Hardware Implementation of Signature Verification in the New Post-Quantum Signature Scheme MAYO

ECE 646 Project

Kowsyap Pranay Kumar Musumuri

INTRODUCTION

Post-Quantum Cryptography (PQC)

A New Era in Cryptography

The challenge? Quantum computers.

- They're powerful. They solve problems like **integer factorization** and **discrete logarithms** easily.
- The problem? These are the foundations of today's encryption.

The solution? Post-Quantum Cryptography (PQC).

- PQC designs cryptographic systems that stay secure even if quantum computers become common.
- The problem? These are the foundations of today's encryption.

A promising approach? Multivariate cryptography.

- It's based on solving multivariate quadratic (MQ) equations over finite fields.
- Why is this important? Even **quantum computers** find these problems hard to crack.

Multivariate Cryptography

Multivariate Quadratic Solving

Example: Solve for integers x and y:

$$x + 5x^2 + 3xy = 4 \bmod 7$$
$$x^2 + 5xy + 5y^2 = 1 \bmod 7$$

Solution: x = 6 and y = 0.

For only 2 variables this is still doable, but for more variables this problem quickly becomes very difficult.

E.g. The current record mod 31 is solving a system of only 22 equations in 22 variables! (~1 core-year of computation effort)

Oil-and-Vinegar Signature Scheme

Foundation of Multivariate Quadratic Schemes

Solves multivariate polynomial equations with:

- **Oil variables**: Limited interaction.
- **Vinegar variables**: Freely interact with oil and themselves.

Key Generation: Public key (polynomials) and private key (trapdoor structure).

Decryption: Uses the private key to solve equations efficiently.

Pros:

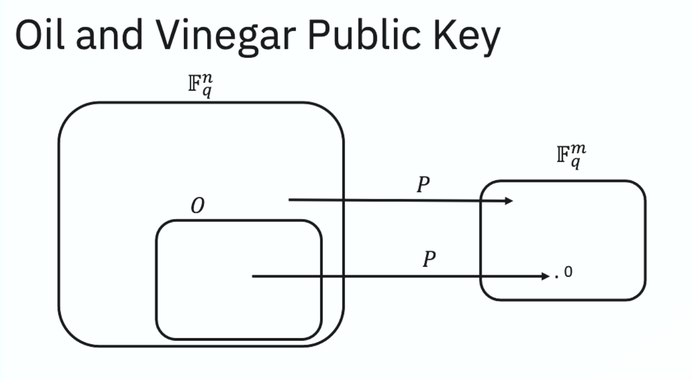
- Well studied
- Small signatures
- Fast

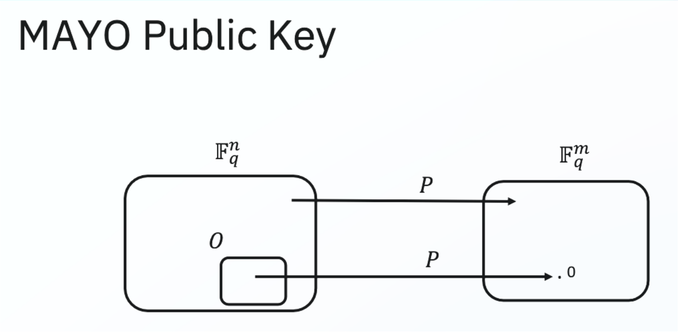
Cons:

- Larger Public Keys (44KB)

Why MAYO?

Evolution of the OV Scheme





Why move beyond OV?

- The main issue? Large public keys. Up to 44KB

MAYO's Key Improvements?

- Uses a **smaller** o (number of oil variables) for optimization.
- C Public key size? Reduced to $O(o^2m)$ instead of $O(m^3)$.

The result?

- Retains OV's advantages (small signatures and fast operations).t's based on solving multivariate quadratic (MQ) equations over finite fields.
- Compact public keys.

Overview

MAYO Signature Scheme

A Post-Quantum Signature Solution

What is MAYO?

- A variant of the Oil-and-Vinegar (OV) scheme.
- Features smaller public keys than traditional OV.

Differences?

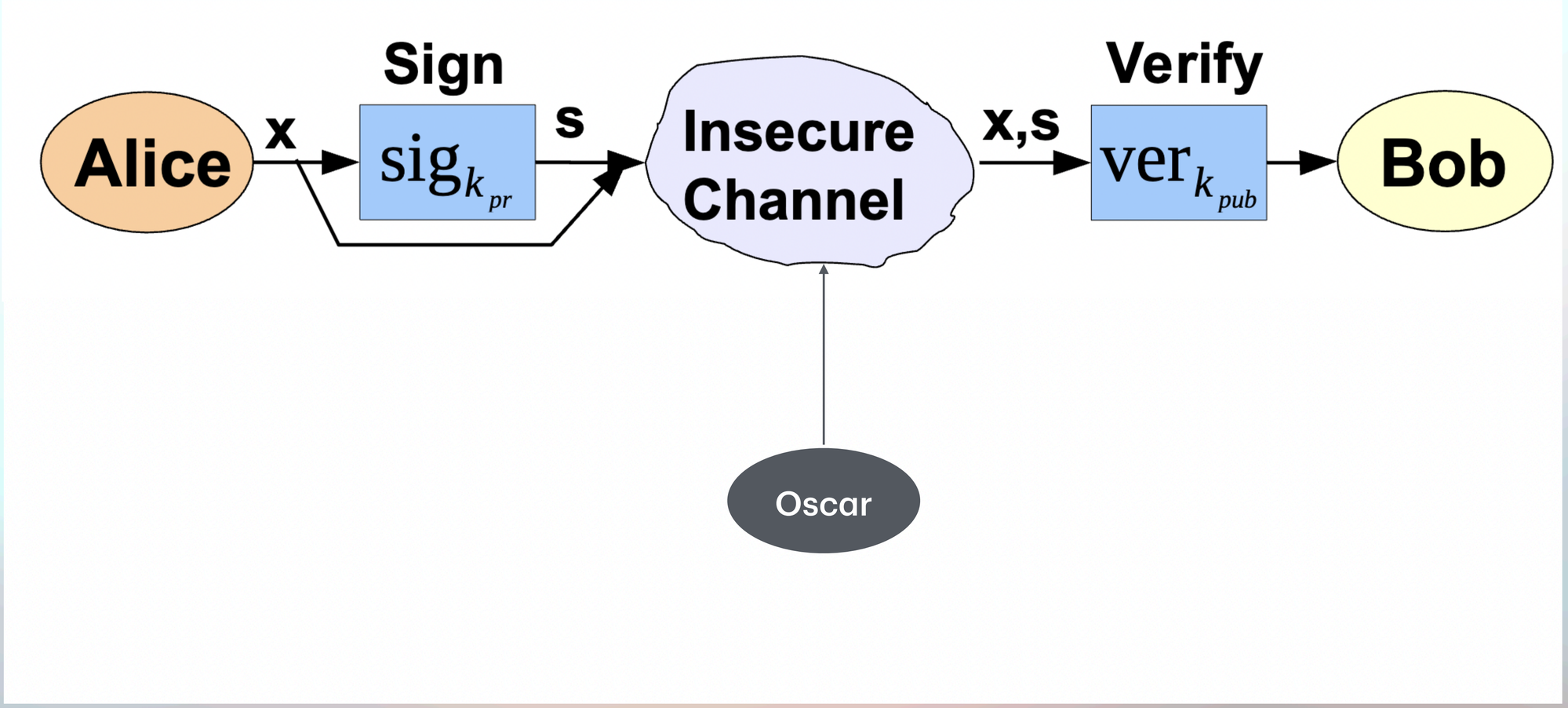
- In MAYO, the dimension of O (oil space) is too small: o<m.
- Analogy? A smaller needle in a smaller haystack, keeping the problem hard but efficient.

Problem?

- Sampling valid signatures becomes harder when o<m.
- **Solution?** Whip up the Oil-and-Vinegar map $P(x)$ into a k-fold larger map $P_*(x_1,\ldots,x_k)$. i.e, Publicly adds extra variables to O, making it solvable.

The Result?

- Public key size reduced 58x (1.1 KB vs. 67 KB in OV).
- Signature size increased slightly (321 B vs. 96 B).


Sign
Verify
Alice
x
$sig_{k_{pr}}$
s
Insecure Channel
x,s
$ver_{k_{pub}}$
Bob
Oscar

Parameter Sets of MAYO

Flexibility for Security and Efficiency

Key parameters

- (n, m, o, k, q) Control the trade-off between security, key size, and signature size.
- Features smaller public keys than traditional OV.

Differences?

- Smaller o: Reduces public key size ($O(o^2m)$ instead of $O(m^3)$).
- Larger k: Ensures signature sampling remains efficient.

Problem?

- Adjust parameters to balance **security** and **practicality**.

Example

- For NIST security level 1:
- (n,m,o,k,q)=(66,64,8,9,16), Public Key Size: 1.1 KB, Signature Size: 321 B.

	Oil & Vinegar GF(16)	Oil & Vinegar GF(256)	MAYO 1 $o = 8$	MAYO2 $o = 18$
# Variables	160	112	66 x 9	66 x 16
# Equations	64	44	64	69
Finite Field	GF(16)	GF(256)	GF(16)	GF(16)
Pk size	67 KB	44 KB	1.1 KB	5.4 KB
Signature size	96 B	128 B	321 B	180 B

Size of o gives a trade-off between signature size and pk size.

Importance and Specialty of MAYO

Why Choose MAYO?

Why is MAYO special?

- Compact public keys: Makes it suitable for resource-constrained systems.
- Efficient signing and verification: Balances security and practicality.
- Post-quantum security: Designed to resist attacks from quantum computers.

Advantages over OV

- 58x smaller public keys for the same security level.
- Scales well with higher security levels.

Ideal Use Cases

- IoT devices.
- Embedded systems with limited resources.
- Secure communications in the quantum era.

Parameter set security level	MAYO_1 1	MAYO_2 1	MAYO_3 3	MAYO_5 5
n	66	78	99	133
m	64	64	96	128
o	8	18	10	12
k	9	4	11	12
q	16	16	16	16
salt_bytes	24	24	32	40
digest_bytes	32	32	48	64
pk_seed_bytes	16	16	16	16
$f(z)$	$f_{64}(z)$	$f_{64}(z)$	$f_{96}(z)$	$f_{128}(z)$
secret key size	24 B	24 B	32 B	40 B
public key size	1168 B	5488 B	2656 B	5008 B
signature size	321 B	180 B	577 B	838 B
expanded sk size	69 KB	92 KB	230 KB	553 KB
expanded pk size	70 KB	97 KB	233 KB	557 KB

MAYO Verification

MAYO Verify

Algorithm

Purpose: Verifies if a given signature is valid or invalid.

Inputs:

- Expanded public key (*epk*)
- Message (*M*)
- Signature (*sig*)

Key Steps:

- Decode inputs
- Hash the message
- Compute P*(s)
- Compare results

Algorithm 9 MAYO.Verify(epk, M, sig)

Input: Expanded public key epk $\in \mathcal{B}^{\text{epk_bytes}}$
Input: Message M $\in \mathcal{B}^*$
Input: Signature sig $\in \mathcal{B}^{\text{sig_bytes}}$
Constant: $\mathbf{E} \in \mathbb{F}_q^{m \times m}$ # Represents multiplication by z in $\mathbb{F}_q[z]/(f(z))$
Output: An integer result to indicate if sig is valid (result $= 0$) or invalid (result < 0).

//Decode epk.
P1_bytestring $\leftarrow$ epk[0 : P1_bytes]
P2_bytestring $\leftarrow$ epk[P1_bytes : P1_bytes + P2_bytes]
P3_bytestring $\leftarrow$ epk[P1_bytes + P2_bytes : P1_bytes + P2_bytes + P3_bytes]
$\{\mathbf{P}_i^{(1)}\}_{i\in[m]} \leftarrow \text{Decode}_{\mathbf{P}^{(1)}}(\text{P1_bytestring})$ // $\mathbf{P}_i^{(1)} \in \mathbb{F}_q^{(n-o)\times(n-o)}$ upper triangular
$\{\mathbf{P}_i^{(2)}\}_{i\in[m]} \leftarrow \text{Decode}_{\mathbf{P}^{(2)}}(\text{P2_bytestring})$ // $\mathbf{P}_i^{(2)} \in \mathbb{F}_q^{(n-o)\times o}$
$\{\mathbf{P}_i^{(3)}\}_{i\in[m]} \leftarrow \text{Decode}_{\mathbf{P}^{(3)}}(\text{P3_bytestring})$ // $\mathbf{P}_i^{(3)} \in \mathbb{F}_q^{o\times o}$ upper triangular

//Decode sig.
salt $\leftarrow$ sig[$\lceil nk/2 \rceil$: $\lceil nk/2 \rceil$ + salt_bytes]
$\mathbf{s} \leftarrow \text{Decode}_{vec}(kn, \text{sig})$
for i from 0 to $k-1$ **do**
  $\mathbf{s}_i \leftarrow \mathbf{s}[i * n : (i+1) * n]$

//Hash message and derive $\mathbf{t}$.
M_digest $\leftarrow$ SHAKE256(M, digest_bytes) // M_digest $\in \mathcal{B}^{\text{digest_bytes}}$
$\mathbf{t} \leftarrow \text{Decode}_{vec}(m, \text{SHAKE256}(\text{M_digest} \parallel \text{salt}, \lceil m \log(q)/8 \rceil))$ // $\mathbf{t} \in \mathbb{F}_q^m$

//Compute $\mathcal{P}^*(\mathbf{s})$.
$\mathbf{y} \leftarrow \mathbf{0}_m$ // $\mathbf{y} \in \mathbb{F}_q^m$
$\ell \leftarrow 0$
for i from 0 to $k-1$ **do**
  for j from $k-1$ to i **do**

$$\mathbf{u} \leftarrow \begin{cases} \left\{ \mathbf{s}_i^\top \begin{pmatrix} \mathbf{P}_a^{(1)} & \mathbf{P}_a^{(2)} \\ \mathbf{0} & \mathbf{P}_a^{(3)} \end{pmatrix} \mathbf{s}_i \right\}_{a\in[m]} & \text{if } i = j \\ \left\{ \mathbf{s}_i^\top \begin{pmatrix} \mathbf{P}_a^{(1)} & \mathbf{P}_a^{(2)} \\ \mathbf{0} & \mathbf{P}_a^{(3)} \end{pmatrix} \mathbf{s}_j + \mathbf{s}_j^\top \begin{pmatrix} \mathbf{P}_a^{(1)} & \mathbf{P}_a^{(2)} \\ \mathbf{0} & \mathbf{P}_a^{(3)} \end{pmatrix} \mathbf{s}_i \right\}_{a\in[m]} & \text{if } i \neq j \end{cases} \quad // \mathbf{u} \in \mathbb{F}_q^m$$

    $\mathbf{y} \leftarrow \mathbf{y} + \mathbf{E}^\ell \mathbf{u}$
    $\ell \leftarrow \ell + 1$

//Accept signature if $\mathbf{y} = \mathbf{t}$.
if $\mathbf{y} = \mathbf{t}$ **then**
  return 0
return -1

MAYO Verify

Pseudocode in Detail

Input Decoding

- Split and decode *epk* into *P1, P1, P3*
- ***P1*** : Represents $F_q^{n-o \times n-o}$ (upper triangular matrix)
- ***P2*** : Represents $F_q^{n-o \times o}$
- ***P3*** : Represents $F_q^{o \times o}$ (upper triangular matrix)

```
//Decode epk.
P1_bytestring ← epk[0 : P1_bytes]
P2_bytestring ← epk[P1_bytes : P1_bytes + P2_bytes]
P3_bytestring ← epk[P1_bytes + P2_bytes : P1_bytes + P2_bytes + P3_bytes]
{P_i^(1)}_{i∈[m]} ← Decode_{P^(1)}(P1_bytestring)        // P_i^(1) ∈ F_q^{(n-o)×(n-o)} upper triangular
{P_i^(2)}_{i∈[m]} ← Decode_{P^(2)}(P2_bytestring)        // P_i^(2) ∈ F_q^{(n-o)×o}
{P_i^(3)}_{i∈[m]} ← Decode_{P^(3)}(P3_bytestring)        // P_i^(3) ∈ F_q^{o×o} upper triangular
```

MAYO Verify

Pseudocode in Detail

Hash and Derive Target

- Use **SHAKE256** to generate a digest of the message *M*
  - Hash M along with the extracted salt.
  - Derive t, the expected result vector in F_q^m

//Hash message and derive $\mathbf{t}$.
M_digest $\leftarrow$ SHAKE256(M, digest_bytes)
$\mathbf{t} \leftarrow \text{Decode}_{vec}(m, \text{SHAKE256}(\text{M_digest} \parallel \text{salt}, \lceil m \log(q)/8 \rceil))$

Purpose:

- Ensure that the verification process is tied to the original message and signature.
- Prevent any tampering with *M*

MAYO Verify

Pseudocode in Detail

Evaluation

- Iterate through *k*-fold expanded map.
- Perform matrix multiplications and evaluate *P*(s)*

//Compute $\mathcal{P}^*(\mathbf{s})$.

$\mathbf{y} \leftarrow \mathbf{0}_m$ // $\mathbf{y} \in \mathbb{F}_q^m$

$\ell \leftarrow 0$

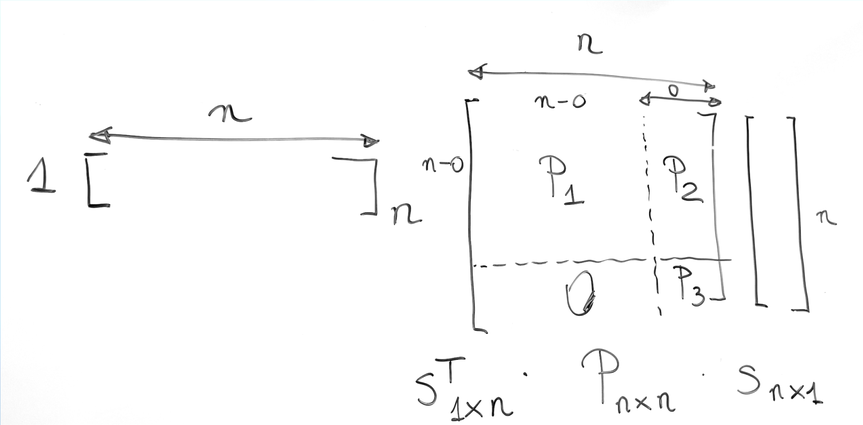
for i from 0 to $k-1$ **do**

  for j from $k-1$ to i **do**

$$\mathbf{u} \leftarrow \begin{cases} \left\{ \mathbf{s}_i^\top \begin{pmatrix} \mathbf{P}_a^{(1)} & \mathbf{P}_a^{(2)} \\ \mathbf{0} & \mathbf{P}_a^{(3)} \end{pmatrix} \mathbf{s}_i \right\}_{a \in [m]} & \text{if } i = j \\ \left\{ \mathbf{s}_i^\top \begin{pmatrix} \mathbf{P}_a^{(1)} & \mathbf{P}_a^{(2)} \\ \mathbf{0} & \mathbf{P}_a^{(3)} \end{pmatrix} \mathbf{s}_j + \mathbf{s}_j^\top \begin{pmatrix} \mathbf{P}_a^{(1)} & \mathbf{P}_a^{(2)} \\ \mathbf{0} & \mathbf{P}_a^{(3)} \end{pmatrix} \mathbf{s}_i \right\}_{a \in [m]} & \text{if } i \neq j \end{cases}$$ // $\mathbf{u} \in \mathbb{F}_q^m$

    $\mathbf{y} \leftarrow \mathbf{y} + \mathbf{E}^\ell \mathbf{u}$

    $\ell \leftarrow \ell + 1$



Constant: $\mathbf{E} \in \mathbb{F}_q^{m \times m}$ # Represents multiplication by z in $\mathbb{F}_q[z]/(f(z))$

$$\begin{aligned} f_{64}(z) &= z^{64} + x^3z^3 + xz^2 + x^3 \\ f_{96}(z) &= z^{96} + xz^3 + xz + x \\ f_{128}(z) &= z^{128} + xz^4 + x^2z^3 + x^3z + x^2 \end{aligned}$$

MAYO Verify

Pseudocode in Detail

Compare results

- Check if y=t
- If yes: **Valid** Signature.
- Else: **Invalid** Signature.

```
//Accept signature if y = t.
if y = t then
    return 0
return −1
```

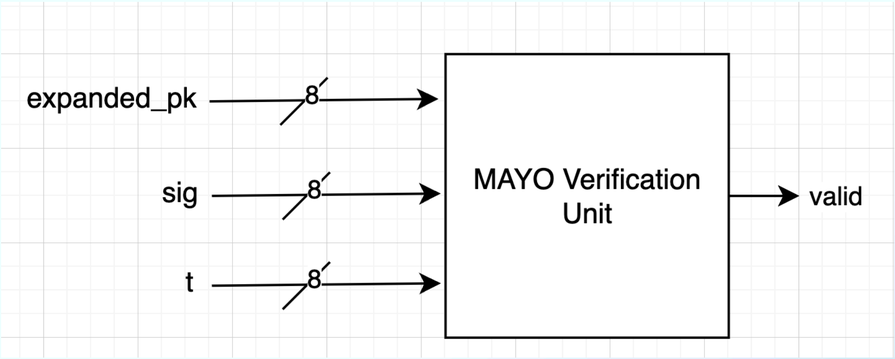
Hardware Implementation

Hardware Implementation Overview

Transitioning from Algorithm to Hardware

Key Goals

- Optimize for **low latency**
- Efficiently handle **GF arithmetic** operations in hardware.
- Maintain scalability for different **parameter sets**.

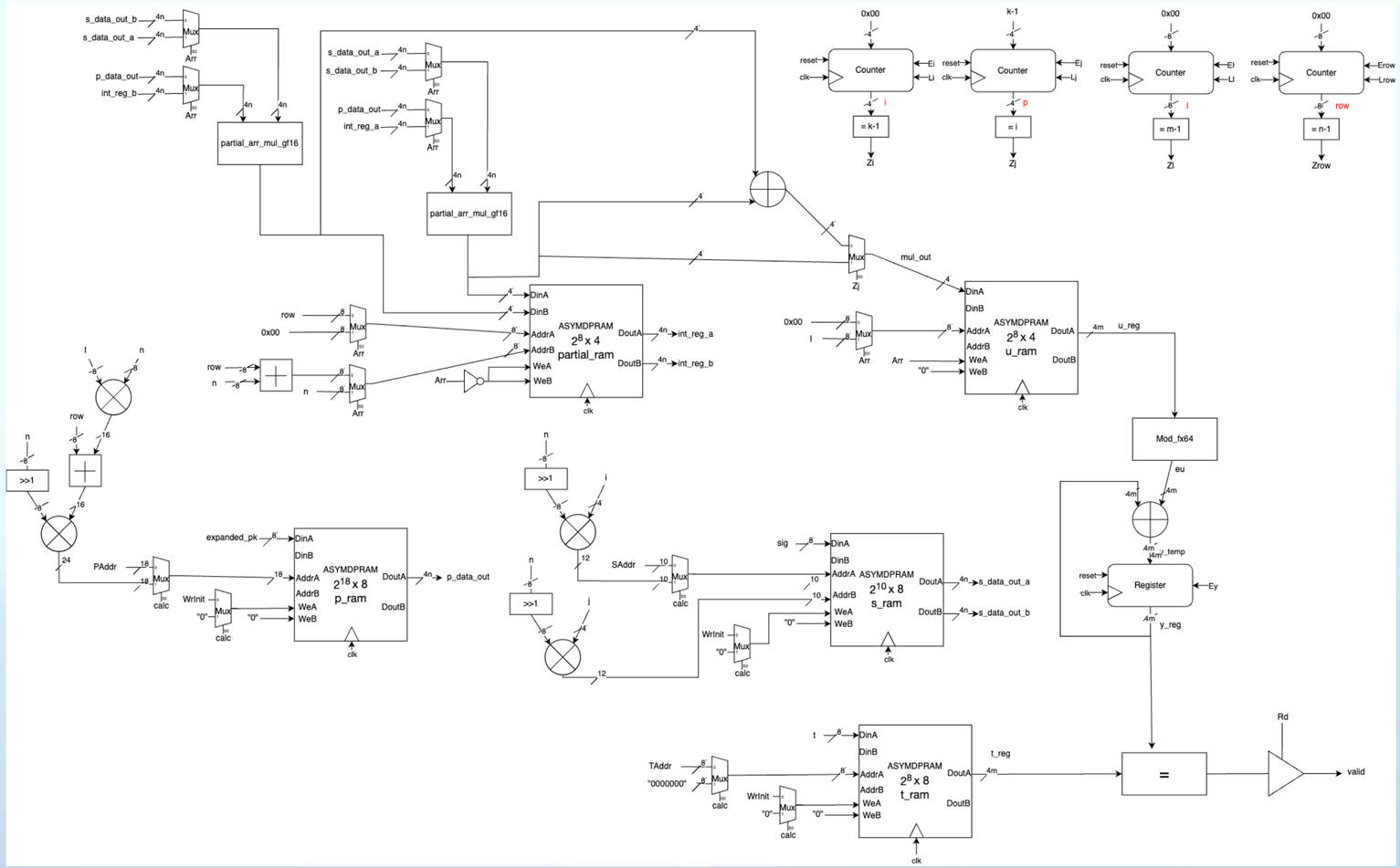


Inputs

- ***expanded_pk*** (organized and decoded into P1 ,P2 ,P3 using a **JavaScript implementation**)
- ***sig*** : (provided by the official MAYO optimized C implementation)
- ***t*** - (derived from the same C implementation, representing the expected hash digest of the message)

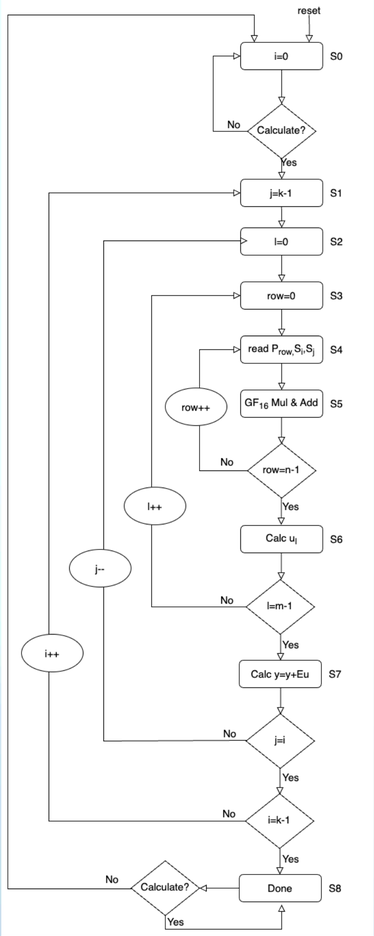
Datapath Design

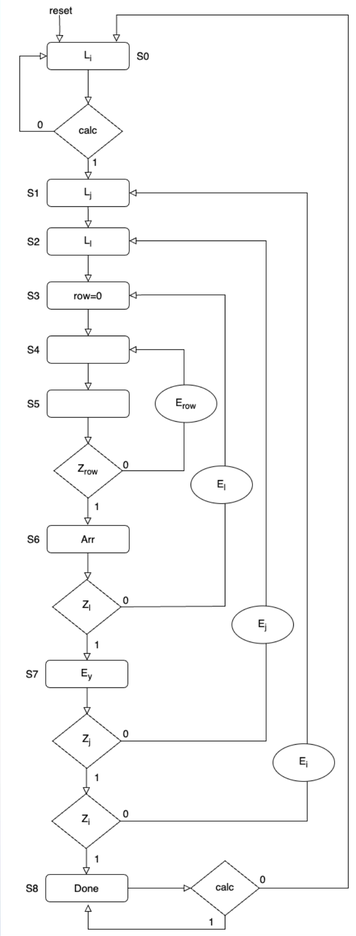
Core Computational Unit



Controller Design

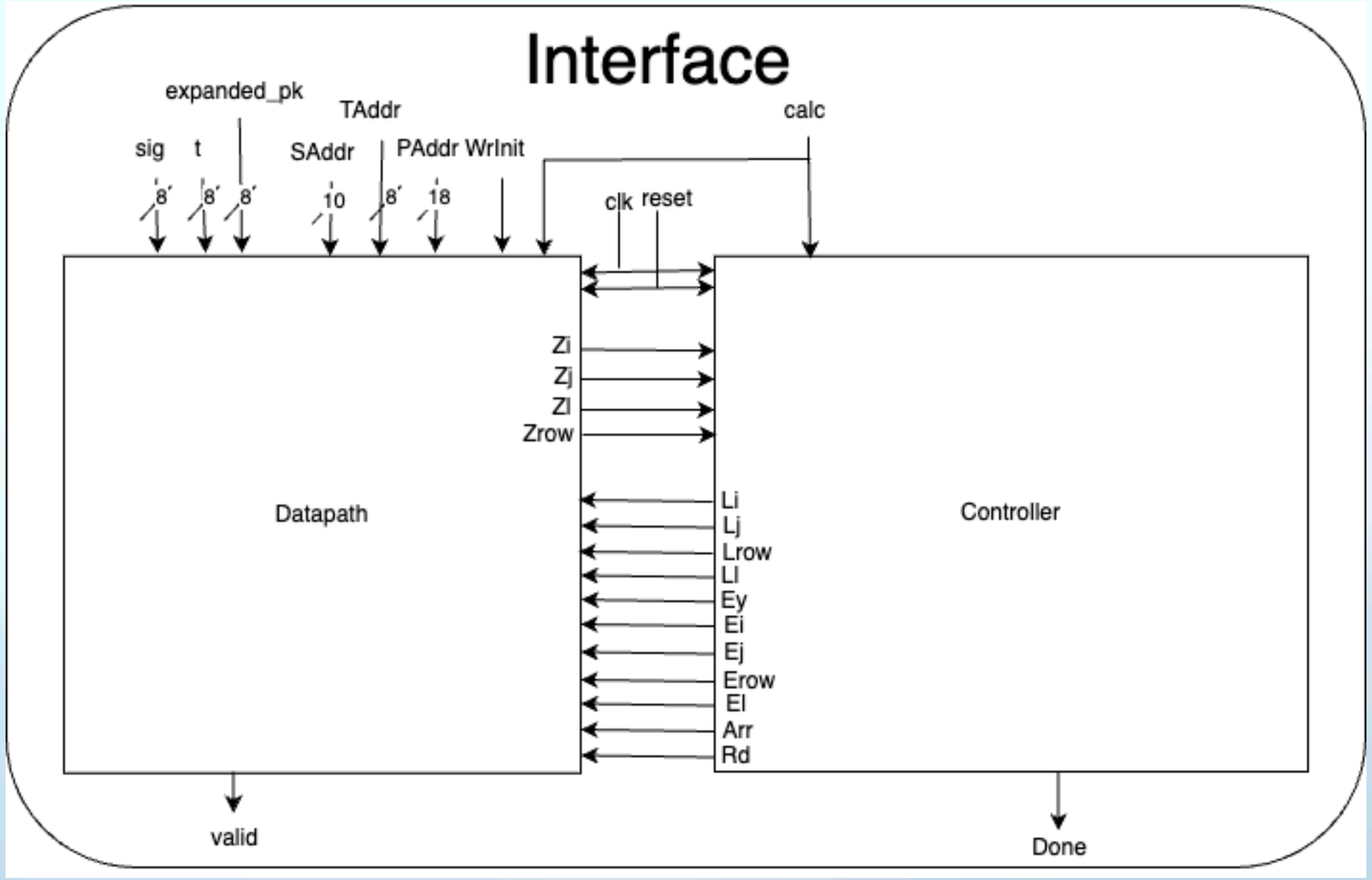
Control Logic for Operations





Interface

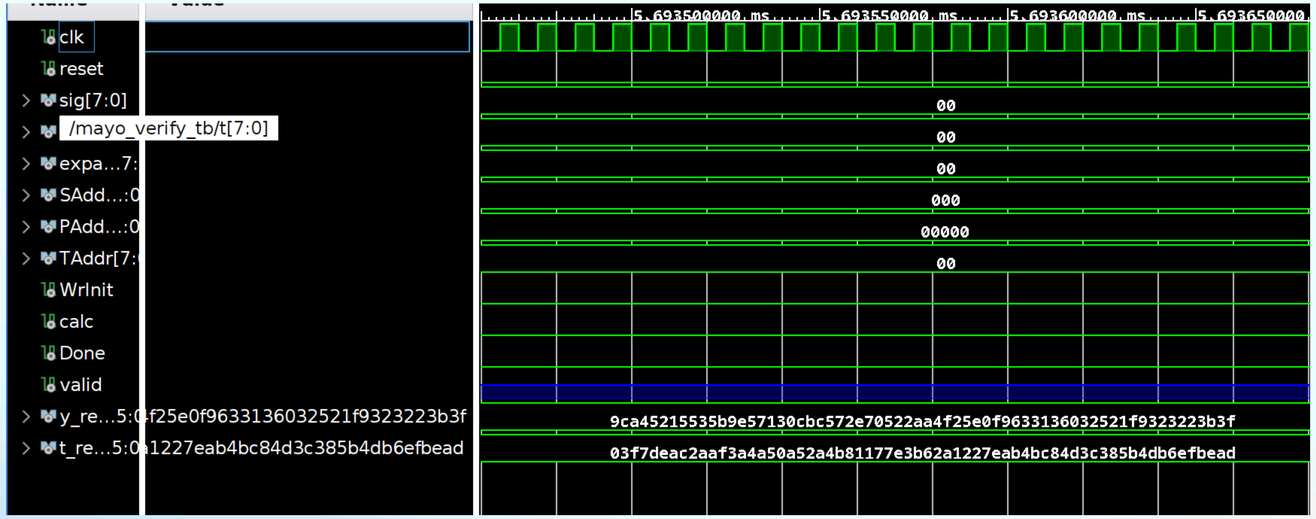
Inputs and Outputs of the Hardware Module



Implementation Results

Timing Simulations

Validation - Mayo




```
[
  9, 12, 10,  4, 5,  2, 1,  5,  5,  3,  5, 11,
  9, 14,  5,  7, 1,  3, 0, 12, 11, 12,  5,  7,
  2, 14,  7,  0, 5,  2, 2, 10, 10,  4, 15,  2,
  5, 14,  0, 15, 9,  6, 3,  3,  1,  3,  6,  0,
  3,  2,  5,  2, 1, 15, 9,  3,  2,  3,  2,  2,
  3, 11,  3, 15
]
```

Execution Times

Latency and Throughput Analysis

Mayo1

ET(cycles) = k[(k+1)((n+1)m+1)+1]+1

= 9[10 * 4289 +1]+1

= 386020 cycles

ET(ns) = ET(cycles) * T_{clk} = 386020 cycles * 10 ns

= 3860200 ns = 3.86 ms

Throughput = 1/ET(ns) = 1000/3.86

= 259 (operations/sec)

Mayo2

ET(cycles) = k[(k+1)((n+1)m+1)+1]+1

= 4[5 * 5057 +1]+1

= 101145 cycles

ET(ns) = ET(cycles) * T_{clk} = 101145 cycles * 10 ns

= 1011450 ns = 1.01 ms

Throughput = 1/ET(ns) = 1000/1.01

= 990 (operations/sec)

Challenges & Future Directions

Challenges

Obstacles in Implementing MAYO.Verify

- Translating mathematical operations (polynomials, GF arithmetic) into hardware modules.
- Memory bottlenecks for storing expanded public keys ($P1$,$P2$,$P3$).
- Ensuring low latency without exceeding chip area constraints.
- Handling various parameter sets efficiently in a unified design.
- Verifying the correctness of arithmetic and logical operations in hardware.

Future Work

Enhancements and Extensions

Completed work

- Designed and implemented the **datapath** and **controller**.
- Developed the hardware implementation for MAYO Verify.
- Performed **timing simulations** in Vivado to validate functionality.

Next steps

- **Synthesis** -> *Optimize the design for hardware resources (LUTs, registers, DSPs).*
- **Implementation** -> Map the design onto physical hardware and test for real-world performance under various parameter sets.
- Explore **power optimization** techniques and **fault tolerance** mechanisms to improve efficiency.

Summary & References

Conclusion

Final Takeaways

- **MAYO** offers smaller public keys and efficient verification, making it suitable for post-quantum cryptography.
- Successfully implemented and validated the functional correctness of **MAYO.Verify** in hardware.
- Next steps involves completing **synthesis** and **implementation** to prepare for deployment.
- Future goals focus on **energy efficiency**, and **fault tolerance**.
- **Hardware** solutions are crucial for **post-quantum cryptography** due to their **speed** and **efficiency** compared to **software** implementations.

Bibliography

References and Resources

- W. Beullens, MAYO: Practical Post-Quantum Signatures from Oil-and-Vinegar Maps, PQC Seminar, September 24, 2024, available at https://csrc.nist.gov/csrc/media/Projects/post-quantum-cryptography/documents/pqc-seminars/presentations/20-mayo-09242024.pdf

- O. Sayari, S. Marzougui, T. Aulbach, J. Kramer, and J.-P. Seifert, "HaMAYO: A Fault-Tolerant Reconfigurable Hardware Implementation of the MAYO Signature Scheme," in Constructive Side- Channel Analysis and Secure Design, COSADE 2024, pp. 240–259, Springer 2024, available at https://eprint.iacr.org/2023/1135

- F. Hirner, M. Streibl, F. Krieger, A. Can Mert, S.S. Roy, "Whipping the Multivariate-based MAYO Signature Scheme using Hardware Platforms," ACM CCS 2024 and Cryptology ePrint Archive, paper 2023-1267, available at https://eprint.iacr.org/2023/1267

Thank You