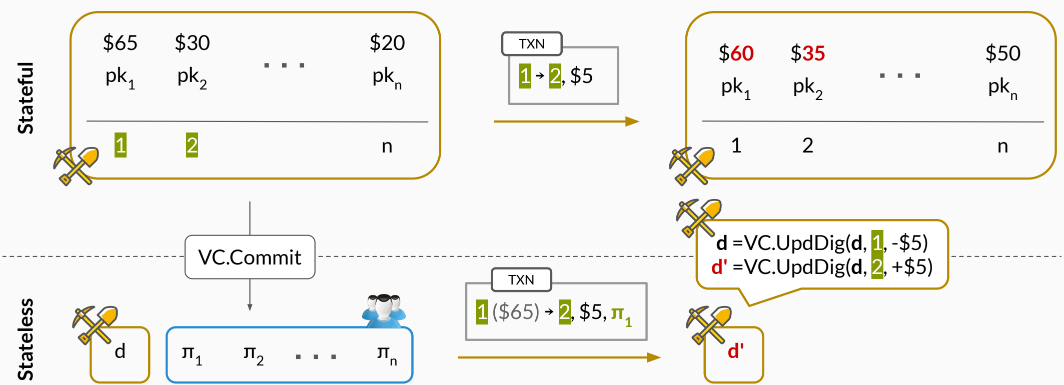
Previous Work Outsources State via *Authenticated Data Structures*

[CPZ18] proposed stateless validation using Vector Commitments (VCs)!



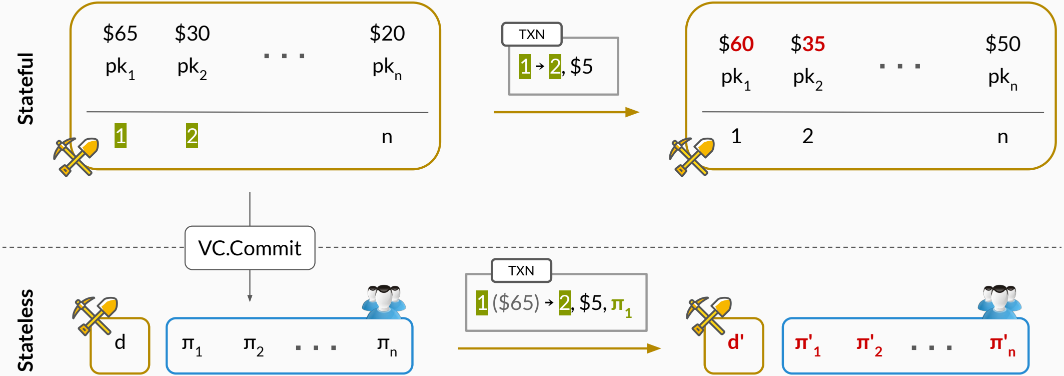
Previous Work Outsources State via *Authenticated Data Structures*

[CPZ18] proposed stateless validation using Vector Commitments (VCs)!



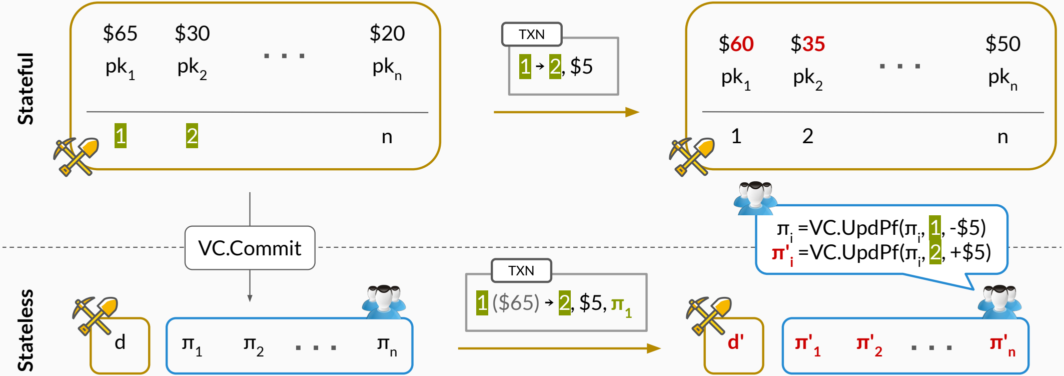
Previous Work Outsources State via *Authenticated Data Structures*

[CPZ18] proposed stateless validation using Vector Commitments (VCs)!



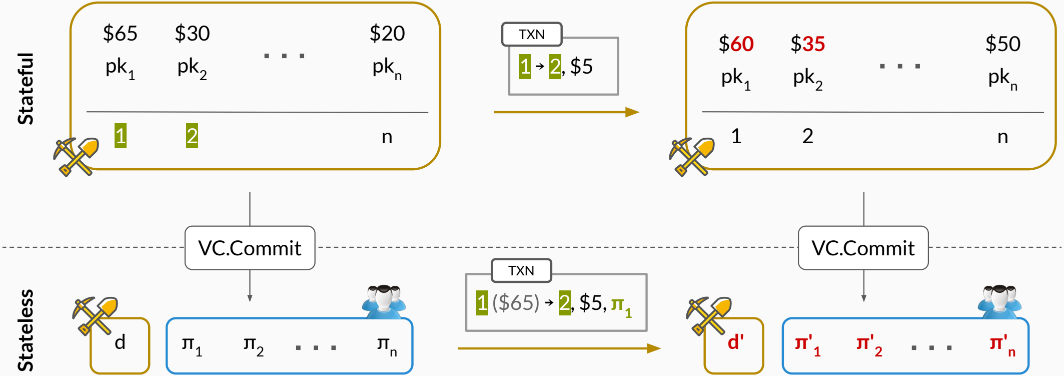
Previous Work Outsources State via *Authenticated Data Structures*

[CPZ18] proposed stateless validation using Vector Commitments (VCs)!



Previous Work Outsources State via *Authenticated Data Structures*

[CPZ18] proposed stateless validation using Vector Commitments (VCs)!



Requirements of VC Scheme for Stateless Validation

Requirements of VC Scheme for Stateless Validation

- Updatable proofs and digest after change (j, δ_j)

Requirements of VC Scheme for Stateless Validation

- Updatable proofs and digest after change (j, δ_j)
  - Only require a *static* **update key** upk_j

Requirements of VC Scheme for Stateless Validation

- Updatable proofs and digest after change (j, δ_j)
  - Only require a *static* **update key** upk_j
  - Some schemes require *dynamic* **update hints**, e.g., the proof π_j

Requirements of VC Scheme for Stateless Validation

- Updatable proofs and digest after change (j, δ_j)
  - Only require a *static* **update key** upk_j
  - Some schemes require *dynamic* **update hints**, e.g., the proof π_j
  - *Problematic:* User i must include not just π_i in the TXN, but also π_j

Requirements of VC Scheme for Stateless Validation

- Updatable proofs and digest after change (j, δ_j)
  - Only require a *static* **update key** upk_j
  - Some schemes require *dynamic* **update hints**, e.g., the proof π_j
  - *Problematic:* User i must include not just π_i in the TXN, but also π_j
- Aggregate many proofs $(\pi_i)_{i \in I}$ into one small **subvector proof** π_I

Requirements of VC Scheme for Stateless Validation

- Updatable proofs and digest after change (j, δ_j)
    - Only require a *static* **update key** upk_j
    - Some schemes require *dynamic* **update hints**, e.g., the proof π_j
    - *Problematic:* User i must include not just π_i in the TXN, but also π_j
- Aggregate many proofs $(\pi_i)_{i \in I}$ into one small **subvector proof** π_I
- Pre-compute all n proofs fast!

Requirements of VC Scheme for Stateless Validation

- Updatable proofs and digest after change (j, δ_j)
  - Only require a *static* **update key** upk_j
  - Some schemes require *dynamic* **update hints**, e.g., the proof π_j
  - *Problematic:* User i must include not just π_i in the TXN, but also π_j
- Aggregate many proofs $(\pi_i)_{i \in I}$ into one small **subvector proof** π_I
- Pre-compute all n proofs fast!
  - Useful for *proof-serving nodes.*

Requirements of VC Scheme for Stateless Validation

- Updatable proofs and digest after change (j, δ_j)
  - Only require a *static* **update key** upk_j
  - Some schemes require *dynamic* **update hints**, e.g., the proof π_j
  - *Problematic:* User i must include not just π_i in the TXN, but also π_j
- Aggregate many proofs $(\pi_i)_{i \in I}$ into one small **subvector proof** π_I
- Pre-compute all n proofs fast!
  - Useful for *proof-serving nodes.*
- Concrete efficiency!

Our Contribution: Aggregatable Subvector Commitments (aSVCs)

Our Contribution: Aggregatable Subvector Commitments (aSVCs)

Table 1: Asymptotic comparison to previous (aS)VCs. n is the size of the vector $\vec{v}$.

(aS)VC scheme	Public parameters	Proof size	Update key size	Digest update	Aggr. b proofs	Prove all

Our Contribution: Aggregatable Subvector Commitments (aSVCs)

Table 1: Asymptotic comparison to previous (aS)VCs. n is the size of the vector $\vec{v}$.

(aS)VC scheme	Public parameters	Proof size	Update key size	Digest update	Aggr. b proofs	Prove all
Merkle trees [Mer88]	1	$\log n$	✗	✗	✗	n

Our Contribution: Aggregatable Subvector Commitments (aSVCs)

Table 1: Asymptotic comparison to previous (aS)VCs. n is the size of the vector $\vec{v}$.

(aS)VC scheme	Public parameters	Proof size	Update key size	Digest update	Aggr. b proofs	Prove all
Merkle trees [Mer88]	1	$\log n$	✗	✗	✗	n
CDHK [CDHK15]	n	1	n	✔	✗	n^2
CPZ [CPZ18]	n	$\log n$	$\log n$	✔	✗	$n \log n$
TCZ [TCZ+20, Tom20]	n	$\log n$	$\log n$	✔	✗	$n \log n$
Pointproofs [GRWZ20]	n	1	n	✔	$b_{\mathbb{G}}$	n^2

Our Contribution: Aggregatable Subvector Commitments (aSVCs)

Table 1: Asymptotic comparison to previous (aS)VCs. n is the size of the vector $\vec{v}$.

(aS)VC scheme	Public parameters	Proof size	Update key size	Digest update	Aggr. b proofs	Prove all
Merkle trees [Mer88]	1	$\log n$	✗	✗	✗	n
CDHK [CDHK15]	n	1	n	✔	✗	n^2
CPZ [CPZ18]	n	$\log n$	$\log n$	✔	✗	$n \log n$
TCZ [TCZ⁺20, Tom20]	n	$\log n$	$\log n$	✔	✗	$n \log n$
Pointproofs [GRWZ20]	n	1	n	✔	$b_{\mathbb{G}}$	n^2
BBF [BBF19]	1	$1_{\mathbb{G}_?}$	✗	✗	$b \log n_{\mathbb{G}_?}$	$n \log n_{\mathbb{G}_?}$
CFG_1 [CFG⁺20]	1	$1_{\mathbb{G}_?}$	✗	✗	$b \log b \log n_{\mathbb{G}_?}$	$n \log^2 n_{\mathbb{G}_?}$

Our Contribution: Aggregatable Subvector Commitments (aSVCs)

Table 1: Asymptotic comparison to previous (aS)VCs. n is the size of the vector $\vec{v}$.

(aS)VC scheme	Public parameters	Proof size	Update key size	Digest update	Aggr. b proofs	Prove all
Merkle trees [Mer88]	1	$\log n$	✗	✗	✗	n
CDHK [CDHK15]	n	1	n	✔	✗	n^2
CPZ [CPZ18]	n	$\log n$	$\log n$	✔	✗	$n \log n$
TCZ [TCZ+20, Tom20]	n	$\log n$	$\log n$	✔	✗	$n \log n$
Pointproofs [GRWZ20]	n	1	n	✔	$b_{\mathbb{G}}$	n^2
BBF [BBF19]	1	$1_{\mathbb{G}_?}$	✗	✗	$b \log n_{\mathbb{G}_?}$	$n \log n_{\mathbb{G}_?}$
CFG_1 [CFG+20]	1	$1_{\mathbb{G}_?}$	✗	✗	$b \log b \log n_{\mathbb{G}_?}$	$n \log^2 n_{\mathbb{G}_?}$
CFG_2 [CF13, LM19, CFG+20]	1	$1_{\mathbb{G}_?}$	$1_{\mathbb{G}_?}$	✔	$b \log^2 b_{\mathbb{G}_?}$	$n \log^2 n_{\mathbb{G}_?}$

Our Contribution: Aggregatable Subvector Commitments (aSVCs)

Table 1: Asymptotic comparison to previous (aS)VCs. n is the size of the vector $\vec{v}$.

(aS)VC scheme	Public parameters	Proof size	Update key size	Digest update	Aggr. b proofs	Prove all
Merkle trees [Mer88]	1	$\log n$	✗	✗	✗	n
CDHK [CDHK15]	n	1	n	✔	✗	n^2
CPZ [CPZ18]	n	$\log n$	$\log n$	✔	✗	$n \log n$
TCZ [TCZ+20, Tom20]	n	$\log n$	$\log n$	✔	✗	$n \log n$
Pointproofs [GRWZ20]	n	1	n	✔	$b_{\mathbb{G}}$	n^2
BBF [BBF19]	1	$1_{\mathbb{G}_?}$	✗	✗	$b \log n_{\mathbb{G}_?}$	$n \log n_{\mathbb{G}_?}$
CFG_1 [CFG+20]	1	$1_{\mathbb{G}_?}$	✗	✗	$b \log b \log n_{\mathbb{G}_?}$	$n \log^2 n_{\mathbb{G}_?}$
CFG_2 [CF13, LM19, CFG+20]	1	$1_{\mathbb{G}_?}$	$1_{\mathbb{G}_?}$	✔	$b \log^2 b_{\mathbb{G}_?}$	$n \log^2 n_{\mathbb{G}_?}$
Our aSVC	n	1	1	✔	$b \lg^2 b_{\mathbb{F}} + b_{\mathbb{G}}$	$n \log n$

Some limitations

Unlike schemes based on hidden-order groups [CF13, LM19, BBF19, CFG⁺20], we have:

Some limitations

Unlike schemes based on hidden-order groups [CF13, LM19, BBF19, CFG+20], we have:

- Trusted setup

Some limitations

Unlike schemes based on hidden-order groups [CF13, LM19, BBF19, CFG+20], we have:

- Trusted setup
- No *incremental* aggregation & no dis-aggregation [CFG+20]

Some limitations

Unlike schemes based on hidden-order groups [CF13, LM19, BBF19, CFG+20], we have:

- Trusted setup
- No *incremental* aggregation & no dis-aggregation [CFG+20]
- No space-time trade-off for proof pre-computation [BBF19, CFG+20]

Background

KZG Constant-sized Polynomial Commitments [KZG10]

KZG Constant-sized Polynomial Commitments [KZG10]

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

Committing: Given $\phi \in \mathbb{F}_p[X]$

KZG Constant-sized Polynomial Commitments [KZG10]

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

Committing: Given $\phi \in \mathbb{F}_p[X]$, where $\phi(X) = \sum_{i=0}^{n} \phi_i X^i = \langle \phi_0, \phi_1, \dots, \phi_n \rangle$

KZG Constant-sized Polynomial Commitments [KZG10]

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

Committing: Given $\phi \in \mathbb{F}_p[X]$, where $\phi(X) = \sum_{i=0}^{n} \phi_i X^i = \langle \phi_0, \phi_1, \ldots, \phi_n \rangle$ 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 $\tau \in \mathbb{F}_p$ is unknown.

Committing: Given $\phi \in \mathbb{F}_p[X]$, where $\phi(X) = \sum_{i=0}^{n} \phi_i X^i = \langle \phi_0, \phi_1, \ldots, \phi_n \rangle$ of degree $\le n$:

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

$$\tag{2}$$

KZG Constant-sized Polynomial Commitments [KZG10]

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

Committing: Given $\phi \in \mathbb{F}_p[X]$, where $\phi(X) = \sum_{i=0}^{n} \phi_i X^i = \langle \phi_0, \phi_1, \dots, \phi_n \rangle$ of degree $\le n$:

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

$$= \left(g^{\tau^n}\right)^{\phi_n} \left(g^{\tau^{n-1}}\right)^{\phi_{n-1}} \dots (g^{\tau})^{\phi_1} (g)^{\phi_0} \tag{2}$$

KZG Constant-sized Polynomial Commitments [KZG10]

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

Committing: Given $\phi \in \mathbb{F}_p[X]$, where $\phi(X) = \sum_{i=0}^{n} \phi_i X^i = \langle \phi_0, \phi_1, \ldots, \phi_n \rangle$ of degree $\le n$:

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

$$= \left(g^{\tau^n}\right)^{\phi_n} \left(g^{\tau^{n-1}}\right)^{\phi_{n-1}} \ldots (g^{\tau})^{\phi_1} (g)^{\phi_0} \tag{2}$$

Homomorphism: For all field elements a, b and polynomials $\phi(X), \psi(X)$, we have:

KZG Constant-sized Polynomial Commitments [KZG10]

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

Committing: Given $\phi \in \mathbb{F}_p[X]$, where $\phi(X) = \sum_{i=0}^{n} \phi_i X^i = \langle \phi_0, \phi_1, \dots, \phi_n \rangle$ of degree $\le n$:

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

$$= \left(g^{\tau^n}\right)^{\phi_n} \left(g^{\tau^{n-1}}\right)^{\phi_{n-1}} \dots (g^{\tau})^{\phi_1} (g)^{\phi_0} \tag{2}$$

Homomorphism: For all field elements a, b and polynomials $\phi(X), \psi(X)$, we have:

$$c(a \cdot \phi + b \cdot \psi) = g^{a\cdot\phi(\tau)+b\cdot\psi(\tau)} \tag{3}$$

(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 $\tau \in \mathbb{F}_p$ is unknown.

Committing: Given $\phi \in \mathbb{F}_p[X]$, where $\phi(X) = \sum_{i=0}^{n} \phi_i X^i = \langle \phi_0, \phi_1, \ldots, \phi_n \rangle$ of degree $\le n$:

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

$$= \left(g^{\tau^n}\right)^{\phi_n} \left(g^{\tau^{n-1}}\right)^{\phi_{n-1}} \ldots (g^{\tau})^{\phi_1} (g)^{\phi_0} \tag{2}$$

Homomorphism: For all field elements a, b and polynomials $\phi(X), \psi(X)$, we have:

$$c(a \cdot \phi + b \cdot \psi) = g^{a \cdot \phi(\tau) + b \cdot \psi(\tau)} \tag{3}$$

$$= \left(g^{\phi(\tau)}\right)^{a} \left(g^{\psi(\tau)}\right)^{b} \tag{4}$$

$$\tag{5}$$

KZG Constant-sized Polynomial Commitments [KZG10]

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

Committing: Given $\phi \in \mathbb{F}_p[X]$, where $\phi(X) = \sum_{i=0}^{n} \phi_i X^i = \langle \phi_0, \phi_1, \ldots, \phi_n \rangle$ of degree $\le n$:

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

$$= \left(g^{\tau^n}\right)^{\phi_n} \left(g^{\tau^{n-1}}\right)^{\phi_{n-1}} \ldots (g^{\tau})^{\phi_1} (g)^{\phi_0} \tag{2}$$

Homomorphism: For all field elements a, b and polynomials $\phi(X), \psi(X)$, we have:

$$c(a \cdot \phi + b \cdot \psi) = g^{a\cdot\phi(\tau)+b\cdot\psi(\tau)} \tag{3}$$

$$= \left(g^{\phi(\tau)}\right)^a \left(g^{\psi(\tau)}\right)^b \tag{4}$$

$$= c(\phi)^a c(\psi)^b \tag{5}$$

VCs from Lagrange Basis Polynomials [CDHK15]

VCs from Lagrange Basis Polynomials [CDHK15]

Represent vector $\vec{v}$ with a polynomial ϕ s.t. $\phi(i) = v_i$:

VCs from Lagrange Basis Polynomials [CDHK15]

Represent vector $\vec{v}$ with a polynomial ϕ s.t. $\phi(i) = v_i$:

$$\phi(X) = \sum_{i=0}^{n-1} v_i \cdot L_i(X) \tag{6}$$

(7)

VCs from Lagrange Basis Polynomials [CDHK15]

Represent vector $\vec{v}$ with a polynomial ϕ s.t. $\phi(i) = v_i$:

$$\phi(X) = \sum_{i=0}^{n-1} v_i \cdot L_i(X) \tag{6}$$

$$L_i(X) = \prod_{\substack{j \in [0,n) \\ j \neq i}} \frac{X - j}{i - j} \tag{7}$$

VCs from Lagrange Basis Polynomials [CDHK15]

Represent vector $\vec{v}$ with a polynomial ϕ s.t. $\phi(i) = v_i$:

$$\phi(X) = \sum_{i=0}^{n-1} v_i \cdot L_i(X) \tag{6}$$

$$L_i(X) = \prod_{\substack{j \in [0,n) \\ j \neq i}} \frac{X - j}{i - j} \tag{7}$$

Applying the KZG homomorphism to Equation (6):

VCs from Lagrange Basis Polynomials [CDHK15]

Represent vector $\vec{v}$ with a polynomial ϕ s.t. $\phi(i) = v_i$:

$$\phi(X) = \sum_{i=0}^{n-1} v_i \cdot L_i(X) \tag{6}$$

$$L_i(X) = \prod_{\substack{j \in [0,n) \\ j \neq i}} \frac{X - j}{i - j} \tag{7}$$

Applying the KZG homomorphism to Equation (6):

$$c(\phi) = \prod_{i=0}^{n-1} c(L_i)^{v_i} \tag{8}$$

VCs from Lagrange Basis Polynomials [CDHK15]

Represent vector $\vec{v}$ with a polynomial ϕ s.t. $\phi(i) = v_i$:

$$\phi(X) = \sum_{i=0}^{n-1} v_i \cdot L_i(X) \tag{6}$$

$$L_i(X) = \prod_{\substack{j \in [0,n) \\ j \neq i}} \frac{X - j}{i - j} \tag{7}$$

Applying the KZG homomorphism to Equation (6):

$$c(\phi) = \prod_{i=0}^{n-1} c(L_i)^{v_i} \tag{8}$$

Note: Public parameters include commitments $c(L_i)$.

VCs from Lagrange Basis Polynomials [CDHK15]

Represent vector $\vec{v}$ with a polynomial ϕ s.t. $\phi(i) = v_i$:

$$\phi(X) = \sum_{i=0}^{n-1} v_i \cdot L_i(X) \tag{6}$$

$$L_i(X) = \prod_{\substack{j\in[0,n)\\ j\neq i}} \frac{X - j}{i - j} \tag{7}$$

Applying the KZG homomorphism to Equation (6):

$$c(\phi) = \prod_{i=0}^{n-1} c(L_i)^{v_i} \tag{8}$$

Note: Public parameters include commitments $c(L_i)$. *Can derive from* g^{τ^i}*'s.*

Updating the Digest = Updating Polynomial & Updating Commitment

Updating the Digest = Updating Polynomial & Updating Commitment

Assume v_i changed to $v_i + \delta_i$.

Updating the Digest = Updating Polynomial & Updating Commitment

Assume v_i changed to $v_i + \delta_i$. Old polynomial was:

$$\phi(X) = \sum_{i=0}^{n-1} v_i \cdot L_i(X) \tag{9}$$

Updating the Digest = Updating Polynomial & Updating Commitment

Assume v_i changed to $v_i + \delta_i$. Old polynomial was:

$$\phi(X) = \sum_{i=0}^{n-1} v_i \cdot L_i(X) \tag{9}$$

Updated polynomial will be:

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

Updating the Digest = Updating Polynomial & Updating Commitment

Assume v_i changed to $v_i + \delta_i$. Old polynomial was:

$$\phi(X) = \sum_{i=0}^{n-1} v_i \cdot L_i(X) \tag{9}$$

Updated polynomial will be:

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

Updated commitment will be:

$$c(\phi') = c(\phi) \cdot c\left(L_i\right)^{\delta_i} \tag{11}$$

Updating the Digest = Updating Polynomial & Updating Commitment

Assume v_i changed to $v_i + \delta_i$. Old polynomial was:

$$\phi(X) = \sum_{i=0}^{n-1} v_i \cdot L_i(X) \tag{9}$$

Updated polynomial will be:

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

Updated commitment will be:

$$c(\phi') = c(\phi) \cdot c\left(L_i\right)^{\delta_i} \tag{11}$$

Thus, for our purposes, each upk_i will include $c\left(L_i\right)$.

Proof π_i for v_i Refresher

Proof π_i for v_i Refresher

A proof π_i for v_i must convince that $\phi(i) = v_i$

Proof π_i for v_i Refresher

A proof π_i for v_i must convince that $\phi(i) = v_i \Leftrightarrow \phi \bmod (X - i) = v_i$.

Proof π_i for v_i Refresher

A proof π_i for v_i must convince that $\phi(i) = v_i \Leftrightarrow \phi \bmod (X - i) = v_i$.

$$q_i(X) = \frac{\phi(X) - v_i}{X - i} \tag{12}$$

$$\tag{13}$$

Proof π_i for v_i Refresher

A proof π_i for v_i must convince that $\phi(i) = v_i \Leftrightarrow \phi \bmod (X - i) = v_i$.

$$q_i(X) = \frac{\phi(X) - v_i}{X - i} \tag{12}$$

$$\pi_i = c\left(q_i\right) = g^{q_i(\tau)} \tag{13}$$

Proof π_i for v_i Refresher

A proof π_i for v_i must convince that $\phi(i) = v_i \Leftrightarrow \phi \bmod (X - i) = v_i$.

$$q_i(X) = \frac{\phi(X) - v_i}{X - i} \tag{12}$$

$$\pi_i = c\left(q_i\right) = g^{q_i(\tau)} \tag{13}$$

To verify, use pairings:

Proof π_i for v_i Refresher

A proof π_i for v_i must convince that $\phi(i) = v_i \Leftrightarrow \phi \bmod (X - i) = v_i$.

$$q_i(X) = \frac{\phi(X) - v_i}{X - i} \tag{12}$$

$$\pi_i = c\left(q_i\right) = g^{q_i(\tau)} \tag{13}$$

To verify, use pairings:

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

$$\tag{15}$$

Proof π_i for v_i Refresher

A proof π_i for v_i must convince that $\phi(i) = v_i \Leftrightarrow \phi \bmod (X - i) = v_i$.

$$q_i(X) = \frac{\phi(X) - v_i}{X - i} \tag{12}$$

$$\pi_i = c\left(q_i\right) = g^{q_i(\tau)} \tag{13}$$

To verify, use pairings:

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

$$\phi(\tau) - v_i = q_i(\tau)(\tau - i) \tag{15}$$

Our Techniques

Outline

Background

Our Techniques

Updating Proofs

Aggregating Proofs into Subvector Proofs

Conclusion

New Technique: Updating Proof $\pi_i = c(q_i)$ After Change (i, δ_i)

New Technique: Updating Proof $\pi_i = c(q_i)$ After Change (i, δ_i)

We know $\pi_i' = c\left(q_i'\right)$, where:

New Technique: Updating Proof $\pi_i = c(q_i)$ After Change (i, δ_i)

We know $\pi_i' = c\left(q_i'\right)$, where:

$$q_i'(X) = \frac{\phi'(X) - (v_i + \delta_i)}{X - i} \tag{16}$$

(17)

(18)

(19)

New Technique: Updating Proof $\pi_i = c(q_i)$ After Change (i, δ_i)

We know $\pi_i' = c\left(q_i'\right)$, where:

$$q_i'(X) = \frac{\phi'(X) - (v_i + \delta_i)}{X - i} \tag{16}$$

$$= \frac{\left(\phi(X) + \delta_i L_i(X)\right) - v_i - \delta_i}{X - i} \tag{17}$$

(18)

(19)

New Technique: Updating Proof $\pi_i = c(q_i)$ After Change (i, δ_i)

We know $\pi'_i = c\left(q'_i\right)$, where:

$$
\begin{aligned}
q'_i(X) &= \frac{\phi'(X) - (v_i + \delta_i)}{X - i} && (16)\\
&= \frac{\left(\phi(X) + \delta_i L_i(X)\right) - v_i - \delta_i}{X - i} && (17)\\
&= \frac{\phi(X) - v_i}{X - i} - \frac{\delta_i (L_i(X) - 1)}{X - i} && (18)\\
& && (19)
\end{aligned}
$$

New Technique: Updating Proof $\pi_i = c(q_i)$ After Change (i, δ_i)

We know $\pi_i' = c\left(q_i'\right)$, where:

$$q_i'(X) = \frac{\phi'(X) - (v_i + \delta_i)}{X - i} \tag{16}$$

$$= \frac{\left(\phi(X) + \delta_i L_i(X)\right) - v_i - \delta_i}{X - i} \tag{17}$$

$$= \frac{\phi(X) - v_i}{X - i} - \frac{\delta_i(L_i(X) - 1)}{X - i} \tag{18}$$

$$q_i'(X) = q_i(X) + \delta_i \left(\frac{L_i(X) - 1}{X - i}\right) \tag{19}$$

New Technique: Updating Proof $\pi_i = c(q_i)$ After Change (i, δ_i)

We know $\pi_i' = c\left(q_i'\right)$, where:

$$\begin{aligned} q_i'(X) &= \frac{\phi'(X) - (v_i + \delta_i)}{X - i} && (16) \\ &= \frac{\left(\phi(X) + \delta_i L_i(X)\right) - v_i - \delta_i}{X - i} && (17) \\ &= \frac{\phi(X) - v_i}{X - i} - \frac{\delta_i (L_i(X) - 1)}{X - i} && (18) \\ q_i'(X) &= q_i(X) + \delta_i \left(\frac{L_i(X) - 1}{X - i} \right) && (19) \end{aligned}$$

Applying KZG homomorphism, it follows that:

New Technique: Updating Proof $\pi_i = c(q_i)$ After Change (i, δ_i)

We know $\pi_i' = c\left(q_i'\right)$, where:

$$q_i'(X) = \frac{\phi'(X) - (v_i + \delta_i)}{X - i} \tag{16}$$

$$= \frac{\left(\phi(X) + \delta_i L_i(X)\right) - v_i - \delta_i}{X - i} \tag{17}$$

$$= \frac{\phi(X) - v_i}{X - i} - \frac{\delta_i(L_i(X) - 1)}{X - i} \tag{18}$$

$$q_i'(X) = q_i(X) + \delta_i\left(\frac{L_i(X) - 1}{X - i}\right) \tag{19}$$

Applying KZG homomorphism, it follows that:

$$\pi_i' = c\left(q_i'\right) = c\left(q_i\right) \cdot c\left(\frac{L_i(X) - 1}{X - i}\right)^{\delta_i} \tag{20}$$

New Technique: Updating Proof $\pi_i = c(q_i)$ After Change (i, δ_i)

We know $\pi_i' = c\left(q_i'\right)$, where:

$$q_i'(X) = \frac{\phi'(X) - (v_i + \delta_i)}{X - i} \tag{16}$$

$$= \frac{\big(\phi(X) + \delta_i L_i(X)\big) - v_i - \delta_i}{X - i} \tag{17}$$

$$= \frac{\phi(X) - v_i}{X - i} - \frac{\delta_i (L_i(X) - 1)}{X - i} \tag{18}$$

$$q_i'(X) = q_i(X) + \delta_i \left(\frac{L_i(X) - 1}{X - i}\right) \tag{19}$$

Applying KZG homomorphism, it follows that:

$$\pi_i' = c\left(q_i'\right) = c\left(q_i\right) \cdot c\left(\frac{L_i(X) - 1}{X - i}\right)^{\delta_i} \tag{20}$$

Thus, each upk_i must include $c\left(\frac{L_i(X)-1}{X-i}\right)$.

New Technique: Updating Proof $\pi_i = c(q_i)$ After Change (i, δ_i)

We know $\pi_i' = c\left(q_i'\right)$, where:

$$q_i'(X) = \frac{\phi'(X) - (v_i + \delta_i)}{X - i} \tag{16}$$

$$= \frac{\left(\phi(X) + \delta_i L_i(X)\right) - v_i - \delta_i}{X - i} \tag{17}$$

$$= \frac{\phi(X) - v_i}{X - i} - \frac{\delta_i(L_i(X) - 1)}{X - i} \tag{18}$$

$$q_i'(X) = q_i(X) + \delta_i \left(\frac{L_i(X) - 1}{X - i}\right) \tag{19}$$

Applying KZG homomorphism, it follows that:

$$\pi_i' = c\left(q_i'\right) = c\left(q_i\right) \cdot c\left(\frac{L_i(X) - 1}{X - i}\right)^{\delta_i} \tag{20}$$

Thus, each upk_i must include $c\left(\frac{L_i(X)-1}{X-i}\right)$. *Can derive these from g^{τ^i}'s!*

New Technique: Updating Proof $\pi_i = c(q_i)$ After Change (j, δ_j), $j \neq i$

New Technique: Updating Proof $\pi_i = c(q_i)$ After Change (j, δ_j), $j \neq i$

Once again, we know $\pi_i' = c\left(q_i'\right)$, where:

New Technique: Updating Proof $\pi_i = c(q_i)$ After Change (j, δ_j), $j \neq i$

Once again, we know $\pi_i' = c\left(q_i'\right)$, where:

$$q_i'(X) = \frac{\phi'(X) - v_i}{X - i} \tag{21}$$

(22)

(23)

(24)

New Technique: Updating Proof $\pi_i = c(q_i)$ After Change (j, δ_j), $j \neq i$

Once again, we know $\pi_i' = c\left(q_i'\right)$, where:

$$q_i'(X) = \frac{\phi'(X) - v_i}{X - i} \tag{21}$$

$$= \frac{\left(\phi(X) + \delta_j L_j(X)\right) - v_i}{X - i} \tag{22}$$

(23)

(24)

New Technique: Updating Proof $\pi_i = c(q_i)$ After Change (j, δ_j), $j \neq i$

Once again, we know $\pi_i' = c\left(q_i'\right)$, where:

$$q_i'(X) = \frac{\phi'(X) - v_i}{X - i} \tag{21}$$

$$= \frac{\left(\phi(X) + \delta_j L_j(X)\right) - v_i}{X - i} \tag{22}$$

$$= \frac{\phi(X) - v_i}{X - i} - \frac{\delta_j L_j(X)}{X - i} \tag{23}$$

$$\tag{24}$$

New Technique: Updating Proof $\pi_i = c(q_i)$ After Change (j, δ_j), $j \neq i$

Once again, we know $\pi_i' = c\left(q_i'\right)$, where:

$$q_i'(X) = \frac{\phi'(X) - v_i}{X - i} \tag{21}$$

$$= \frac{\left(\phi(X) + \delta_j L_j(X)\right) - v_i}{X - i} \tag{22}$$

$$= \frac{\phi(X) - v_i}{X - i} - \frac{\delta_j L_j(X)}{X - i} \tag{23}$$

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

New Technique: Updating Proof $\pi_i = c(q_i)$ After Change (j, δ_j), $j \neq i$

Once again, we know $\pi_i' = c\left(q_i'\right)$, where:

$$
\begin{aligned}
q_i'(X) &= \frac{\phi'(X) - v_i}{X - i} && (21)\\
&= \frac{\left(\phi(X) + \delta_j L_j(X)\right) - v_i}{X - i} && (22)\\
&= \frac{\phi(X) - v_i}{X - i} - \frac{\delta_j L_j(X)}{X - i} && (23)\\
q_i'(X) &= q_i(X) + \delta_j \left(\frac{L_j(X)}{X - i}\right) && (24)
\end{aligned}
$$

Applying KZG homomorphism, it follows that:

New Technique: Updating Proof $\pi_i = c(q_i)$ After Change (j, δ_j), $j \neq i$

Once again, we know $\pi_i' = c\left(q_i'\right)$, where:

$$q_i'(X) = \frac{\phi'(X) - v_i}{X - i} \tag{21}$$

$$= \frac{\left(\phi(X) + \delta_j L_j(X)\right) - v_i}{X - i} \tag{22}$$

$$= \frac{\phi(X) - v_i}{X - i} - \frac{\delta_j L_j(X)}{X - i} \tag{23}$$

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

Applying KZG homomorphism, it follows that:

$$\pi_i' = c\left(q_i'\right) = c\left(q_i\right) \cdot c\left(\frac{L_j(X)}{X - i}\right)^{\delta_j} \tag{25}$$

New Technique: Updating Proof $\pi_i = c(q_i)$ After Change (j, δ_j), $j \neq i$

Once again, we know $\pi_i' = c\left(q_i'\right)$, where:

$$q_i'(X) = \frac{\phi'(X) - v_i}{X - i} \tag{21}$$

$$= \frac{\left(\phi(X) + \delta_j L_j(X)\right) - v_i}{X - i} \tag{22}$$

$$= \frac{\phi(X) - v_i}{X - i} - \frac{\delta_j L_j(X)}{X - i} \tag{23}$$

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

Applying KZG homomorphism, it follows that:

$$\pi_i' = c\left(q_i'\right) = c\left(q_i\right) \cdot c\left(\frac{L_j(X)}{X - i}\right)^{\delta_j} \tag{25}$$

Problem: To update any π_i after a change to j, need $c\left(\frac{L_j(X)}{X-i}\right)$, $\forall i \neq j$

New Technique: Updating Proof $\pi_i = c(q_i)$ After Change (j, δ_j), $j \neq i$

Once again, we know $\pi_i' = c\left(q_i'\right)$, where:

$$q_i'(X) = \frac{\phi'(X) - v_i}{X - i} \tag{21}$$

$$= \frac{\left(\phi(X) + \delta_j L_j(X)\right) - v_i}{X - i} \tag{22}$$

$$= \frac{\phi(X) - v_i}{X - i} - \frac{\delta_j L_j(X)}{X - i} \tag{23}$$

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

Applying KZG homomorphism, it follows that:

$$\pi_i' = c\left(q_i'\right) = c\left(q_i\right) \cdot c\left(\frac{L_j(X)}{X - i}\right)^{\delta_j} \tag{25}$$

Problem: To update any π_i after a change to j, need $c\left(\frac{L_j(X)}{X-i}\right), \forall i \neq j \Rightarrow O(n)$-sized upk_j.

New Technique: Updating Proof $\pi_i = c(q_i)$ After Change (j, δ_j), $j \neq i$

Once again, we know $\pi_i' = c\left(q_i'\right)$, where:

$$q_i'(X) = \frac{\phi'(X) - v_i}{X - i} \tag{21}$$

$$= \frac{\left(\phi(X) + \delta_j L_j(X)\right) - v_i}{X - i} \tag{22}$$

$$= \frac{\phi(X) - v_i}{X - i} - \frac{\delta_j L_j(X)}{X - i} \tag{23}$$

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

Applying KZG homomorphism, it follows that:

$$\pi_i' = c\left(q_i'\right) = c\left(q_i\right) \cdot c\left(\frac{L_j(X)}{X - i}\right)^{\delta_j} \tag{25}$$

Problem: To update any π_i after a change to j, need $c\left(\frac{L_j(X)}{X-i}\right), \forall i \neq j \Rightarrow O(n)$-sized upk_j.

Solution: Compute $c\left(\frac{L_j(X)}{X-i}\right)$ in $O(1)$ time from information in upk_i and upk_j.

New Technique: Compute $c\left(\frac{L_j(X)}{X-i}\right)$ in $O(1)$ Time

New Technique: Compute $c\left(\frac{L_j(X)}{X-i}\right)$ in $O(1)$ Time

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

New Technique: Compute $c\left(\frac{L_j(X)}{X-i}\right)$ in $O(1)$ Time

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

$$\frac{L_j(X)}{X - i} = \frac{1}{A'(j)} \cdot \frac{A(X)}{(X - j)(X - i)} \tag{26}$$

Next, use **partial fraction decomposition** to rewrite:

New Technique: Compute $c\left(\frac{L_j(X)}{X-i}\right)$ in $O(1)$ Time

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

$$\frac{L_j(X)}{X - i} = \frac{1}{A'(j)} \cdot \frac{A(X)}{(X - j)(X - i)} \tag{26}$$

Next, use **partial fraction decomposition** to rewrite:

$$\frac{A(X)}{(X - i)(X - j)} = \frac{1}{i - j} \cdot \frac{A(X)}{X - i} + \frac{1}{j - i} \cdot \frac{A(X)}{X - j} \tag{27}$$

New Technique: Compute $c\left(\frac{L_j(X)}{X-i}\right)$ in $O(1)$ Time

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

$$\frac{L_j(X)}{X - i} = \frac{1}{A'(j)} \cdot \frac{A(X)}{(X - j)(X - i)} \tag{26}$$

Next, use **partial fraction decomposition** to rewrite:

$$\frac{A(X)}{(X - i)(X - j)} = \frac{1}{i - j} \cdot \frac{A(X)}{X - i} + \frac{1}{j - i} \cdot \frac{A(X)}{X - j} \tag{27}$$

Now, replacing Equation (27) into Equation (26):

$$\frac{L_j(X)}{X - i} = \frac{1}{A'(j)} \cdot \left(\frac{1}{i - j} \cdot \frac{A(X)}{X - i} + \frac{1}{j - i} \cdot \frac{A(X)}{X - j}\right) \tag{28}$$

New Technique: Compute $c\left(\frac{L_j(X)}{X-i}\right)$ in $O(1)$ Time

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

$$\frac{L_j(X)}{X - i} = \frac{1}{A'(j)} \cdot \frac{A(X)}{(X - j)(X - i)} \tag{26}$$

Next, use **partial fraction decomposition** to rewrite:

$$\frac{A(X)}{(X - i)(X - j)} = \frac{1}{i - j} \cdot \frac{A(X)}{X - i} + \frac{1}{j - i} \cdot \frac{A(X)}{X - j} \tag{27}$$

Now, replacing Equation (27) into Equation (26):

$$\frac{L_j(X)}{X - i} = \frac{1}{A'(j)} \cdot \left(\frac{1}{i - j} \cdot \frac{A(X)}{X - i} + \frac{1}{j - i} \cdot \frac{A(X)}{X - j}\right) \tag{28}$$

As a result:

New Technique: Compute $c\left(\frac{L_j(X)}{X-i}\right)$ in $O(1)$ Time

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

$$\frac{L_j(X)}{X - i} = \frac{1}{A'(j)} \cdot \frac{A(X)}{(X - j)(X - i)} \tag{26}$$

Next, use **partial fraction decomposition** to rewrite:

$$\frac{A(X)}{(X - i)(X - j)} = \frac{1}{i - j} \cdot \frac{A(X)}{X - i} + \frac{1}{j - i} \cdot \frac{A(X)}{X - j} \tag{27}$$

Now, replacing Equation (27) into Equation (26):

$$\frac{L_j(X)}{X - i} = \frac{1}{A'(j)} \cdot \left(\frac{1}{i - j} \cdot \frac{A(X)}{X - i} + \frac{1}{j - i} \cdot \frac{A(X)}{X - j}\right) \tag{28}$$

As a result:

$$c\left(\frac{L_j(X)}{X - i}\right) = \left(c\left(\frac{A(X)}{X - i}\right)^{\frac{1}{i-j}} \cdot c\left(\frac{A(X)}{X - j}\right)^{\frac{1}{j-i}}\right)^{\frac{1}{A'(j)}} \tag{29}$$

New Technique: Compute $c\left(\frac{L_j(X)}{X-i}\right)$ in $O(1)$ Time

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

$$\frac{L_j(X)}{X-i} = \frac{1}{A'(j)} \cdot \frac{A(X)}{(X-j)(X-i)} \tag{26}$$

Next, use **partial fraction decomposition** to rewrite:

$$\frac{A(X)}{(X-i)(X-j)} = \frac{1}{i-j} \cdot \frac{A(X)}{X-i} + \frac{1}{j-i} \cdot \frac{A(X)}{X-j} \tag{27}$$

Now, replacing Equation (27) into Equation (26):

$$\frac{L_j(X)}{X-i} = \frac{1}{A'(j)} \cdot \left(\frac{1}{i-j} \cdot \frac{A(X)}{X-i} + \frac{1}{j-i} \cdot \frac{A(X)}{X-j}\right) \tag{28}$$

As a result:

$$c\left(\frac{L_j(X)}{X-i}\right) = \left(c\left(\frac{A(X)}{X-i}\right)^{\frac{1}{i-j}} \cdot c\left(\frac{A(X)}{X-j}\right)^{\frac{1}{j-i}}\right)^{\frac{1}{A'(j)}} \tag{29}$$

Thus, each upk_i must include $c\left(\frac{A(X)}{X-i}\right)$ and $A'(i)$.

New Technique: Compute $c\left(\frac{L_j(X)}{X-i}\right)$ in $O(1)$ Time

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

$$\frac{L_j(X)}{X - i} = \frac{1}{A'(j)} \cdot \frac{A(X)}{(X - j)(X - i)} \tag{26}$$

Next, use **partial fraction decomposition** to rewrite:

$$\frac{A(X)}{(X - i)(X - j)} = \frac{1}{i - j} \cdot \frac{A(X)}{X - i} + \frac{1}{j - i} \cdot \frac{A(X)}{X - j} \tag{27}$$

Now, replacing Equation (27) into Equation (26):

$$\frac{L_j(X)}{X - i} = \frac{1}{A'(j)} \cdot \left(\frac{1}{i - j} \cdot \frac{A(X)}{X - i} + \frac{1}{j - i} \cdot \frac{A(X)}{X - j}\right) \tag{28}$$

As a result:

$$c\left(\frac{L_j(X)}{X - i}\right) = \left(c\left(\frac{A(X)}{X - i}\right)^{\frac{1}{i-j}} \cdot c\left(\frac{A(X)}{X - j}\right)^{\frac{1}{j-i}}\right)^{\frac{1}{A'(j)}} \tag{29}$$

Thus, each upk_i must include $c\left(\frac{A(X)}{X-i}\right)$ and $A'(i)$. *Can derive from g^{τ^i}'s!*

Outline

Background

Our Techniques

Updating Proofs

Aggregating Proofs into Subvector Proofs

Conclusion

Aggregating Proofs

Aggregating Proofs

Given many $(\pi_i)_{i \in I}$, can aggregate into succinct **subvector proof** π_I.

Aggregating Proofs

Given many $(\pi_i)_{i \in I}$, can aggregate into succinct **subvector proof** π_I.

High-level ideas, thanks to Drake and Buterin:

Aggregating Proofs

Given many $(\pi_i)_{i \in I}$, can aggregate into succinct **subvector proof** π_I.

High-level ideas, thanks to Drake and Buterin:

- Each π_i is a commitment to quotient q_i of division $\frac{\phi(X)}{X-i}$

Aggregating Proofs

Given many $(\pi_i)_{i \in I}$, can aggregate into succinct **subvector proof** π_I.

High-level ideas, thanks to Drake and Buterin:

- Each π_i is a commitment to quotient q_i of division $\frac{\phi(X)}{X-i}$
- π_I is a commitment to quotient q_I of division $\frac{\phi(X)}{\prod_{i \in I}(X-i)}$ (see [KZG10])

Aggregating Proofs

Given many $(\pi_i)_{i\in I}$, can aggregate into succinct **subvector proof** π_I.

High-level ideas, thanks to Drake and Buterin:

- Each π_i is a commitment to quotient q_i of division $\frac{\phi(X)}{X-i}$
- π_I is a commitment to quotient q_I of division $\frac{\phi(X)}{\prod_{i\in I}(X-i)}$ (see [KZG10])
- Compute c_i's such that $\frac{1}{\prod_{i\in I}(X-i)} = \sum_{i\in I} c_i \frac{1}{X-i}$

Aggregating Proofs

Given many $(\pi_i)_{i \in I}$, can aggregate into succinct **subvector proof** π_I.

High-level ideas, thanks to Drake and Buterin:

- Each π_i is a commitment to quotient q_i of division $\frac{\phi(X)}{X-i}$
- π_I is a commitment to quotient q_I of division $\frac{\phi(X)}{\prod_{i \in I}(X-i)}$ (see [KZG10])
- Compute c_i's such that $\frac{1}{\prod_{i \in I}(X-i)} = \sum_{i \in I} c_i \frac{1}{X-i}$
- Then, $q_I(X) = \sum_{i \in I} c_i \cdot q_i(X)$

Aggregating Proofs

Given many $(\pi_i)_{i \in I}$, can aggregate into succinct **subvector proof** π_I.

High-level ideas, thanks to Drake and Buterin:

- Each π_i is a commitment to quotient q_i of division $\frac{\phi(X)}{X-i}$
- π_I is a commitment to quotient q_I of division $\frac{\phi(X)}{\prod_{i \in I}(X-i)}$ (see [KZG10])
- Compute c_i's such that $\frac{1}{\prod_{i \in I}(X-i)} = \sum_{i \in I} c_i \frac{1}{X-i}$
- Then, $q_I(X) = \sum_{i \in I} c_i \cdot q_i(X)$
- Thus, $\pi_I = \prod_{i \in I} \pi_i^{c_i}$

Conclusion

Summary

Summary

Main contribution: Concretely-efficient aSVC from KZG polynomial commitments

Summary

Main contribution: Concretely-efficient aSVC from KZG polynomial commitments

- Constant-sized, aggregatable and updatable proofs

Summary

Main contribution: Concretely-efficient aSVC from KZG polynomial commitments

- Constant-sized, aggregatable and updatable proofs
- Constant-sized update keys

Summary

Main contribution: Concretely-efficient aSVC from KZG polynomial commitments

- Constant-sized, aggregatable and updatable proofs
- Constant-sized update keys
- Quasilinear-time proof pre-computation, via Feist-Khovratovich (FK) technique [FK20]

Summary

Main contribution: Concretely-efficient aSVC from KZG polynomial commitments

- Constant-sized, aggregatable and updatable proofs
- Constant-sized update keys
- Quasilinear-time proof pre-computation, via Feist-Khovratovich (FK) technique [FK20]

Other contributions (not in this talk):

Summary

Main contribution: Concretely-efficient aSVC from KZG polynomial commitments

- Constant-sized, aggregatable and updatable proofs
- Constant-sized update keys
- Quasilinear-time proof pre-computation, via Feist-Khovratovich (FK) technique [FK20]

Other contributions (not in this talk):

- New security definition for KZG batch proofs, proven secure under n-SBDH

Summary

Main contribution: Concretely-efficient aSVC from KZG polynomial commitments

- Constant-sized, aggregatable and updatable proofs
- Constant-sized update keys
- Quasilinear-time proof pre-computation, via Feist-Khovratovich (FK) technique [FK20]

Other contributions (not in this talk):

- New security definition for KZG batch proofs, proven secure under n-SBDH
- Derive all upk_i's from g^{τ^i}'s fast

Summary

Main contribution: Concretely-efficient aSVC from KZG polynomial commitments

- Constant-sized, aggregatable and updatable proofs
- Constant-sized update keys
- Quasilinear-time proof pre-computation, via Feist-Khovratovich (FK) technique [FK20]

Other contributions (not in this talk):

- New security definition for KZG batch proofs, proven secure under n-SBDH
- Derive all upk_i's from g^{τ^i}'s fast
    - Keeps trusted setup simple

Summary

Main contribution: Concretely-efficient aSVC from KZG polynomial commitments

- Constant-sized, aggregatable and updatable proofs
- Constant-sized update keys
- Quasilinear-time proof pre-computation, via Feist-Khovratovich (FK) technique [FK20]

Other contributions (not in this talk):

- New security definition for KZG batch proofs, proven secure under n-SBDH
- Derive all upk_i's from g^{τ^i}'s fast
  - Keeps trusted setup simple
  - Keeps public parameters *updatable*

Summary

Main contribution: Concretely-efficient aSVC from KZG polynomial commitments

- Constant-sized, aggregatable and updatable proofs
- Constant-sized update keys
- Quasilinear-time proof pre-computation, via Feist-Khovratovich (FK) technique [FK20]

Other contributions (not in this talk):

- New security definition for KZG batch proofs, proven secure under n-SBDH
- Derive all upk_i's from g^{τ^i}'s fast
    - Keeps trusted setup simple
    - Keeps public parameters *updatable*
- Subtleties of VC-based stateless cryptocurrencies

Summary

Main contribution: Concretely-efficient aSVC from KZG polynomial commitments

- Constant-sized, aggregatable and updatable proofs
- Constant-sized update keys
- Quasilinear-time proof pre-computation, via Feist-Khovratovich (FK) technique [FK20]

Other contributions (not in this talk):

- New security definition for KZG batch proofs, proven secure under n-SBDH
- Derive all upk_i's from g^{τ^i}'s fast
    - Keeps trusted setup simple
    - Keeps public parameters *updatable*
- Subtleties of VC-based stateless cryptocurrencies
    - Keeping track of transaction counters

Summary

Main contribution: Concretely-efficient aSVC from KZG polynomial commitments

- Constant-sized, aggregatable and updatable proofs
- Constant-sized update keys
- Quasilinear-time proof pre-computation, via Feist-Khovratovich (FK) technique [FK20]

Other contributions (not in this talk):

- New security definition for KZG batch proofs, proven secure under n-SBDH
- Derive all upk_i's from g^{τ^i}'s fast
    - Keeps trusted setup simple
    - Keeps public parameters *updatable*
- Subtleties of VC-based stateless cryptocurrencies
    - Keeping track of transaction counters
    - Verifiable update keys

Summary

Main contribution: Concretely-efficient aSVC from KZG polynomial commitments

- Constant-sized, aggregatable and updatable proofs
- Constant-sized update keys
- Quasilinear-time proof pre-computation, via Feist-Khovratovich (FK) technique [FK20]

Other contributions (not in this talk):

- New security definition for KZG batch proofs, proven secure under n-SBDH
- Derive all upk_i's from g^{τ^i}'s fast
    - Keeps trusted setup simple
    - Keeps public parameters *updatable*
- Subtleties of VC-based stateless cryptocurrencies
    - Keeping track of transaction counters
    - Verifiable update keys
    - DoS attacks on new user registration

Thank you!

Paper is too long? **Read our blogpost!**

https://alinush.github.io/2020/05/06/aggregatable-subvector-commitments-for-stateless-cryptocurrencies.html

Outline

- Roots of Unity Benefits
- Decomposition of $1/((X-i)(X-j))$
- Decomposition of $1/A_I(X)$
- Requirements of VC Scheme

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:

- Can pre-compute 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:

- Can pre-compute all n proofs in $O(n \log n)$ time via FK technique [FK20].
- 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:

- Can pre-compute all n proofs in $O(n \log n)$ time via FK technique [FK20].
- Our public parameters can be *efficiently* derived from “powers-of-tau” g^{τ^i}’s:
  - $c\left(L_i\right)$’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:

- Can pre-compute all n proofs in $O(n \log n)$ time via FK technique [FK20].
- Our public parameters can be *efficiently* derived from "powers-of-tau" g^{τ^i}'s:
  - $c\left(L_i\right)$'s via an inverse FFT [Vir17]
  - $c\left(\frac{A(X)}{(X-i)}\right)$'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:

- Can pre-compute all n proofs in $O(n \log n)$ time via FK technique [FK20].
- Our public parameters can be *efficiently* derived from "powers-of-tau" g^{τ^i}'s:
  - $c\left(L_i\right)$'s via an inverse FFT [Vir17]
  - $c\left(\frac{A(X)}{(X-i)}\right)$'s via the Feist-Khovratovich (FK) technique [FK20]
  - $c\left(\frac{L_i(X)-1}{X-i}\right)$'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:

- Can pre-compute all n proofs in $O(n \log n)$ time via FK technique [FK20].
- Our public parameters can be *efficiently* derived from "powers-of-tau" g^{τ^i}'s:
  - $c\left(L_i\right)$'s via an inverse FFT [Vir17]
  - $c\left(\frac{A(X)}{(X-i)}\right)$'s via the Feist-Khovratovich (FK) technique [FK20]
  - $c\left(\frac{L_i(X)-1}{X-i}\right)$'s via our new, FK-like, technique (see [TAB+20, Sec 3.4.5])
- Since g^{τ^i}'s are updatable and rest are derivable from them, 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:

- Can pre-compute all n proofs in $O(n \log n)$ time via FK technique [FK20].
- Our public parameters can be *efficiently* derived from "powers-of-tau" g^{τ^i}'s:
  - $c\left(L_i\right)$'s via an inverse FFT [Vir17]
  - $c\left(\frac{A(X)}{(X-i)}\right)$'s via the Feist-Khovratovich (FK) technique [FK20]
  - $c\left(\frac{L_i(X)-1}{X-i}\right)$'s via our new, FK-like, technique (see [TAB+20, Sec 3.4.5])
- Since g^{τ^i}'s are updatable and rest are derivable from them, our public parameters are *updatable*.
- Can remove $A'(i)$ from upk_i.

Outline

Roots of Unity Benefits

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

Decomposition of $1/A_I(X)$

Requirements of VC Scheme

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

Note that:

$$\frac{1}{i-j}\cdot\frac{A(X)}{X-i}+\frac{1}{j-i}\cdot\frac{A(X)}{X-j} = \frac{1}{i-j}\cdot\frac{A(X)(X-j)}{(X-i)(X-j)}+\frac{1}{j-i}\cdot\frac{A(X)(X-i)}{(X-j)(X-i)} \tag{30}$$

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

$$= \frac{\frac{1}{i-j}A(X)[(X-j)-(X-i)]}{(X-i)(X-j)} \tag{32}$$

$$= \frac{\frac{1}{i-j}A(X)(-j+i)}{(X-i)(X-j)} \tag{33}$$

$$= \frac{A(X)}{(X-i)(X-j)} \tag{34}$$

Outline

Roots of Unity Benefits

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

Decomposition of $1/A_I(X)$

Requirements of VC Scheme

Partial Fraction Decomposition From Lagrange Interpolation

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

$$L_i(X) = \prod_{j \in I, j \neq i} \frac{X - j}{i - j} = \frac{A_I(X)}{A_I{}'(i)(X - i)}, \text{ where } A_I(X) = \prod_{i \in I}(X - i) \tag{35}$$

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 $\phi(X) = 1$:

$$\phi(X) = \sum_{i \in I} v_i L_i(X) \Leftrightarrow \tag{36}$$

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

$$\frac{1}{A_I(X)} = \sum_{i \in I} \frac{1}{A_I{}'(i)(X - i)} \Leftrightarrow \tag{38}$$

$$\frac{1}{A_I(X)} = \sum_{i \in I} \frac{1}{A_I{}'(i)} \cdot \frac{1}{(X - i)} \Rightarrow \tag{39}$$

$$c_i = \frac{1}{A_I{}'(i)} \tag{40}$$

Outline

Requirements of VC Scheme

Let $\vec{v} = [v_0, v_1, \dots, v_{n-1}]$ be a vector of size n.

Requirements of VC Scheme

Let $\vec{v} = [v_0, v_1, \ldots, v_{n-1}]$ be a vector of size n.

- $vrk, prk, (upk_i)_{i\in[0,n)}, \pi^*, d^* \leftarrow VC.KeyGen(1^\lambda, n)$

Requirements of VC Scheme

Let $\vec{v} = [v_0, v_1, \ldots, v_{n-1}]$ be a vector of size n.

- $vrk, prk, (upk_i)_{i \in [0,n)}, \pi^*, d^* \leftarrow VC.KeyGen(1^\lambda, n)$
    - vrk is a *small* global verification key

Requirements of VC Scheme

Let $\vec{v} = [v_0, v_1, \ldots, v_{n-1}]$ be a vector of size n.

- $vrk, prk, (upk_i)_{i \in [0,n)}, \pi^*, d^* \leftarrow VC.KeyGen(1^\lambda, n)$
    - vrk is a *small* global verification key
    - prk is an $O(n)$-sized proving key

Requirements of VC Scheme

Let $\vec{v} = [v_0, v_1, \ldots, v_{n-1}]$ be a vector of size n.

- $vrk, prk, (upk_i)_{i \in [0,n)}, \pi^*, d^* \leftarrow VC.KeyGen(1^\lambda, n)$
  - vrk is a *small* global verification key
  - prk is an $O(n)$-sized proving key
  - upk_i is a *small* user-specific update key

Requirements of VC Scheme

Let $\vec{v} = [v_0, v_1, \ldots, v_{n-1}]$ be a vector of size n.

- $vrk, prk, (upk_i)_{i \in [0,n)}, \pi^*, d^* \leftarrow VC.KeyGen(1^\lambda, n)$
  - vrk is a *small* global verification key
  - prk is an $O(n)$-sized proving key
  - upk_i is a *small* user-specific update key
  - π^* is a *small* proof w.r.t. d^* that v_i is 0, for any position i

Requirements of VC Scheme

Let $\vec{v} = [v_0, v_1, \ldots, v_{n-1}]$ be a vector of size n.

- $vrk, prk, (upk_i)_{i \in [0,n)}, \pi^*, d^* \leftarrow VC.KeyGen(1^\lambda, n)$
  - vrk is a *small* global verification key
  - prk is an $O(n)$-sized proving key
  - upk_i is a *small* user-specific update key
  - π^* is a *small* proof w.r.t. d^* that v_i is 0, for any position i
- $d = VC.Commit(prk, (v_i)_{i \in [0,n-1)})$

Requirements of VC Scheme

Let $\vec{v} = [v_0, v_1, \ldots, v_{n-1}]$ be a vector of size n.

- $vrk, prk, (upk_i)_{i \in [0,n)}, \pi^*, d^* \leftarrow VC.KeyGen(1^\lambda, n)$
    - vrk is a *small* global verification key
    - prk is an $O(n)$-sized proving key
    - upk_i is a *small* user-specific update key
    - π^* is a *small* proof w.r.t. d^* that v_i is 0, for any position i
- $d = VC.Commit(prk, (v_i)_{i \in [0,n-1)})$
- $d' = VC.UpdDig(d, \delta_i, i, upk_i)$

Requirements of VC Scheme

Let $\vec{v} = [v_0, v_1, \dots, v_{n-1}]$ be a vector of size n.

- $vrk, prk, (upk_i)_{i \in [0,n)}, \pi^*, d^* \leftarrow VC.KeyGen(1^\lambda, n)$
  - vrk is a *small* global verification key
  - prk is an $O(n)$-sized proving key
  - upk_i is a *small* user-specific update key
  - π^* is a *small* proof w.r.t. d^* that v_i is 0, for any position i
- $d = VC.Commit(prk, (v_i)_{i \in [0,n-1)})$
- $d' = VC.UpdDig(d, \delta_i, i, upk_i)$
- $\pi_i' = VC.UpdProof(\pi_i, \delta_j, j, upk_i, upk_j)$

Requirements of VC Scheme

Let $\vec{v} = [v_0, v_1, \dots, v_{n-1}]$ be a vector of size n.

- $vrk, prk, (upk_i)_{i \in [0,n)}, \pi^*, d^* \leftarrow VC.KeyGen(1^\lambda, n)$
  - vrk is a *small* global verification key
  - prk is an $O(n)$-sized proving key
  - upk_i is a *small* user-specific update key
  - π^* is a *small* proof w.r.t. d^* that v_i is 0, for any position i
- $d = VC.Commit(prk, (v_i)_{i \in [0,n-1)})$
- $d' = VC.UpdDig(d, \delta_i, i, upk_i)$
- $\pi_i' = VC.UpdProof(\pi_i, \delta_j, j, upk_i, upk_j)$
- $\pi_I = VC.AggregateProofs(I, (\pi_i)_{i \in I})$

Requirements of VC Scheme

Let $\vec{v} = [v_0, v_1, \ldots, v_{n-1}]$ be a vector of size n.

- $vrk, prk, (upk_i)_{i \in [0,n)}, \pi^*, d^* \leftarrow VC.KeyGen(1^\lambda, n)$
  - vrk is a *small* global verification key
  - prk is an $O(n)$-sized proving key
  - upk_i is a *small* user-specific update key
  - π^* is a *small* proof w.r.t. d^* that v_i is 0, for any position i
- $d = VC.Commit(prk, (v_i)_{i \in [0,n-1)})$
- $d' = VC.UpdDig(d, \delta_i, i, upk_i)$
- $\pi_i' = VC.UpdProof(\pi_i, \delta_j, j, upk_i, upk_j)$
- $\pi_I = VC.AggregateProofs(I, (\pi_i)_{i \in I})$
- $\{T, F\} \leftarrow VC.VerifyPos(vrk, d, (v_i)_{i \in I}, I, \pi_I)$

Requirements of VC Scheme

Let $\vec{v} = [v_0, v_1, \ldots, v_{n-1}]$ be a vector of size n.

- $vrk, prk, (upk_i)_{i\in[0,n)}, \pi^*, d^* \leftarrow VC.KeyGen(1^\lambda, n)$
  - vrk is a *small* global verification key
  - prk is an $O(n)$-sized proving key
  - upk_i is a *small* user-specific update key
  - π^* is a *small* proof w.r.t. d^* that v_i is 0, for any position i
- $d = VC.Commit(prk, (v_i)_{i\in[0,n-1)})$
- $d' = VC.UpdDig(d, \delta_i, i, upk_i)$
- $\pi_i' = VC.UpdProof(\pi_i, \delta_j, j, upk_i, upk_j)$
- $\pi_I = VC.AggregateProofs(I, (\pi_i)_{i\in I})$
- $\{T, F\} \leftarrow VC.VerifyPos(vrk, d, (v_i)_{i\in I}, I, \pi_I)$
- $(\pi_i)_{i\in[0,n)} \leftarrow VC.ProveAll(prk, \vec{v})$

Requirements of VC Scheme

Let $\vec{v} = [v_0, v_1, \ldots, v_{n-1}]$ be a vector of size n.

- $vrk, prk, (upk_i)_{i \in [0,n)}, \pi^*, d^* \leftarrow VC.KeyGen(1^\lambda, n)$
    - vrk is a *small* global verification key
    - prk is an $O(n)$-sized proving key
    - upk_i is a *small* user-specific update key
    - π^* is a *small* proof w.r.t. d^* that v_i is 0, for any position i
- $d = VC.Commit(prk, (v_i)_{i \in [0,n-1)})$
- $d' = VC.UpdDig(d, \delta_i, i, upk_i)$
- $\pi_i' = VC.UpdProof(\pi_i, \delta_j, j, upk_i, upk_j)$
- $\pi_I = VC.AggregateProofs(I, (\pi_i)_{i \in I})$
- $\{T, F\} \leftarrow VC.VerifyPos(vrk, d, (v_i)_{i \in I}, I, \pi_I)$
- $(\pi_i)_{i \in [0,n)} \leftarrow VC.ProveAll(prk, \vec{v})$

References i

Dan Boneh, Benedikt Bünz, and Ben Fisch.
Batching Techniques for Accumulators with Applications to IOPs and Stateless Blockchains.
In *CRYPTO'19*, 2019.

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`.

References ii

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

Jan Camenisch, Maria Dubovitskaya, Kristiyan Haralambiev, and Markulf Kohlweiss.
Composable and Modular Anonymous Credentials: Definitions and Practical Constructions.
In *ASIACRYPT'15*, 2015.

Dario Catalano and Dario Fiore.
Vector Commitments and Their Applications.
In *PKC'13*, 2013.

- Matteo Campanelli, Dario Fiore, Nicola Greco, Dimitris Kolonelos, and Luca Nizzardo.
  Vector Commitment Techniques and Applications to Verifiable Decentralized Storage, 2020.
  `https://eprint.iacr.org/2020/149`.
- Alexander Chepurnoy, Charalampos Papamanthou, and Yupeng Zhang.
  Edrax: A Cryptocurrency with Stateless Transaction Validation, 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, 2020.
`https://eprint.iacr.org/2020/419`.

Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg.
Constant-Size Commitments to Polynomials and Their Applications.
In *ASIACRYPT'10*, 2010.

Russell W. F. Lai and Giulio Malavolta.
Subvector Commitments with Application to Succinct Arguments.
In *CRYPTO'19*, 2019.

References v

Ralph C. Merkle.
A Digital Signature Based on a Conventional Encryption Function.
In Carl Pomerance, editor, *CRYPTO '87*, pages 369–378, Berlin, Heidelberg, 1988. Springer Berlin Heidelberg.

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 *FC'17*, 2017.

Alin Tomescu, Ittai Abraham, Vitalik Buterin, Justin Drake, Dankrad Feist, and Dmitry Khovratovich.
Aggregatable Subvector Commitments for Stateless Cryptocurrencies, 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 *IEEE S&P'20*, 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, 3rd edition, 2013.