High-Speed Hardware for NTRUEncrypt-SVES: Lessons Learned

Malik Umar Sharif,
and
Kris Gaj
George Mason University
USA

Partially supported by NIST under grant no. 60NANB15D058

Co-Author

Malik Umar Sharif

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

Our Objective

Paving the way for the future

comprehensive, fair, and efficient

hardware benchmarking of PQC algorithms

through

1. **Uniform Hardware API**

   inspired by the APIs used in the SHA-3 and CAESAR competitions

2. **Efficient Methodology**

   based on the manually written VHDL/Verilog code

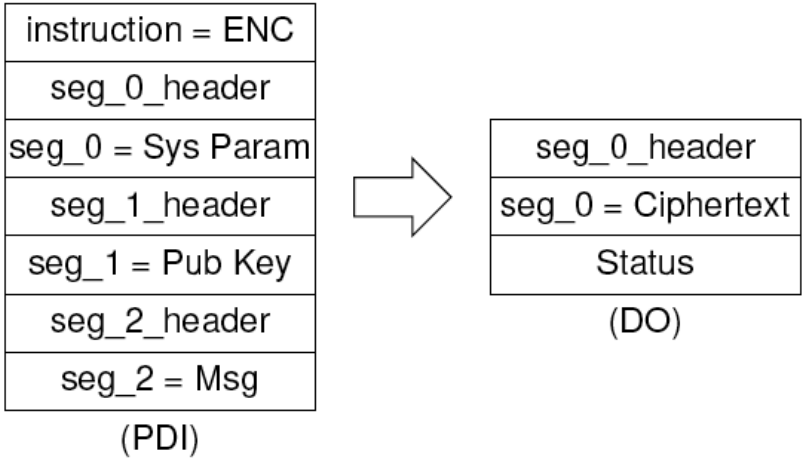
   or High-Level Synthesis

Proposed Uniform Hardware API

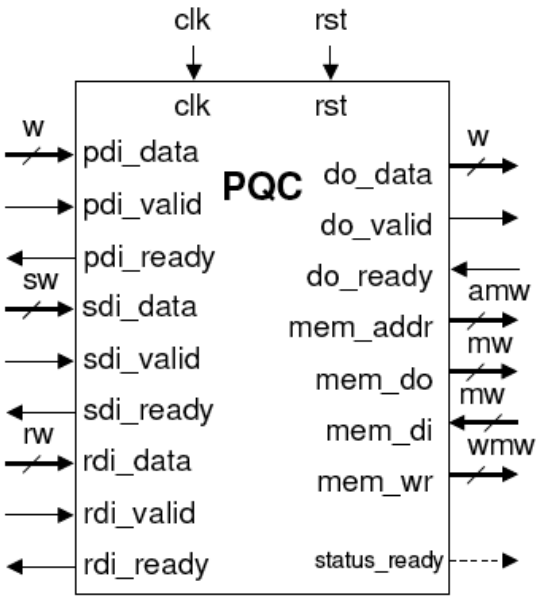
Minimum Compliance Criteria

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

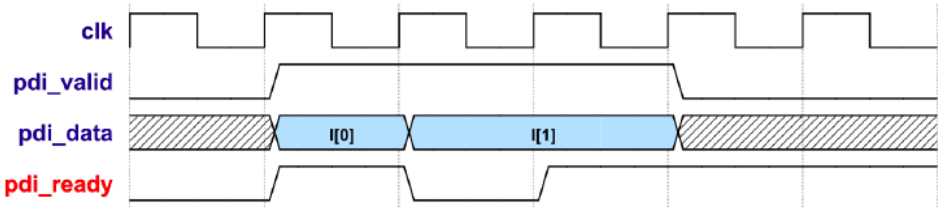
Communication Protocol



Interface



Timing Characteristics



Specification: https://cryptography.gmu.edu/athena

Algorithm Selected for a Pilot Study

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, 2009

Parameter sets:

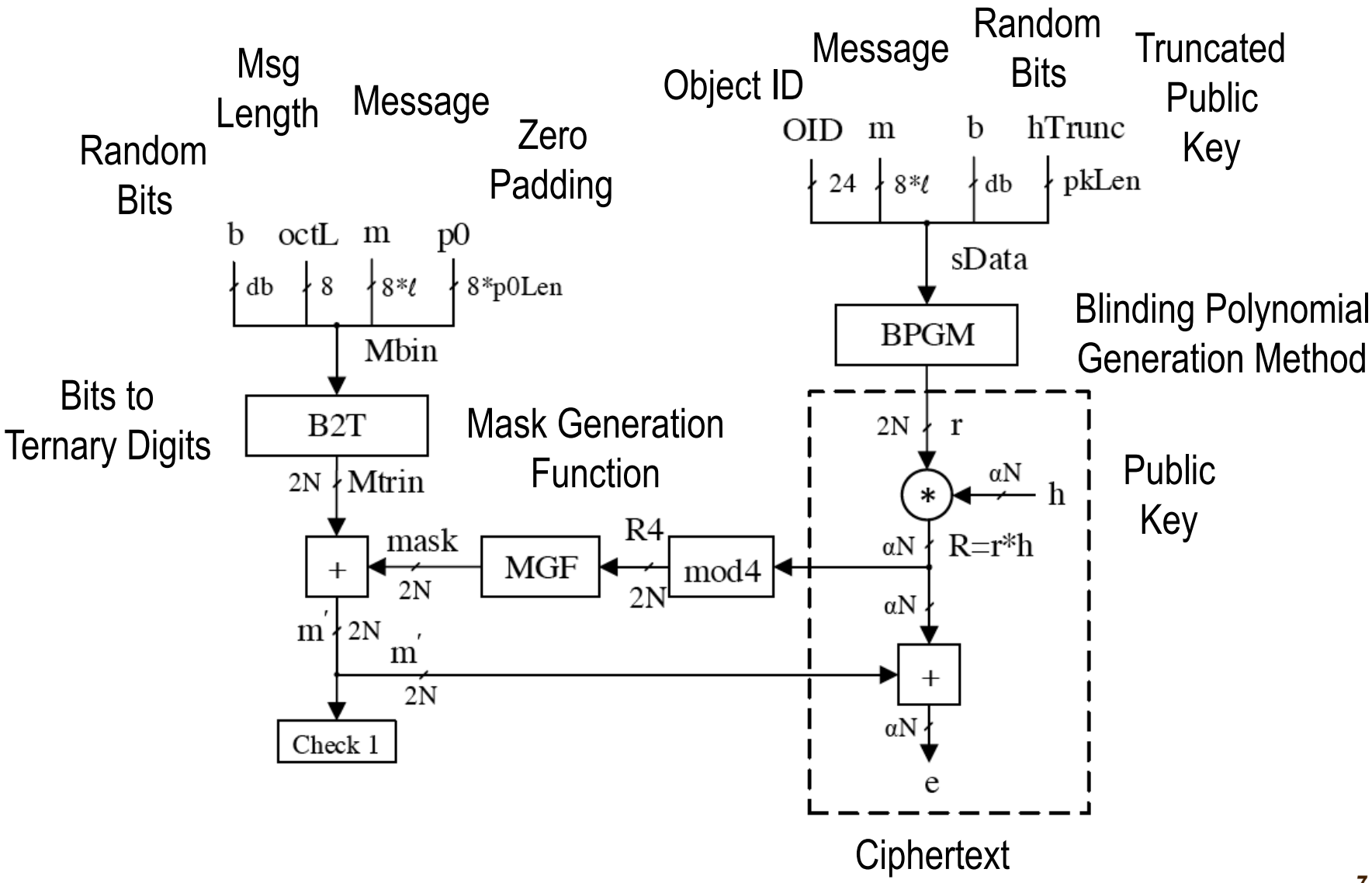
- **Optimized for speed**
- **192-bit security: ees1087ep1: p=3, q=2048, N=1087, df=dr=63**
- **256-bit security: ees1499ep1: p=3, q=2048, N=1499, df=dr=79**

Security Level	Public Key Size	Private Key Size
192	1495 B	174 B
256	2062 B	218 B

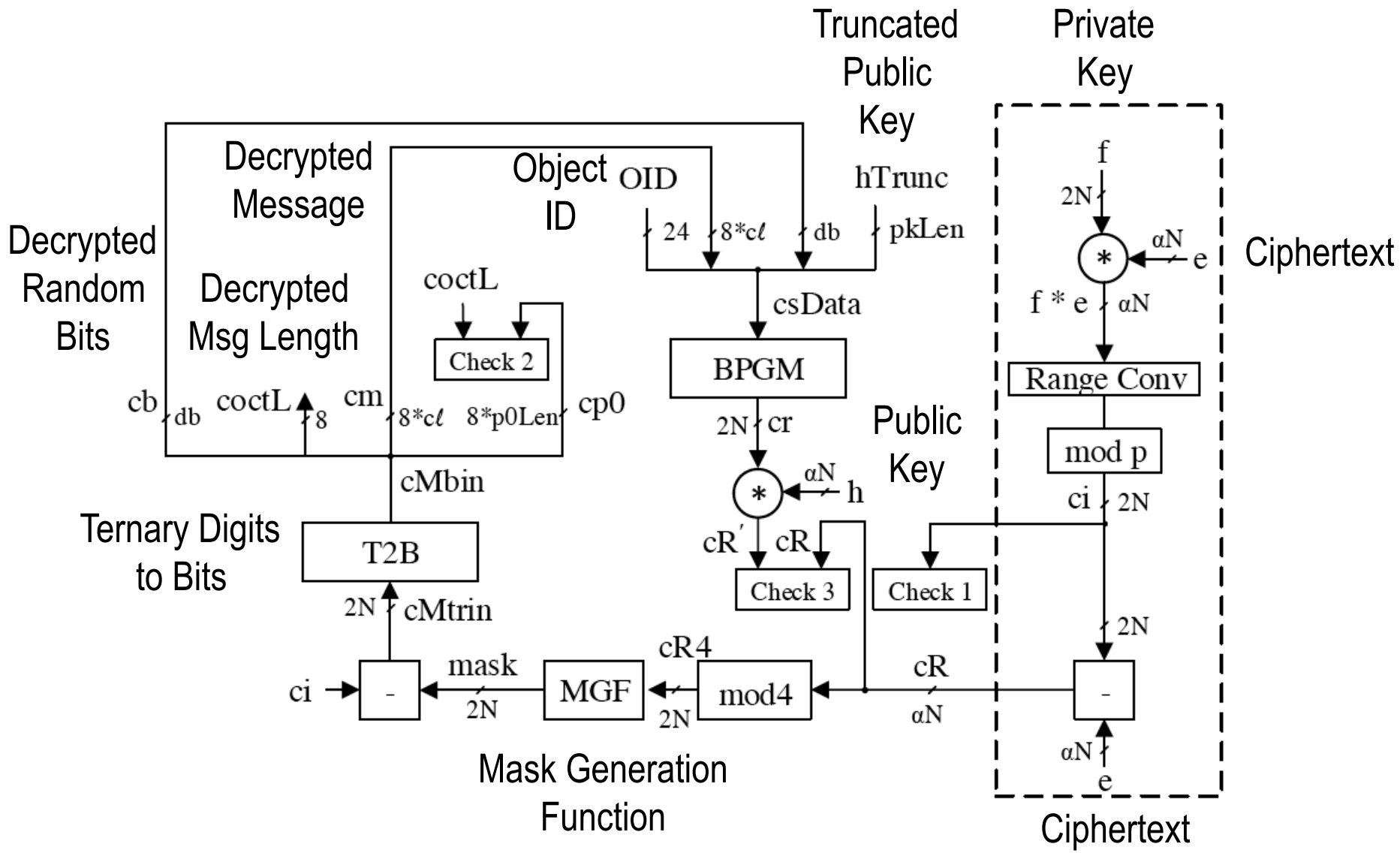
Implementation Assumptions

- 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

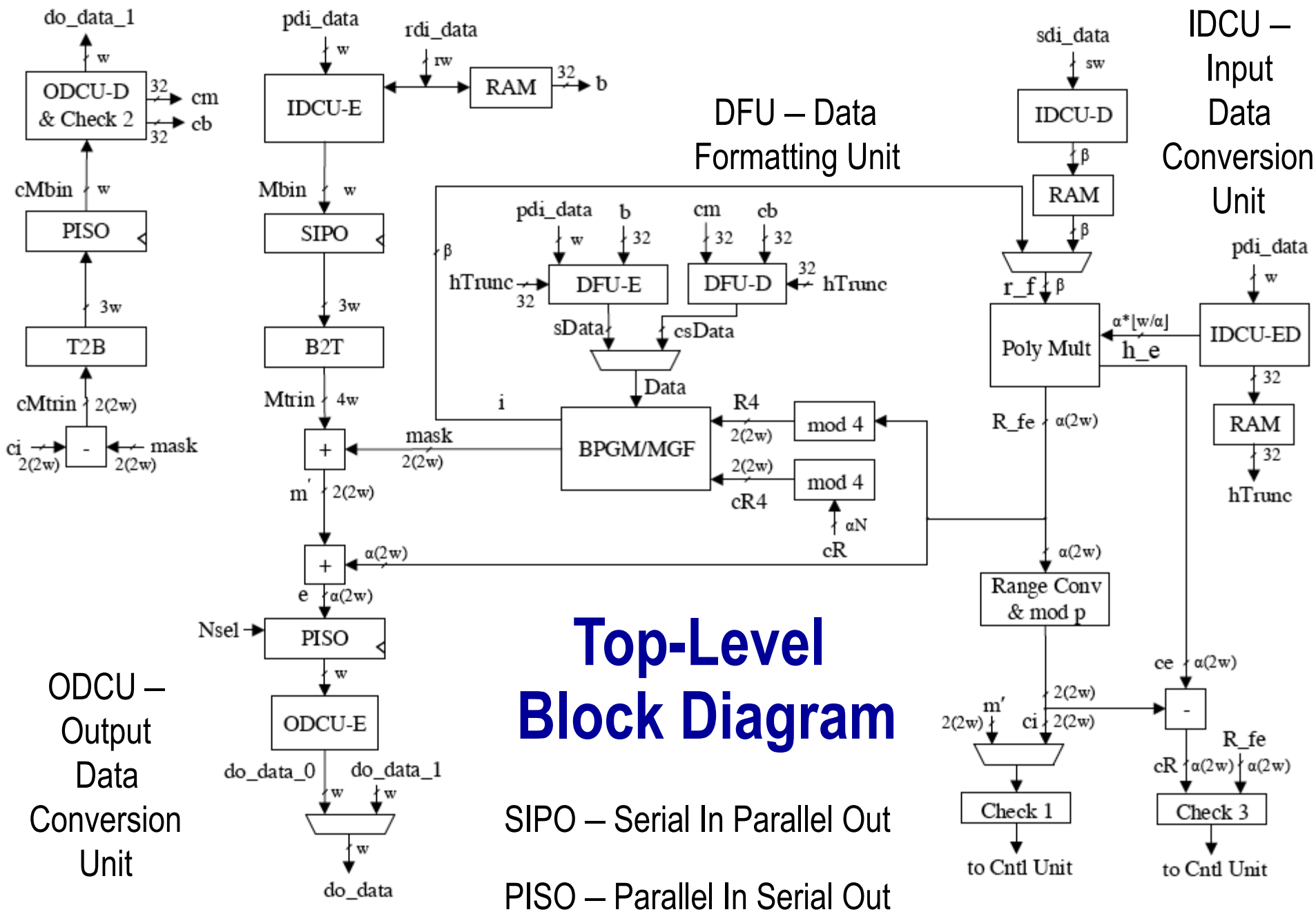
NTRUEncrypt – Flow Diagram for Encryption



NTRUEncrypt – Flow Diagram for Decryption



Top-Level Block Diagram



SIPO – Serial In Parallel Out

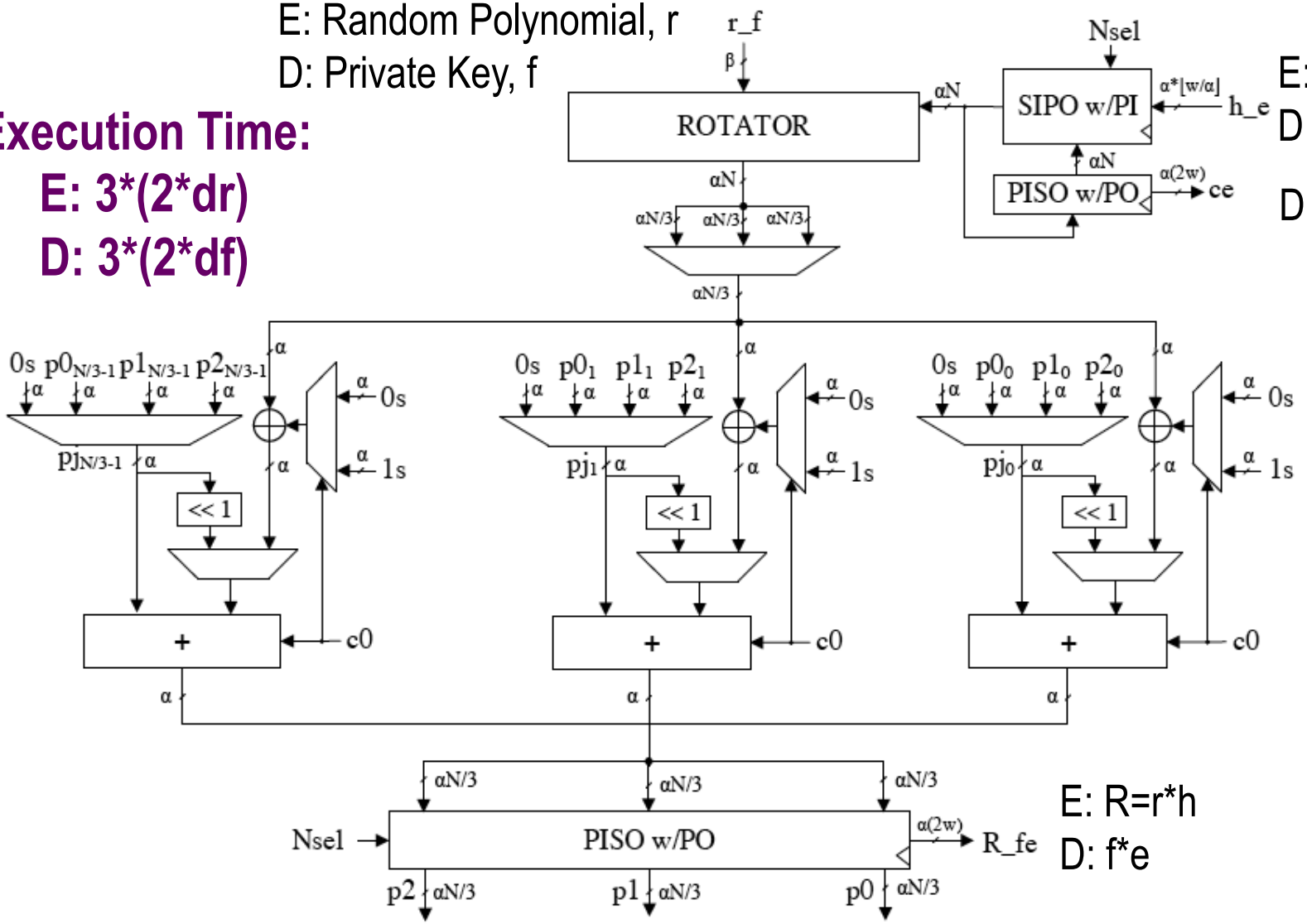
PISO – Parallel In Serial Out

Block Diagram of Polynomial Multiplier

E: Random Polynomial, r
D: Private Key, f

Execution Time:

E: 3*(2*dr)
D: 3*(2*df)

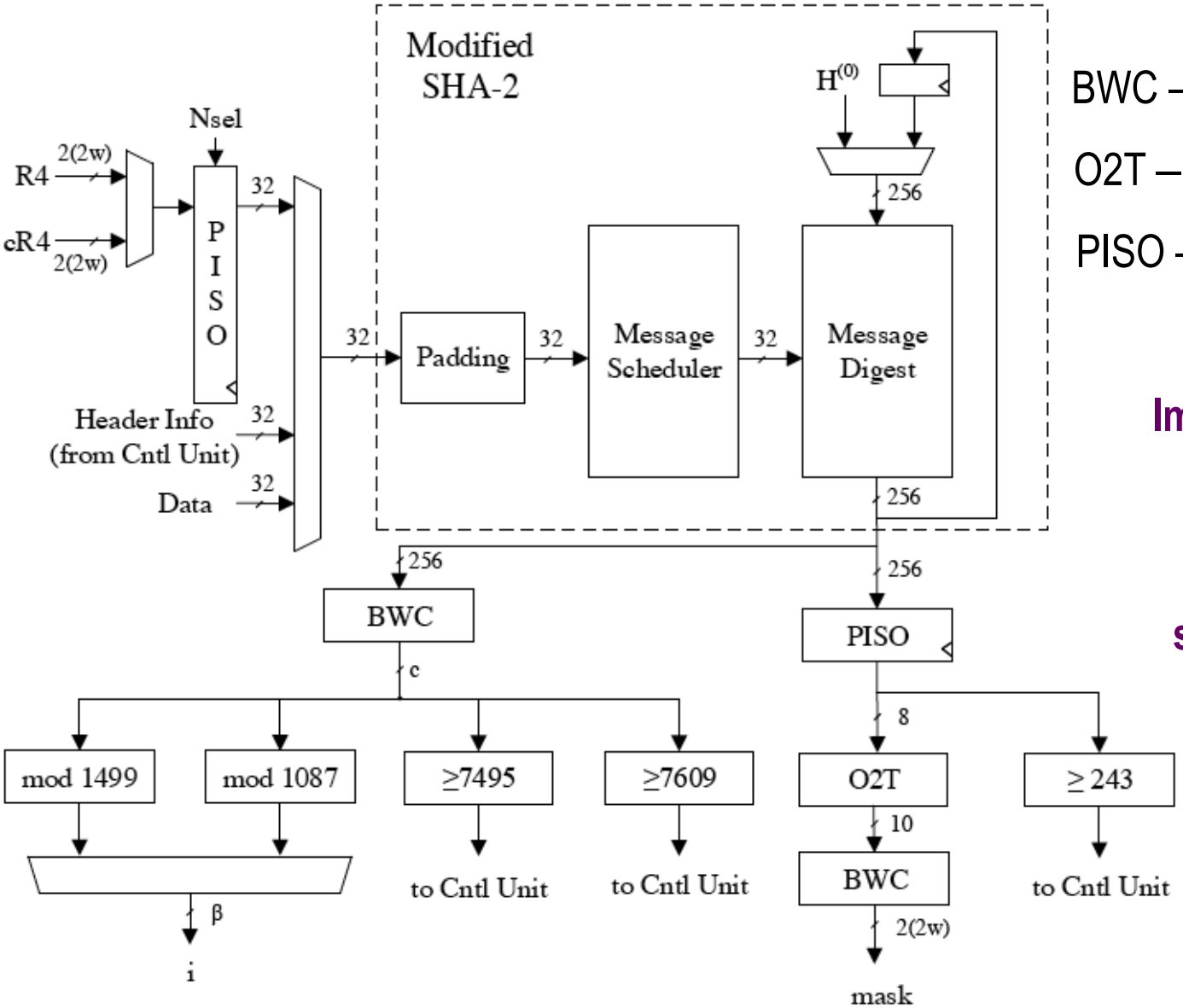


E: Public Key, h
D: Ciphertext, e

D: Ciphertext, e

E: R=r*h
D: f*e

Block Diagram of Blinding Polynomial Generation Method (BPGM) / Mask Generation Function (MGF)



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

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 Multiplicaover Truncated Polynomial Ring for NTRUEncrypt System," IEEE International Symposium on Circuits and Systems, ISCAS 2016

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 standard 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 standard required):

- Using SHA-2 instead of SHA-1 in lower-security parameter sets
- Eliminating (or at least reducing) the dependence of the execution time on message size

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
- New PQC Hardware API, paving the way for the fair evaluation of candidates in the NIST standardization process

Thank you!

http:/cryptography.gmu.edu

http:/cryptography.gmu.edu/athena