Modern Cryptography

Post-Quantum Cycles - Kyber, Saber and Dilithium

Dr. Sara Ricci

Brno University of Technology
ricci@vut.cz

Table of contents

1. Hash Functions
2. Comparison of NIST competitors
3. MLWE and MLRW problems
4. Kyber and Saber schemes
5. CRYSTALS-Dilithium Signature
6. Appendix: NTT transform

Table of contents

Hash Functions

Cryptographic Hash Function

Definition

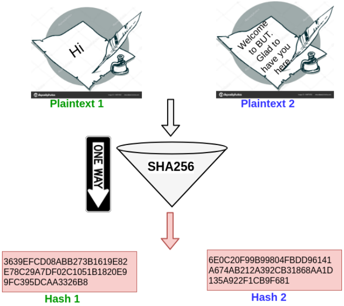
A collision-resistant hash function *is a map used for representation of long (any length) string by short (fixed-length) string.*

Collision-resistant

For given output of hash function $\mathcal{H}(m)$, it is not computationally feasible to find m, m' such that $\mathcal{H}(m) = \mathcal{H}(m')$.

Applications:

- Message integrity
- Digital signature
- Password verification
- Proof-of-work
- File or data identifier
- also KEM



Cryptographic Hash Function Families

Block Cipher-design

Based on: ad-hoc design principles.

Examples: SHA-2 (2012), SHA-3 (2015), RIPEMD (2011), BLAKE2 (2009)

Operations: bitwise op., modular additions, compression funct.

Provable Secure-design 1

Based on: **standard** number theoretic problems.

Examples: VSH (IF, 2005), ECOH (ECC, 2008)

Provable Secure-design 2

Based on: **post-quantum** number theoretic problems.

Examples: FSB (code-based, 2003), SWIFFT (lattice-based, 2008),

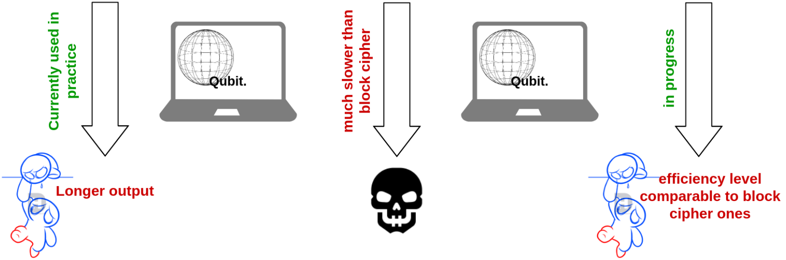


Table of contents

Comparison of NIST competitors

NIST Finalist (2020)

NIST, July 2020			
	Signature	KEM/Encryption	Overall
Lattice-based	5 (2)	21 (3)	26 (5)
Code-based	2 (0)	17 (1)	19 (1)
Multi-variate	7 (1)	2 (0)	9 (1)
Symmetric/Hash-based	3 (0)		3 (0)
Other	2 (0)	5 (0)	7 (0)
Total	19 (3)	45 (4)	64 (7)

Note (July 2022)

4 schemes for standardization:

- *Dilithium (Signature, lattice)*
- *Falcon (Signature, lattice)*
- *PHINICS+ (Signature, hash)*
- *Kyber (KEM, lattice)*

4th Round Candidates (KEM):

- *BIKE (code)*
- *Classic McEliece (code)*
- *HQC (code)*
- *SIKE (synergy)*

---

https://csrc.nist.gov/CSRC/media/Presentations/Let-s-Get-Ready-to-Rumble-The-NIST-PQC-Competiti/images-media/PQCrypto-April2018_Moody.pdf

Traditional Cryptography

Post-quantum cryptography is not ready, we need time to:

- improve the efficiency of post-quantum cryptography $\rightarrow$ fast.
- build confidence in post-quantum cryptography $\rightarrow$ attack the schemes.
- improve the usability of post-quantum cryptography $\rightarrow$ scenarios.

Effciency: current situation

Symmetric Key Size (bits)	RSA and Diffie-Hellman Key Size (bits)	Elliptic Curve Key Size (bits)
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	521

Table 1: NIST Recommended Key Sizes

Memory Issue and Computational Cost (Signatures)

Efficiency

Traditional Cryptography: **Memory**			
Scheme	Sec. Level [b]	Total key size [B]	Signature [B]
RSA signature	128	384	384
ECDSA	128	$\approx$ **33**	$\approx$ **131**

NIST Signatures: **Memory**					
Scheme	Type	Sec. Level [b]	Secret Key [B]	Public Key [B]	Signature [B]
Dilithium	lattice	125	-	1 472	2 701
Falcon	lattice	$\gg$ 128	-	1 441	993.91
GeMSS	multivariate	128	14 208	417 408	48
LUOV	multivariate	128	32	7 300	1 700
MQDSS	multivariate	128	32	62	32 882
Picnic	symmetric/hash	128	32	64	195 458
qTESLA	lattice	$\gg$ 128	12 320	39 712	6 176
Rainbow	multivariate	$\gg$ 128	511 400	206 700	156
SPHINCS+	hash	128	64	32	16 976

Computational cost measurements on ARM Cortex-A53					
Scheme	Type	Sec.	Key Pair Generation	Signing	Verification
Dilithium	lattice	125	0.1	0.5	0.1
Falcon	lattice	$\gg$ 128	34.8	3.2	0.3
MQDSS	multivariate	128	1.2	98.4	72.9
Picnic	symmetric/hash	128	0.1	61.7	41.9
qTESLA	lattice	$\gg$ 128	1.1	0.8	0.2
SPHINCS+	hash	128	3.5	110.0	4.7

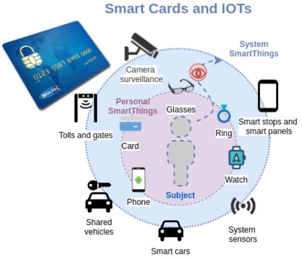
Malina, L., Ricci, S., Dzurenda, P., Smekal, D., Hajny, J., Gerlich, T.: *Towards Practical Deployment of Post-quantum Cryptography on Constrained Platforms and Hardware-Accelerated Platforms.*

Usability

Post-quantum cryptography is not ready, we need time to:

- improve the efficiency of post-quantum cryptography → fast.
- build confidence in post-quantum cryptography → attack the schemes.
- improve the usability of post-quantum cryptography → scenarios.

Usability



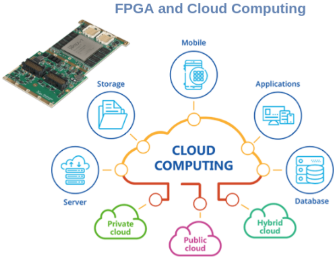


Table of contents

MLWE and MLWR problems

MLWE and MLWR problems

	Base	Ring	Module
Random error	Learning With Errors (LWE)	Ring-Learning With Errors (RLWE)	Module Learning With Errors (MLWE)
Rounding	Learning With Rounding (LWR)	Ring-Learning With Rounding (RLWR)	Module Learning With Rounding (MLWR)

$(a_0, a_1, \ldots, a_{n-1}) \in \mathbb{Z}_q^n$ $\Longleftrightarrow$ $a_0 + a_1 x + \cdots + a_{n-1}x^{n-1} \in \mathbb{Z}_q[x]/\langle x^n + 1\rangle$

operations modulus ***q***

$15 \bmod 7 = 2(7) + 1 \equiv 1 \bmod 7$

each number in $\mathbb{Z}$ can be represented as a reminder/class $[0]_7, [1]_7, \ldots, [6]_7$

LWE

Given (**A**, ***As+e***)

$$\underbrace{\begin{pmatrix} 14 & 15 & 5 & 2 \\ 13 & 14 & 14 & 6 \\ 6 & 10 & 13 & 1 \\ & \vdots & & \\ 6 & 7 & 16 & 2 \end{pmatrix}}_{\mathbf{A}} \underbrace{\begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \end{pmatrix}}_{\mathbf{s}} \approx \underbrace{\begin{pmatrix} 8 \\ 16 \\ 3 \\ \vdots \\ 3 \end{pmatrix}}_{\mathbf{As}+\mathbf{e}}$$

our goal is to recover ***s***

operations modulus ***x^n+1*** in $\mathbb{Z}_q$

$x^3 + 3x \bmod x^2 + 1 = x(x^2 + 1) + 2x \equiv 2x \bmod x^2 + 1$

each polynomial in $\mathbb{Z}_q[x]$ has a unique representation $a_0 + a_1 x + a_2 x^2 \in \mathbb{Z}_7[x]/\langle x^2 + 1\rangle$

R-LWE

Given (**A**, ***As+e***)

$$\underbrace{\begin{matrix} 2 - x + 2x^2 - x^3 \\ -1 - x - x^2 \\ -x + x^2 - 3x^3 \end{matrix}}_{\mathbf{A}} \quad \underbrace{1 - x + x^3}_{\mathbf{s}} \approx \underbrace{\begin{matrix} 3 - 4x + 2x^2 - 3x^3 \\ 1 + x - x^2 \\ \cdots \end{matrix}}_{\mathbf{As}+\mathbf{e}}$$

our goal is to recover ***s***

MLWE and Number Theoretic Transform (NTT)

Why MLWE?

It is always a matter of speed. MLWE has less complicated algebraic structure than RLWE and it is faster than LWE.

Note

RLWE is restricted to ideal lattice. It is not known if this restriction makes the problem easier to be solved.

Why do we need NTT?

We need a way to multiply polynomials with polynomials in fast way. To do so, we need to represent polynomials as "numbers".

Module LWE problem

Definition

The decisional Module-LWE problem asks to recover a secret vector s, given a matrix A and the vector b given by

$$b = \begin{pmatrix} a_{11}(x) & a_{12}(x) & a_{13}(x) \\ a_{21}(x) & a_{22}(x) & a_{23}(x) \\ a_{31}(x) & a_{32}(x) & a_{33}(x) \end{pmatrix} \begin{pmatrix} s_{11}(x) \\ s_{21}(x) \\ s_{31}(x) \end{pmatrix} + \begin{pmatrix} e_{11}(x) \\ e_{21}(x) \\ e_{31}(x) \end{pmatrix}$$

$$b = A \quad s + e$$

Note

The elements of A, s and e are polynomials.

Note

In this case, the matrix A has rank 3.

LWE vs LWR problems

Note (In LWE)

In LWE problem, the error values e is generated by a probability distribution.

Note (In LWR)

The error values e can also be generated by scaling and rounding.

Let q be the polynomial modulus and $p < q$. Then, the vector b is given by

$$b = \lfloor \frac{p}{q} As \rceil$$

Note

In this way, the error is deterministic.

Note

If p and q are powers of two, scaling and rounding operations are very efficient.

Table of contents

Kyber and Saber schemes

A Generic LWE Key Exchange

Alice		**Bob**
generate s		generate s'
$b = As + e$		$b' = A^T s' + e'$
	$\xrightarrow{\quad b \quad}$	
	$\xleftarrow{\quad b' \quad}$	
$v = s^T b'$		$v' = s'^T b$

$$v \approx v'^T$$

Why $v \approx v'^T$?

Alice: $v = s^T(A^T s' + e') = \boxed{s^T A^T s'} + s^T e'$
Bob: $v'^T = (s'^T b)^T = b^T s' = (As + e)^T s' = \boxed{s^T A^T s'} + e^T s'$.

Cryptographic Primitives

Key Exchanges

- Both parties generate a *pk*
- Both parties use each other's *pk* to obtain a shared secret

$$\Downarrow$$

Public-key Encryption

- Alice generates a public key *pk*
- Bob uses Alice's *pk* to encrypt a message
- Only Alice can decrypt it

$$\Downarrow$$

Key-Encapsulation Methods (KEM)

- Alice generates a public key *pk*
- Bob uses Alice's *pk* to encapsulate a random key
- Only Alice can decapsulate it

CRYSTALS-Kyber KEM

CRYSTALS-Kyber KEM:

- is part of the Cryptographic Suite for Algebraic Lattices (CRYSTALS) with Dilithium signature,
- published by Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, Damien Stehlé in 2017,
- German-Belgian-Dutch-American-Swiss-French collaboration,
- lattice-based KEM,
- quantum-resistant and only PQC NIST KEM for standardization,
- based on Module-LWE problem.

The Kyber scheme (sketch)

All polynomials belong to $\mathbb{Z}_q[x]/(x^{256}+1)$ with $q = 3329$

Key Generation

$seed_A \leftarrow random()$
$A = gen(seed_A)$
$s \leftarrow small\_vec\_sec()$
$e \leftarrow small\_vec\_err()$
$b = A^T s + e$

$\xrightarrow{seed_A,\, b}$

Encryption

$A = gen(seed_A)$
$s' \leftarrow small\_vec\_sec()$
$e' \leftarrow small\_vec\_err()$
$b' = As' + e'$
$c = b^T s' + \frac{q}{2} m$

$\xleftarrow{b',\, c}$

Decryption

$v = b'^T s$
$m = \frac{2}{q}(c - v)$

Note

Different security levels are achieved by changing the rank of A (the number of polynomials) and the random distributions.

The Kyber scheme (sketch)

All polynomials belong to $\mathbb{Z}_q[x]/(x^{256}+1)$ with $q = 3329$

Key Generation

$seed_A \leftarrow random()$
$A = gen(seed_A)$
$s \leftarrow small\_vec\_sec()$
$e \leftarrow small\_vec\_err()$
$b = A^T s + e$

$\xrightarrow{seed_A,\, b}$

Encryption

$A = gen(seed_A)$
$s' \leftarrow small\_vec\_sec()$
$e' \leftarrow small\_vec\_err()$
$b' = As' + e'$
$c = b^T s' + \frac{q}{2} m$

$\xleftarrow{b',\, c}$

Decryption

$v = b'^T s$
$m = \frac{2}{q}(c - v)$

Failure probability

Decryption may fail when the errors are too large, but the failure probability is very small ($< 2^{100}$)

Saber KEM

Saber KEM:

- published by Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, Frederik Vercauteren in 2017,
- Belgian work,
- lattice-based KEM,
- quantum-resistant and 3rd round PQC NIST KEM finalist,
- based on Module-LWR problem.

The Saber scheme (sketch)

All polynomials belong to $\mathbb{Z}_q[x]/(x^{256}+1)$ with $q = 2^{13}$

Key Generation

$seed_A \leftarrow random()$
$A = gen(seed_A)$
$s \leftarrow small\_vec\_sec()$
$b = \lfloor \frac{p}{q} A^T s \rceil$

$\xrightarrow{seed_A,\ b}$

Encryption

$A = gen(seed_A)$
$s' \leftarrow small\_vec()$
$b' = \lfloor \frac{p}{q} A s' \rceil$
$c = \lfloor \frac{T}{p} b^T s' + \frac{T}{2} m \rceil$

$\xleftarrow{b',\ c}$

Decryption

$v = b'^T s$
$m = \lfloor \frac{2}{q}(v - \frac{p}{T} c) \rceil$

Comparison between Kyber and Saber

Kyber and Saber are both modern, fast and secure protocols.

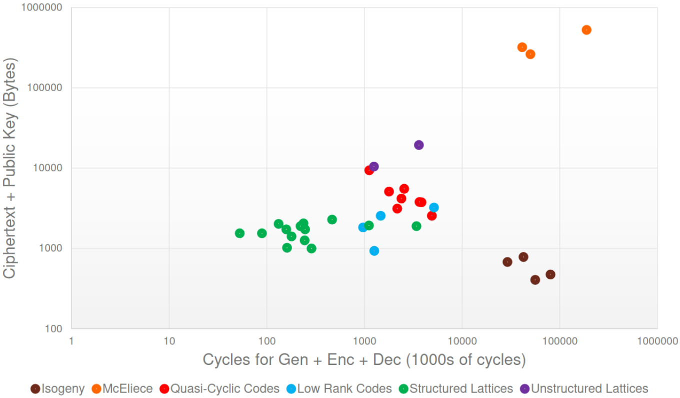
Kyber

- prime modulus
- NTT alg. is required
- Fastest performance (esp. in SW)
- LWE is better studied
- Several implementation studies

Saber

- power-of-two modulus
- Flexible multiplication alg.
- Faster in HW
- Slightly smaller
- Better side-channel protected performance

Both protocols are fast ... and small(ish)



Dustin Moody *The 2nd Round of the NIST PQC Standardization Process-Opening Remarks at PQC 2019*

Table of contents

CRYSTALS-Dilithium Signature

CRYSTALS-Dilithium Signature

CRYSTALS-Dilithium signature:

- is part of the Cryptographic Suite for Algebraic Lattices (CRYSTALS) with Kyber encryption scheme,
- published by Leo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé in 2017,
- Dutch-German-American-Swiss-French collaboration,
- lattice-based signature,
- quantum-resistant and PQC NIST finalist,
- based on Module-LWE problem.

CRYSTALS-Dilithium Signature

Parameters:

- $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^{256} + 1)$ with $q = 8380417 = 2^{23} - 2^{13} + 1$
- Dilithium specifies four sets of parameters
  matrix $\mathbf{A}$ dimensions $(k, l) = (3, 2), (4, 3), (5, 4)$ and $(6, 5)$.
- The functions $\texttt{HighBits}_q$ and $\texttt{LowBits}_q$ allow reducing the public key by a factor of around 2.5.

CRYSTALS-Dilithium (sketch)

Key Generation

$\mathbf{A} \in \mathcal{R}_q^{k \times l}$ ← **matrix of polynomials**

$(\mathbf{s_1}, \mathbf{s_2}) \in \mathcal{S}_\eta^l \times \mathcal{S}_\eta^k$ ← **polynomials**

$\mathbf{t} = \mathbf{A}\mathbf{s_1} + \mathbf{s_2}$

return **public key:** $pk = (\mathbf{A}, t)$

private key: $sk = (s_1, s_2)$

Module-Learning With Error Problem

$14s_1 + 15s_2 + 5s_3 + 2s_4 \approx 8 \bmod 17$
$13s_1 + 14s_2 + 14s_3 + 6s_4 \approx 16 \bmod 17$
$6s_1 + 10s_2 + 13s_3 + 1s_4 \approx 3 \bmod 17$
$10s_1 + 4s_2 + 12s_3 + 16s_4 \approx 12 \bmod 17$
$\vdots$
$6s_1 + 7s_2 + 16s_3 + 2s_4 \approx 3 \bmod 17$

In this case: $\mathbf{s} \in \mathbb{Z}_{17}^4$

and the equation is correct up to some small additive error (say, **±1**)

our goal is to recover **s**

$$\underbrace{\begin{pmatrix} 14 & 15 & 5 & 2 \\ 13 & 14 & 14 & 6 \\ 6 & 10 & 13 & 1 \\ & \vdots & & \\ 6 & 7 & 16 & 2 \end{pmatrix}}_{\mathbf{A}} \underbrace{\begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \end{pmatrix}}_{\mathbf{s}} = \underbrace{\begin{pmatrix} 8 \\ 16 \\ 3 \\ \vdots \\ 3 \end{pmatrix}}_{\mathbf{As} + \mathbf{e}}$$

Simplified!

Signing Algorithm (sk, M)

$\mathbf{z} = \perp$

while $\mathbf{z} = \perp$ **do**

$\mathbf{y} \in \mathcal{S}_{\gamma_1 - 1}^l$

$\mathbf{w_1} = \mathrm{HighBits}(\mathbf{A}\mathbf{y}, 2\gamma_2)$

$c \in B_{60} = \mathcal{H}(M || \mathbf{w_1})$

$\mathbf{z} = \mathbf{y} + c\mathbf{s_1}$

if $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ **or** $\|\mathrm{LowBits}(\mathbf{A}\mathbf{y} - c\mathbf{s_2},$

return $\sigma = (\mathbf{z}, c)$

Verifying Algorithm (pk, M, σ)

$\mathbf{w}_1' = \mathrm{HighBits}(\mathbf{A}\mathbf{z} - c\mathbf{t}, 2\gamma_2)$

accept if $\|\mathbf{z}\|_\infty \leq \gamma_1 - \beta$ **and** $c = \mathcal{H}(M || \mathbf{w}_1')$

Ducas L. et al. Crystals-dilithium: A lattice-based digital signature scheme.

CRYSTALS-Dilithium (sketch)

Alice

Bob

Key Generation

$\mathbf{A} \in \mathcal{R}_q^{k \times l}$ ← matrix of polynomials

$(\mathbf{s_1}, \mathbf{s_2}) \in \mathcal{S}_\eta^l \times \mathcal{S}_\eta^k$ ← polynomials

$\mathbf{t} = \mathbf{A}\mathbf{s_1} + \mathbf{s_2}$

return public key: $pk = (\mathbf{A}, t)$

private key: $sk = (s_1, s_2)$

Signing Algorithm (sk, M)

$\mathbf{z} = \perp$

Fiat-Shamir with Aborts

while $\mathbf{z} = \perp$ **do**

$\mathbf{y} \in \mathcal{S}_{\gamma_1 - 1}^l$

$\mathbf{w_1} = \mathrm{HighBits}(\mathbf{A}\mathbf{y}, 2\gamma_2)$

$c \in B_{60} = \mathcal{H}(M || \mathbf{w_1})$

$\mathbf{z} = \mathbf{y} + c\mathbf{s_1}$

if $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ **or** $\|\mathrm{LowBits}(\mathbf{A}\mathbf{y} - c\mathbf{s_2}, 2\gamma_2)\|_\infty \geq \gamma_2 - \beta$ **then** $\mathbf{z} = \perp$

If **z** is small enough, we have a valid signature

return $\sigma = (\mathbf{z}, c)$

Verifying Algorithm (pk, M, σ)

$\mathbf{A}\mathbf{z} - c\mathbf{t} = \mathbf{A}\mathbf{y} + c\mathbf{A}\mathbf{s_1} - c\mathbf{A}\mathbf{s_1} - c\mathbf{s_2} \approx \mathbf{A}\mathbf{y}$ →

$\mathbf{w}'_1 = \mathrm{HighBits}(\mathbf{A}\mathbf{z} - c\mathbf{t}, 2\gamma_2)$

accept if $\|\mathbf{z}\|_\infty \leq \gamma_1 - \beta$ **and** $c = \mathcal{H}(M || \mathbf{w}'_1)$

Ducas L. et al. Crystals-dilithium: A lattice-based digital signature scheme.

Summary

Hash, MLWE and MLRW problems:

- Post-quantum hash functions based on complexity provblems are still under development.
- MLWE has less complicated algebraic structure than RLWE and it is faster than LWE.
- the error values can also be generated by scaling and rounding, i.e. LWR problem

Kyber, Saber and Dilithium:

- Kyber, Saber, and Dilithium are fast and secure protocols that can replace classical protocols in most applications.
- Kyber KEM and Dilithium signature are based on MLWE problem with prime modulus. They need NTT
- Saber KEM is based on MLWR problem with power-of-two modulus. It has deterministic error.

References

Articles:

- Pollard, J.M.: *The fast Fourier transform in a finite field*, 1971.
- Bos J, et al.: *CRYSTALS-Kyber: a CCA-secure module-lattice-based KEM.* 2018. Vercauteren IF. *SABER: Mod-LWR Based KEM (Round 2 Submission)*, 2018.
- Ducas L. et al.: *CRYSTALS-Dilithium: A lattice-based digital signature scheme*, 2018.
- Albrecht, M. R. and Deo, A.: *Large Modulus Ring-LWE $\geq$ Module-LWE*, 2017.
- Malina, L., Ricci, S., Dzurenda, P., Smekal, D., Hajny, J., Gerlich, T.: *Towards Practical Deployment of Post-quantum Cryptography on Constrained Platforms and Hardware-Accelerated Platforms*, 2020.
- Ricci, S., Malina, L., Jedlicka, P., Smekal, D., Hajny, J., Cibik, P., Dzurenda, P., Dobias, P.: *Implementing CRYSTALS-Dilithium Signature Scheme on FPGAs*, 2021.
- Ricci, S., Jedlicka, P., Cibik, P., Dzurenda, P., Malina, L., Hajny, J.: *Towards CRYSTALS-Kyber VHDL Implementation*, 2021.

Thank you for attention!

ricci@vut.cz
https://axe.utko.feec.vutbr.cz/

SCAN ME

Evaluation of a Polynomial

Polynomial

A polynomial b of degree $n-1$ can be written as

$$b(x) = \sum_{1=0}^{n-1} b_i x^i$$

Example

We are in $\mathbb{Z}_5[x]$*. Then* $b(x) = x^2 - x + 3 = 0x^3 + 1x^2 - 1x + 3x^0$*.*

Note

To evaluate a polynomial, we have to assign a value to x*. For example, if* $x = 1, 2$ *or* 4 *then:*

$b(1) = 1^2 - 1 + 3 = 3 \mod 5$
$b(2) = 2^2 - 2 + 3 = 0 \mod 5$
$b(4) = 4^2 - 4 + 3 = 0 \mod 5$

Unique polynomial and NTT^{-1}

Unique polynomial

there exists a unique n-th degree polynomial that passes through $n+1$ points in the plane.

Example

We know that $b(x)$ has degree 2 in $\mathbb{Z}_5[x]$ and that $b(1)=3, b(2)=0$ and $b(4)=0$. Let us find $b(x)$.
It has degree 2, therefore: $b(x) = b_2x^2 + b_1x + b_0 \mod 5$.
We need to find the values of b_2, b_1, b_0.

$$\begin{aligned} 3 &= b_2 + b_1 + b_0 \\ 0 &= 4b_2 + 2b_1x + b_0 \\ 0 &= b_2 + 4b_1 + b_0 \end{aligned}$$

We solve the system and we obtain: $b_2 = 1, b_1 = -1$ and $b_0 = 3$, therefore, $b(x) = x^2 - x + 3$.

Note

NTT^{-1} is equivalent to "reconstruct" a polynomial knowing its evaluation in several points.

NTT requirements

Requirements

NTT can be applied if:

- n divides $q-1$. Note that we are in $\mathbb{Z}_q/(x^n+1)$.
- exists α in $\mathbb{Z}_q$ such that

$$\alpha^n = 1 \mod q,$$

$$\alpha^k \neq 1 \mod q \text{ for each } k < n$$

Example

We consider $\mathbb{Z}_5[x]/(x^4+1)$. Let us see if this ring has the right requirements:

- $n=4$ *and* $q=5$, *therefore* 4 *divides* $4=q-1$.
- $\alpha = 2$ *works:*

$$2^0 = 1,\ \ 2^1 = 2,\ \ 2^2 = 4,\ \ 2^3 = 3,\ \ 2^4 = 1$$

Note

In order to multiply two elements via NTT, those elements have to be transformed into NTT form.

NTT form of a polynomial

Polynomial and its NTT form

A polynomial can be written as

$$b(x) = \sum_{1=0}^{n-1} b_i x^i$$

NTT form of b is

$$NTT(b)_\alpha = (B_0, \ldots, B_{n-1}) \text{ where } B_j = b(\alpha^j) = \sum_{i=0}^{n-1} b_i (\alpha^j)^i \mod q$$

The polynomial is evaluated in α^j.

Example

We are still in $\mathbb{Z}_5[x]/(x^4+1)$ *and* $\alpha = 2$. *We consider*

$$b(x) = x^2 - x + 3 = 0x^3 + 1x^2 - 1x + 3x^0$$

therefore, $NTT(b)_2 = (3, 0, 0, 4)$, *where* $B_0 = b(2^0) = 1^2 - 1 + 3 = 3 \mod 5$

NTT algorithm

Note

NTT is invertible.
If we can pass from a polynomials $b(x)$ to its NTT form $NTT(b)_\alpha = (B_0, \ldots, B_{n-1})$, we can also pass from a NTT form $(B_0, \ldots, B_{n-1})$ to the polynomial $NTT^{-1}(B_0, \ldots, B_{n-1}) = b(x)$.

Note

Note that NTT is the "evaluation of a polynomial" procedure.

Multiplication of two polynomials

Given $a(x)$ and $b(x)$ two polynomial of degree $n-1$. We want to compute $c(x) = a(x)b(x)$ then

$$C_j = A_j B_j$$

and $c(x) = NTT^{-1}(C_0, \ldots, C_{n-1})$