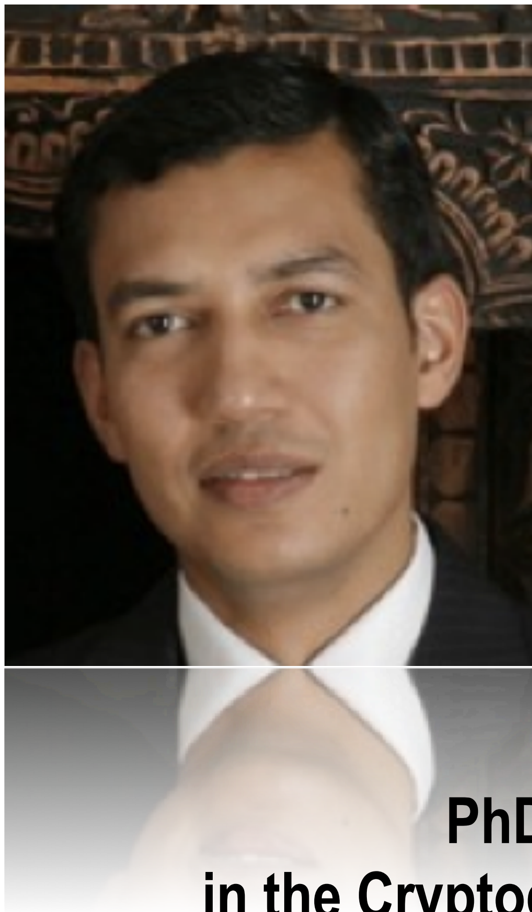
Lessons Learned from High-Speed Implementation and Benchmarking of Two Post-Quantum Public-Key Cryptosystems

Malik Umar Sharif,
Ahmed Ferozpuri, and
<u>Kris Gaj</u>
George Mason University
USA

Partially supported by NIST under grant no. 60NANB15D058

Co-Authors

Malik Umar Sharif



Ahmed Ferozpuri

PhD Students
in the Cryptographic Engineering
Research Group (CERG) at GMU

Post-Quantum Cryptography

- Quantum Computers could potentially break all current American federal standards in the area of public-key cryptography (RSA, ECC, Diffie-Hellman)
- Increasing key sizes would be futile
- Public key cryptographic families presumed resistant against quantum computing cryptanalysis collectively referred to as Post-Quantum Cryptography (PQC)
- PQC algorithms capable of
  - being implemented using any traditional methods, including software and hardware
  - running efficiently on any modern computing platforms: PCs, tablets, smartphones, servers with hardware accelerators, etc.

Post-Quantum Cryptography Efforts

- New public-key cryptographic families: mid-1990s-present
- Series of PQCrypto Conferences: 2006-present
- NIST Workshop on Cybersecurity in a Post-Quantum World 2015
- NIST announcement of standardization plans at PQCrypto 2016: Feb. 2016
- NIST Call for Proposals and Request for Nominations for Public-Key Post-Quantum Cryptographic Algorithms: Dec. 2016

  Deadline for submitting candidates: November 30, 2017

- Time of Standard Development + Time of Standard Deployment
  + Max. Protection Time
  must be smaller than
  Time to Develop Sufficiently Large Quantum Computer

Promising PQC Families

Family	Encryption	Signature	Key Agreement
Hash-based		XX	
Code-based	XX	X	
Lattice-based	XX	X	
Multivariate	X	XX	
Supersingular Elliptic Curve Isogeny			XX

XX – high-confidence candidates,   X – medium-confidence candidates

Our Objectives

Paving the way for the future comprehensive, fair, and efficient hardware benchmarking of PQC candidates through

1. **Uniform Hardware API**
2. **Uniform & Efficient Development Process based on**
    a. detailed flow diagrams
    b. choice of supported parameter sets
    c. top-level & lower-level block diagrams
    d. cycle-based timing analysis
    e. Algorithmic State Machine (ASM) charts
    f. Register-Transfer Level (RTL) code
    g. software-generated test vectors
    h. comprehensive testbenches
    i. results of synthesis and implementation
    j. analysis of results & lessons learned

Proposed Uniform Hardware API

Minimum Compliance Criteria

- Encryption & decryption, or
  Signature generation & verification
- External key generation (e.g., in software)
- Permitted data port widths, etc.

Communication Protocol

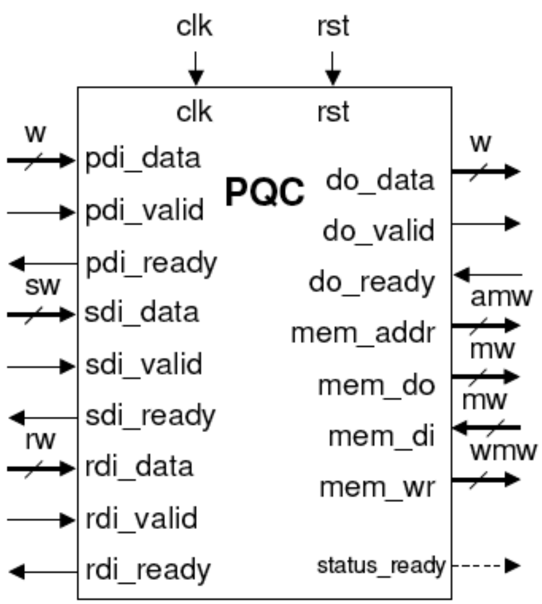
instruction = ENC
seg_0_header
seg_0 = Sys Param
seg_1_header
seg_1 = Pub Key
seg_2_header
seg_2 = Msg

(PDI)

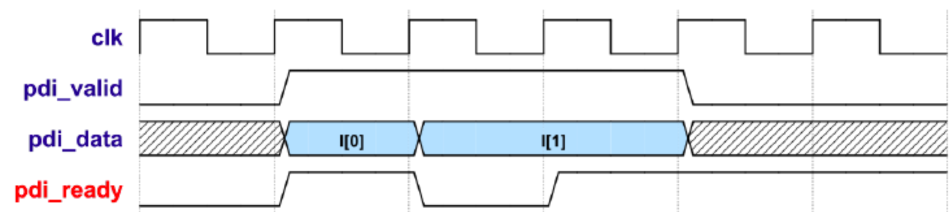
seg_0_header
seg_0 = Ciphertext
Status

(DO)

Interface



Timing Characteristics



Algorithms Selected for a Pilot Study

1. NTRUEncrypt Short Vector Encryption Scheme (SVES)

   fully compliant with

   IEEE 1363.1 Standard Specification for Public Key Cryptographic Techniques Based on Hard Problems over Lattices

   Parameter sets:

   - Optimized for speed
   - 192-bit security: ees1087ep1: p=3, q=2048, N=1087, df=dr=63
   - 256-bit security: ees1499ep1: p=3, q=2048, N=1499, df=dr=79

2. Multivariate Rainbow Signature Scheme

   Parameter set:

   - (17,12)(1,12)
   - 80-bit security level

NTRUEncrypt Standards

- IEEE 1363.1 Standard Specification for Public Key Cryptographic Techniques Based on Hard Problems over Lattices, 2009
- Financial Services Industry's Accredited Standards Committee X9, ANSI X9.98-2010, Lattice-Based Polynomial Public Key Establishment Algorithm for the Financial Services Industry, 2010
- Consortium for Efficient Embedded Security, Efficient Embedded Security Standards (EESS), EESS #1: Implementation Aspects of NTRUEncrypt, 2015
- J. Schanck, W. Whyte, Z. Zhang, "Quantum-Safe Hybrid (QSH) Ciphersuite for Transport Layer Security (TLS) version 1.3," TLS Working Group Internet Draft, Oct. 2016 (work in progress)

Implementation Assumptions

- Optimization for speed
  - Minimum Latency
  - Maximum Number of Operations per Second
- Application: high-end servers supporting a very large number of TLS, IPSec, and other protocol transactions
- Key generation performed externally, e.g., in software
- No countermeasures against side-channel attacks
- Full Compliance with Existing Standards (if available)

NTRUEncrypt – Core Functionality (1)

Parameters:

N – prime

p – small prime, typically 3

q – power of 2, typically 2048

Basic Operations:

Polynomial Multiplication, Addition, Subtraction in the ring $Z/qZ[X]/X^N-1$

Private Key:

f = 1+pF, where F – random polynomial with small coefficients {-1, 0, 1}

Public Key:

$h = f^{-1} * g \bullet p$, and

g – random polynomial with small coefficients {-1, 0, 1}

NTRUEncrypt – Core Functionality (2)

Encryption:

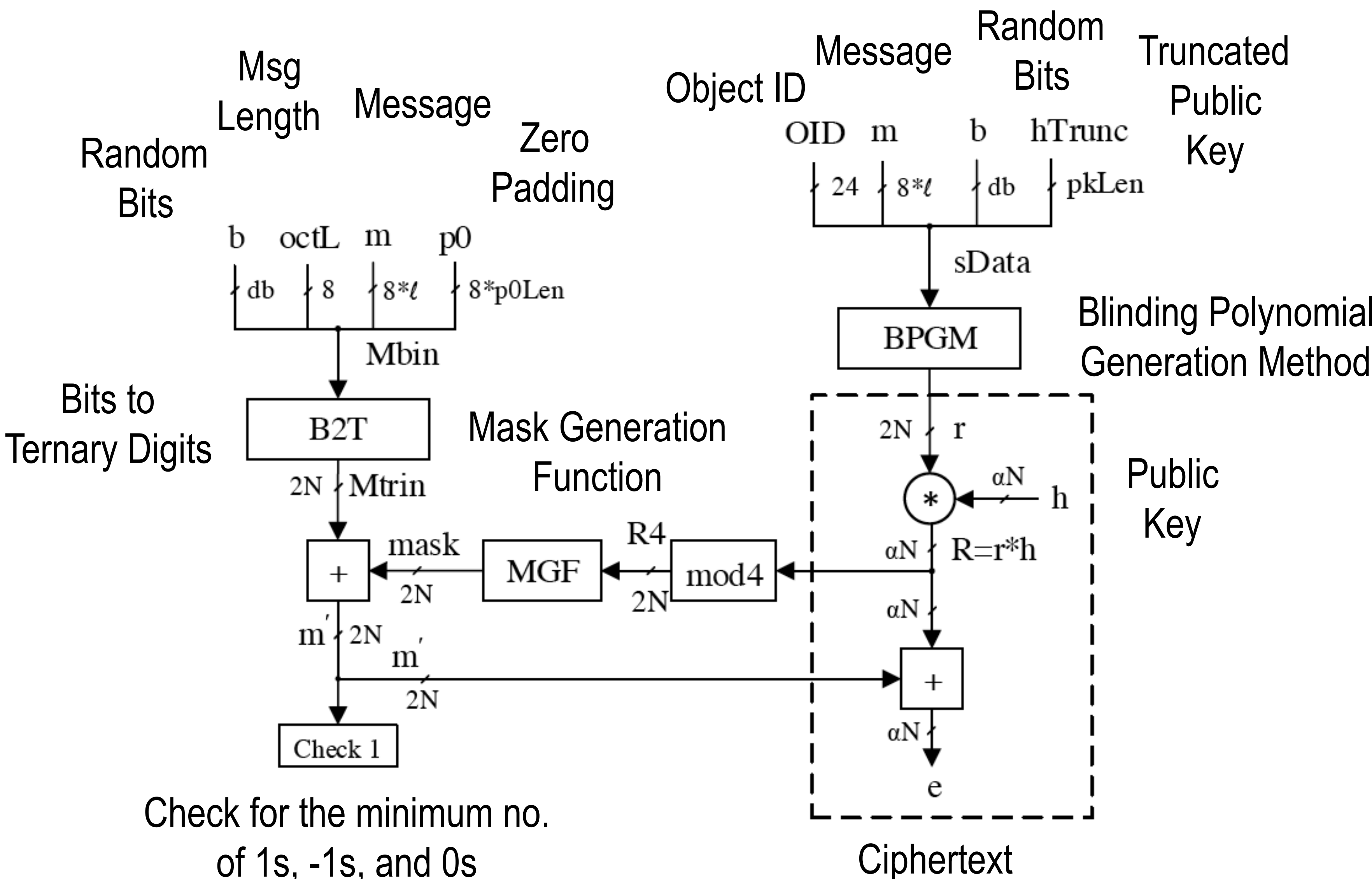
$$e = r * h + m \pmod{q}$$

where r is a random polynomial with small coefficients

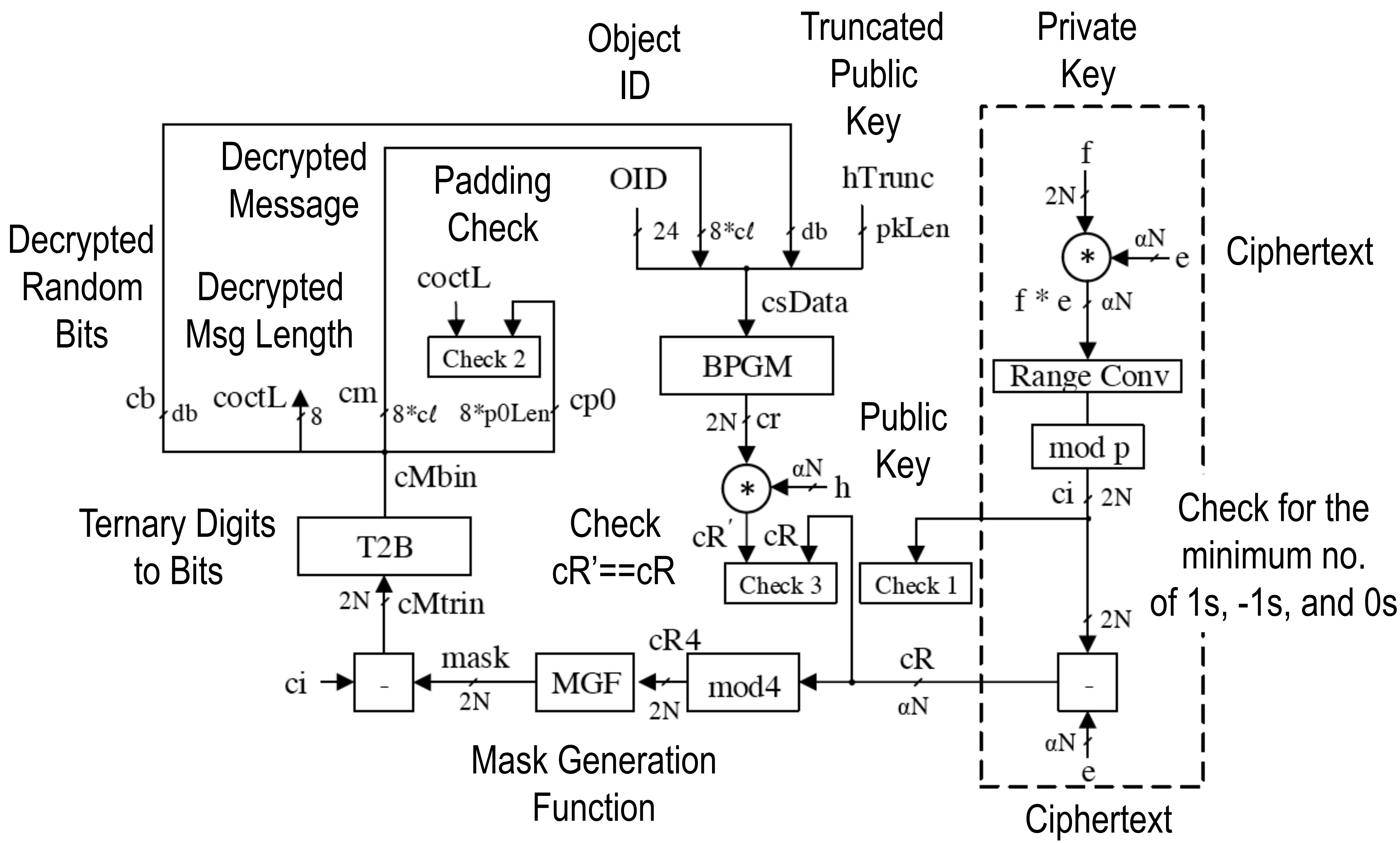
Decryption:

1) calculate $f * e \pmod{q}$
2) shift coefficients of the obtained polynomial to the range $[-q/2, q/2)$,
3) reduce the obtained coefficients mod p

NTRUEncrypt – Flow Diagram for Encryption



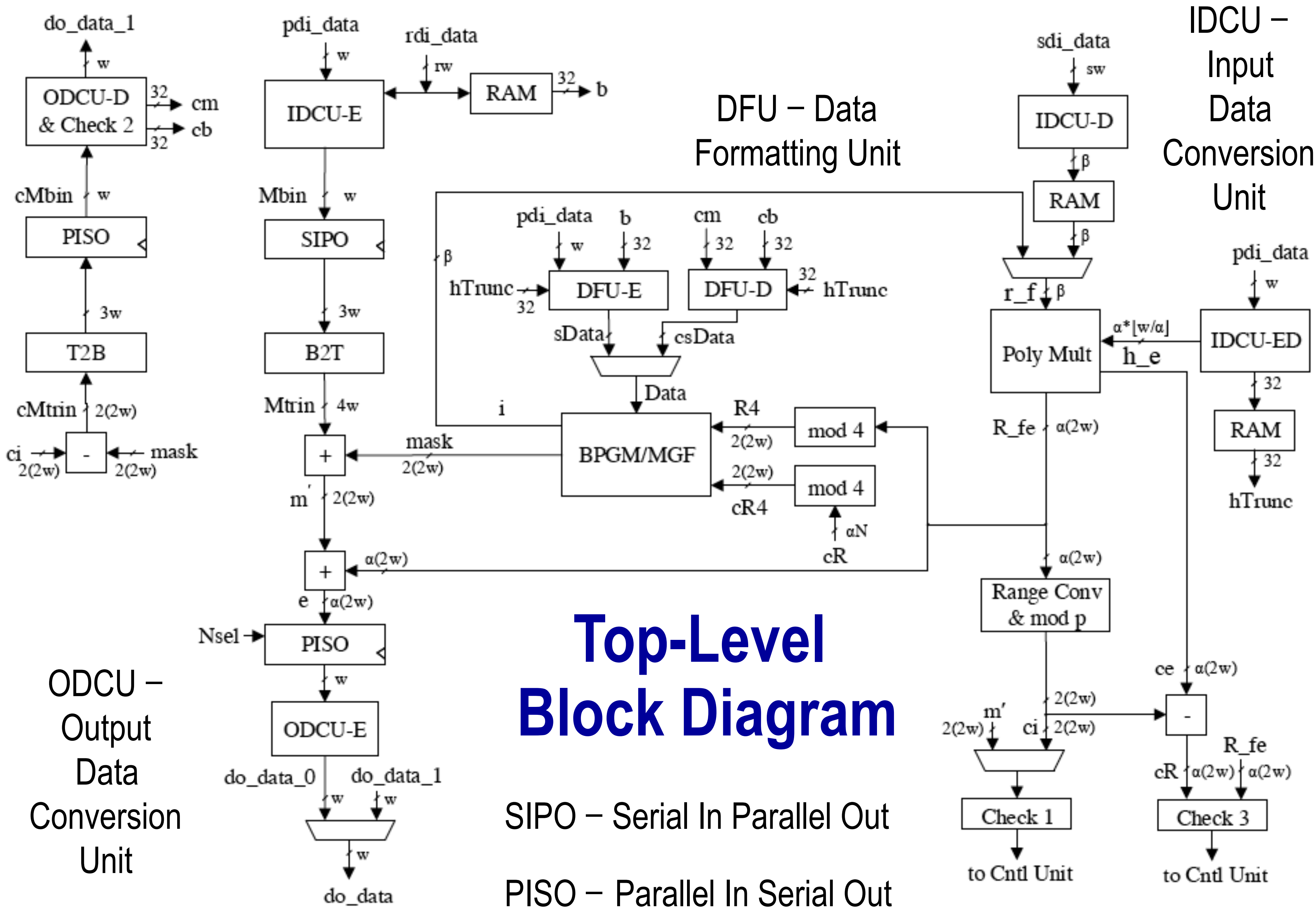
NTRUEncrypt – Flow Diagram for Decryption



NTRUEncrypt: Supported Parameter Sets

Parameter Set		ees1499ep1	ees1087ep1
Name	Description		
PARAMETERS OF ALGORITHM – BASIC			
N	Dimension (rank) of the polynomial ring	1499	1087
dr	No. of 1s and no. of -1s in r	79	63
df	No. of 1s and no. of -1s in F	79	63
db	No. of random bits of b	256	192
dm0	The minimum number of 0s, 1s and -1s in m' and ci, used in Check 1	79	63
maxMsg LenBytes	Maximum message length in bytes	247	178
pkLen	No. of bits of h to include in sData	256	192
q	"Big" modulus	2048	2048
p	"Small" modulus	3	3
c	Polynomial index generation constant	13	13
hiLen	Hash function input block size in bits	512	512
hoLen	Hash function output block size in bits	256	256

Top-Level Block Diagram



SIPO – Serial In Parallel Out

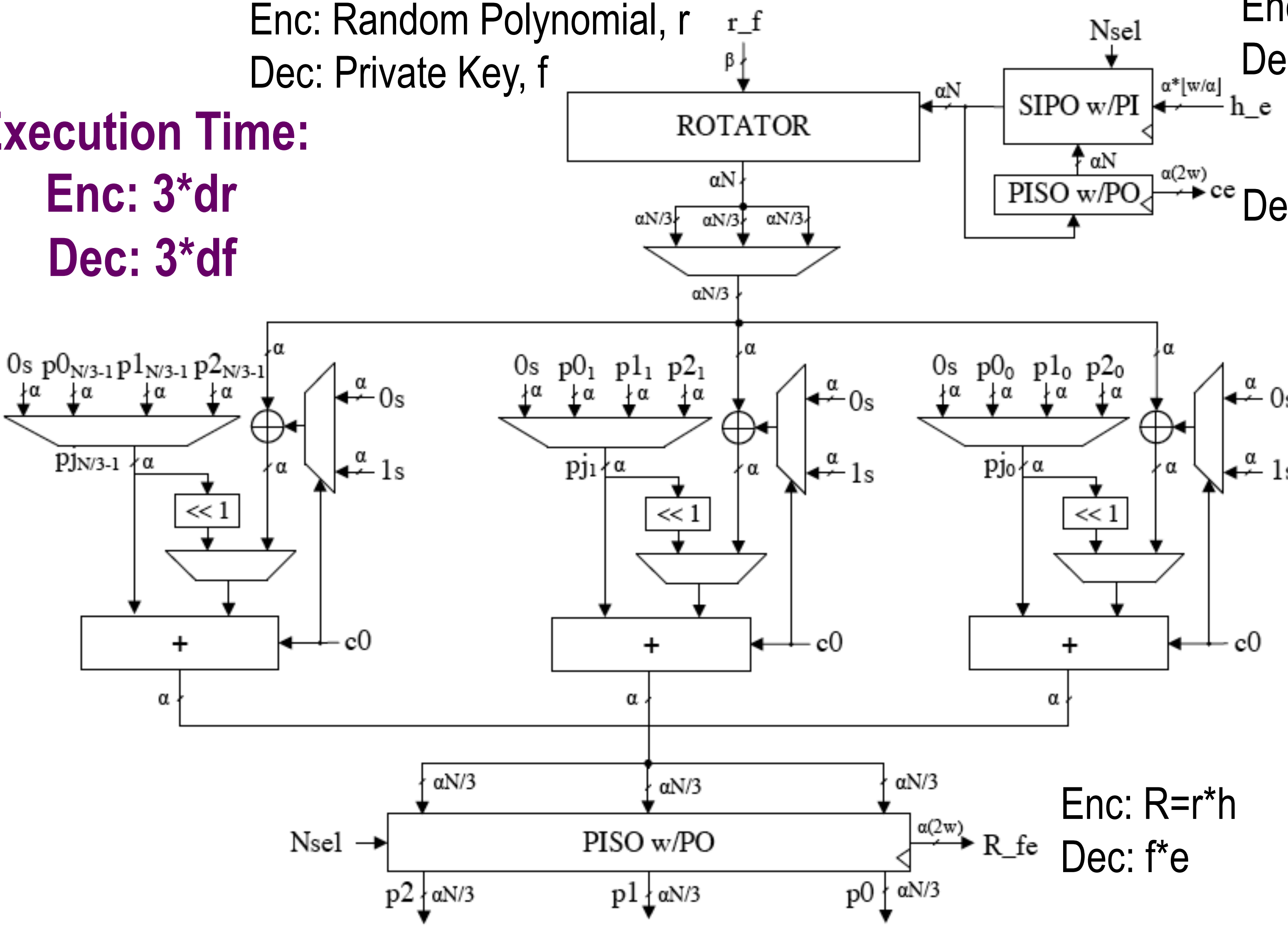
PISO – Parallel In Serial Out

Block Diagram of Polynomial Multiplier

Enc: Random Polynomial, r
Dec: Private Key, f

Enc: Public Key, h
Dec: Ciphertext, e

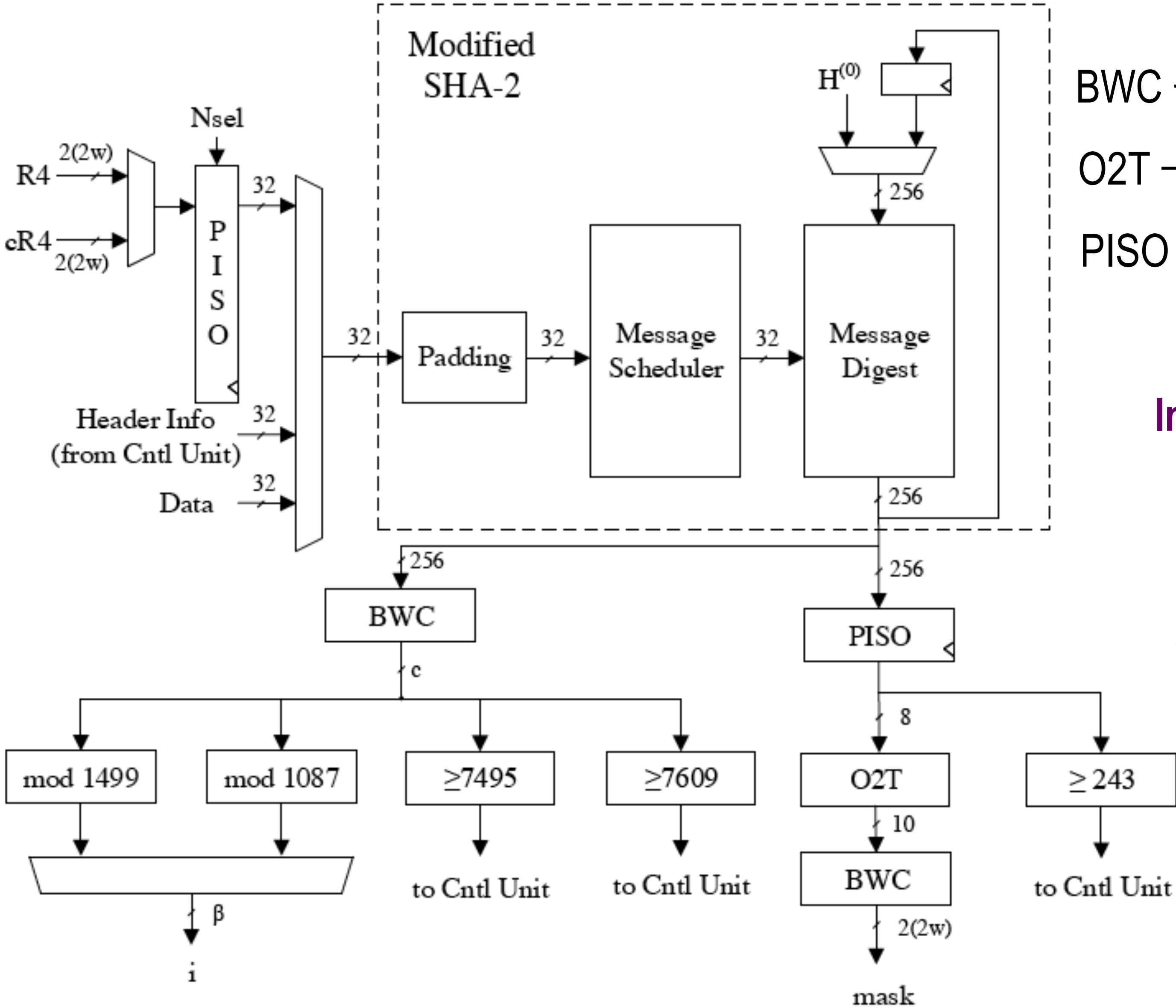
Execution Time:
Enc: 3*dr
Dec: 3*df



Dec: Ciphertext, e

Enc: R=r*h
Dec: f*e

Block Diagram of Blinding Polynomial Generation Method / Mask Generation Function



BWC – Bit Width Conversion

O2T – Octets to Ternary Digits

PISO – Parallel In Serial Out

Implementation of SHA-2 modified to share computations for hashing of strings starting from the same common substring

Sharing Computations for Multiple Overlapping Inputs

Case 1: t+n-1 input blocks

h(sData||C1)=h(**sData_0, …, sData_t-1, sData_t||C1**)

h(sData||C2)=h(sData_0, …, sData_t-1, **sData_t||C2**)

h(sData||C3)=h(sData_0, …, sData_t-1, **sData_t||C3**)

……

h(sData||Cn)=h(sData_0, …, sData_t-1, **sData_t||Cn**)

Case 2: t+2(n-1) input blocks

h(sData||C1)=h(**sData_0, …, sData_t-1, sData_t||C1_0, C1_1**)

h(sData||C2)=h(sData_0, …, sData_t-1, **sData_t||C2_0, C2_1**)

h(sData||C3)=h(sData_0, …, sData_t-1, **sData_t||C3_0, C3_1**)

……

h(sData||Cn)=h(sData_0, …, sData_t-1, **sData_t||Cn_0, Cn_1)**

Implementation Platforms

Hardware:

FPGA Family: Xilinx Kintex-7 UltraSCALE
Device: XCKU035-FFVA1156
Technology: 20nm CMOS

Software:

Cortex A9 ARM Core of Zynq 7020

Major Component Operations Resource Utilization & Performance

Operation	LUTs: Slices	Clk Freq. [MHz]
Poly Mult	140,512 : 25,099	74.4
BPGM	1971 : 421	171.0
MGF		
B2T	64 : 34	904.0
T2B	64 : 35	984.3
Poly Add	1338 : 272	316.3
Poly Sub 1	1221 : 258	331.2
Poly Sub 2	74 : 64	540.2

PolyMult contributes to over 90% of area and limits clock frequency

Comparison with Previous Work on Implementing Polynomial Multiplication

Source	Resources	Clk Freq. [MHz]	Latency [cycles]	Latency [μs]
Parameter set: ees1499ep1				
Liu et al., 2016*	83,949 LEs	63.6	867	13.6
This Work	140,512 LUTs	74.4	474	6.4
	Speed-up	x1.17	x1.83	x 2.14
Parameter set: ees1087ep1				
Liu et al., 2016*	60,876 LEs	73.7	638	8.7
This Work	140,512 LUTs	74.4	378	5.1
	Speed-up	x1.01	x1.69	x 1.70

* B. Liu and H. Wu, “Efficient Multiplication Architecture over Truncated Polynomial Ring for NTRUEncrypt System,” IEEE International Symposium on Circuits and Systems, ISCAS 2016. **Platform: Altera Cyclone IV EP4CE115F23C7**.

Profiling of Software Implementation on Cortex A9 ARM

Software Function	Hardware Equivalent	Clock cycles	% of Total Time
ntru_gen_poly	Performing BPGM on sData & calculating R using Poly Mult (in a pipelined fashion)	24,779	2.3%
ntru_octets_2_elements		12,728	1.2%
ntru_ring_mult_product_indices		950,892	89.4%
ntru_coeffs_mod4_2_octets	Calculating cR4 using mod 4 & mask using MGF	9,427	0.9%
ntru_mgftp1		30,703	2.9%
ntru_bits_2_trits		3,020	0.3%
adding Mtrin to mask	Calculating m' using Poly Add & performing Check 1	8,108	0.8%
ntru_poly_check_min_weight		6,910	0.6%
add_m'		8,672	0.8%
elements_2_octets	Unloading ciphertext e	13,549	1.3%
Total		**1,068,788**	**100.0%**

Profiling of Hardware Implementation on Xilinx Virtex-7

Operation	Latency (clock cycles)	% of Total Time	Latency (clock cycles)	% of Total Time
	ees1499ep1		ees1087ep1	
ENCRYPTION				
Performing BPGM on sData & calculating R using Poly Mult (in a pipelined fashion)	890	38.8%	701	39.5%
Calculating cR4 using mod 4 & mask using MGF	1005	43.8%	787	44.3%
Calculating m' using Poly Add & performing Check 1	97	4.2%	70	3.9%
Unloading ciphertext e	300	13.1%	218	12.3%
Total	**2292**	100%	**1776**	100%

Hash Function Bottleneck in Hardware

Software

- **Poly Mult amounts to about 90% of the total execution time**

Hardware

- **Execution time dominated by hash-based**
  - **MGF: Mask Generation Function: 44%**
  - **BPGM: Blinding Polynomial Generation Method: 39.5%**
- **Poly Mult almost completely overlapped with the computations of BPGM through the use of pipelining**
- **Poly Mult naturally parallelizable**
- **Hash function naturally sequential**

Possible Improvements

To Address the Hash Function Bottleneck:

Architecture-Level:

- Unrolled Implementation of SHA-2

Algorithmic-Level (changes in the IEEE & EES standards required):

- SHA-3 instead of SHA-2
- Pseudorandom function based on the pipelined AES

To Address Other Encountered Problems:

Algorithmic-Level (changes in the IEEE & EES standards required):

- Eliminating (or at least reducing) the dependence of the execution time on message size

Rainbow – Core Functionality (1)

Parameters:

o1=o2=12 : # of Layer 1/Layer 2 oil variables
v1=17 : # of Layer 1 vinegar variables
v2'=1 : # of random Layer 2 vinegar variables
v2=v1+o1+v2'=30 : # of Layer 2 vinegar variables

n = v2+o2 = 42 : total # of variables; signature size
m = o_1+o_2 = 24 : message size

Basic Operations:

Solving System of Equations
Polynomial Multiplication
with irreducible polynomial $x^8 + x^6 + x^3 + x^2 + 1$
Polynomial Addition

Rainbow – Core Functionality (2)

Public Key:

Map F', which consists of o_1+o_2 multivariate quadratic polynomials of n variables

$F' = L_1 \circ F \circ L_2$

where "$\circ$" denotes composition of two maps,

F consists of *randomly chosen* quadratic polynomials of special form

L_1, L_2 are *randomly chosen* invertible affine transformations

Private Key:

Used as a trap-door to find a solution to F'(sgn_out) = msg_in

Consists of maps L_1^{-1}, L_2^{-1}, and F,

F is the center mapping, with 2 layers,

It contains multivariate oil-vinegar polynomial sets P_1 and P_2,

Rainbow – Core Functionality (3)

Multivariate Oil-Vinegar Polynomials

Consist of terms of type;

- **vinegar-vinegar (VV), $\alpha_{ij}x_ix_j$,** where x_i, x_j are vinegar variables
- **vinegar-oil (VO), $\alpha_{ij}x_ix_j$,** where x_i is a vinegar, x_j is an oil variable
- **vinegar only (V), β_ix_i,** where x_i is a vinegar variable
- **oil only (O), β_ix_i,** where x_i is an oil variable
- **constant (C), η**

The set of all polynomials of a given Rainbow layer, l, is denoted by P_l. Furthermore, let an element of P_l, called q_k, be made of terms VV, VO, V, O, and C, corresponding to the types described above.

Since all coefficients α_{ij}, β_i, and η are elements of GF(2^8) and thus, have a size of 1 byte, therefore we have,

$$|q_k| = |VV| + |VO| + |V| + |O| + 1$$

Rainbow: Flow Diagram for Signature Generation

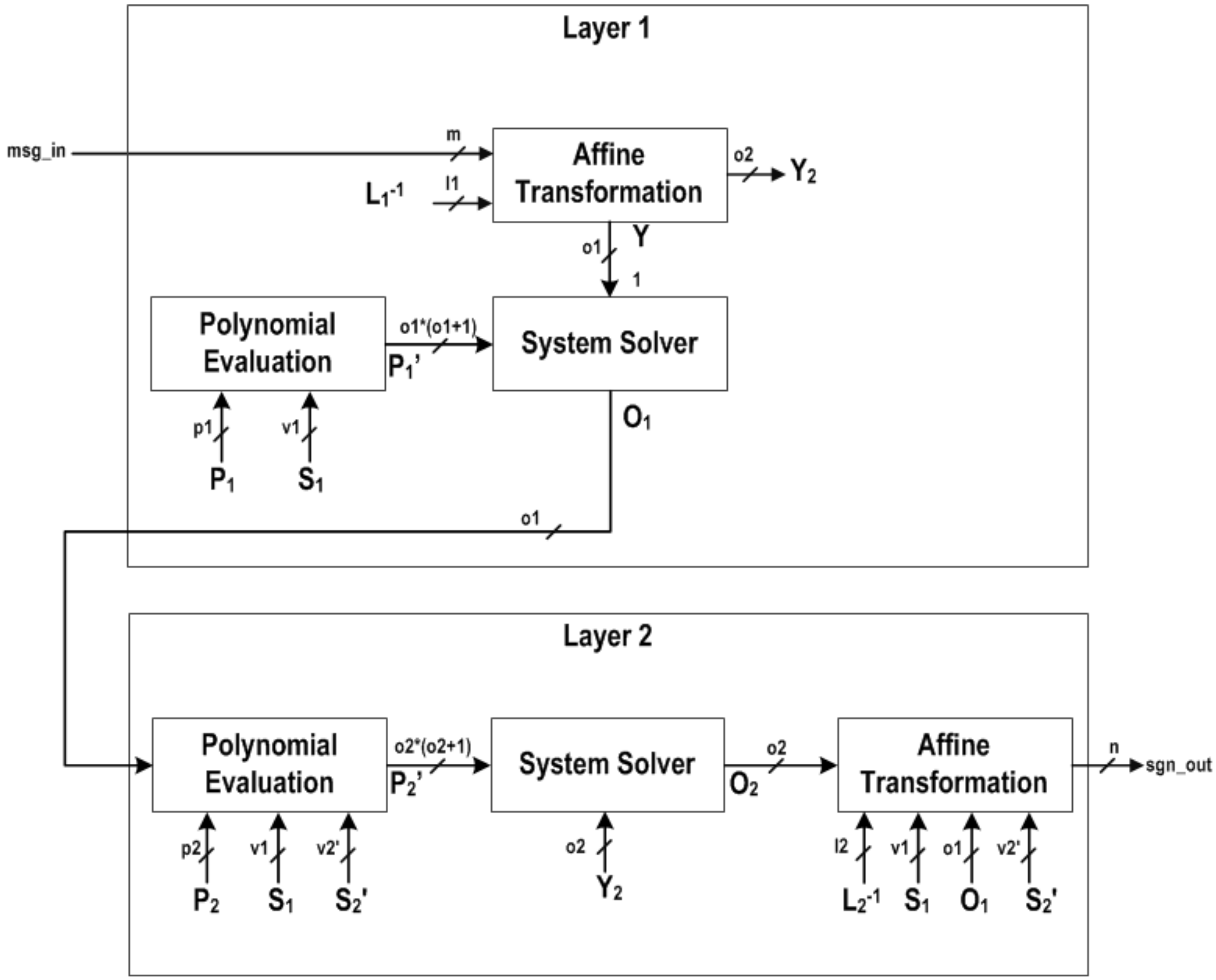
Oil-vinegar sets:
$S1 = \{x_1, ..., x_{v1}\} = \{x_1, ..., x_{17}\}$
$O1 = \{x_{v1+1}, ..., x_{v1+o1}\} = \{x_{18}, ..., x_{29}\}$
$S2' = x_{v2} = x_{30}$
$S2 = S1 \mid O1 \mid S2' = \{x_1, ..., x_{30}\}$
$O2 = \{x_{v2+1}, ..., x_n\} = \{x_{31}, ..., x_{42}\}$

Oil-vinegar parameters:
o1 = o2 = 12
v1 = 17
v2' = 1
v2 = v1 + o1 + v1' = 30

Affine Transformation parameters:
m = o1 + o2 = 24
l1 = (o1+o2)*(o1+o2+1) = 24*25
= 600
l2 = n*(n+1) = 42*43 = 1806

Polynomial parameters:
p1 = 4,644
p2 = 10,416

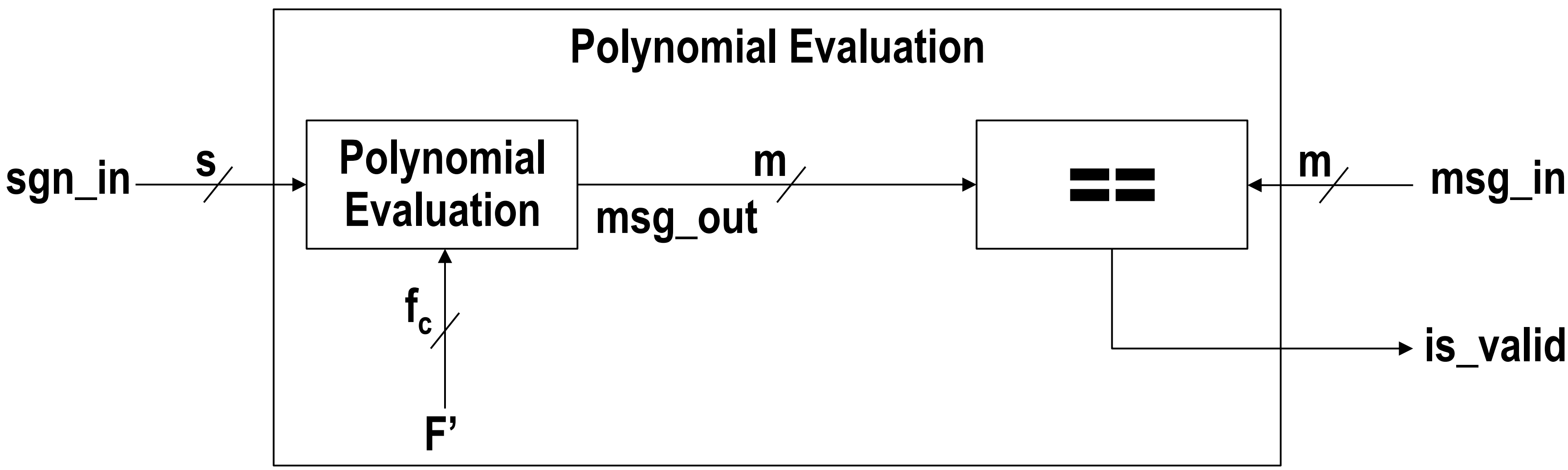
All sizes in bytes



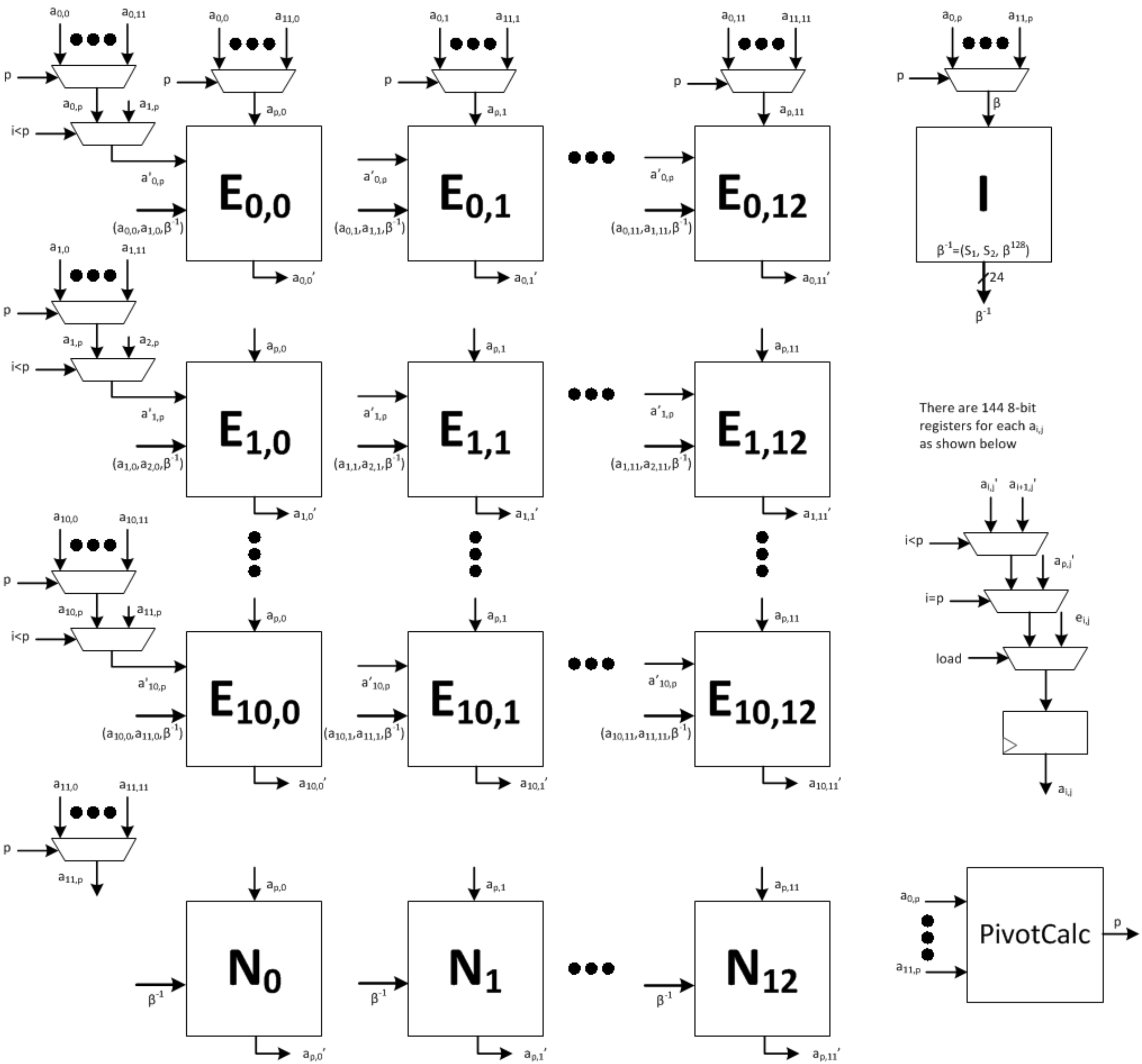
Rainbow – Flow Diagram for Signature Verification

Signature Verification:

sgn_in: signature, msg_in: message
Evaluate F'(sgn_in) = msg_in ?



f_c = 22,704 bytes



Block Diagram of System Solver

E – Elimination

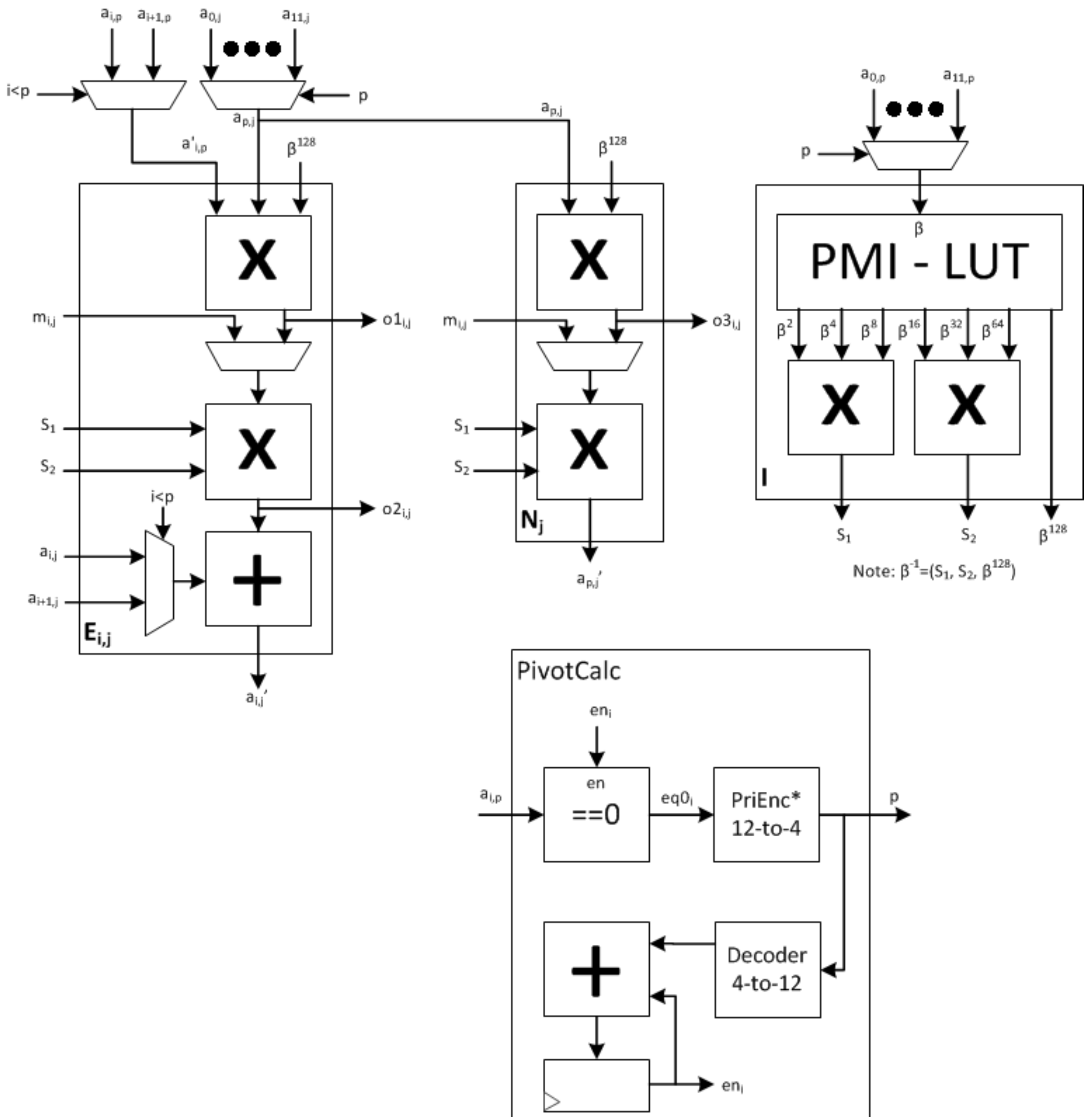
N – Normalization

I – Inversion

~36k LUTs

~60 MHz

12 clk cycles



Building Blocks

$E_{i,j}$ – Elimination

N_j – Normalization

I – Inversion

PivotCalc – Pivot Calculation

Rainbow Signature Scheme - Results

Component Name	Resource Utilization [LUTs]
2-input Multiplier	34
3-input Multiplier	91
I-cell (Partial Inversion)	67
N-cell (Normalization)	131
E-cell (Elimination)	198
12 x N-cell	1,572
11 x 12 x E-cell	26,136
Total N+E+I cells	**27,856**
Multiplexing Logic	**5,166**
Total System Solver	**33,022 (67%)**
Total Area	**47,881**

NTRUEncrypt vs. Rainbow Signature Scheme

Security Levels:

NTRU: **Parameter sets supporting 112, 128, 192, & 256 bit security levels**
Rainbow: **Most published parameter sets at 80-90 bit security levels**

Key Sizes:

	Security Level	Public Key Size	Private Key Size
NTRU	192	1495 B	174 B
	256	2062 B	218 B
Rainbow	80	**22704 B**	**17466 B**

Comparative Analysis of Implementation Difficulties

Feature	NTRUEncrypt	Rainbow SS
High-security levels	Easy to implement	**Challenging to implement**
Key sizes	Small	**Very Large**
Support for multiple parameter sets swapped at run time	Relatively easy to implement	**Challenging to implement**
Component operations	Standard: variable rotator, hash function	**Complex: System of Linear Equation Solver**
Dependence of the execution time on message size	**Strong**	Weak

Conclusions

- First hardware implementation of the full NTRUEncrypt-SVES scheme
- Hardware optimization for speed revealed the hash function bottleneck
- Changes in the NTRUEncrypt standards may be required to overcome this bottleneck
- State of the art implementation of the Rainbow Signature Scheme comparable to the earlier results by Tang et al. from PQCrypto 2011
- New PQC Hardware API, paving the way for the fair evaluation of candidates in the NIST standardization process

Future Work

- Constant Time Implementations
- Extension of the Rainbow implementation to higher security levels and multiple parameter sets
- Lightweight Implementations
- Resistance to Side-Channel Attacks
- Hardware Benchmarking of Candidates in the NIST Standardization Effort for the New Public-Key Post-Quantum Cryptographic Algorithms
- Possible use of High-Level Synthesis to speed-up the development and benchmarking process

Thank you!

Questions?

Questions?

http:/cryptography.gmu.edu