Uni/Multi Variate Polynomial Embeddings for zkSNARKs

Guang Gong

Department of Electrical and Computer Engineering
University of Waterloo
CANADA
https://uwaterloo.ca/scholar/ggong

Boolean Functions and their Applications (BFA)
September 3-8, 2023, Fleischer's Hotel, Voss, Norway

Outline

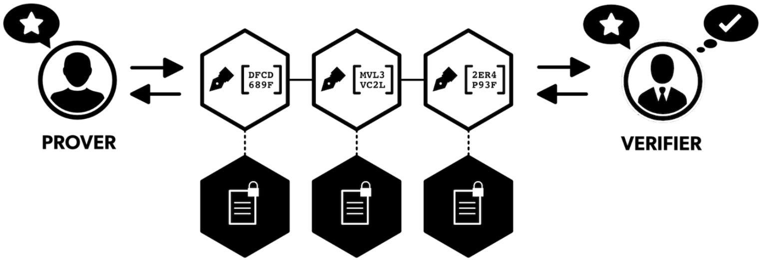
- Overview of zero-knowledge proofs and **zkSNARKs**
- **Polynomial embeddings** for R1CS relation
- **Uni/Multi variate** polynomial embeddings for R1CS - Polaris/Spartan protocols
- Efficiency comparisons for different encoding methods
- Concluding remarks and some open problems

Motivation: blockchain privacy

- **Blockchain**, a decentralized peer-to-peer (P2P) ledger system, in addition of applications in cryptocurrency, is gaining interest to many different applications, such as
  - decentralized identity management,
  - supply chain management,
  - private data management,
  - ...
- Blockchains can provide **trusted** consensus, computation, and immutable data between untrusted entities.
- However, those applications need **privacy**!
- Tool for blockchain privacy: zero-knowledge proofs.

Zero-Knowledge Proofs

Loosely speaking, zero-knowledge proofs are **proofs** that yields **nothing** beyond the validity of the assertion.



Zero-Knowledge Proofs (cont.)

$\mathcal{P}$rover
Alice

X = "I have x Bitcoin"

$\mathcal{V}$erifier
Bob

I believe X is true.
But I do not know why!

- **Completeness**: $\mathcal{P}$ can convince $\mathcal{V}$ if X is true
- **Soundness**: No malicious $\mathcal{P}^*$ cannot convince $\mathcal{V}$ if X is not true
- **Zero Knowledge**: $\mathcal{V}^*$ learns nothing except for the validity of X

UNIVERSITY OF WATERLOO

Zero-Knowledge Proofs (cont.)

$\mathcal{P}$rover
Alice

X = "I have x Bitcoin"

$\mathcal{V}$erifier
Bob

I believe X is true.
But I do not know why!

- **Completeness**: $\mathcal{P}$ can convince $\mathcal{V}$ if X is true
- **Soundness**: No malicious $\mathcal{P}^*$ cannot convince $\mathcal{V}$ if X is not true
- **Zero Knowledge**: $\mathcal{V}^*$ learns nothing except for the validity of X

Zero-Knowledge Proofs (cont.)

$\mathcal{P}$rover
Alice

X = "I have x Bitcoin"

$\mathcal{V}$erifier
Bob

I believe X is true.
But I do not know why!

- **Completeness**: $\mathcal{P}$ can convince $\mathcal{V}$ if X is true
- **Soundness**: No malicious $\mathcal{P}^*$ cannot convince $\mathcal{V}$ if X is not true
- **Zero Knowledge**: $\mathcal{V}^*$ learns nothing except for the validity of X

Zero-Knowledge Proofs (cont.)

$\mathcal{P}$rover
Alice

X = "I have x Bitcoin"

$\mathcal{V}$erifier
Bob

I believe X is true.
But I do not know why!

- **Completeness**: $\mathcal{P}$ can convince $\mathcal{V}$ if X is true
- **Soundness**: No malicious $\mathcal{P}^*$ cannot convince $\mathcal{V}$ if X is not true
- **Zero Knowledge**: $\mathcal{V}^*$ learns nothing except for the validity of X

UNIVERSITY OF WATERLOO

ZKP efficiency

- **Prover complexity**: Computational cost for the prover to run the protocol.
- **Round** complexity: Number of transmissions between prover and verifier.
- **Proof length (or communication)**: Total size of communication between prover and verifier.
- **Verifier** complexity: Computational cost for the verifier.
- **Setup cost:** Size of setup parameters, e.g. a **common reference string (CRS)**, and computational cost of creating the setup.

How about integrity of computation?

$\mathcal{P}$rover
Alice

y →

$\mathcal{V}$Verifier
Bob

- How can Alice to prove to Bob that a hash value $y = h(x)$ is correctly evaluated without sending Bob the pre-image x?
- In other words, how can the prover convince the verifier the following NP statement without giving out x:

$$X = \{ \textbf{ I know that } x \textbf{ such that } y = f(x). \}$$

Verifiable computation

The integrity of computation is achieved by **verifiable computation**. It can be done through representing an algorithm/program as a circuit.

A special ZK class: **zkSNARK**

zkSNARK

zero-**k**nowledge **S**uccinct **N**on-interactive **AR**gument of **K**nowledge.

Properties of zkSNARK

- **Z**ero-**K**nowledge: does not leak any information about witness
- **S**uccinct: Proof size is independent of NP witness sizes, i.e., the computing complexity of the prover/verifier and communication (i.e., the proof length) are computationally bounded.
- **N**on-interactive: only one message is sent by prover.
- **AR**gument of **K**nowledge.

UNIVERSITY OF WATERLOO

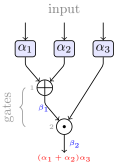
Constructions of zkSNARKs

A general approach for zkSNARKs consists of four steps:

1. Convert a program/algorithm to an **arithmetic circuit**.
2. Convert the arithmetic circuit to **polynomials**.
3. Build an argument to **prove** something about the polynomial using **(fully) homomorphic encryption** or **probabilistic checkable proof (PCP)** with error correcting codes.
4. **Add** zero-knowledge and using **Fiat-Shamir transform** to convert interactive to non-interactive if not done in the steps 2 and 3.

In the rest of the talk, we focus on Step 2.

Some recent zkSNARKs



Properties of different zkSNARK schemes			
scheme	setup	security	implementation
QAP/QSP based (GGPR13, Groth16) (BCTV14a)	private	KOE	libsnark (BCTV14) Pinocchio, Zcach Hawk
Bullet proof (BCCGP16)	public	DLOG	experiments
Marlin (CHMMVW20)	private	Strong DH	experiments
Spartan$_{DL,OR}$ (Setty20)	public	DLOG, (CRH, PRG)	experiments
Ligero (AHIV17)	public	CRH, PRG	Ligero cryptocurrency
Stark (BBHR18)	public	CRH, PRG	libstark
Aurora (BCRSVW19)	public	CRH, PRG	libiop
Virgo (ZXZS20)	public	CRH, PRG	security below 128 bit
Polaris (HG2022)	public	CRH, PRG	partial tests

Rank 1 Constraint Satisfiability (R1CS) Relation

From now on, we assume that we have obtained R1CS relation from a circuit converted from a given algorithm/program.

R1CS instance

$\mathcal{T} = (\mathbb{F}, A, B, C, v, m, n)$ and corresponding **witness** w

- A, B, C are $m \times m$ matrices over a large finite field $\mathbb{F}$ representing the computation circuit
- v is the public input and output vector of the instance
- w is the private input vector of the instance
- there are at most n non-zero entries in each matrix

UNIVERSITY OF WATERLOO

R1CS relation

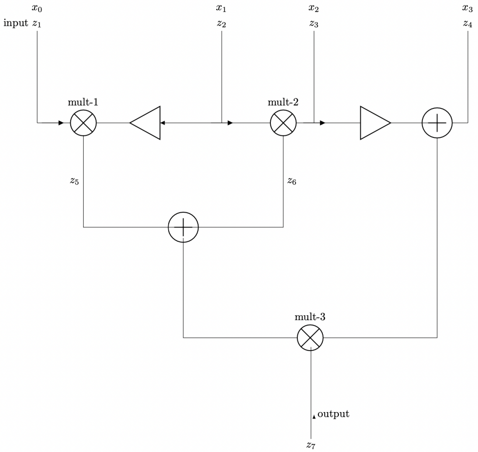
There exists a witness $w \in \mathbb{F}^{m-|v|-1}$ such that

$$(A \cdot z) \circ (B \cdot z) - (C \cdot z) = \vec{\mathbf{0}},$$

where $z := (1, w, v) \in \mathbb{F}^m$, "$\cdot$" is the matrix-vector product, and "$\circ$" denotes the Hadamard product (i.e., term-wise product).

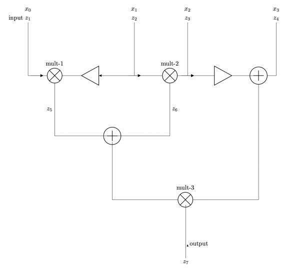
- The **goal** of a zkSNARK scheme is to prove the above relation.
- R1CS relation generalizes the problem of arithmetic circuit satisfiability.
- For the three matrices A, B, C, the vectors Az, Bz and Cz represent the **left input, right input and output** vectors of the **multiplicative gates** in the circuit respectively. The witness w consists of the circuit's private input and wire values.

Example – a Boolean circuit with three AND gates



Example – R1CS instance

- $z = (z_0, z_1, \cdots, z_7)$ where $z_0 = 1$.



AND i	$g_l \cdot g_r - g_o = 0$
1	$z_1 \cdot (1 \oplus z_2) - z_5 = 0$
2	$z_2 \cdot z_3 - z_6 = 0$
3	$(z_5 \oplus z_6) \cdot (1 \oplus z_3 \oplus z_4) - z_7 = 0$

- Encoding the circuit to an R1CS instance: A

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} \implies A\mathbf{z} = \begin{pmatrix} z_1 \\ z_2 \\ z_5 \oplus z_6 \end{pmatrix}$$

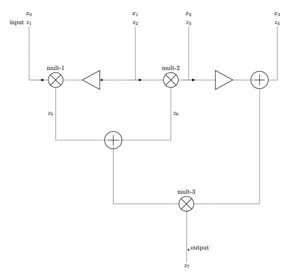
- Encoding the circuit to an R1CS instance: B, C

$$B = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{pmatrix} \implies B\mathbf{z} = \begin{pmatrix} 1 \oplus z_2 \\ z_3 \\ 1 \oplus z_3 \oplus z_4 \end{pmatrix}$$

$$C = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \implies C\mathbf{z} = \begin{pmatrix} z_5 \\ z_6 \\ z_7 \end{pmatrix}$$

Example – R1CS instance

- $z = (z_0, z_1, \cdots, z_7)$ where $z_0 = 1$.



AND i	$g_l \cdot g_r - g_o = 0$
1	$z_1 \cdot (1 \oplus z_2) - z_5 = 0$
2	$z_2 \cdot z_3 - z_6 = 0$
3	$(z_5 \oplus z_6) \cdot (1 \oplus z_3 \oplus z_4) - z_7 = 0$

- **Encoding the circuit to an R1CS instance: A**

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} \Longrightarrow A\mathbf{z} = \begin{pmatrix} z_1 \\ z_2 \\ z_5 \oplus z_6 \end{pmatrix}$$

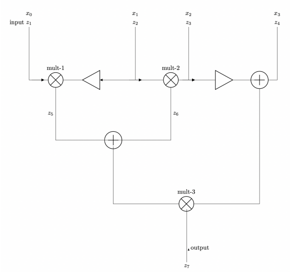
- **Encoding the circuit to an R1CS instance: B, C**

$$B = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{pmatrix} \Longrightarrow B\mathbf{z} = \begin{pmatrix} 1 \oplus z_2 \\ z_3 \\ 1 \oplus z_3 \oplus z_4 \end{pmatrix}$$

$$C = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \Longrightarrow C\mathbf{z} = \begin{pmatrix} z_5 \\ z_6 \\ z_7 \end{pmatrix}$$

Example – R1CS instance

- $z = (z_0, z_1, \cdots, z_7)$ where $z_0 = 1$.



AND i	$g_l \cdot g_r - g_o = 0$
1	$z_1 \cdot (1 \oplus z_2) - z_5 = 0$
2	$z_2 \cdot z_3 - z_6 = 0$
3	$(z_5 \oplus z_6) \cdot (1 \oplus z_3 \oplus z_4) - z_7 = 0$

- **Encoding the circuit to an R1CS instance: A**

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} \Longrightarrow A\mathbf{z} = \begin{pmatrix} z_1 \\ z_2 \\ z_5 \oplus z_6 \end{pmatrix}$$

- **Encoding the circuit to an R1CS instance: B, C**

$$B = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{pmatrix} \Longrightarrow B\mathbf{z} = \begin{pmatrix} 1 \oplus z_2 \\ z_3 \\ 1 \oplus z_3 \oplus z_4 \end{pmatrix}$$

$$C = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \Longrightarrow C\mathbf{z} = \begin{pmatrix} z_5 \\ z_6 \\ z_7 \end{pmatrix}$$

UNIVERSITY OF WATERLOO

Example – R1CS instance (cont.)

- R1CS relation

$$(A\mathbf{z}) \circ (B\mathbf{z}) - C\mathbf{z} = \mathbf{0} \quad (1)$$

where $\circ$ is the bit-wise Hadamard product in this case.

- In this case, we have a R1CS instance

$$(\mathbb{F}, A, B, C, v, m, n) = (GF(2^{3t}), A, B, C, 1, 8, 6)$$

where $m = 8$, the size of $\mathbf{z}$, $n = 6$, the maximum among the number of nonzero entries in each matrix, and

$$w = (z_1, \cdots, z_6), v = z_7.$$

- If we take

$$(z_1, z_2, z_3, z_4) = (1011) \Rightarrow (z_5, z_6, z_7) = (101)$$

then (1) is true. So, this is an R1CS instance. But if we take $\mathbf{z}' = 11011100$, then (1) is not true.

Example – R1CS instance (cont.)

- R1CS relation
$$(A\mathbf{z}) \circ (B\mathbf{z}) - C\mathbf{z} = \mathbf{0} \qquad (1)$$
where $\circ$ is the bit-wise Hadamard product in this case.
- In this case, we have a R1CS instance
$$(\mathbb{F}, A, B, C, v, m, n) = (GF(2^{3t}), A, B, C, 1, 8, 6)$$
where $m = 8$, the size of $\mathbf{z}$, $n = 6$, the maximum among the number of nonzero entries in each matrix, and
$$w = (z_1, \cdots, z_6), v = z_7.$$
- If we take
$$(z_1, z_2, z_3, z_4) = (1011) \Rightarrow (z_5, z_6, z_7) = (101)$$
then (1) is true. So, this is an R1CS instance. But if we take $\mathbf{z}' = 11011100$, then (1) is not true.

Example – R1CS instance (cont.)

- R1CS relation

$$(A\mathbf{z}) \circ (B\mathbf{z}) - C\mathbf{z} = \mathbf{0} \tag{1}$$

where $\circ$ is the bit-wise Hadamard product in this case.

- In this case, we have a R1CS instance

$$(\mathbb{F}, A, B, C, v, m, n) = (GF(2^{3t}), A, B, C, 1, 8, 6)$$

where $m = 8$, the size of $\mathbf{z}$, $n = 6$, the maximum among the number of nonzero entries in each matrix, and

$$w = (z_1, \cdots, z_6), v = z_7.$$

- If we take

$$(z_1, z_2, z_3, z_4) = (1011) \Rightarrow (z_5, z_6, z_7) = (101)$$

then (1) is true. So, this is an R1CS instance. But if we take $\mathbf{z}' = 11011100$, then (1) is not true.

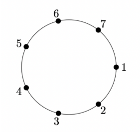
Encoding Methods

Two different methods to encode R1CS:

- to represent the matrices as **biivariate** polynomials and vector $\mathbf{z}$ as a **univariate** polynomial and
- to represent them as **multi-variate** polynomials.

Example

$\mathbf{z} = (11011101)$, let $\mathbb{F}_{2^3}$ be defined by the primitive polynomial $t(x) = x^3 + x + 1$ and $t(\alpha) = 0$:



univariate poly

$\mathbf{z} = (11011101)$

$f(x) = 1 + Tr(x) + Tr(\alpha^3 x^3)$

$z_0 = 1, z_i = f(\alpha^{i-1}),$

$i = 1, \cdots, 7$

$\Longrightarrow$

Trace representation of the sequence

(x_2, x_1, x_0)	$\mathbf{z}$
000	1
001	1
010	0
011	1
100	1
101	1
110	0
111	1

multivariate poly

$g(x_0, x_1, x_2) = 1 + x_1 + x_0 x_1$

$z_i = g(i_0, i_1, i_2),$

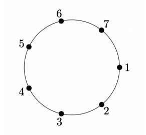
$i = i_0 + i_1 2 + i_2 2^2, i_j \in \mathbb{F}_2$

$\Longrightarrow$

Golay sequence representation

Example

$\mathbf{z} = (11011101)$, let $\mathbb{F}_{2^3}$ be defined by the primitive polynomial $t(x) = x^3 + x + 1$ and $t(\alpha) = 0$:



univariate poly
$\mathbf{z} = (11011101)$
$f(x) = 1 + Tr(x) + Tr(\alpha^3 x^3)$
$z_0 = 1, z_i = f(\alpha^{i-1})$,
$i = 1, \cdots, 7$
$\Longrightarrow$
Trace representation of the sequence

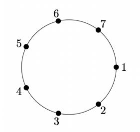
(x_2, x_1, x_0)	$\mathbf{z}$
000	1
001	1
010	0
011	1
100	1
101	1
110	0
111	1

multivariate poly

$g(x_0, x_1, x_2) = 1 + x_1 + x_0 x_1$
$z_i = g(i_0, i_1, i_2)$,
$i = i_0 + i_1 2 + i_2 2^2, i_j \in \mathbb{F}_2$
$\Longrightarrow$
Golay sequence representation

Example

$\mathbf{z} = (11011101)$, let $\mathbb{F}_{2^3}$ be defined by the primitive polynomial $t(x) = x^3 + x + 1$ and $t(\alpha) = 0$:



univariate poly
$\mathbf{z} = (\mathbf{11011101})$
$f(x) = 1 + Tr(x) + Tr(\alpha^3 x^3)$
$z_0 = 1, z_i = f(\alpha^{i-1})$,
$i = 1, \cdots, 7$
$\Longrightarrow$
Trace representation of the sequence

(x_2, x_1, x_0)	$\mathbf{z}$
000	**1**
001	**1**
010	**0**
011	**1**
100	1
101	1
110	0
111	1

multivariate poly

$g(x_0, x_1, x_2) = 1 + x_1 + x_0 x_1$
$z_i = g(i_0, i_1, i_2)$,
$i = i_0 + i_1 2 + i_2 2^2, i_j \in \mathbb{F}_2$
$\Longrightarrow$
Golay sequence representation

Detour: some basic properties of uni/multi variate polynomials

- Given any sequence of length $N = 2^s$ over $\mathbb{F}$, say $\mathbf{u} = (u_0, \cdots, u_{2^s-1})$, we can represent it as a univariate polynomial, say $f(x)$ through Lagrange interpolation over the evaluating set $H = \{\alpha_0, \cdots, \alpha_{2^s-1}\} \subset \mathbb{F}$:

$$f(x) = \sum_{i=0}^{2^s-1} u_i \sigma_i(x), f(\alpha_i) = u_i, i = 0, \cdots, 2^s - 1,$$

  where $\{\sigma_i(x)\}$ is the Lagrange basis.
- The request of $N = 2^s$ is to facilitate a fast computation through **Fast Fourier transform (FFT)** and inverse FFT (Lagrange interpolation).

Bivariate polynomial $\Delta_H(x, y)$

- Let H be an **s-dimensional affine space** of $\mathbb{F}$ (so in this case, $\mathbb{F}$ has characteristic 2), and

$$Z_H(x) = \prod_{a \in H}(x + a) = x^{2^s} + \sum_{i=1}^{s} c_i x^{2^{i-1}}, c_i \in \mathbb{F}$$

  a linearized polynomial.

- Define

$$\Delta_H(x, y) = \frac{Z_H(x) + Z_H(y)}{x + y}, \tag{2}$$

- Then the Lagrange basis element $\sigma_i(x)$ becomes

$$\sigma_i(x) = \frac{\Delta_H(x, \alpha_i)}{c_1} = \frac{1}{c_1}\frac{Z_H(x)}{x + \alpha_i}, 0 \leq i < 2^s$$

  where c_1 is the coefficient of x in $Z_H(x)$.

- The matrices A, B, C can be represented by bivariate polynomial $\Delta_H(x, y)$, and the witness vector $\mathbf{z}$ can be represented by $Z_H(y)$.

Bivariate polynomial $\Delta_H(x, y)$

- Let H be an **s-dimensional affine space** of $\mathbb{F}$ (so in this case, $\mathbb{F}$ has characteristic 2), and

$$Z_H(x) = \prod_{a \in H}(x + a) = x^{2^s} + \sum_{i=1}^{s} c_i x^{2^{i-1}}, c_i \in \mathbb{F}$$

a linearized polynomial.

- Define

$$\Delta_H(x, y) = \frac{Z_H(x) + Z_H(y)}{x + y}, \tag{2}$$

- Then the Lagrange basis element $\sigma_i(x)$ becomes

$$\sigma_i(x) = \frac{\Delta_H(x, \alpha_i)}{c_1} = \frac{1}{c_1}\frac{Z_H(x)}{x + \alpha_i}, 0 \le i < 2^s$$

where c_1 is the coefficient of x in $Z_H(x)$.

- The matrices A, B, C can be represented by bivariate polynomial $\Delta_H(x, y)$, and the witness vector $\mathbf{z}$ can be represented by $Z_H(y)$.

UNIVERSITY OF WATERLOO

Bivariate polynomial $\Delta_H(x, y)$

- Let H be an **s-dimensional affine space** of $\mathbb{F}$ (so in this case, $\mathbb{F}$ has characteristic 2), and

$$Z_H(x) = \prod_{a \in H}(x + a) = x^{2^s} + \sum_{i=1}^{s} c_i x^{2^{i-1}}, c_i \in \mathbb{F}$$

  a linearized polynomial.

- Define

$$\Delta_H(x, y) = \frac{Z_H(x) + Z_H(y)}{x + y}, \qquad (2)$$

- Then the Lagrange basis element $\sigma_i(x)$ becomes

$$\sigma_i(x) = \frac{\Delta_H(x, \alpha_i)}{c_1} = \frac{1}{c_1}\frac{Z_H(x)}{x + \alpha_i}, 0 \le i < 2^s$$

  where c_1 is the coefficient of x in $Z_H(x)$.

- The matrices A, B, C can be represented by bivariate polynomial $\Delta_H(x, y)$, and the witness vector $\mathbf{z}$ can be represented by $Z_H(y)$.

UNIVERSITY OF WATERLOO

Multivariate polynomial encodings of sequences

- For a sequence $\mathbf{u} = (u_0, \cdots, u_{N-1})$, we associate it with a function $f(t): \mathbb{Z}_N \to \mathbb{F}$ by

$$f(t) = u_t, 0 \leq t < N$$

i.e.,

$$\mathbf{u} = (f(0), f(1), \cdots, f(N-1)).$$

- For any $\forall x \in \mathbb{Z}_N$,

$$x = \sum_{v=0}^{s-1} x_v \cdot 2^v \leftrightarrow \mathbf{x} = (x_0, x_1, \cdots, x_{s-1}), x_v \in \{0, 1\}.$$

- Let

$$\delta_t(\boldsymbol{x}) = \prod_{i=0}^{s-1} (x_i t_i + (1 - x_i)(1 - t_i)). \quad (3)$$

- Then any function f: $\mathbb{Z}_N \to \mathbb{F}$ can be represented by

$$f(\boldsymbol{x}) = \sum_{t=0}^{2^s - 1} f(t) \delta_t(\boldsymbol{x}). \quad (4)$$

The representation of Golay sequences!

UNIVERSITY OF WATERLOO

Embedding of multilinear extension

- When x_i and t_i take values in $\mathbb{F}$,

$$\tilde{f}(\boldsymbol{x}) = \sum_{\boldsymbol{t} \in \{0,1\}^s} f(\boldsymbol{t})\delta_{\boldsymbol{t}}(\boldsymbol{x}), \boldsymbol{x} \in \mathbb{F}^s. \quad (5)$$

  is called a **embedding of $f(x)$ or a multi-linear extension (MLE) of** $f(\boldsymbol{x})$ from $\{0,1\}^s \mapsto \mathbb{F}$ to $\mathbb{F}^s \mapsto \mathbb{F}$.

UNIVERSITY OF WATERLOO

Uni/multi variate embeddings of R1CS

For a given $m \times m$ matrix $A = (a_{ij})$ over $\mathbb{F}$, the prover needs to compute the following Lagrange interpolated polynomials ($m = 2^s$):

-
$$A(x, y) = \frac{1}{c_1^2} \sum_{(i,j)\in[2^s]^2} a_{ij} \Delta_H(x, \alpha_i) \Delta_H(y, \alpha_j) \quad \textbf{Univariate in Polaris}$$
$$A(\mathbf{x}, \mathbf{y}) = \sum_{(i,j)\in[2^s]^2} a_{ij} \delta_{(i,j)}(\mathbf{x}, \mathbf{y}), (\mathbf{x}, \mathbf{y}) \in (\mathbb{F}^s)^2 \quad \textbf{MLE in Spartan} \tag{6}$$
  Note that $[2^s] = \{0, 1, \cdots, 2^s - 1\}$.
- From the property of $\Delta(x, y)$, we have the following simplified formulae
$$A(x, y) = \frac{1}{c_1^2} \sum_{(i,j)\in[2^s]^2} a_{ij} \frac{\mathbb{Z}_H(x)}{x+\alpha_i} \cdot \frac{\mathbb{Z}_H(y)}{y+\alpha_j} \tag{7}$$
- Similarly, we have $B(\cdot, \cdot), C(\cdot, \cdot)$.

Uni/multi variate embeddings of R1CS

For a given $m \times m$ matrix $A = (a_{ij})$ over $\mathbb{F}$, the prover needs to compute the following Lagrange interpolated polynomials ($m = 2^s$):

- $$A(x,y) = \frac{1}{c_1^2} \sum_{(i,j)\in[2^s]^2} a_{ij} \Delta_H(x,\alpha_i)\Delta_H(y,\alpha_j) \qquad \textbf{Univariate in Polaris}$$
  $$A(\mathbf{x},\mathbf{y}) = \sum_{(i,j)\in[2^s]^2} a_{ij}\delta_{(i,j)}(\mathbf{x},\mathbf{y}), (\mathbf{x},\mathbf{y}) \in (\mathbb{F}^s)^2 \qquad \textbf{MLE in Spartan} \tag{6}$$
  Note that $[2^s] = \{0, 1, \cdots, 2^s - 1\}$.
- From the property of $\Delta(x,y)$, we have the following simplified formulae
  $$A(x,y) = \frac{1}{c_1^2} \sum_{(i,j)\in[2^s]^2} a_{ij} \frac{\mathbb{Z}_H(x)}{x+\alpha_i} \cdot \frac{\mathbb{Z}_H(y)}{y+\alpha_j} \tag{7}$$
- Similarly, we have $B(\cdot,\cdot), C(\cdot,\cdot)$.

Uni/multi variate embeddings of R1CS

For a given $m \times m$ matrix $A = (a_{ij})$ over $\mathbb{F}$, the prover needs to compute the following Lagrange interpolated polynomials ($m = 2^s$):

- $$A(x, y) = \frac{1}{c_1^2} \sum_{(i,j) \in [2^s]^2} a_{ij} \Delta_H(x, \alpha_i) \Delta_H(y, \alpha_j) \qquad \textbf{Univariate in Polaris}$$
  $$A(\mathbf{x}, \mathbf{y}) = \sum_{(i,j) \in [2^s]^2} a_{ij} \delta_{(i,j)}(\mathbf{x}, \mathbf{y}), (\mathbf{x}, \mathbf{y}) \in (\mathbb{F}^s)^2 \qquad \textbf{MLE in Spartan} \tag{6}$$
  Note that $[2^s] = \{0, 1, \cdots, 2^s - 1\}$.
- From the property of $\Delta(x, y)$, we have the following simplified formulae
  $$A(x, y) = \frac{1}{c_1^2} \sum_{(i,j) \in [2^s]^2} a_{ij} \frac{\mathbb{Z}_H(x)}{x + \alpha_i} \cdot \frac{\mathbb{Z}_H(y)}{y + \alpha_j} \tag{7}$$
- Similarly, we have $B(\cdot, \cdot), C(\cdot, \cdot)$.

Uni/multi variate embeddings of R1CS (cont.)

-

Univariate	MLE
$\bar{A}(x) = \sum_{y \in H} A(x,y)Z(y)$ $\bar{B}(x) = \sum_{y \in H} B(x,y)Z(y)$ $\bar{C}(x) = \sum_{y \in H} C(x,y)Z(y)$	$\bar{A}(\mathbf{x}) = \sum_{\mathbf{y} \in \mathbb{Z}_2^s} A(\mathbf{x},\mathbf{y})Z(\mathbf{y})$ $\bar{B}(\mathbf{x}) = \sum_{\mathbf{y} \in \mathbb{Z}_2^s} B(\mathbf{x},\mathbf{y})Z(\mathbf{y})$ $\bar{C}(\mathbf{x}) = \sum_{\mathbf{y} \in \mathbb{Z}_2^s} C(\mathbf{x},\mathbf{y})Z(\mathbf{y})$

- Define $F_w(\cdot)$ that is used to encode the vector $\mathbf{z}$:

Univariate	MLE
$F_w(x) = \bar{A}(x) \cdot \bar{B}(x) - \bar{C}(x)$	$F_w(\mathbf{x}) = \bar{A}(\mathbf{x}) \cdot \bar{B}(\mathbf{x}) - \bar{C}(\mathbf{x})$

Lemma

A pair $(\mathcal{T}, w)$ is a valid instance-witness pair, i.e., $(\mathcal{T}, w) \in \mathcal{R}_{\mathrm{R1CS}}$ if and only if

- $F_w(x) = 0$ for any $x \in H$ if it is encoded by the univariate polynomial and
- $F_w(\mathbf{x}) = 0$ for any $x \in \{0,1\}^s$ if it is encoded by the MLE.

Uni/multi variate embeddings of R1CS (cont.)

-

Univariate	MLE
$\bar{A}(x) = \sum_{y \in H} A(x,y)Z(y)$ $\bar{B}(x) = \sum_{y \in H} B(x,y)Z(y)$ $\bar{C}(x) = \sum_{y \in H} C(x,y)Z(y)$	$\bar{A}(\mathbf{x}) = \sum_{\mathbf{y} \in \mathbb{Z}_2^s} A(\mathbf{x},\mathbf{y})Z(\mathbf{y})$ $\bar{B}(\mathbf{x}) = \sum_{\mathbf{y} \in \mathbb{Z}_2^s} B(\mathbf{x},\mathbf{y})Z(\mathbf{y})$ $\bar{C}(\mathbf{x}) = \sum_{\mathbf{y} \in \mathbb{Z}_2^s} C(\mathbf{x},\mathbf{y})Z(\mathbf{y})$

- Define $F_w(\cdot)$ that is used to encode the vector $\mathbf{z}$:

Univariate	MLE
$F_w(x) = \bar{A}(x) \cdot \bar{B}(x) - \bar{C}(x)$	$F_w(\mathbf{x}) = \bar{A}(\mathbf{x}) \cdot \bar{B}(\mathbf{x}) - \bar{C}(\mathbf{x})$

Lemma

A pair $(\mathcal{T}, w)$ is a valid instance-witness pair, i.e., $(\mathcal{T}, w) \in \mathcal{R}_{\mathrm{R1CS}}$ if and only if

- $F_w(x) = 0$ for any $x \in H$ if it is encoded by the **univariate polynomial** and
- $F_w(\mathbf{x}) = 0$ for any $x \in \{0,1\}^s$ if it is encoded by the MLE.

Uni/multi variate embeddings of R1CS (cont.)

-

Univariate	MLE
$\bar{A}(x) = \sum_{y \in H} A(x,y)Z(y)$ $\bar{B}(x) = \sum_{y \in H} B(x,y)Z(y)$ $\bar{C}(x) = \sum_{y \in H} C(x,y)Z(y)$	$\bar{A}(\mathbf{x}) = \sum_{\mathbf{y} \in \mathbb{Z}_2^s} A(\mathbf{x},\mathbf{y})Z(\mathbf{y})$ $\bar{B}(\mathbf{x}) = \sum_{\mathbf{y} \in \mathbb{Z}_2^s} B(\mathbf{x},\mathbf{y})Z(\mathbf{y})$ $\bar{C}(\mathbf{x}) = \sum_{\mathbf{y} \in \mathbb{Z}_2^s} C(\mathbf{x},\mathbf{y})Z(\mathbf{y})$

- Define $F_w(\cdot)$ that is used to encode the vector $\mathbf{z}$:

Univariate	MLE
$F_w(x) = \bar{A}(x) \cdot \bar{B}(x) - \bar{C}(x)$	$F_w(\mathbf{x}) = \bar{A}(\mathbf{x}) \cdot \bar{B}(\mathbf{x}) - \bar{C}(\mathbf{x})$

Lemma

A pair $(\mathcal{T}, w)$ is a valid instance-witness pair, i.e., $(\mathcal{T}, w) \in \mathcal{R}_{\mathrm{R1CS}}$ if and only if

- $F_w(x) = 0$ for any $x \in H$ if it is encoded by the **univariate polynomial** and
- $F_w(\mathbf{x}) = 0$ for any $\boldsymbol{x} \in \{0,1\}^s$ if it is encoded by the **MLE**.

Polaris Protocol: Univariate encoding

Given an R1CS instance over $\mathbb{F}$ $\mathcal{T} = (\mathbb{F}, A, B, C, v, m, n)$, encoded by univariate polynomials over H, an affine space of $\mathbb{F}$. $\mathcal{V}$ in Polaris checks

$$F_w(r_x) \stackrel{?}{=} G(r_x) \cdot \mathbb{Z}_H(r_x)$$

from the claims of $\mathcal{P}$.

- **Quad-check**: $\mathcal{P}$ computes $\bar{A}(r_x) = v_A$, $\bar{B}(r_x) = v_B$, and $\bar{C}(r_x) = v_C$, $G(r_x) = \eta$ and send (v_A, v_B, v_C, η) to $\mathcal{V}$ where $G(x)$ is committed through a polynomial commitment scheme. $\mathcal{V}$ computes $\gamma = \mathcal{Z}_H(r_x)$, verifies $\eta = G(r_x)$ by the polynomial commitment. If it is successful, $\mathcal{V}$ checks

  $$v_A \cdot v_B - v_C \stackrel{?}{=} \eta \cdot \gamma$$

  If it is true, continue. Otherwise, it rejects.

- **Lin-check**: $\mathcal{V}$ chooses r_A, r_B, $r_C \in \mathbb{F}$ uniformly at random, sends them to $\mathcal{P}$, and computes $c = r_A \cdot v_A + r_B \cdot v_B + r_C \cdot v_C$. $\mathcal{P}$ and $\mathcal{V}$ invoke the **univariate sumcheck protocol** together with **GKR protocol** to verify

  $$c \stackrel{?}{=} \sum_{y \in H} Q_{r_x}(y)$$

  where

  $$Q_{r_x}(y) := (r_A \cdot A(r_x, y) + r_B \cdot B(r_x, y) + r_C \cdot C(r_x, y)) \cdot Z(y).$$

Summary of encoding R1CS relation in Polaris

- **Quad-check. Product checking polynomial** $F_w(x)$ is converted to **Poly-SAT**

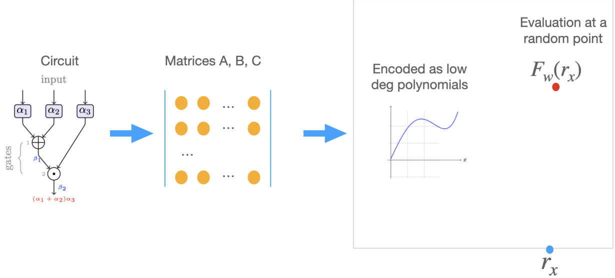
$$F_w(x) = \mathbb{Z}_H(x) \cdot G(x)$$

$$\Downarrow \Uparrow \text{soundness}$$

$$F_w(r_x) = \mathbb{Z}_H(r_x) \cdot G(r_x) \text{ for a random } r_x \in \mathbb{F} \setminus H$$

- **Lin-check. Univariate sum check together with GKR protocol** This is to check whether the validity of three evaluations: $v_A = \bar{A}(r_x)$, $v_B = \bar{B}(r_x)$, $v_C = \bar{C}(r_x)$ through a random combination:

$$c = r_A v_A + r_B v_B + r_C v_C$$



UNIVERSITY OF WATERLOO

Spartan Protocol: multivariate encoding

Given an R1CS instance over $\mathbb{F}$, $\mathcal{T} = (\mathbb{F}, A, B, C, v, m, n)$, encoded by multivariate polynomials. The verifier needs to check $\tilde{F}_w(\boldsymbol{x}) = 0, \forall \boldsymbol{x} \in \{0,1\}^s$. This converts to check

$\sum_{\boldsymbol{x}\in\{0,1\}^s} \tilde{F_w}(\boldsymbol{x})\delta_{\boldsymbol{t}_0}(\boldsymbol{x}) = 0$ **through the mutivariate sumcheck protocol converted to check**

$$\tilde{F_w}(\boldsymbol{x})\delta_{\boldsymbol{t}_0}(\boldsymbol{x}) = e_x, \boldsymbol{t}_0, \boldsymbol{r}_x \in_R \mathbb{F}^s.$$

- **Quad-check**: So $\mathcal{P}$ computes three claims: $\tilde{A}(\boldsymbol{r}_x) = v'_A$, $\tilde{B}(\boldsymbol{r}_x) = v'_B$, and $\tilde{C}(\boldsymbol{r}_x) = v'_C$, sends them to $\mathcal{V}$ and commits e_x. $\mathcal{V}$ computes $\delta_{\boldsymbol{t}_0}(\boldsymbol{r}_x)$ and checks

$$(v'_A v'_B - v'_C)\delta_{\boldsymbol{t}_0}(\boldsymbol{r}_x) \stackrel{?}{=} e_x.$$

  If it is true, continue. Otherwise, it rejects.
- **Lin-check**: $\mathcal{V}$ chooses $r'_A, r'_B, r'_C \in \mathbb{F}$ uniformly at random, sends them to $\mathcal{P}$, and computes $c' = r'_A \cdot v'_A + r'_B \cdot v'_B + r'_C \cdot v'_C$. $\mathcal{P}$ and $\mathcal{V}$ invoke the **multivariate sumcheck** protocol to verify

$$c' = \sum_{\mathbf{y}\in\{0,1\}^s} Q'_{\boldsymbol{r}_x}(\mathbf{y}) \Longrightarrow \text{to check } Q'_{\boldsymbol{r}_x}(\boldsymbol{r}_y) \stackrel{?}{=} e_y, \boldsymbol{r}_y \in_R \mathbb{F}^s, e_y \in \mathbb{F}.$$

Summary of two protocols

Recall $[2] = \{0, 1\}$.

Univariate	MLE
R1CS instance $F_w(x), x \in \mathbb{F}$	$F_w(\mathbf{x}), \mathbf{x} \in \mathbb{F}^s$
$F_w(x) = 0, \forall x \in H$ $F_w(x) = G_w(x)\mathbb{Z}_H(x)$	$F_w(\mathbf{x}) = 0, \forall \mathbf{x} \in [2]^s$ $J_w(\mathbf{t}) = \sum_{\mathbf{t}\in[2]^s} F_w(\mathbf{x})\delta_{\mathbf{t}}(\mathbf{x})$
To check $F_w(r_x) \stackrel{?}{=} G_w(r_x)\mathbb{Z}_H(r_x), r_x \in_R \mathbb{F}$	To prove $J_w(\mathbf{t})$ a zero polynomial $J_w(\mathbf{t}_0) = 0, \mathbf{t}_0 \in_R \mathbb{F}^s$ invoking the multi sumcheck protocol $\Longrightarrow F_w(\boldsymbol{r}_x)\delta_{\mathbf{t}_0}(\boldsymbol{r}_x) \stackrel{?}{=} e_x, \boldsymbol{r}_x \in_R \mathbb{F}^s, e_x \in \mathbb{F}$
Quad-check: $v_A \cdot v_B - v_C \stackrel{?}{=} G_w(r_x)\mathbb{Z}_H(r_x)$ **Lin-check**: $c = r_A \cdot v_A + r_B \cdot v_B + r_C \cdot v_C$ $\Longrightarrow$ $c \stackrel{?}{=} \sum_{y\in H} Q_{r_x}(y)$ Univariate sumcheck and GKR	**Quad-check**: $(v'_A v'_B - v'_C)\delta_{\mathbf{t}_0}(\boldsymbol{r}_x) \stackrel{?}{=} e_x$ **Lin-check**: $c' = r'_A v'_A + r'_B v'_B + r'_C v'_C$ $\Longrightarrow$ $c' \stackrel{?}{=} \sum_{\mathbf{y}\in[2]^s} Q_{\boldsymbol{r}_x}(\mathbf{y})$ second time multi sumcheck

Efficiency analysis

Univariate poly in Polaris

- $\mathcal{P}$: complexity is bounded by the complexity of computing $G(x) = F_w(x)/Z_H(x)$. The most efficient way is to apply **additive FFT, bounded by** $O(s2^s)$.
- **Proof size** is bounded by $O(s^2)$.
- $\mathcal{V}$: the complexity is bounded by $O(s^2)$ from the univariate sumcheck and GKR protocol.

Multivariate poly in Spartan

- $\mathcal{P}$: It does not actually compute $\tilde{F}_w(\boldsymbol{x})$ instead it only needs to evaluate $\tilde{F}_w(\boldsymbol{x})$ at a random point $\boldsymbol{r}_x \in \mathbb{F}^s$. So the complexity for the prover is **linear** on 2^s.
- The proof size is similar as the univariate case.
- $\mathcal{V}$: this has a problem to make it **logarithmic on** 2^s, like the univariate case. Spartan gets the result by using special memory structure.

Efficiency analysis

Univariate poly in Polaris

- $\mathcal{P}$: complexity is bounded by the complexity of computing $G(x) = F_w(x)/Z_H(x)$. The most efficient way is to apply **additive FFT, bounded by** $O(s2^s)$.
- **Proof size** is bounded by $O(s^2)$.
- $\mathcal{V}$: the complexity is bounded by $O(s^2)$ from the univariate sumcheck and GKR protocol.

Multivariate poly in Spartan

- $\mathcal{P}$: It does not actually compute $\tilde{F}_w(\boldsymbol{x})$ instead it only needs to evaluate $\tilde{F}_w(\boldsymbol{x})$ at a random point $\boldsymbol{r}_x \in \mathbb{F}^s$. So the complexity for the prover is **linear** on 2^s.
- The proof size is similar as the univariate case.
- $\mathcal{V}$: this has a problem to make it **logarithmic on** 2^s, like the univariate case. Spartan gets the result by using special memory structure.

Efficiency analysis

Univariate poly in Polaris

- $\mathcal{P}$: complexity is bounded by the complexity of computing $G(x) = F_w(x)/Z_H(x)$. The most efficient way is to apply **additive FFT, bounded by** $O(s2^s)$.
- **Proof size** is bounded by $O(s^2)$.
- $\mathcal{V}$: the complexity is bounded by $O(s^2)$ from the univariate sumcheck and GKR protocol.

Multivariate poly in Spartan

- $\mathcal{P}$: It does not actually compute $\tilde{F}_w(\boldsymbol{x})$ instead it only needs to evaluate $\tilde{F}_w(\boldsymbol{x})$ at a random point $\boldsymbol{r}_x \in \mathbb{F}^s$. So the complexity for the prover is **linear** on 2^s.
- The proof size is similar as the univariate case.
- $\mathcal{V}$: this has a problem to make it **logarithmic on** 2^s, like the univariate case. Spartan gets the result by using special memory structure.

Problem on the number of multiplication gates

- How does the number of **multiplication gates** effect the performance of zkSNARKs?
- The **degrees** of the polynomials or the number of variables of multivariate polynomials involved in R1CS are determined by the number of multiplication gates of the circuit.
- For example, in Zcash, one needs to prove $y = SHA256(x)$ where x is the number of Bitcoin for which the user wishes to spend. $SHA256$ has about **23k AND gates** and proof is based on a Merkle tree with high 64 . In this case $s = \lceil \log(64 \times 23000) \rceil = 21$.
  - The size of H is 2^{21}, and those polynomials has degree $2^{21} - 1$ for $\bar{M}(x), M \in \{A, B, C\}$.
  - In the multivariate case, the number of **variables** is 21 and there are 2^{21} monomials involved in the computation.
- Thus it requests the underline hash functions should have **minimal multiplicative complexity** $\rightarrow$ MiMC for symmetric-key cryptography!

Problem on the number of multiplication gates

- How does the number of **multiplication gates** effect the performance of zkSNARKs?
- The **degrees** of the polynomials or the number of variables of multivariate polynomials involved in R1CS are determined by the number of multiplication gates of the circuit.
- For example, in Zcash, one needs to prove $y = SHA256(x)$ where x is the number of Bitcoin for which the user wishes to spend. $SHA256$ has about **23k AND gates** and proof is based on a Merkle tree with high 64 . In this case $s = \lceil \log(64 \times 23000) \rceil = 21$.
  - The size of H is 2^{21}, and those polynomials has degree $2^{21} - 1$ for $\bar{M}(x), M \in \{A, B, C\}$.
  - In the multivariate case, the number of **variables** is 21 and there are 2^{21} monomials involved in the computation.
- Thus it requests the underline hash functions should have **minimal multiplicative complexity** $\rightarrow$ MiMC for symmetric-key cryptography!

Problem on the number of multiplication gates

- How does the number of **multiplication gates** effect the performance of zkSNARKs?
- The **degrees** of the polynomials or the number of variables of multivariate polynomials involved in R1CS are determined by the number of multiplication gates of the circuit.
- For example, in Zcash, one needs to prove $y = SHA256(x)$ where x is the number of Bitcoin for which the user wishes to spend. $SHA256$ has about **23k AND gates** and proof is based on a Merkle tree with high 64 . In this case $s = \lceil \log(64 \times 23000) \rceil = 21$.
  - The size of H is 2^{21}, and those polynomials has degree $2^{21} - 1$ for $\bar{M}(x), M \in \{A, B, C\}$.
  - In the multivariate case, the number of **variables** is 21 and there are 2^{21} monomials involved in the computation.
- Thus it requests the underline hash functions should have **minimal multiplicative complexity** $\rightarrow$ MiMC for symmetric-key cryptography!

Problem on the number of multiplication gates

- How does the number of **multiplication gates** effect the performance of zkSNARKs?
- The **degrees** of the polynomials or the number of variables of multivariate polynomials involved in R1CS are determined by the number of multiplication gates of the circuit.
- For example, in Zcash, one needs to prove $y = SHA256(x)$ where x is the number of Bitcoin for which the user wishes to spend. $SHA256$ has about **23k AND gates** and proof is based on a Merkle tree with high 64 . In this case $s = \lceil \log(64 \times 23000) \rceil = 21$.
  - The size of H is 2^{21}, and those polynomials has degree $2^{21} - 1$ for $\bar{M}(x), M \in \{A, B, C\}$.
  - In the multivariate case, the number of **variables** is 21 and there are 2^{21} monomials involved in the computation.
- Thus it requests the underline hash functions should have **minimal multiplicative complexity** $\rightarrow$ MiMC for symmetric-key cryptography!

Can we do better?

By selecting a **special** H, for example, to take H as a subfield of $\mathbb{F}$ instead of an affine subspace (or a multiplicative coset of $\mathbb{F}$).

- In this case, we have

$$Z_H(x) = x^{2^s} + x$$

$\Longrightarrow$ no computation needed!

- Can we reduce the **degrees** of those uni/multi variate polynomials?

- Yes, we have **algebraic attacks/selective DFT attack** in our area to make it happen (undergoing).

Can we do better?

By selecting a **special** H, for example, to take H as a subfield of $\mathbb{F}$ instead of an affine subspace (or a multiplicative coset of $\mathbb{F}$).

- In this case, we have

$$Z_H(x) = x^{2^s} + x$$

$\Longrightarrow$ no computation needed!

- Can we reduce the **degrees** of those uni/multi variate polynomials?
- Yes, we have **algebraic attacks/selective DFT attack** in our area to make it happen (undergoing).

Can we do better?

By selecting a **special** H, for example, to take H as a subfield of $\mathbb{F}$ instead of an affine subspace (or a multiplicative coset of $\mathbb{F}$).

- In this case, we have

$$Z_H(x) = x^{2^s} + x$$

$\Longrightarrow$ no computation needed!

- Can we reduce the **degrees** of those uni/multi variate polynomials?

- Yes, we have **algebraic attacks/selective DFT attack** in our area to make it happen (undergoing).

Can we do better?

By selecting a **special** H, for example, to take H as a subfield of $\mathbb{F}$ instead of an affine subspace (or a multiplicative coset of $\mathbb{F}$).

- In this case, we have
$$Z_H(x) = x^{2^s} + x$$
$\Longrightarrow$ no computation needed!

- Can we reduce the **degrees** of those uni/multi variate polynomials?

- Yes, we have **algebraic attacks/selective DFT attack** in our area to make it happen (undergoing).

Concluding remarks

- We have presented how **uni/multi variate polynomial embeddings** work for R1CS.
- As examples, we use **Polaris and Spartan** for demonstrating those post zkSNARK schemes. Those constructions of zkSNARKS are **quantum secure**, since they only involve polynomial operations and hash functions.
- We have showed that the computation of univariate polynomial embeddings can be optimized by selecting affine space/multiplicative cosets as a subfield.
- Applications are immense, but our focus is for **implementing blockchain privacy**.
- Currently, we are investigating to their **concrete computational cost** for both embeddings.

Remarks on some related areas

1. Recently, NIST called the **post-quantum** secure digital signature schemes which has the deadline in June 2023. Currently it has 50 submissions.
2. A **zkSNARK** scheme with post-quantum security is naturally a post-quantum secure digital signature scheme. (E.g. Picnic style digital signatures are in this class.)
3. In other words, let $pk = F(sk)$ where F is either an encryption or a hash function. A zkSNARK to prove the NP statement:

   "I know sk such that $pk = F(sk)$"

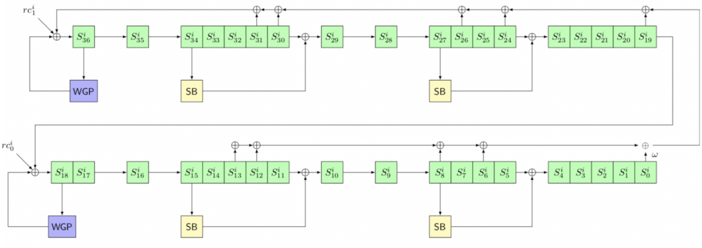
   without giving out sk to verifiers yields a signature scheme where the proof is the signature, sk is the **signing** key and pk is the **verification** key.
4. However, we need the underline **symmetric key** algorithm F is MiMC.

Open problems on MiMC design

- How **small** can we go to get MiMC symmetric key algorithm at a designated security level?
- If we take H as a multiplicative coset of $\mathbb{F}$, where $|H| = 2^s$. Then $\mathbb{F}$ has to be a **prime field**, i.e., $\mathbb{F} = GF(q)$ where q is a prime or a power of a prime $\neq 2$.
  - Can we find **good permutations** of K^t where K is a subfield of $\mathbb{F}$, with $|K^t| \approx |H|$?
  - In other words, the permutations with good **nonlinearity**, differential uniformity or APN property, ...

Open problems on MiMC design (cont.)

- We have proposed to apply WAGE's (NIST LWC Round 2 Candidate) structure for obtaining MiMC for the binary field case. However, even for WG permutations of $\mathbb{F}_{2^n}$, we do not know the above mentioned properties for nonbinary fields.



WAGE one round function

- WAGE, an authenticated WG encryption, is obtained by taking parameters of LFSR of order 37 over $\mathbb{F}_{2^7}$ in the WG stream cipher with additionally added nonlinear operations SB.

UNIVERSITY OF WATERLOO

Open problem on uni/multi variate poly. interp./eval

- For multivariate polynomial embedded R1CS (e.g. Spartan), at the end, the verifier has to evaluate $Q_{\boldsymbol{r}_x}(\mathbf{y})$ at a random point $\boldsymbol{r}_y = (r_0, r_1, \cdots, r_{s-1})$ in $\mathbb{F}^s$ in order to check the equality (we shorten $Q_{\boldsymbol{r}_x}(\mathbf{y})$ as $Q(\mathbf{y})$):

$$Q(\boldsymbol{r}_y) \stackrel{?}{=} e_y, \boldsymbol{r}_y \in_R \mathbb{F}^s, e_y \in \mathbb{F}$$

- We may consider the coefficients of $Q(\mathbf{y})$ as a vector (or equivalently a **sequence**), say $\boldsymbol{o} = (o_0, \cdots, 0_{d-1})$ and its **monomial** terms $r_0^{e_0} r_1^{e_1} \cdots r_{s-1}^{e_{s-1}}$ as another vector, say $\boldsymbol{p} = (p_0, \cdots, p_{d-1})$ where d is the number of monomials in $Q(\mathbf{y})$.
- In this way, we can interpolate $\boldsymbol{o}$ and $\boldsymbol{p}$ over another affine space of $\mathbb{F}$, say H', say $O(x)$ and $P(x)$ respectively, Thus

$$Q(\boldsymbol{r}_y) = \sum_{a \in H'} O(a)P(a) \stackrel{?}{=} e_y.$$

So this is converted to the **univariate** polynomial sumcheck for polynomial $S(x) = O(x)P(x)$ (used in Virgo in [ZXZS20]) → Polaris' verification!

- Can we do this **conversion** with time complexity $O(|H'|)$ instead of $O(|H'| \log |H'|)$?

UNIVERSITY OF WATERLOO

Open problem on uni/multi variate poly. interp./eval

- For multivariate polynomial embedded R1CS (e.g. Spartan), at the end, the verifier has to evaluate $Q_{\boldsymbol{r}_x}(\mathbf{y})$ at a random point $\boldsymbol{r}_y = (r_0, r_1, \cdots, r_{s-1})$ in $\mathbb{F}^s$ in order to check the equality (we shorten $Q_{\boldsymbol{r}_x}(\mathbf{y})$ as $Q(\mathbf{y})$):

$$Q(\boldsymbol{r}_y) \stackrel{?}{=} e_y, \boldsymbol{r}_y \in_R \mathbb{F}^s, e_y \in \mathbb{F}$$

- We may consider the coefficients of $Q(\mathbf{y})$ as a vector (or equivalently a **sequence**), say $\boldsymbol{o} = (o_0, \cdots, 0_{d-1})$ and its **monomial** terms $r_0^{e_0} r_1^{e_1} \cdots r_{s-1}^{e_{s-1}}$ as another vector, say $\boldsymbol{p} = (p_0, \cdots, p_{d-1})$ where d is the number of monomials in $Q(\mathbf{y})$.
- In this way, we can interpolate $\boldsymbol{o}$ and $\boldsymbol{p}$ over another affine space of $\mathbb{F}$, say H', say $O(x)$ and $P(x)$ respectively, Thus

$$Q(\boldsymbol{r}_y) = \sum_{a \in H'} O(a)P(a) \stackrel{?}{=} e_y.$$

So this is converted to the **univariate** polynomial sumcheck for polynomial $S(x) = O(x)P(x)$ (used in Virgo in [ZXZS20]) → Polaris' verification!

- Can we do this **conversion** with time complexity $O(|H'|)$ instead of $O(|H'| \log |H'|)$?

Open problem on uni/multi variate poly. interp./eval

- For multivariate polynomial embedded R1CS (e.g. Spartan), at the end, the verifier has to evaluate $Q_{\boldsymbol{r}_x}(\mathbf{y})$ at a random point $\boldsymbol{r}_y = (r_0, r_1, \cdots, r_{s-1})$ in $\mathbb{F}^s$ in order to check the equality (we shorten $Q_{\boldsymbol{r}_x}(\mathbf{y})$ as $Q(\mathbf{y})$):

$$Q(\boldsymbol{r}_y) \stackrel{?}{=} e_y, \boldsymbol{r}_y \in_R \mathbb{F}^s, e_y \in \mathbb{F}$$

- We may consider the coefficients of $Q(\mathbf{y})$ as a vector (or equivalently a **sequence**), say $\boldsymbol{o} = (o_0, \cdots, 0_{d-1})$ and its **monomial** terms $r_0^{e_0} r_1^{e_1} \cdots r_{s-1}^{e_{s-1}}$ as another vector, say $\boldsymbol{p} = (p_0, \cdots, p_{d-1})$ where d is the number of monomials in $Q(\mathbf{y})$.
- In this way, we can interpolate $\boldsymbol{o}$ and $\boldsymbol{p}$ over another affine space of $\mathbb{F}$, say H', say $O(x)$ and $P(x)$ respectively, Thus

$$Q(\boldsymbol{r}_y) = \sum_{a \in H'} O(a)P(a) \stackrel{?}{=} e_y.$$

So this is converted to the **univariate** polynomial sumcheck for polynomial $S(x) = O(x)P(x)$ (used in Virgo in [ZXZS20]) $\rightarrow$ Polaris' verification!

- Can we do this **conversion** with time complexity $O(|H'|)$ instead of $O(|H'| \log |H'|)$?

Open problem on uni/multi variate poly. interp./eval

- For multivariate polynomial embedded R1CS (e.g. Spartan), at the end, the verifier has to evaluate $Q_{\boldsymbol{r}_x}(\mathbf{y})$ at a random point $\boldsymbol{r}_y = (r_0, r_1, \cdots, r_{s-1})$ in $\mathbb{F}^s$ in order to check the equality (we shorten $Q_{\boldsymbol{r}_x}(\mathbf{y})$ as $Q(\mathbf{y})$):

$$Q(\boldsymbol{r}_y) \stackrel{?}{=} e_y, \boldsymbol{r}_y \in_R \mathbb{F}^s, e_y \in \mathbb{F}$$

- We may consider the coefficients of $Q(\mathbf{y})$ as a vector (or equivalently a **sequence**), say $\boldsymbol{o} = (o_0, \cdots, 0_{d-1})$ and its **monomial** terms $r_0^{e_0} r_1^{e_1} \cdots r_{s-1}^{e_{s-1}}$ as another vector, say $\boldsymbol{p} = (p_0, \cdots, p_{d-1})$ where d is the number of monomials in $Q(\mathbf{y})$.
- In this way, we can interpolate $\boldsymbol{o}$ and $\boldsymbol{p}$ over another affine space of $\mathbb{F}$, say H', say $O(x)$ and $P(x)$ respectively, Thus

$$Q(\boldsymbol{r}_y) = \sum_{a \in H'} O(a)P(a) \stackrel{?}{=} e_y.$$

So this is converted to the **univariate** polynomial sumcheck for polynomial $S(x) = O(x)P(x)$ (used in Virgo in [ZXZS20]) $\rightarrow$ Polaris' verification!

- Can we do this **conversion** with time complexity $O(|H'|)$ instead of $O(|H'| \log |H'|)$?

UNIVERSITY OF WATERLOO

References

- Guang Gong, Hybrid uni/multivariate encoded R1CS, 2023. Internal report.
- Shihui Fu and Guang Gong, **Polaris:** Transparent Succinct Zero-Knowledge Arguments for R1CS with Efficient Verifier, *the Proceedings on Privacy Enhancing Technologies (PPETs)*, 2022 (1), pp. 544 - 564.
- Nusa Zidaric, Kalikinkar Mandal, Guang Gong, Mark Aagaard, The Welch-Gong stream cipher - evolutionary path. *Cryptogr. Commun. (2023).* https://doi.org/10.1007/s12095-023-00656-0.

Thanks! Questions?