Linearization Trick in SNARKs

Janno Siim
University of Tartu, Estonia

Estonian-Latvian Theory Days 2026

Background

Talk is based on my recent PKC26 paper: `https://eprint.iacr.org/2026/458`

Commitment Scheme

A commitment lets a prover lock value m and reveal it later.

Commitment Scheme

A commitment lets a prover lock value m and reveal it later.

$$(C, decom) \leftarrow \mathbf{Com}(m),$$

Commitment Scheme

A commitment lets a prover lock value m and reveal it later.

$$(C, \mathit{decom}) \leftarrow \mathbf{Com}(m), \qquad \mathbf{Verify}(C, m, \mathit{decom}) \in \{\text{Accept}, \text{Reject}\}$$

Commitment Scheme

A commitment lets a prover lock value m and reveal it later.

$$(C, \mathit{decom}) \leftarrow \mathbf{Com}(m), \qquad \mathbf{Verify}(C, m, \mathit{decom}) \in \{\text{Accept}, \text{Reject}\}$$

Security properties:

Commitment Scheme

A commitment lets a prover lock value m and reveal it later.

$$(C, decom) \leftarrow \textbf{Com}(m), \qquad \textbf{Verify}(C, m, decom) \in \{\text{Accept}, \text{Reject}\}$$

Security properties:

- **Hiding**: C reveals nothing about m before opening.

Commitment Scheme

A commitment lets a prover lock value m and reveal it later.

$$(C, \mathit{decom}) \leftarrow \mathbf{Com}(m), \qquad \mathbf{Verify}(C, m, \mathit{decom}) \in \{\text{Accept}, \text{Reject}\}$$

Security properties:

- **Hiding**: C reveals nothing about m before opening.
- **Binding**: Prover cannot open to $m' \neq m$.

Polynomial Commitment Schemes (PCS)

A PCS for degree $\leq n$ polynomials allows a Prover to:

Polynomial Commitment Schemes (PCS)

A PCS for degree $\leq n$ polynomials allows a Prover to:

- **Commit**: Send a short string C that locks a polynomial $f(X)$ with $\deg(f) \leq n$.

Polynomial Commitment Schemes (PCS)

A PCS for degree $\leq n$ polynomials allows a Prover to:

- **Commit**: Send a short string C that locks a polynomial $f(X)$ with $\deg(f) \leq n$.
- **Open**: For any α, provide $y = f(\alpha)$ and a short proof π.

Polynomial Commitment Schemes (PCS)

A PCS for degree $\leq n$ polynomials allows a Prover to:

- **Commit**: Send a short string C that locks a polynomial $f(X)$ with $\deg(f) \leq n$.
- **Open**: For any α, provide $y = f(\alpha)$ and a short proof π.
- **Verify**: A Verifier checks the evaluation y using C and π.

Polynomial Commitment Schemes (PCS)

A PCS for degree $\leq n$ polynomials allows a Prover to:

- **Commit**: Send a short string C that locks a polynomial $f(X)$ with $\deg(f) \leq n$.
- **Open**: For any α, provide $y = f(\alpha)$ and a short proof π.
- **Verify**: A Verifier checks the evaluation y using C and π.

Security property (intuition):

Polynomial Commitment Schemes (PCS)

A PCS for degree $\leq n$ polynomials allows a Prover to:

- **Commit**: Send a short string C that locks a polynomial $f(X)$ with $\deg(f) \leq n$.
- **Open**: For any α, provide $y = f(\alpha)$ and a short proof π.
- **Verify**: A Verifier checks the evaluation y using C and π.

Security property (intuition):

- Prover knows f of $\deg(f) \leq n$ such that $f(\alpha) = y$.

Polynomial Commitment Schemes (PCS)

A PCS for degree $\leq n$ polynomials allows a Prover to:

- **Commit**: Send a short string C that locks a polynomial $f(X)$ with $\deg(f) \leq n$.
- **Open**: For any α, provide $y = f(\alpha)$ and a short proof π.
- **Verify**: A Verifier checks the evaluation y using C and π.

Security property (intuition):

- Prover knows f of $\deg(f) \leq n$ such that $f(\alpha) = y$.

Let look at one construction based on pairings.

Pairings

Setup: Let p be prime and let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be cyclic additive groups of order p.

Pairings

Setup: Let p be prime and let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be cyclic additive groups of order p.

$$e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$$

Pairings

Setup: Let p be prime and let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be cyclic additive groups of order p.

$$e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$$

Notation:

- Respective generators: $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_T$.

Pairings

Setup: Let p be prime and let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be cyclic additive groups of order p.

$$e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$$

Notation:

- Respective generators: $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_T$.
- For $x \in \mathbb{F}_p$, let $[x]_1 := x\mathcal{P}_1$, $[x]_2 := x\mathcal{P}_2$, and $[x]_T := x\mathcal{P}_T$.

Pairings

Setup: Let p be prime and let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be cyclic additive groups of order p.

$$e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$$

Notation:

- Respective generators: $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_T$.
- For $x \in \mathbb{F}_p$, let $[x]_1 := x\mathcal{P}_1$, $[x]_2 := x\mathcal{P}_2$, and $[x]_T := x\mathcal{P}_T$.

Properties:

- **Bilinear:** $e([a]_1, [b]_2) = (ab) \cdot e([1]_1, [1]_2)$ for $a, b \in \mathbb{F}_p$.

Pairings

Setup: Let p be prime and let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be cyclic additive groups of order p.

$$e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$$

Notation:

- Respective generators: $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_T$.
- For $x \in \mathbb{F}_p$, let $[x]_1 := x\mathcal{P}_1$, $[x]_2 := x\mathcal{P}_2$, and $[x]_T := x\mathcal{P}_T$.

Properties:

- **Bilinear:** $e([a]_1, [b]_2) = (ab) \cdot e([1]_1, [1]_2)$ for $a, b \in \mathbb{F}_p$.
- **Non-degenerate:** $e([1]_1, [1]_2) \neq [0]_T$.

Pairings

Setup: Let p be prime and let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be cyclic additive groups of order p.

$$e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$$

Notation:

- Respective generators: $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_T$.
- For $x \in \mathbb{F}_p$, let $[x]_1 := x\mathcal{P}_1$, $[x]_2 := x\mathcal{P}_2$, and $[x]_T := x\mathcal{P}_T$.

Properties:

- **Bilinear:** $e([a]_1, [b]_2) = (ab) \cdot e([1]_1, [1]_2)$ for $a, b \in \mathbb{F}_p$.
- **Non-degenerate:** $e([1]_1, [1]_2) \neq [0]_T$.
- **Efficient:** group operations and $e(\cdot, \cdot)$ efficiently computable.

KZG Polynomial Commitment

A popular PCS based on pairings:

KZG Polynomial Commitment

A popular PCS based on pairings:

- **Setup:** Commitment key $\mathsf{ck} = ([1]_1, [\tau]_1, \ldots, [\tau^n]_1, [1]_2, [\tau]_2)$ for random τ.

KZG Polynomial Commitment

A popular PCS based on pairings:

- **Setup:** Commitment key $\mathsf{ck} = ([1]_1, [\tau]_1, \ldots, [\tau^n]_1, [1]_2, [\tau]_2)$ for random τ.
- **Commit**: For a polynomial $f(X) = \sum_{i=0}^{n} a_i X^i$, compute $C = \sum_{i=0}^{n} a_i [\tau^i]_1$.

KZG Polynomial Commitment

A popular PCS based on pairings:

- **Setup:** Commitment key $\mathsf{ck} = ([1]_1, [\tau]_1, \ldots, [\tau^n]_1, [1]_2, [\tau]_2)$ for random τ.
- **Commit**: For a polynomial $f(X) = \sum_{i=0}^{n} a_i X^i$, compute $C = \sum_{i=0}^{n} a_i [\tau^i]_1$.
- **Open**: To open at α, send $y = f(\alpha)$ and commitment π to $Q(X) = (f(X) - y)/(X - \alpha)$.

KZG Polynomial Commitment

A popular PCS based on pairings:

- **Setup:** Commitment key $\mathsf{ck} = ([1]_1, [\tau]_1, \ldots, [\tau^n]_1, [1]_2, [\tau]_2)$ for random τ.
- **Commit**: For a polynomial $f(X) = \sum_{i=0}^{n} a_i X^i$, compute $C = \sum_{i=0}^{n} a_i [\tau^i]_1$.
- **Open**: To open at α, send $y = f(\alpha)$ and commitment π to $Q(X) = (f(X) - y)/(X - \alpha)$.
- **Verify**: Check $e(\pi, [\tau - \alpha]_2) = e(C - [y]_1, [1]_2)$.

KZG Polynomial Commitment

A popular PCS based on pairings:

- **Setup:** Commitment key $\mathsf{ck} = ([1]_1, [\tau]_1, \ldots, [\tau^n]_1, [1]_2, [\tau]_2)$ for random τ.
- **Commit**: For a polynomial $f(X) = \sum_{i=0}^{n} a_i X^i$, compute $C = \sum_{i=0}^{n} a_i [\tau^i]_1$.
- **Open**: To open at α, send $y = f(\alpha)$ and commitment π to $Q(X) = (f(X) - y)/(X - \alpha)$.
- **Verify**: Check $e(\pi, [\tau - \alpha]_2) = e(C - [y]_1, [1]_2)$.

Security intuition:

KZG Polynomial Commitment

A popular PCS based on pairings:

- **Setup:** Commitment key $\mathsf{ck} = ([1]_1, [\tau]_1, \ldots, [\tau^n]_1, [1]_2, [\tau]_2)$ for random τ.
- **Commit**: For a polynomial $f(X) = \sum_{i=0}^{n} a_i X^i$, compute $C = \sum_{i=0}^{n} a_i [\tau^i]_1$.
- **Open**: To open at α, send $y = f(\alpha)$ and commitment π to $Q(X) = (f(X) - y)/(X - \alpha)$.
- **Verify**: Check $e(\pi, [\tau - \alpha]_2) = e(C - [y]_1, [1]_2)$.

Security intuition:

- Verifier checks $Q(\tau)(\tau - \alpha) = f(\tau) - y$.

KZG Polynomial Commitment

A popular PCS based on pairings:

- **Setup:** Commitment key $\mathsf{ck} = ([1]_1, [\tau]_1, \ldots, [\tau^n]_1, [1]_2, [\tau]_2)$ for random τ.
- **Commit**: For a polynomial $f(X) = \sum_{i=0}^{n} a_i X^i$, compute $C = \sum_{i=0}^{n} a_i [\tau^i]_1$.
- **Open**: To open at α, send $y = f(\alpha)$ and commitment π to $Q(X) = (f(X) - y)/(X - \alpha)$.
- **Verify**: Check $e(\pi, [\tau - \alpha]_2) = e(C - [y]_1, [1]_2)$.

Security intuition:

- Verifier checks $Q(\tau)(\tau - \alpha) = f(\tau) - y$.
- τ is unknown random value $\Rightarrow$

KZG Polynomial Commitment

A popular PCS based on pairings:

- **Setup:** Commitment key $\mathsf{ck} = ([1]_1, [\tau]_1, \ldots, [\tau^n]_1, [1]_2, [\tau]_2)$ for random τ.
- **Commit**: For a polynomial $f(X) = \sum_{i=0}^{n} a_i X^i$, compute $C = \sum_{i=0}^{n} a_i [\tau^i]_1$.
- **Open**: To open at α, send $y = f(\alpha)$ and commitment π to $Q(X) = (f(X) - y)/(X - \alpha)$.
- **Verify**: Check $e(\pi, [\tau - \alpha]_2) = e(C - [y]_1, [1]_2)$.

Security intuition:

- Verifier checks $Q(\tau)(\tau - \alpha) = f(\tau) - y$.
- τ is unknown random value $\Rightarrow$
- $Q(X)(X - \alpha) = f(X) - y$

KZG Polynomial Commitment

A popular PCS based on pairings:

- **Setup:** Commitment key $\mathsf{ck} = ([1]_1, [\tau]_1, \ldots, [\tau^n]_1, [1]_2, [\tau]_2)$ for random τ.
- **Commit**: For a polynomial $f(X) = \sum_{i=0}^{n} a_i X^i$, compute $C = \sum_{i=0}^{n} a_i [\tau^i]_1$.
- **Open**: To open at α, send $y = f(\alpha)$ and commitment π to $Q(X) = (f(X) - y)/(X - \alpha)$.
- **Verify**: Check $e(\pi, [\tau - \alpha]_2) = e(C - [y]_1, [1]_2)$.

Security intuition:

- Verifier checks $Q(\tau)(\tau - \alpha) = f(\tau) - y$.
- τ is unknown random value $\Rightarrow$
- $Q(X)(X - \alpha) = f(X) - y \Rightarrow f(\alpha) - y = Q(\alpha)(\alpha - \alpha) = 0 \Rightarrow$

KZG Polynomial Commitment

A popular PCS based on pairings:

- **Setup:** Commitment key $\mathsf{ck} = ([1]_1, [\tau]_1, \ldots, [\tau^n]_1, [1]_2, [\tau]_2)$ for random τ.
- **Commit**: For a polynomial $f(X) = \sum_{i=0}^{n} a_i X^i$, compute $C = \sum_{i=0}^{n} a_i [\tau^i]_1$.
- **Open**: To open at α, send $y = f(\alpha)$ and commitment π to $Q(X) = (f(X) - y)/(X - \alpha)$.
- **Verify**: Check $e(\pi, [\tau - \alpha]_2) = e(C - [y]_1, [1]_2)$.

Security intuition:

- Verifier checks $Q(\tau)(\tau - \alpha) = f(\tau) - y$.
- τ is unknown random value $\Rightarrow$
- $Q(X)(X - \alpha) = f(X) - y \Rightarrow f(\alpha) - y = Q(\alpha)(\alpha - \alpha) = 0 \Rightarrow$
- $f(\alpha) = y$.

Homomorphic Property

KZG is additively homomorphic:

Homomorphic Property

KZG is additively homomorphic:

- If C_i is a commitment to $f_i(X)$, then $\sum_{i=1}^{m} \lambda_i C_i$ is a commitment to $\sum_{i=1}^{m} \lambda_i f_i(X)$.

Homomorphic Property

KZG is additively homomorphic:

- If C_i is a commitment to $f_i(X)$, then $\sum_{i=1}^{m} \lambda_i C_i$ is a commitment to $\sum_{i=1}^{m} \lambda_i f_i(X)$.

Example:

Homomorphic Property

KZG is additively homomorphic:

- If C_i is a commitment to $f_i(X)$, then $\sum_{i=1}^{m} \lambda_i C_i$ is a commitment to $\sum_{i=1}^{m} \lambda_i f_i(X)$.

Example:

- Let $f(X) = \sum_{i=0}^{n} a_i X^i$ and $g(X) = \sum_{i=0}^{d} b_i X^i$.

Homomorphic Property

KZG is additively homomorphic:

- If C_i is a commitment to $f_i(X)$, then $\sum_{i=1}^{m} \lambda_i C_i$ is a commitment to $\sum_{i=1}^{m} \lambda_i f_i(X)$.

Example:

- Let $f(X) = \sum_{i=0}^{n} a_i X^i$ and $g(X) = \sum_{i=0}^{d} b_i X^i$.
- $C_1 = Com(f) = \sum_{i=0}^{n} a_i [\tau^i]_1$ and $C_2 = Com(g) = \sum_{i=0}^{d} b_i [\tau^i]_1$.

Homomorphic Property

KZG is additively homomorphic:

- If C_i is a commitment to $f_i(X)$, then $\sum_{i=1}^{m} \lambda_i C_i$ is a commitment to $\sum_{i=1}^{m} \lambda_i f_i(X)$.

Example:

- Let $f(X) = \sum_{i=0}^{n} a_i X^i$ and $g(X) = \sum_{i=0}^{d} b_i X^i$.
- $C_1 = Com(f) = \sum_{i=0}^{n} a_i [\tau^i]_1$ and $C_2 = Com(g) = \sum_{i=0}^{d} b_i [\tau^i]_1$.
- $C_1 + C_2 = \sum_{i=0}^{n} (a_i + b_i)[\tau^i]_1 = Com(g + f)$.

SNARKs

Why we care about about PCS?

SNARKs

Why we care about about PCS?

- SNARKs (**S**uccinct **N**on-interactive **AR**guments of **K**nowledge): short proof that **any** computation was done correctly.

SNARKs

Why we care about about PCS?

- SNARKs (**S**uccinct **N**on-interactive **AR**guments of **K**nowledge): short proof that **any** computation was done correctly.
- Widely used in blockchain and elsewhere.

SNARKs

Why we care about about PCS?

- SNARKs (**S**uccinct **N**on-interactive **AR**guments of **K**nowledge): short proof that **any** computation was done correctly.
- Widely used in blockchain and elsewhere.
- PCS is a key component in SNARKs.

The 3-Step SNARKs

Modern SNARKs follow a modular design:

The 3-Step SNARKs

Modern SNARKs follow a modular design:

1. **PIOP**: Interactive information-theoretic proof where the prover sends polynomial oracles.

The 3-Step SNARKs

Modern SNARKs follow a modular design:

1. **PIOP**: Interactive information-theoretic proof where the prover sends polynomial oracles.
   - Verifier can query polynomials.

The 3-Step SNARKs

Modern SNARKs follow a modular design:

1. **PIOP**: Interactive information-theoretic proof where the prover sends polynomial oracles.
   - Verifier can query polynomials.
2. **PCS**: Compress these oracles and allow queries.

The 3-Step SNARKs

Modern SNARKs follow a modular design:

1. **PIOP**: Interactive information-theoretic proof where the prover sends polynomial oracles.
   - Verifier can query polynomials.
2. **PCS**: Compress these oracles and allow queries.
   - Relies on security assumptions.

The 3-Step SNARKs

Modern SNARKs follow a modular design:

1. **PIOP**: Interactive information-theoretic proof where the prover sends polynomial oracles.
   - Verifier can query polynomials.
2. **PCS**: Compress these oracles and allow queries.
   - Relies on security assumptions.
3. **Fiat-Shamir**: A heuristic to achieve non-interactivity.

The Problem: Multiple Openings

In many SNARKs, the verifier needs to check an identity involving committed polynomials $f_1, \ldots, f_m$ and $g_1, \ldots, g_m$:

The Problem: Multiple Openings

In many SNARKs, the verifier needs to check an identity involving committed polynomials $f_1, \ldots, f_m$ and $g_1, \ldots, g_m$:

$$\mathcal{V}(X) = \sum_{i=1}^{m} g_i(X) f_i(X) = 0$$

The Problem: Multiple Openings

In many SNARKs, the verifier needs to check an identity involving committed polynomials $f_1, \ldots, f_m$ and $g_1, \ldots, g_m$:

$$\mathcal{V}(X) = \sum_{i=1}^{m} g_i(X) f_i(X) = 0$$

Naive Verification:

The Problem: Multiple Openings

In many SNARKs, the verifier needs to check an identity involving committed polynomials $f_1, \ldots, f_m$ and $g_1, \ldots, g_m$:

$$\mathcal{V}(X) = \sum_{i=1}^{m} g_i(X) f_i(X) = 0$$

Naive Verification:

- Open every f_i and g_i at a random point α.

The Problem: Multiple Openings

In many SNARKs, the verifier needs to check an identity involving committed polynomials $f_1, \ldots, f_m$ and $g_1, \ldots, g_m$:

$$\mathcal{V}(X) = \sum_{i=1}^{m} g_i(X) f_i(X) = 0$$

Naive Verification:

- Open every f_i and g_i at a random point α.
- **Cost**: $2m$ field elements and evaluation proofs.

The Solution: Linearization (Maller's Trick)

SNARKs Marlin and Plonk introduced a clever optimization for KZG:

The Solution: Linearization (Maller's Trick)

SNARKs Marlin and Plonk introduced a clever optimization for KZG:

1. Prover opens $\bar{g}_i = g_i(\alpha)$ as usual.

The Solution: Linearization (Maller's Trick)

SNARKs Marlin and Plonk introduced a clever optimization for KZG:

1. Prover opens $\bar{g}_i = g_i(\alpha)$ as usual.
2. Using the **homomorphic** property of KZG, the verifier computes a commitment C' to the "linearized" polynomial:

The Solution: Linearization (Maller's Trick)

SNARKs Marlin and Plonk introduced a clever optimization for KZG:

1. Prover opens $\bar{g}_i = g_i(\alpha)$ as usual.
2. Using the **homomorphic** property of KZG, the verifier computes a commitment C' to the "linearized" polynomial:

$$\mathcal{V}'(X) = \sum_{i=1}^{m} \bar{g}_i f_i(X)$$

The Solution: Linearization (Maller's Trick)

SNARKs Marlin and Plonk introduced a clever optimization for KZG:

1. Prover opens $\bar{g}_i = g_i(\alpha)$ as usual.
2. Using the **homomorphic** property of KZG, the verifier computes a commitment C' to the "linearized" polynomial:

$$\mathcal{V}'(X) = \sum_{i=1}^{m} \bar{g}_i f_i(X)$$

3. Prover sends only **one** opening proof for $\mathcal{V}'(\alpha) = 0$.

The Solution: Linearization (Maller's Trick)

SNARKs Marlin and Plonk introduced a clever optimization for KZG:

1. Prover opens $\bar{g}_i = g_i(\alpha)$ as usual.
2. Using the **homomorphic** property of KZG, the verifier computes a commitment C' to the "linearized" polynomial:

$$\mathcal{V}'(X) = \sum_{i=1}^{m} \bar{g}_i f_i(X)$$

3. Prover sends only **one** opening proof for $\mathcal{V}'(\alpha) = 0$.

Impact: Cuts the communication roughly in half.

Widespread Adoption

This "trick" is widely used in pairing-based SNARKs:

Widespread Adoption

This "trick" is widely used in pairing-based SNARKs:

- **Plonk** (and its many variants)

Widespread Adoption

This "trick" is widely used in pairing-based SNARKs:

- **Plonk** (and its many variants)
- **Marlin**

Widespread Adoption

This "trick" is widely used in pairing-based SNARKs:

- **Plonk** (and its many variants)
- **Marlin**
- **Lunar**

Widespread Adoption

This "trick" is widely used in pairing-based SNARKs:

- **Plonk** (and its many variants)
- **Marlin**
- **Lunar**
- **Vampire**

Widespread Adoption

This "trick" is widely used in pairing-based SNARKs:

- **Plonk** (and its many variants)
- **Marlin**
- **Lunar**
- **Vampire**
- **Mercury** & **Samaritan**

Widespread Adoption

This "trick" is widely used in pairing-based SNARKs:

- **Plonk** (and its many variants)
- **Marlin**
- **Lunar**
- **Vampire**
- **Mercury** & **Samaritan**
- ...

Widespread Adoption

This "trick" is widely used in pairing-based SNARKs:

- **Plonk** (and its many variants)
- **Marlin**
- **Lunar**
- **Vampire**
- **Mercury** & **Samaritan**
- ...

*Plonk used in blockchains and **secures (hundreds of) millions of dollars**.*

The Problem: Is it actually secure?

Security requirement (usually):

The Problem: Is it actually secure?

Security requirement (usually):

- The prover must "know" the polynomials f_i, g_i that satisfy $\mathcal{V}(X) = 0$.

The Problem: Is it actually secure?

Security requirement (usually):

- The prover must "know" the polynomials f_i, g_i that satisfy $\mathcal{V}(X) = 0$.
- Formalized as **extractability** property of PCS (many variants).

The Problem: Is it actually secure?

Security requirement (usually):

- The prover must "know" the polynomials f_i, g_i that satisfy $\mathcal{V}(X) = 0$.
- Formalized as **extractability** property of PCS (many variants).
- No proof in original papers.

The Problem: Is it actually secure?

Security requirement (usually):

- The prover must "know" the polynomials f_i, g_i that satisfy $\mathcal{V}(X) = 0$.
- Formalized as **extractability** property of PCS (many variants).
- No proof in original papers.

Concerns about linearization:

The Problem: Is it actually secure?

Security requirement (usually):

- The prover must "know" the polynomials f_i, g_i that satisfy $\mathcal{V}(X) = 0$.
- Formalized as **extractability** property of PCS (many variants).
- No proof in original papers.

Concerns about linearization:

- **Negative Results**: Lipmaa, Parisella, Me (TCC23) and Faonio, Fiore, Russo (CCS24) attack in some scenarios.

Computational Special Soundness

- Work by Lipmaa, Parisella, Me (Eurocrypt 2024)

Computational Special Soundness

- Work by Lipmaa, Parisella, Me (Eurocrypt 2024)
- In SNARKs KZG evaluation point α is random from large space.

Computational Special Soundness

- Work by Lipmaa, Parisella, Me (Eurocrypt 2024)
- In SNARKs KZG evaluation point α is random from large space.
- Extractability notion: **Computational Special Soundness (CSS)**:

Computational Special Soundness

- Work by Lipmaa, Parisella, Me (Eurocrypt 2024)
- In SNARKs KZG evaluation point α is random from large space.
- Extractability notion: **Computational Special Soundness (CSS)**:
  - Computationally bounded adversary $\mathcal{A}$ gets ck (poly $\deg(f) \leq n$) as input

Computational Special Soundness

- Work by Lipmaa, Parisella, Me (Eurocrypt 2024)
- In SNARKs KZG evaluation point α is random from large space.
- Extractability notion: **Computational Special Soundness (CSS)**:
  - Computationally bounded adversary $\mathcal{A}$ gets ck (poly $\deg(f) \leq n$) as input
  - Outputs transcripts $\mathsf{tr}_i = (C, \alpha_i, \pi_i, y_i)$ for $i = 1, 2, \ldots, n+1$.

Computational Special Soundness

- Work by Lipmaa, Parisella, Me (Eurocrypt 2024)
- In SNARKs KZG evaluation point α is random from large space.
- Extractability notion: **Computational Special Soundness (CSS)**:
  - Computationally bounded adversary $\mathcal{A}$ gets ck (poly $\deg(f) \leq n$) as input
  - Outputs transcripts $\mathsf{tr}_i = (C, \alpha_i, \pi_i, y_i)$ for $i = 1, 2, \ldots, n+1$.
  - Same commitment C for all tr_i.

Computational Special Soundness

- Work by Lipmaa, Parisella, Me (Eurocrypt 2024)
- In SNARKs KZG evaluation point α is random from large space.
- Extractability notion: **Computational Special Soundness (CSS)**:
  - Computationally bounded adversary $\mathcal{A}$ gets ck (poly $\deg(f) \leq n$) as input
  - Outputs transcripts $\mathsf{tr}_i = (C, \alpha_i, \pi_i, y_i)$ for $i = 1, 2, \ldots, n+1$.
  - Same commitment C for all tr_i.
  - Distinct α_i.

Computational Special Soundness

- Work by Lipmaa, Parisella, Me (Eurocrypt 2024)
- In SNARKs KZG evaluation point α is random from large space.
- Extractability notion: **Computational Special Soundness (CSS)**:
  - Computationally bounded adversary $\mathcal{A}$ gets ck (poly $\deg(f) \leq n$) as input
  - Outputs transcripts $\mathsf{tr}_i = (C, \alpha_i, \pi_i, y_i)$ for $i = 1, 2, \ldots, n+1$.
  - Same commitment C for all tr_i.
  - Distinct α_i.
  - Exists extractor $\mathsf{Ext}(\mathsf{tr}_1, ..., \mathsf{tr}_{n+1}) \to f(X)$ with $\mathit{Com}(\mathsf{ck}, f) = C$ and $\deg(f) \leq n$ and $f(\alpha_i) = y_i$ for all i.

Computational Special Soundness

- Work by Lipmaa, Parisella, Me (Eurocrypt 2024)
- In SNARKs KZG evaluation point α is random from large space.
- Extractability notion: **Computational Special Soundness (CSS)**:
  - Computationally bounded adversary $\mathcal{A}$ gets ck (poly $\deg(f) \leq n$) as input
  - Outputs transcripts $\mathsf{tr}_i = (C, \alpha_i, \pi_i, y_i)$ for $i = 1, 2, \ldots, n+1$.
  - Same commitment C for all tr_i.
  - Distinct α_i.
  - Exists extractor $\mathsf{Ext}(\mathsf{tr}_1, ..., \mathsf{tr}_{n+1}) \to f(X)$ with $\mathit{Com}(\mathsf{ck}, f) = C$ and $\deg(f) \leq n$ and $f(\alpha_i) = y_i$ for all i.
- Holds for KZG under ARSDH (Adaptive Rational Strong Diffie-Hellman) assumption.

ARSDH Assumption

ARSDH (Adaptive Rational Strong Diffie-Hellman):

- Adversary $\mathcal{A}$ gets $\mathsf{ck} \leftarrow ([1, \tau, \ldots, \tau^n]_1, [1, \tau]_2)$.

ARSDH Assumption

ARSDH (Adaptive Rational Strong Diffie-Hellman):

- Adversary $\mathcal{A}$ gets ck $\leftarrow ([1, \tau, \ldots, \tau^n]_1, [1, \tau]_2)$.
- Outputs $[g]_1, [\varphi]_1$, set $S \subseteq \mathbb{F}$

ARSDH Assumption

ARSDH (Adaptive Rational Strong Diffie-Hellman):

- Adversary $\mathcal{A}$ gets ck $\leftarrow ([1, \tau, \ldots, \tau^n]_1, [1, \tau]_2)$.
- Outputs $[g]_1, [\varphi]_1$, set $S \subseteq \mathbb{F}$
- $\mathcal{A}$ breaks assumption if

ARSDH Assumption

ARSDH (Adaptive Rational Strong Diffie-Hellman):

- Adversary $\mathcal{A}$ gets ck $\leftarrow ([1, \tau, \ldots, \tau^n]_1, [1, \tau]_2)$.
- Outputs $[g]_1, [\varphi]_1$, set $S \subseteq \mathbb{F}$
- $\mathcal{A}$ breaks assumption if
  - $|S| = n + 1$,

ARSDH Assumption

ARSDH (Adaptive Rational Strong Diffie-Hellman):

- Adversary $\mathcal{A}$ gets ck $\leftarrow ([1, \tau, \ldots, \tau^n]_1, [1, \tau]_2)$.
- Outputs $[g]_1, [\varphi]_1$, set $S \subseteq \mathbb{F}$
- $\mathcal{A}$ breaks assumption if
  - $|S| = n + 1$,
  - $[g]_1 \neq [0]_1$,

ARSDH Assumption

ARSDH (Adaptive Rational Strong Diffie-Hellman):

- Adversary $\mathcal{A}$ gets ck $\leftarrow ([1, \tau, \ldots, \tau^n]_1, [1, \tau]_2)$.
- Outputs $[g]_1, [\varphi]_1$, set $S \subseteq \mathbb{F}$
- $\mathcal{A}$ breaks assumption if
  - $|S| = n + 1$,
  - $[g]_1 \neq [0]_1$,
  - $e([g]_1, [1]_2) = e([\varphi]_1, [Z_S(\tau)]_2)$, where $Z_S(X) = \prod_{s \in S}(X - s)$.

Roberto Talk’s

- Lipmaa, Parisella, Me (Crypto 2025)

Roberto Talk's

- Lipmaa, Parisella, Me (Crypto 2025)
- **Sanitized** KZG linearization: special sound under ARSDH.

Roberto Talk's

- Lipmaa, Parisella, Me (Crypto 2025)
- **Sanitized** KZG linearization: special sound under ARSDH.
  - Real-world SNARKs use non-sanitized version.

Roberto Talk's

- Lipmaa, Parisella, Me (Crypto 2025)
- **Sanitized** KZG linearization: special sound under ARSDH.
  - Real-world SNARKs use non-sanitized version.
- Plonk has computational special soundness

Roberto Talk's

- Lipmaa, Parisella, Me (Crypto 2025)
- **Sanitized** KZG linearization: special sound under ARSDH.
  - Real-world SNARKs use non-sanitized version.
- Plonk has computational special soundness
  - ARSDH assumption

Roberto Talk's

- Lipmaa, Parisella, Me (Crypto 2025)
- **Sanitized** KZG linearization: special sound under ARSDH.
  - Real-world SNARKs use non-sanitized version.
- Plonk has computational special soundness
  - ARSDH assumption
  - More specialized assumption related to Plonk: SplitRSDH

Roberto Talk's

- Lipmaa, Parisella, Me (Crypto 2025)
- **Sanitized** KZG linearization: special sound under ARSDH.
  - Real-world SNARKs use non-sanitized version.
- Plonk has computational special soundness
  - ARSDH assumption
  - More specialized assumption related to Plonk: SplitRSDH
  - Needed because of linearization

Roberto Talk's

- Lipmaa, Parisella, Me (Crypto 2025)
- **Sanitized** KZG linearization: special sound under ARSDH.
  - Real-world SNARKs use non-sanitized version.
- Plonk has computational special soundness
  - ARSDH assumption
  - More specialized assumption related to Plonk: SplitRSDH
  - Needed because of linearization
- Can we use only ARSDH?

Formalizing the Trick: Linearization PCS (LPCS)

I define a new primitive to formalize the linearization technique:

Formalizing the Trick: Linearization PCS (LPCS)

I define a new primitive to formalize the linearization technique:

- **Linearization PCS**:

Formalizing the Trick: Linearization PCS (LPCS)

I define a new primitive to formalize the linearization technique:

- **Linearization PCS**:
  - KGen(n) : commitment key ck for degree $\leq n$ polynomials.

Formalizing the Trick: Linearization PCS (LPCS)

I define a new primitive to formalize the linearization technique:

- **Linearization PCS**:
  - KGen(n) : commitment key ck for degree $\leq n$ polynomials.
  - $Com(\text{ck}, \mathbf{f})$: Commits to $\mathbf{f} = (f_1, \ldots, f_m)$.

Formalizing the Trick: Linearization PCS (LPCS)

I define a new primitive to formalize the linearization technique:

- **Linearization PCS**:
  - KGen(n) : commitment key ck for degree $\leq n$ polynomials.
  - *Com*(ck, $\mathbf{f}$) : Commits to $\mathbf{f} = (f_1, \ldots, f_m)$.
  - *Open*(ck, $\mathbf{g}, \mathbf{f}, \alpha$) : Defines $\mathcal{V}(X) = \sum_{i=1}^{m} g_i(X) f_i(X)$

Formalizing the Trick: Linearization PCS (LPCS)

I define a new primitive to formalize the linearization technique:

- **Linearization PCS**:
  - KGen(n) : commitment key ck for degree $\leq n$ polynomials.
  - *Com*(ck, $\mathbf{f}$) : Commits to $\mathbf{f} = (f_1, \ldots, f_m)$.
  - *Open*(ck, $\mathbf{g}, \mathbf{f}, \alpha$) : Defines $\mathcal{V}(X) = \sum_{i=1}^{m} g_i(X) f_i(X)$
    - Sends $y = \mathcal{V}(\alpha)$, and proof π.

Formalizing the Trick: Linearization PCS (LPCS)

I define a new primitive to formalize the linearization technique:

- **Linearization PCS**:
  - KGen(n) : commitment key ck for degree $\leq n$ polynomials.
  - *Com*(ck, $\mathbf{f}$) : Commits to $\mathbf{f} = (f_1, \ldots, f_m)$.
  - *Open*(ck, $\mathbf{g}, \mathbf{f}, \alpha$) : Defines $\mathcal{V}(X) = \sum_{i=1}^{m} g_i(X) f_i(X)$
    - Sends $y = \mathcal{V}(\alpha)$, and proof π.
  - *Verify*(ck, $C, \mathbf{g}_\alpha, \alpha, y, \pi$) : where $\mathbf{g}_\alpha = (g_1(\alpha), \ldots, g_m(\alpha))$ and outputs **Accept** or **Reject**.

Formalizing the Trick: Linearization PCS (LPCS)

I define a new primitive to formalize the linearization technique:

- **Linearization PCS**:
  - KGen(n) : commitment key ck for degree $\leq n$ polynomials.
  - *Com*(ck, $\mathbf{f}$) : Commits to $\mathbf{f} = (f_1, \ldots, f_m)$.
  - *Open*(ck, $\mathbf{g}, \mathbf{f}, \alpha$) : Defines $\mathcal{V}(X) = \sum_{i=1}^{m} g_i(X) f_i(X)$
    - Sends $y = \mathcal{V}(\alpha)$, and proof π.
  - *Verify*(ck, $C, \mathbf{g}_\alpha, \alpha, y, \pi$) : where $\mathbf{g}_\alpha = (g_1(\alpha), \ldots, g_m(\alpha))$ and outputs **Accept** or **Reject**.

Two Extractability Notions:

Formalizing the Trick: Linearization PCS (LPCS)

I define a new primitive to formalize the linearization technique:

- **Linearization PCS**:
  - KGen(n) : commitment key ck for degree $\leq n$ polynomials.
  - *Com*(ck, $\mathbf{f}$) : Commits to $\mathbf{f} = (f_1, \ldots, f_m)$.
  - *Open*(ck, $\mathbf{g}, \mathbf{f}, \alpha$) : Defines $\mathcal{V}(X) = \sum_{i=1}^{m} g_i(X) f_i(X)$
    - Sends $y = \mathcal{V}(\alpha)$, and proof π.
  - *Verify*(ck, $C, \mathbf{g}_\alpha, \alpha, y, \pi$) : where $\mathbf{g}_\alpha = (g_1(\alpha), \ldots, g_m(\alpha))$ and outputs **Accept** or **Reject**.

Two Extractability Notions:

1. **CSS**: Extract all $f_i(X)$ of degree $\leq n$.

Formalizing the Trick: Linearization PCS (LPCS)

I define a new primitive to formalize the linearization technique:

- **Linearization PCS**:
  - KGen(n) : commitment key ck for degree $\leq n$ polynomials.
  - *Com*(ck, $\mathbf{f}$) : Commits to $\mathbf{f} = (f_1, \ldots, f_m)$.
  - *Open*(ck, $\mathbf{g}, \mathbf{f}, \alpha$) : Defines $\mathcal{V}(X) = \sum_{i=1}^{m} g_i(X) f_i(X)$
    - Sends $y = \mathcal{V}(\alpha)$, and proof π.
  - *Verify*(ck, $C, \mathbf{g}_\alpha, \alpha, y, \pi$) : where $\mathbf{g}_\alpha = (g_1(\alpha), \ldots, g_m(\alpha))$ and outputs **Accept** or **Reject**.

Two Extractability Notions:

1. **CSS**: Extract all $f_i(X)$ of degree $\leq n$.
2. **Weak CSS (wCSS)**: Extract only $\mathcal{V}(X)$ of degree $\leq n + d$, where $\deg(g_i) \leq d$.

Security of Plonk

LinKZG: Linearization PCS constructed from KZG.

Security of Plonk

LinKZG: Linearization PCS constructed from KZG.

- **Not CSS**: prior work.

Security of Plonk

LinKZG: Linearization PCS constructed from KZG.

- **Not CSS**: prior work.
- **Weak CSS**: Under a **target group** ARSDH assumption.

Security of Plonk

LinKZG: Linearization PCS constructed from KZG.

- **Not CSS**: prior work.
- **Weak CSS**: Under a **target group** ARSDH assumption.
- **Corollary**: Plonk is secure (knowledge sound) under target group ARSDH.

Security of Plonk

LinKZG: Linearization PCS constructed from KZG.

- **Not CSS**: prior work.
- **Weak CSS**: Under a **target group** ARSDH assumption.
- **Corollary**: Plonk is secure (knowledge sound) under target group ARSDH.
  - $\Rightarrow$ Plonk is simulation extractable under target group ARSDH (Lipmaa, TCC 2025).

ARSDH Assumption

ARSDH (Adaptive Rational Strong Diffie-Hellman):

- Adversary $\mathcal{A}$ gets ck $\leftarrow ([1, \tau, \ldots, \tau^n]_1, [1, \tau]_2)$.
- Outputs $[g]_1, [\varphi]_1$, set $S \subseteq \mathbb{F}$
- $\mathcal{A}$ breaks assumption if
  - $|S| = n + 1$,
  - $[g]_1 \neq [0]_1$,
  - $e([g]_1, [1]_2) = e([\varphi]_1, [Z_S(\tau)]_2)$, where $Z_S(X) = \prod_{s \in S}(X - s)$.

Target Group ARSDH Assumption

Target Group ARSDH (Adaptive Rational Strong Diffie-Hellman):

- Adversary $\mathcal{A}$ gets ck $\leftarrow ([1, \tau, \ldots, \tau^n]_1, [1, \tau, \ldots, \tau^m]_2)$.
- Outputs $[g]_1, [g]_T, [\varphi]_1$, set $S \subseteq \mathbb{F}$
- $\mathcal{A}$ breaks assumption if
  - $|S| = n + m + 1$,
  - $[g]_1 \neq [0]_1$,
  - $[g]_T = e([\varphi]_1, [Z_S(\tau)]_2)$, where $Z_S(X) = \prod_{s \in S}(X - s)$.

Other Linearization PCSs

Any homomorphic PCSs can be turned into linearization PCS:

Other Linearization PCSs

Any homomorphic PCSs can be turned into linearization PCS:

- **Construction**: Prover commits to f_i component-wise; verifier uses homomorphic operations to create a commitment to the linearized polynomial which is opened at α.

Other Linearization PCSs

Any homomorphic PCSs can be turned into linearization PCS:

- **Construction**: Prover commits to f_i component-wise; verifier uses homomorphic operations to create a commitment to the linearized polynomial which is opened at α.
- Are they all secure?

Other Linearization PCSs

Any homomorphic PCSs can be turned into linearization PCS:

- **Construction**: Prover commits to f_i component-wise; verifier uses homomorphic operations to create a commitment to the linearized polynomial which is opened at α.
- Are they all secure?
  - I don't know. :(

Other Linearization PCSs

Any homomorphic PCSs can be turned into linearization PCS:

- **Construction**: Prover commits to f_i component-wise; verifier uses homomorphic operations to create a commitment to the linearized polynomial which is opened at α.
- Are they all secure?
  - I don't know. :(
- **Some are secure**: Bulletproofs, Hyrax, Dory (Pedersen-style commitments).

Other Linearization PCSs

Any homomorphic PCSs can be turned into linearization PCS:

- **Construction**: Prover commits to f_i component-wise; verifier uses homomorphic operations to create a commitment to the linearized polynomial which is opened at α.
- Are they all secure?
  - I don't know. :(
- **Some are secure**: Bulletproofs, Hyrax, Dory (Pedersen-style commitments).
  - Proof non-trivial: either AGM or linear independence of $\{g_i\}_i$.

Other Linearization PCSs

Any homomorphic PCSs can be turned into linearization PCS:

- **Construction**: Prover commits to f_i component-wise; verifier uses homomorphic operations to create a commitment to the linearized polynomial which is opened at α.
- Are they all secure?
  - I don't know. :(
- **Some are secure**: Bulletproofs, Hyrax, Dory (Pedersen-style commitments).
  - Proof non-trivial: either AGM or linear independence of $\{g_i\}_i$.

Efficiency improvements for SNARKs based on these PCSs.

Impact & Open Problems

Impact:

- Formalization of a widely used "trick" in SNARKs.

Impact & Open Problems

Impact:

- Formalization of a widely used "trick" in SNARKs.
- Plonk's security under one non-tailored assumption (Target Group ARSDH).

Impact & Open Problems

Impact:

- Formalization of a widely used "trick" in SNARKs.
- Plonk's security under one non-tailored assumption (Target Group ARSDH).
- Linearization trick in other homomorphic PCSs.

Impact & Open Problems

Impact:

- Formalization of a widely used "trick" in SNARKs.
- Plonk's security under one non-tailored assumption (Target Group ARSDH).
- Linearization trick in other homomorphic PCSs.

Future directions:

Impact & Open Problems

Impact:

- Formalization of a widely used "trick" in SNARKs.
- Plonk's security under one non-tailored assumption (Target Group ARSDH).
- Linearization trick in other homomorphic PCSs.

Future directions:

- Paper mainly focuses on KZG.

Impact & Open Problems

Impact:

- Formalization of a widely used "trick" in SNARKs.
- Plonk's security under one non-tailored assumption (Target Group ARSDH).
- Linearization trick in other homomorphic PCSs.

Future directions:

- Paper mainly focuses on KZG.
- I explored other homomorphic PCSs quite superficially.

Impact & Open Problems

Impact:

- Formalization of a widely used "trick" in SNARKs.
- Plonk's security under one non-tailored assumption (Target Group ARSDH).
- Linearization trick in other homomorphic PCSs.

Future directions:

- Paper mainly focuses on KZG.
- I explored other homomorphic PCSs quite superficially.
- What about lattice-based PCSs?

Impact & Open Problems

Impact:

- Formalization of a widely used "trick" in SNARKs.
- Plonk's security under one non-tailored assumption (Target Group ARSDH).
- Linearization trick in other homomorphic PCSs.

Future directions:

- Paper mainly focuses on KZG.
- I explored other homomorphic PCSs quite superficially.
- What about lattice-based PCSs?

Thank you! Questions?