KpqC Winter Camp
Joong_Ang Univ., Seoul, Korea
Feb. 23(Thu.), 2023

SOLMAE

SOLMAE:
quantum-Secure algOrithm for Long-term Message Authentication and Encryption

Prof. Kwangjo Kim
IACR Fellow

President@ IRCS / Emeritus Professor@ KAIST

Contents

1. Introduction
2. PQC has started
3. Overview of SOLMAE
4. Let's learn lattice and its problem
5. What's NTRU?
6. Details of SOLMAE
7. Security Evaluation
8. Concluding Remarks

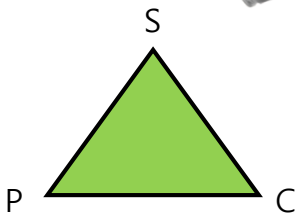
Appendix: Lattice Attack

1. Introduction

SOLMAE

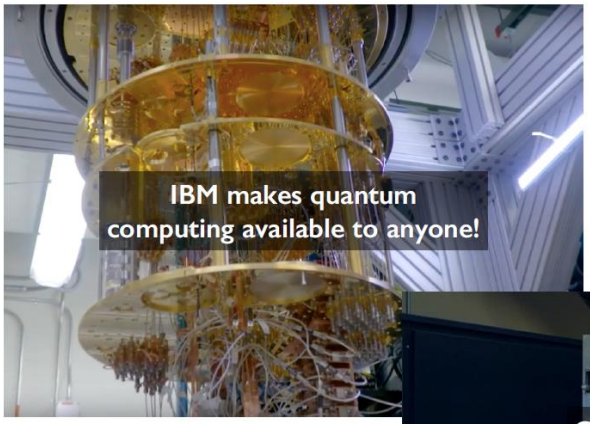
In the beginning

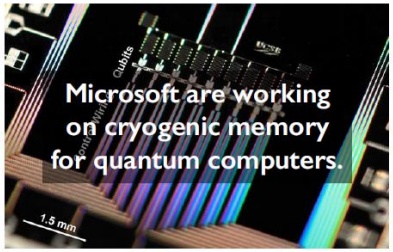
- *"Linearity is a curse to cryptographers* "by A. Shamir
- Lessons learned from the past.
  - Attacking advances much faster than defense.
- One idea can't cover everything → No silver bullet
  - Security/Performance/Cost Tradeoff → Trilemma

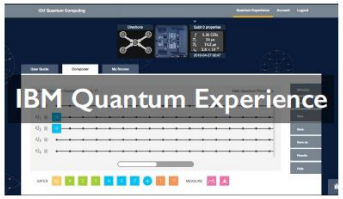


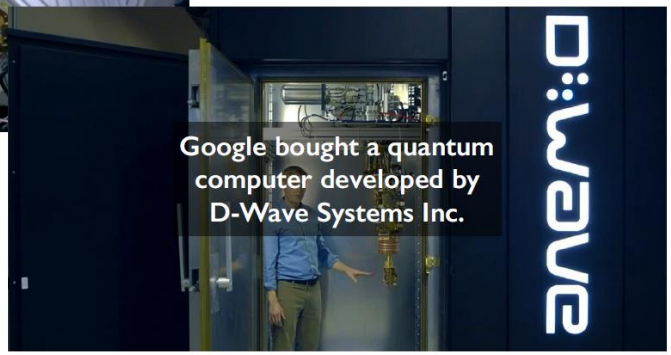
Designing cryptographically-strong primitives such as digital signatures or key encapsulation mechanisms, *etc.* are really a big challenge and could not be accomplished in a short time by one expert. A group of smart designers must understand all the known attacks so far from the theoretical and implementation points of view and anticipate the feasible attacks in the near future. Our team consisting of top-level cryptographers around the world has started to suggest long-term quantum-secure digital signature against quantum attack based on NTRU lattices, well-understood by the cryptographic community since their introduction around two decades ago.

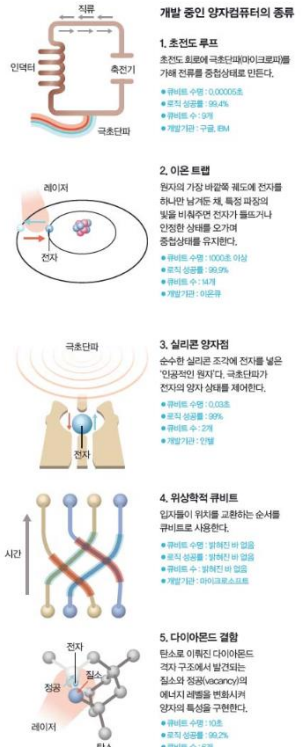
Quantum Computers







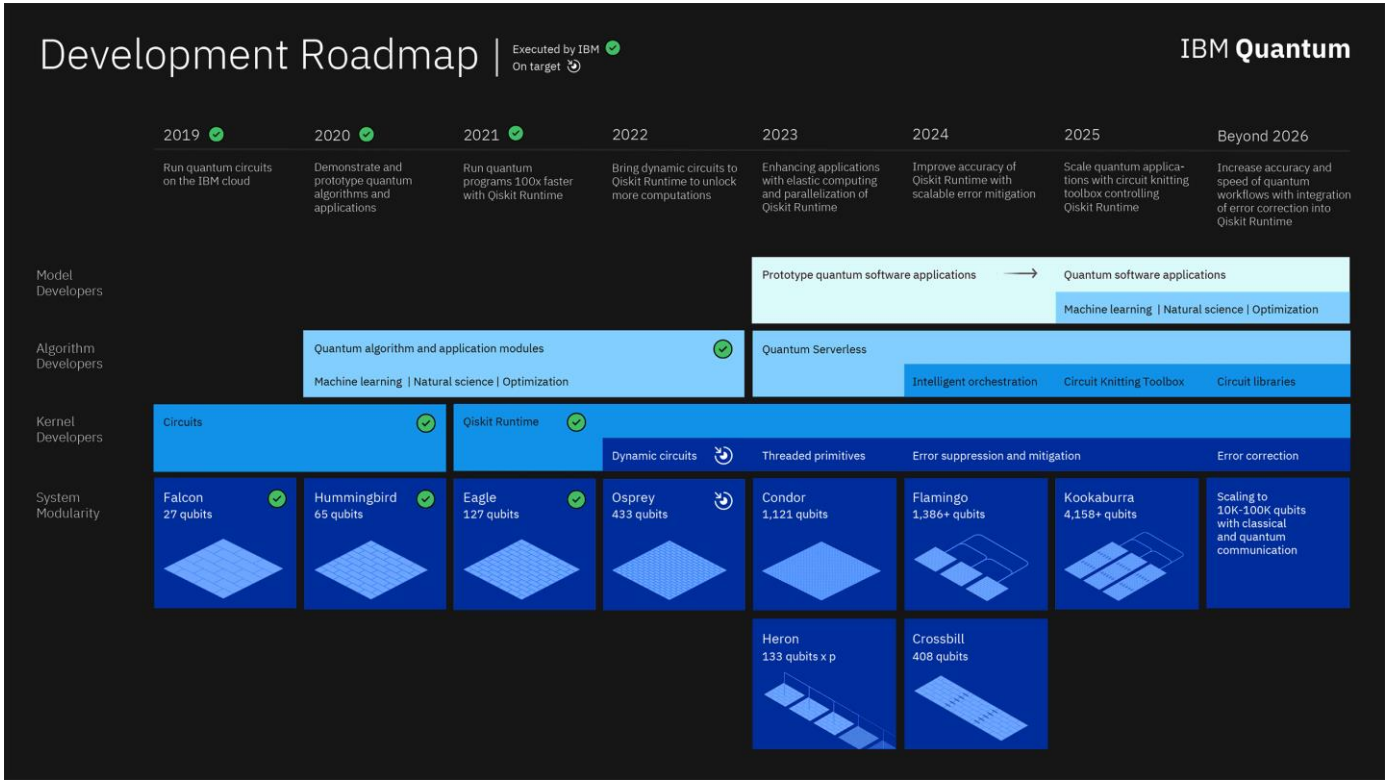




- Quantum mechanics applies to all systems from micro to macro scale and enables superposition and entanglement.
- Reversible computing: if no information is erased, computation may in principle be achieved which is thermodynamically reversible, and require no release of heat [Lan61].

(1) D. Aggarwal et al. "Quantum attacks on Bitcoin, and how to protect against them", arXiv 1710:10377v1. Oct. 28, 2017

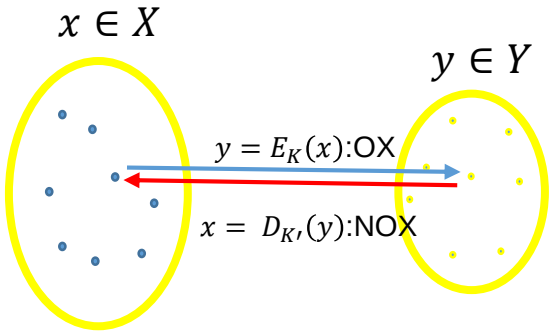
IBM Quantum Computer Roadmap



https://research.ibm.com/blog/ibm-quantum-roadmap-2025

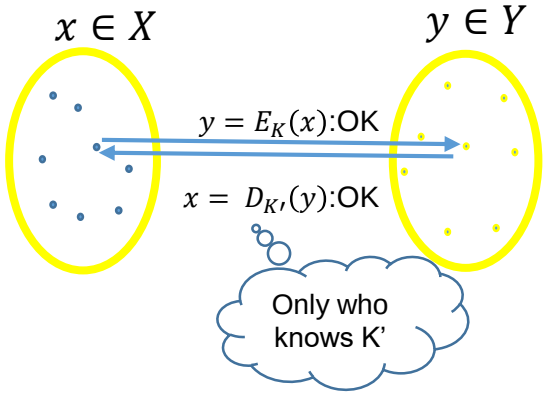
Cryptographic Functions

1-way Function



Ex: Hash Function

Trapdoor 1-way Function



Ex: Public Key Cryptosystem

RSA- PKC/DS

SOLMAE

❖ Key_generation, KeyGen()
- ✓ Select two large (1,024 bits or larger) primes p, q
- ✓ Compute modulus $n = pq$, and $\phi(n) = (p-1)(q-1)$
- ✓ Pick an integer e relatively prime to $\phi(n)$, $gcd(e, \phi(n))=1$
- ✓ Compute d such that $ed = 1 \bmod \phi(n)$
- ✓ Public key (n, e) : public
- ✓ Private key d : keep secret (may hold p and q securely.)

❖ Encryption()/Verification()
- ✓ E: $C = M^e \bmod n$ for $0 < M < n$

❖ Decryption()/Signing()
- ✓ D: $M = C^d \bmod n$

Proof) $C^d = (M^e)^d = M^{ed} = M^{k\phi(n)+1} = M\{M^{\phi(n)}\}^k = M$

❖ Special Property
- ✓ $(M^e \bmod n)^d \bmod n = (M^d \bmod n)^e \bmod n$ for $0 < M < n$

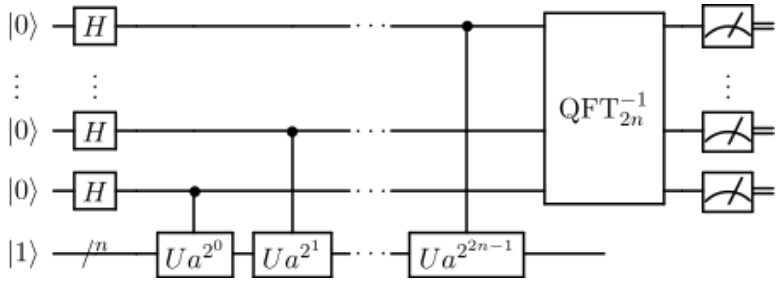
Quantum Attacks

Number-Theoretic KC

- [illegible]
- [illegible]

RSA

Shor Algorithm(1)



- Lattice-based Cryptography
- Code-based Cryptography
- **Post-quantum Cryptography**
- Hash-based Cryptography
- MQE-based Cryptography
- Isogeny-based Cryptography

(1) Peter W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer." SIAM Journal on computing 26.5 (1997): 1484-1509

2. PQC has started

NIST PQC Standardization Project*

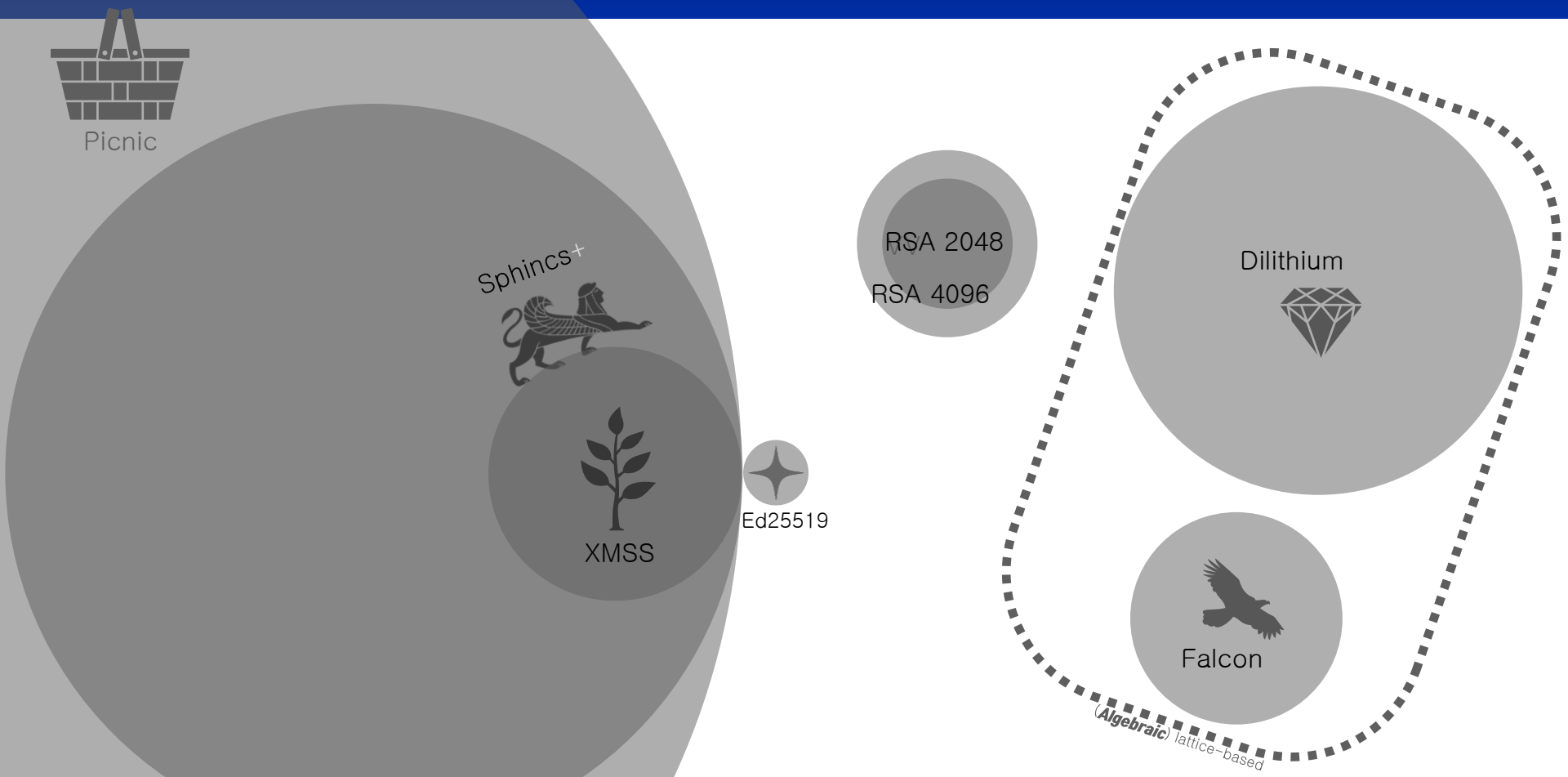
- 2017 : Round 1 (69 submissions)
- 2019 : Round 2 (26 algorithms)
- 2020 : Round 3 (7 finalists, 8 alternates)
- ***2022 : Round 4 (4 finalists, 4 alternatives)***

PKC/KEM			DS	
Algorithm	**Kyber**	**Dilithium**	**Falcon**	**SPHINCS+**
Problem	lattice	lattice	lattice	hash

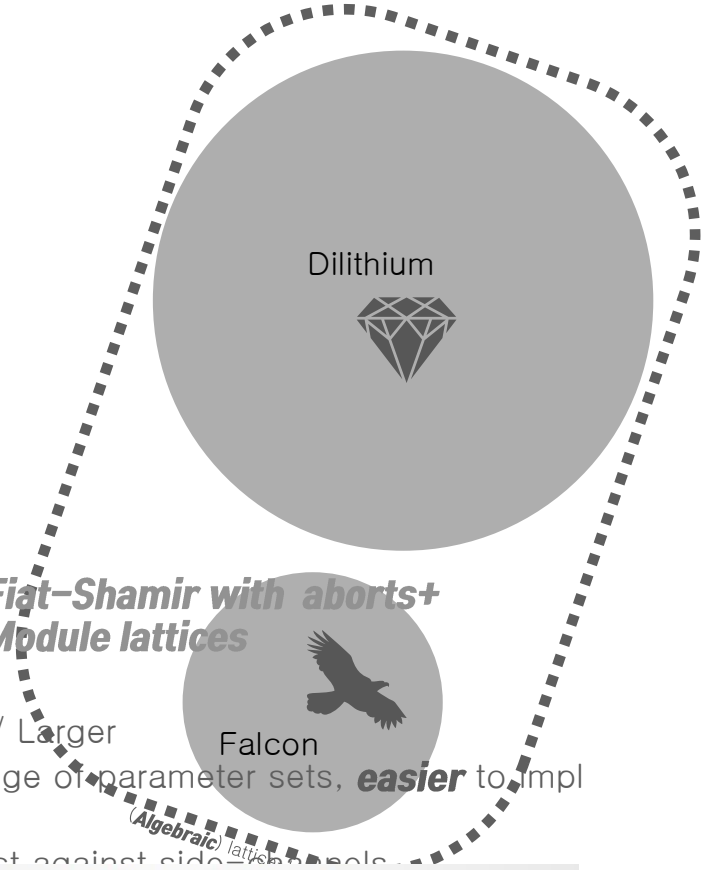
KEM/DS				
Algorithm	**Classic McEliece**	**BIKE**	**SIKE**	**HQC**
Problem	Goppa Code	QC_MDPC	Isogeny	Hamming code

*https://csrc.nist.gov/Projects/post-quantum-cryptography, NISTIR 8413

Panorama of digital signature (1/2)



Panorama of digital signature (2/2)



Hash-and-sign+
NTRU trapdoors

✓ Compact // Fast
X ***Restricted*** parameter set, quite ***hard*** to imple ment and protect against side-channels

From NIST

- SELECTED FOR ITS SMALL BANDWIDTH, FAST VERIFICATION AND SECURITY
- THE IMPLEMENTATION MAY BE COMPLICATED FOR SOME APPLICATIONS
- WE ARE PLANNING TO STANDARDIZE THE PARAMETER SETS FOR FALCON CORRESPONDING TO SECURITY CATEGORIES 1 AND 5
- THE STANDARD WILL COME AFTER THE DILITHIUM STANDARD

Fiat-Shamir with aborts+
Module lattices

✓ Slower // Larger
X ***Large*** range of parameter sets, ***easier*** to impl ement and protect against side-channels

From NIST

SELECTED BASED ON ITS SECURITY, HIGH EFFICIENCY, AND RELATIVELY SIMPLE IMPLEMENTATION

WE RECOMMEND IT BE THE PRIMARY SIGNATURE ALGORITHM USED

WE ARE PLANNING TO STANDARDIZE THE PARAMETER SETS FOR DILITHIUM CORRESPONDING TO SECURITY CATEGORIES 2, 3, AND 5

3. Overview of SOLMAE

Motivation

- New crypto algorithms based on new problem(e.g, braid group, etc.) are not known to provide sufficient security.
- Post-quantum PKC problems (such as NTRU lattices) are mature enough to be trustworthy.
- Diversity is of the utmost importance for standardization.
- FALCON*1 (FAst-Fourier Lattice-based COmpact signatures over NTRU) is one of 4th round 3 digital signatures for NIST PQC Standard (Jul. 5,2022) and future its standard.
- MITAKA team proposed MITAKA*2 which is simpler, parallelizable, maskable variant of FALCON in Eurocrypt2022.
- Collaborated with MITAKA team, Kwangjo wants to make significant contribution in KpqC competition.

1. https://falcon-sign.info/
2. T. Espitau, P.-A. Fouque, F. Gérard, M. Rossi, A. Takahashi, M. Tibouchi, A. Wallet, and Y. Yu. *Mitaka: A simpler, parallelizable, maskable variant of falcon*. In EUROCRYPT 2022, Part III, vol. 13277 of LNCS, pp. 222–253. Springer, Heidelberg, 2022

FALCON Algorithm

SOLMAE

FALCON

Fast-Fourier Lattice-based Compact Signatures over NTRU

About Falcon

Falcon is a cryptographic signature algorithm submitted to NIST Post-Quantum Cryptography Project on November 30th, 2017. It has been designed by: Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, Zhenfei Zhang.

The point of a post-quantum cryptographic algorithm is to keep on ensuring its security characteristics even faced with quantum computers. Quantum computers are deemed feasible, according to our current understanding of the laws of physics, but some significant technological issues remain to be solved in order to build a fully operational unit. Such a quantum computer would very efficiently break the usual asymmetric encryption and digitial signature algorithms based on number theory (RSA, DSA, Diffie-Hellman, ElGamal, and their elliptic curve variants).

Falcon is based on the theoretical framework of Gentry, Peikert and Vaikuntanathan for lattice-based signature schemes. We instantiate that framework over NTRU lattices, with a trapdoor sampler called "fast Fourier sampling". The underlying hard problem is the short integer solution problem (SIS) over NTRU lattices, for which no efficient solving algorithm is currently known in the general case, even with the help of quantum computers.

Algorithm Highlights

Falcon offers the following features:

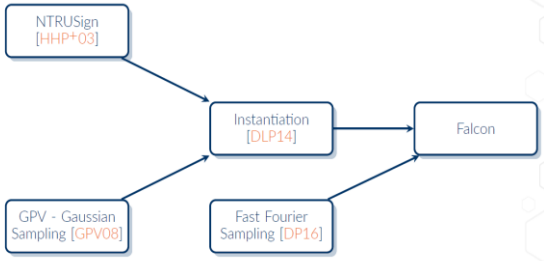
- **Security:** a true Gaussian sampler is used internally, which guarantees negligible leakage of information on the secret key up to a practically infinite number of signatures (more than 2^{64}).
- **Compactness:** thanks to the use of NTRU lattices, signatures are substantially shorter than in any lattice-based signature scheme with the same security guarantees, while the public keys are around the same size.
- **Speed:** use of fast Fourier sampling allows for very fast implementations, in the thousands of signatures per second on a common computer; verification is five to ten times faster.
- **Scalability:** operations have cost $O(n \log n)$ for degree n, allowing the use of very long-term security parameters at moderate cost.
- **RAM Economy:** the enhanced key generation algorithm of Falcon uses less than 30 kilobytes of RAM, a hundredfold improvement over previous designs such as NTRUSign. Falcon is compatible with small, memory-constrained embedded devices.

Performance

While resistance to quantum computers is the main drive for the design and development of Falcon, the algorithm may achieve significant adoption only if it is also reasonably efficient in our current world, where quantum computers do not really exist. Using the reference implementation on a common desktop computer (Intel® Core® i5-8259U at 2.3 GHz, TurboBoost disabled), Falcon achieves the following performance:

variant	keygen (ms)	keygen (RAM)	sign/s	verify/s	pub size	sig size
Falcon-512	8.64	14336	5948.1	27933.0	897	666
Falcon-1024	27.45	28672	2913.0	13650.0	1793	1280

Sizes (key generation RAM usage, public key size, signature size) are expressed in bytes. Key generation time



Keygen(1^λ)

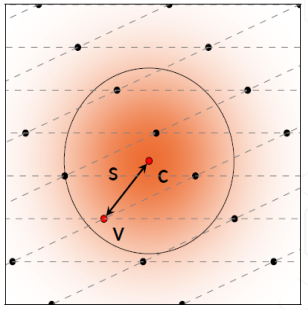
1. Gen. matrices $\mathbf{A}, \mathbf{B}$ s.t.:
   - $\mathbf{B} \cdot \mathbf{A} = 0$
   - $\mathbf{B}$ has small coefficients
2. $\mathsf{pk} := \mathbf{A}, \mathsf{sk} := \mathbf{B}$

Sign($\mathtt{M}, \mathsf{sk} = \mathbf{B}$)

1. Compute $\mathbf{c}$ such that $\mathbf{c} \cdot \mathbf{A} = H(\mathtt{M})$
2. $\mathbf{v} \leftarrow$ vector in $\mathcal{L}(\mathbf{B})$, close to $\mathbf{c}$
3. $\mathtt{sig} := \mathbf{s} = (\mathbf{c} - \mathbf{v})$

Verify($\mathtt{M}, \mathsf{pk} = \mathbf{A}, \mathtt{sig} = \mathbf{s}$)

Check ($\mathbf{s}$ short) & ($\mathbf{s} \cdot \mathbf{A} = H(\mathtt{M})$)



Advantages:

- The most bandwidth-efficient finalist
- Verification (in particular) is fast and RAM efficient
- Extensive research on the security of lattices (and NTRU)
- Side-channel resistance is now better understood [Por19, HPRR20, FKT+20]

What can be improved:

- Key generation and signing remain complex
- Key generation and signing rely on floating-point arithmetic
- More work on side-channel resistance is always welcome

*https://falcon-sign.info/

IRCS 국제사이버보안연구원

KAIST

GPV Framework[GPV'08]

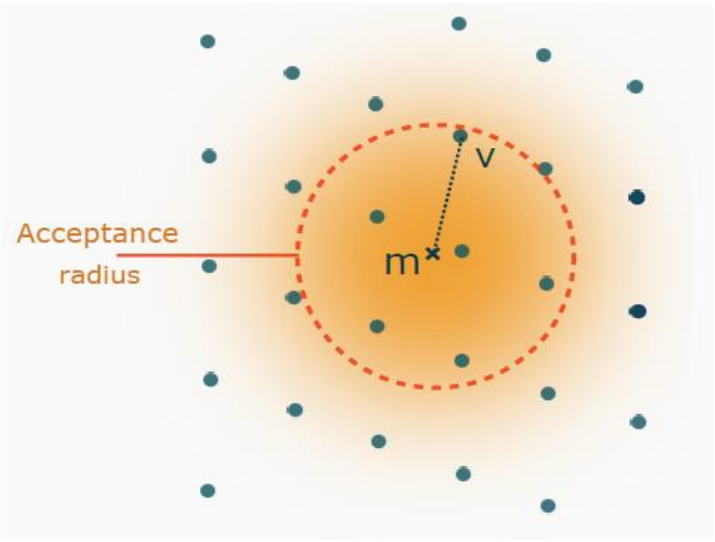
SOLMAE

Simplified $\mathrm{Sign}_{\mathbf{sk},\sigma}(\mathrm{msg})$:

1. $\mathbf{m} = \mathcal{H}(\mathrm{msg})$
2. $\mathbf{v} \leftarrow \mathrm{GaussianSampler}(\mathbf{sk}, \mathbf{m}, \sigma)$
3. Signature: $\mathbf{s} = \mathbf{m} - \mathbf{v}$.

Simplified $\mathrm{Verif}_{\mathcal{L}=\mathbf{pk}}(\mathrm{msg}, \mathbf{s})$:

1. If $\|\mathbf{s}\|$ too big, reject.
2. If $\mathbf{m} - \mathbf{s} \notin \mathcal{L}$, reject.
3. Accept.



Requirements

$\mathbf{CVP}_\gamma$ hard $\Rightarrow$ σ small $\Rightarrow$ **sk** has short vectors

Hard to compute	Easy to generate
sk just from **pk**	**pk** just from **sk**

sk is called *"a trapdoor"*

GPV08. C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *40th ACM STOC*, pp. 197–206. ACM Press, 2008. 3, 5, 6, 13

MITAKA Algorithm

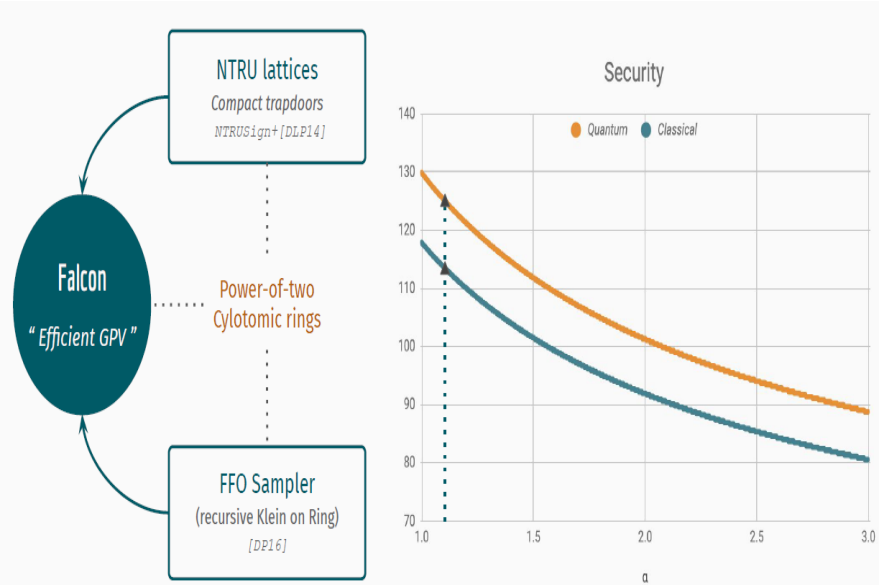
SOLMAE

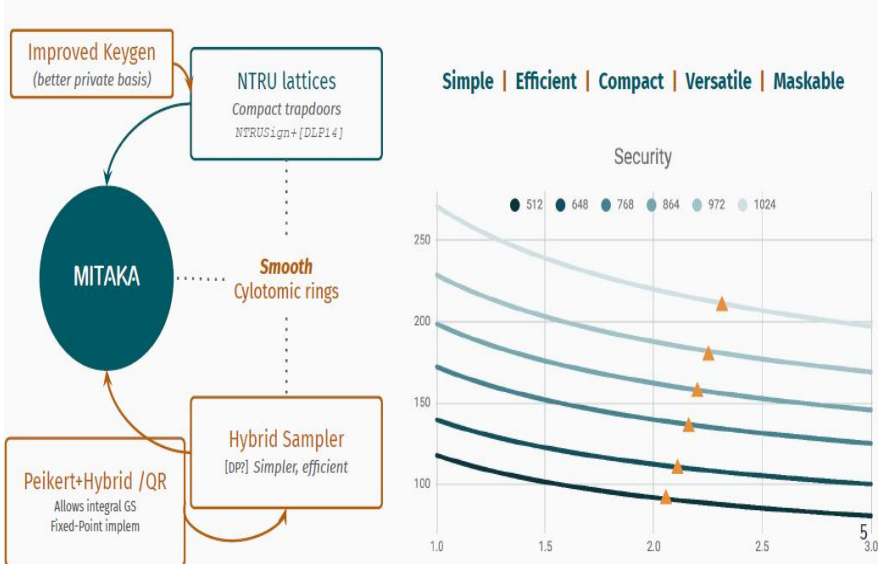
Mitaka

A Simpler, Parallelizable, Maskable Variant of Falcon

Thomas Espitau, Pierre-Alain Fouque, Francois Gérard, Melissa Rossi, Akira Takahashi, Mehdi Tibouchi, Alexandre Wallet, Yang Yu

Eurocrypt 2022





1. Thomas Espitau, Pierre-Alain Fouque, François Gérard, Mélissa Rossi, Akira Takahashi, Mehdi Tibouchi, Alexandre Wallet, Yang Yu **"***Mitaka: A Simpler, Parallelizable, Maskable Variant of Falcon***",**NTT Corporation; Rennes Univ, Inria and IRISA; University of Luxembourg; ANSSI; Aarhus University; Tsinghua University, Proc. of Eurocrypt2022

SOLMAE in a nutshell

SOLMAE

SOLMAE*
Algorithm Specifications

Kwangjo Kim[1], Mehdi Tibouchi[2], Alexandre Wallet[4], Thomas Espitau[2], Akira Takahashi[3], Yang Yu[5], Sylvain Guilley[6], and Seungki Kim[7]

[1] International Research Institute for Cyber Security(IRCS)/KAIST
kkj@kaist.ac.kr
[2] NTT Social Informatics Laboratories, Japan
mehdi.tibouchi.br,thomas.espitau.ax}@hco.ntt.co.jp
[3] Universuty of Edinborgh, Scotland
takahashi.akira.58s@gmail.com
[4] Inria, France
alexandre.wallet@inria.fr
[5] Tsinghua University, China
yang.yu0986@gmail.com
[6] Secure-IC, France
sylvain.guilley@secure-ic.com
[7] University of Cincinnati, USA
seungki.math@gmail.com

Abstract. This document specifies the SOLMAE signature scheme submitted to the Korean Post-Quantum Competition. SOLMAE is a lattice-based signature scheme following the hash-and-sign paradigm (in the style of Gentry–Peikert–Vaikuntanathan signatures), and instantiated over NTRU lattices. In that sense, it is closely related to, and a successor of, several earlier schemes including Ducas–Lyubashevsky–Prest (DLP), FALCON and MITAKA. More precisely, SOLMAE offers the "best of both worlds" between FALCON and MITAKA.

FALCON has the advantage of providing short public keys and signatures (offering essentially the best bandwidth trade-off among post-quantum constructions) as well as high security levels; however, it is plagued by a contrived signing algorithm that makes it very difficult to implement correctly, not very fast for signing and hard to parallelize; it also has very little flexibility in terms of parameter settings. In contrast, MITAKA is much simpler to implement, twice as fast in equal dimension, straightforward to parallelize and fully versatile in terms of parameters; however, it has lower security than FALCON in equal dimension, has an even more contrived key generation algorithm that tends to be quite slow, and has somewhat larger keys and signatures at equivalent security levels.

SOLMAE solves the conundrum of choosing between those two schemes by offering all the advantages of both. It uses the same simple, fast, parallelizable signing algorithm as MITAKA, with flexible parameters. However, by leveraging a novel key generation algorithm that is much faster and achieves higher security, SOLMAE achieves the same high security and short key and signature sizes as FALCON. It is also compatible with recently introduced ellipsoidal lattice Gaussian sampling techniques to further reduce signature sizes. This makes SOLMAE the state-of-the-art in terms of constructing efficient lattice-based signatures over structured lattices. Some further challenges are left in the conclusion.

Keywords: Signature schemes · Lattice-based cryptography · Hash-and-sign paradigm · Module lattices · Lattice Gaussian sampling

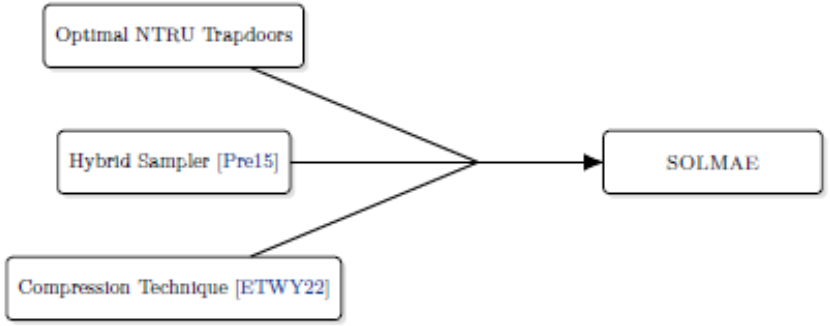


Fig. 1: Overview of SOLMAE

Table 2: Performance comparison between SOLMAE and FALCON.

		SOLMAE–512	SOLMAE–1024	FALCON–512	FALCON–1024
KeyGen time	Mcycles	27	65	—	—
	time (ms)	7.5	18	5.0	15
pk size	Bytes	896	1792	896	1792
Sign time	kcycles	387	775	—	—
	time (μs)	108	216	220	441
sgn size	Bytes	666	1375	666	1280
Verif time	kcycles	40	84	—	—
	time (μs)	11	23	18	36

2X Faster

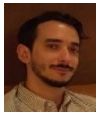
IRCS 국제사이버보안연구원

KAIST

SOLMAE Team

- Korean PI
  - Kwangjo Kim (President@IRCS, Emeritus Prof.@KAIST), https://caislab.kaist.ac.kr/~kkj
- Member (7 persons)
  - Mehdi Tibouchi (NTT, Japan), https://www.normalesup.org/~tibouchi/
  - Thomas Espitau (NTT Japan), https://espitau.github.io/
  - Alexandre Wallet (INRIA Rennes, France), https://awallet.github.io/
  - Yang Yu(Tsinghua University, China), https://yuyang-crypto.github.io/
  - Akira Takahashi(U. of Edinburgh, Scotland), https://akiratk0355.github.io/
  - Sylvain Guilley (Secure-IC, France), https://perso.telecom-paristech.fr/guilley/
  - Seungki Kim(U. of Cincinnati, USA), https://sites.google.com/view/seungki/home



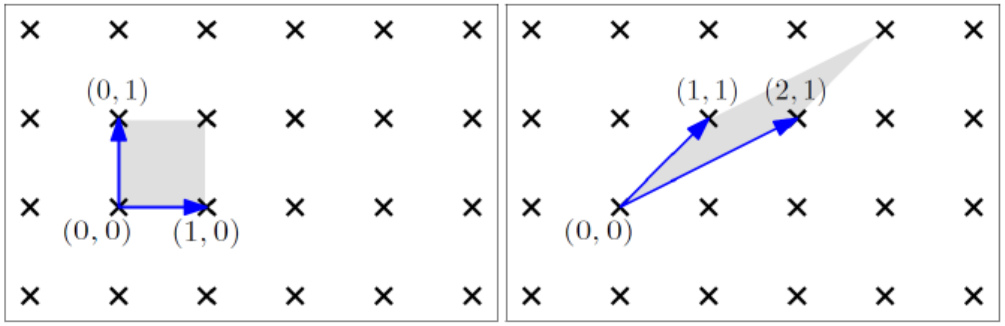





4. Let’s learn lattice and its problem

Lattice

A lattice can have different basis:



(a) A basis of $\mathbb{Z}^2$ (b) Another basis of $\mathbb{Z}^2$

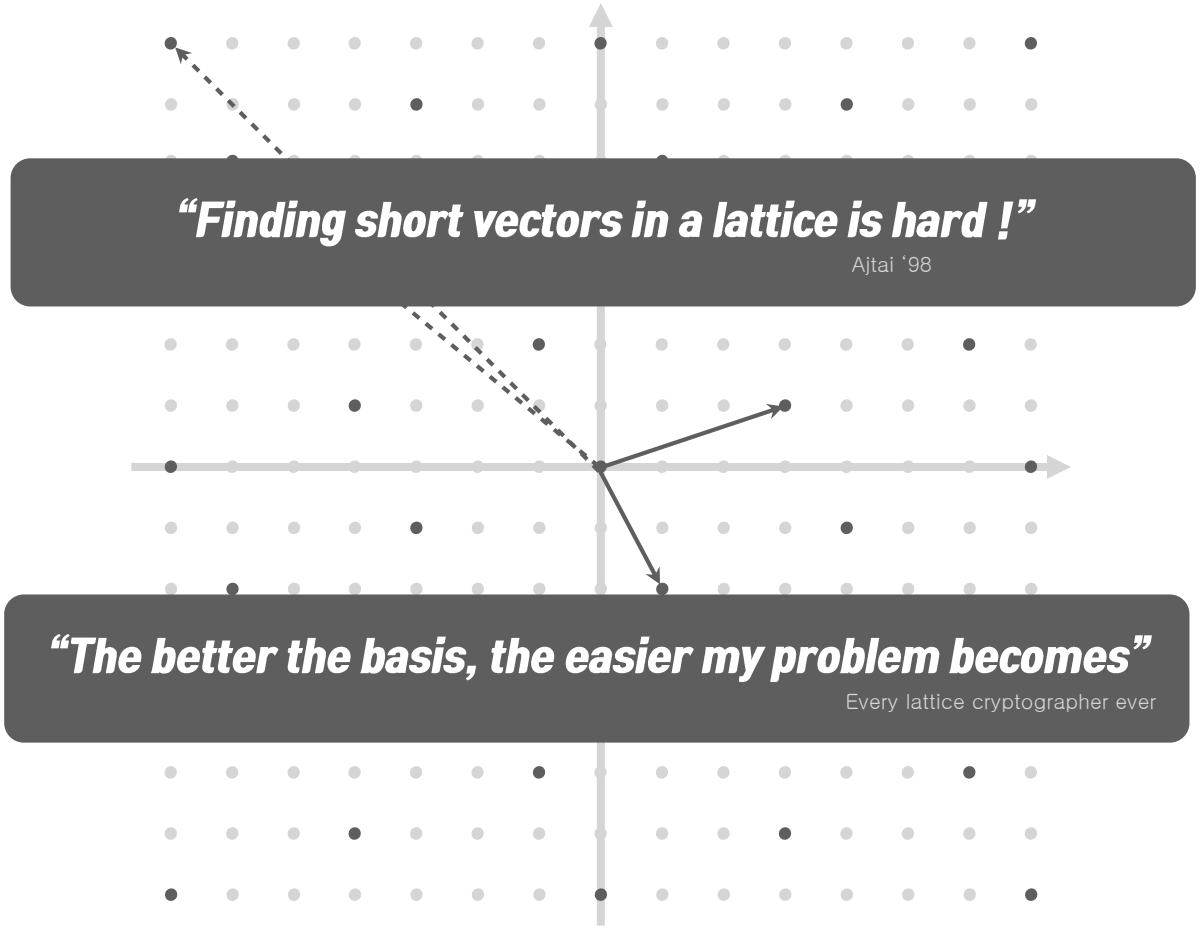
Figure: *Different Basis of same Lattice*

Fact. A lattice has infinite number of bases.
Fact. Integer lattices only have bases with integer entries.
A lattice is the set of integer combinations of any of its bases.

A lattice with basis $\mathbf{B}$ is denoted by $\mathcal{L}(\mathbf{B})$ (denoted by $\Lambda(\mathbf{B})$ in some literature). The notion are abused to non-basis $\mathbf{B}$ in some literature.

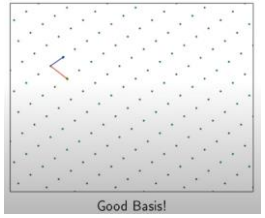
Lattice : Good basis vs. Bad basis

Lattice problem



Lattice Problem : SVP (Shortest Vector Problem)

Good basis is easy to find SVP.



$$L = \{z_1 b_1 + z_2 b_2\} = \{z_1 \begin{bmatrix} 5 \\ 1 \end{bmatrix} + z_2 \begin{bmatrix} -2 \\ 8 \end{bmatrix}\}$$

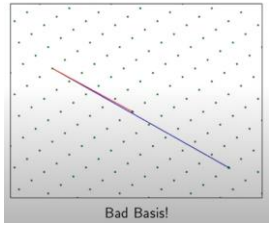
Now we want to find the nearest point to $\begin{bmatrix} 27 \\ 8 \end{bmatrix}$

$$\begin{cases} 5z_1 - 2z_2 = 27 \\ 1z_1 + 8z_2 = 8 \end{cases} \Rightarrow \begin{cases} z_1 = 5.52 \\ z_2 = 0.309 \end{cases} \Rightarrow (z_1, z_2) = (6, 0)$$

$$z_1 \begin{bmatrix} 5 \\ 1 \end{bmatrix} + z_2 \begin{bmatrix} -2 \\ 8 \end{bmatrix} = 6 \begin{bmatrix} 5 \\ 1 \end{bmatrix} + 0 \begin{bmatrix} -2 \\ 8 \end{bmatrix} = \begin{bmatrix} 30 \\ 6 \end{bmatrix}$$

We determine if the angle between the points is around 90 degrees, and if it is, we should be able to solve for the nearest point.

Bad basis is difficult to find SVP.



$$L = \{z_1 b_1 + z_2 b_2\} = \{z_1 \begin{bmatrix} 37 \\ 41 \end{bmatrix} + z_2 \begin{bmatrix} 103 \\ 113 \end{bmatrix}\}$$

Now we want to find the nearest point to $\begin{bmatrix} 27 \\ 8 \end{bmatrix}$

$$\begin{cases} 37z_1 - 103z_2 = 27 \\ 41z_1 + 113z_2 = 8 \end{cases} \Rightarrow \begin{cases} z_1 = -53.023 \\ z_2 = 19.309 \end{cases} \Rightarrow (z_1, z_2) = (-53, 19)$$

$$z_1 \begin{bmatrix} 37 \\ 41 \end{bmatrix} + z_2 \begin{bmatrix} 103 \\ 113 \end{bmatrix} = -53 \begin{bmatrix} 37 \\ 41 \end{bmatrix} + 19 \begin{bmatrix} 103 \\ 113 \end{bmatrix} = \begin{bmatrix} -4 \\ -26 \end{bmatrix}$$

This gives us an incorrect point of $(-4, -26)$ and which is not near $(27, 8)$!

Borrowed from M.B. Niasar," Introduction to lattice-based cryptography", FAU, USA. 2020

Hash-and-sign over lattices(1/3)

Sign (sk, *msg*)

1. ***m*** <— Hash (msg)
2. ***v*** <— Discrete Gaussian sample (***m***)
3. Return **s** = (***m-v***)

Verif (pk, *msg*, *s*)

1. Assert ||***s***|| small
2. Assert ***s***-Hash(*msg*) is in L
3. Accept

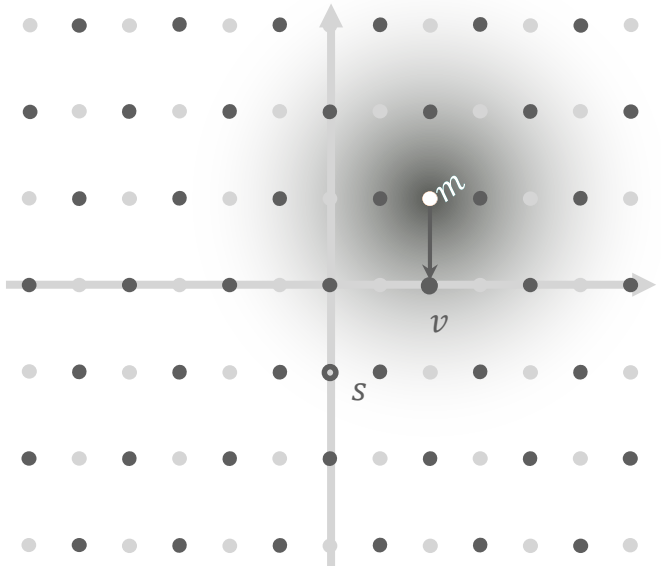
Hash-and-sign over lattices(2/3)

Sign (sk, *msg*)

1. ***m*** <— Hash (msg)
2. ***v*** <— Discrete Gaussian sample (***m***)
3. Return **s** = (***m-v***)

Verif (pk, *msg*, *s*)

1. Assert ||***s*** || small
2. Assert ***s***-Hash(*msg*) is in L
3. Accept



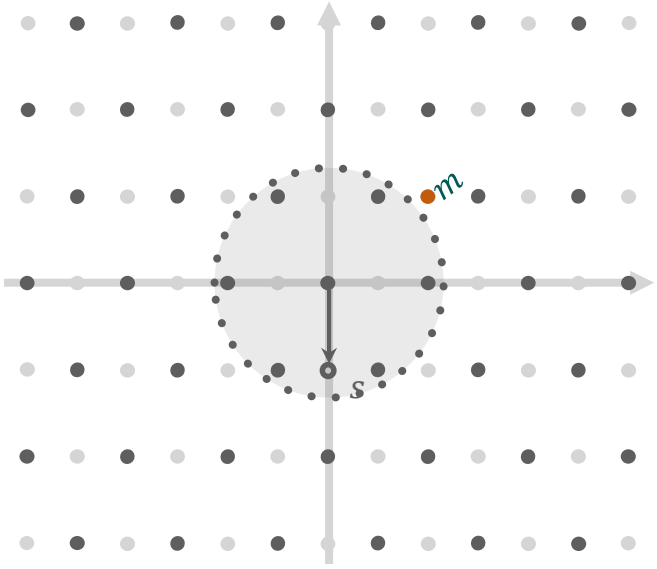
Hash-and-sign over lattices (3/3)

Sign(sk, *msg*)

1. ***m*** <— Hash (msg)
2. ***v*** <— Discrete Gaussian sample (***m***)
3. Return **s** = (***m-v***)

Verif(pk, *msg*, *s*)

1. Assert ||***s*** || small
2. Assert ***s***-Hash(*msg*) is in L
3. Accept



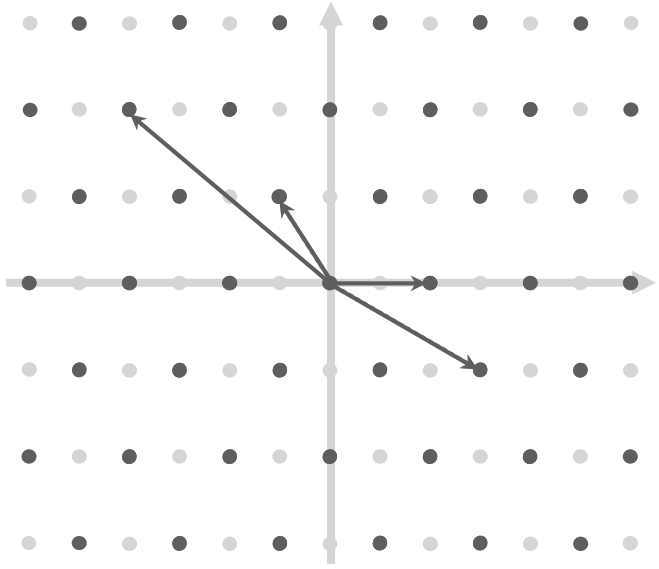
Hash-and-sign over lattices : key recovery

SOLMAE

Key Recovery

Find the secret key directly

- Lattice reduction / SVP (***find short vectors***)
- Should be hard
  - ➡ Large dimension
  - ➡ ''***bad***'' public basis



"Finding short vectors in a lattice is hard !"
Ajtai '98

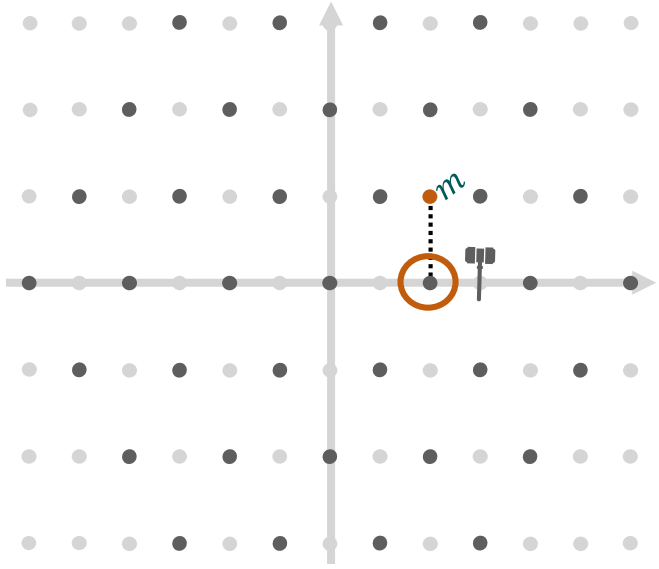
SVP: Shortest Vector Problem

Hash-and-sign over lattices : forgery

Forging a signature

Find a lattice point close to the hash

- CVP instance (***finding a lattice point close enough***)
- Should be hard
  - ➡ small distance
  - ➡ gaussian sample with small variance
  - ➡ ''***good***'' private basis (***short vectors***)



"The better the basis, the easier my problem becomes"

Every lattice cryptographer ever

CVP: Closest Vector Problem

Shortest Vector Problem(SVP)

Definition

Shortest Vector Problem (SVP): Given lattice basis $\mathbf{B}$ of $\mathcal{L}$, find the shortest nonzero vector on $\mathcal{L}$.

SVP is **NP**-hard.

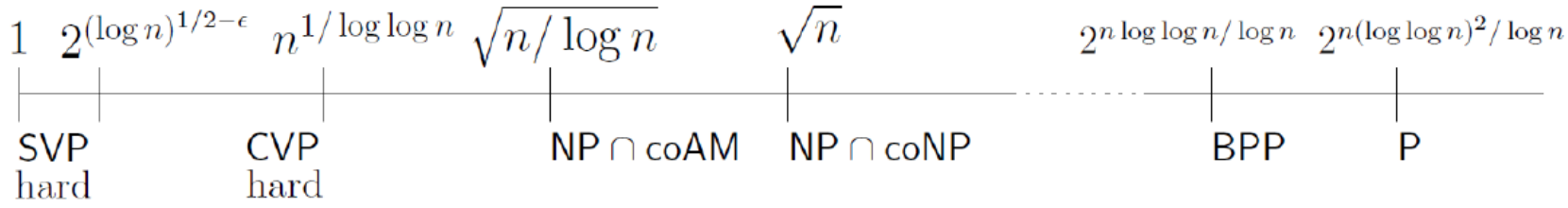
Definition

γ-Approximate Shortest Vector Problem (SVP_γ): Given the lattice basis $\mathbf{B}$ of $\mathcal{L}$, find a non-zero vector $\mathbf{z}$ on lattice $\mathcal{L}$ s.t. $\|\mathbf{z}\| \leq \gamma \cdot \lambda_1(\mathcal{L})$.

Remark. SVP_γ is written to γ-SVP in some literature.
SVP_γ is extremely hard for some γ, but get easier when γ grows very large.

Status of SVP and others

Hardness:



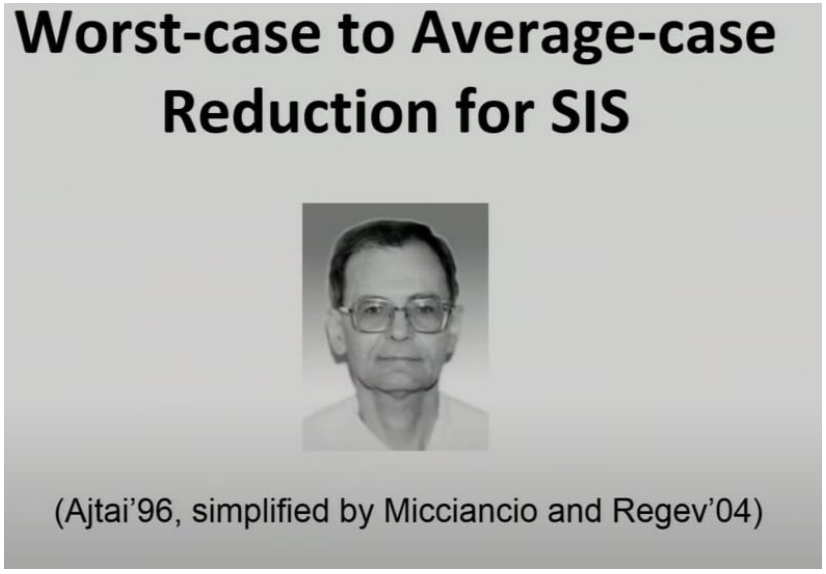
- **NP**-complete for not very small γ
- The hardest among lattice problems
- No known quantum acceleration
- No known subexponential algorithm for $\gamma \leq \sqrt{n}$

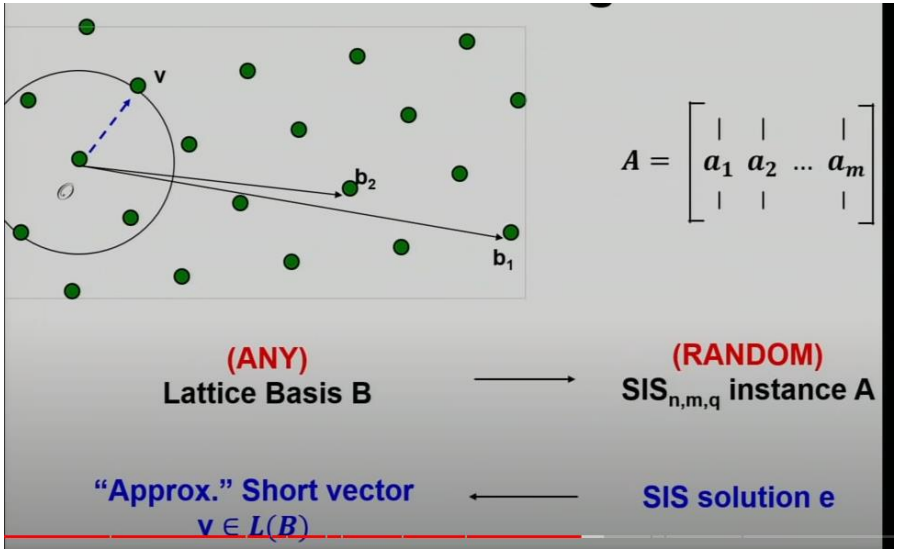
Cryptographic Importance:

- $\gamma = n^c$: the hardness basic of average-case problems (e.g., LWE)

Borrowed from W. Liu presentation

Worst-case to Average-case Reduction for SIS





SIS: Short Integer Solution

5. What is NTRU?

NTRU

- History
  - In 1996, the first version of the system, which was called NTRU, was developed by mathematicians Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman
  - They founded the company NTRU Cryptosystems, Inc., with Daniel Lieman and were given a patent on the cryptosystem
  - The name "NTRU“ stands for
    - ✓Number Theorists 'R' Us or
    - ✓Number Theory Research Unit.

- NTRU from Wikipedia
  - NTRU is an open-source public-key cryptosystem that uses lattice-based cryptography to encrypt and decrypt data of two algorithms:
    - NTRUEncrypt, which is used for encryption
    - NTRUSign, which is used for digital signatures
  - NTRUEncrypt was patented, but placed in the public domain(2017)
  - NTRUSign is patented, but can be used by software under the GPL
  - Resistant to attacks using Shor's algorithm.

NTRU lattice and trapdoor

- $\mathscr{R} = \mathbb{Z}[x]/(x^d + 1)$ where d is a power of 2
- Given $f, g \in \mathscr{R}$ such that f is invertible modulo some prime $q \in \mathbb{Z}$ (usually $q = 12289$), and $h = f^{-1}g \bmod q$
- The NTRU module determined by h is $\{(u, v) \in \mathbb{Z}^2 : uh - v = 0 \mod q\}$
- Two bases of this free module:

$$B_h = \begin{bmatrix} 1 & h \\ 0 & q \end{bmatrix} \quad \mathrm{B}_{f,g} = \begin{bmatrix} f & g \\ F & G \end{bmatrix}$$

  where $F, G \in \mathscr{R}$ such that $fG - gF = q$.

- $B_{f,g}$ is the trapdoor(secret key) for B_h (public key). NTRUEncrypt only need (f, g), but NTRUSign and FALCON also need (F, G).

6. Details of SOLMAE

ToC of SOLMAE Specification

Table of Contents

1 Introduction . . . 3
1.1 Design rationale . . . 3
1.2 Advantages and limitations . . . 6
2 Preliminaries . . . 6
3 Specifications . . . 8
3.1 High-level view of the SOLMAE's signature scheme . . . 8
3.2 Design of `KeyGen` . . . 8
3.3 Design of `Sign`: . . . 11
3.4 Design of `Sample`: . . . 12
3.5 Design of `Verif`: . . . 14
3.6 Miscellanous . . . 14
3.7 List of parameters . . . 16
4 Preliminary performance analysis . . . 16
4.1 Description of platform . . . 16
4.2 Performance of our reference implementation . . . 18
5 Security . . . 18
5.1 Model for lattice reduction . . . 18
5.2 Key recovery attack . . . 19
5.3 Signature forgery by reduction to Approx-CVP. . . . 19
5.4 On the other attacks on SOLMAE . . . 20
5.5 Concrete security . . . 20
5.6 Side-Channel Resilience . . . 21
6 Summary or Conclusion . . . 21

Design Rationale

Two publications related with SOLMAE are:

- MITAKA paper presented at Eurocrypt2022 [EFG+22] and
- Compression paper presented at Crypto2022 [ETWY22].

1.1 Design rationale

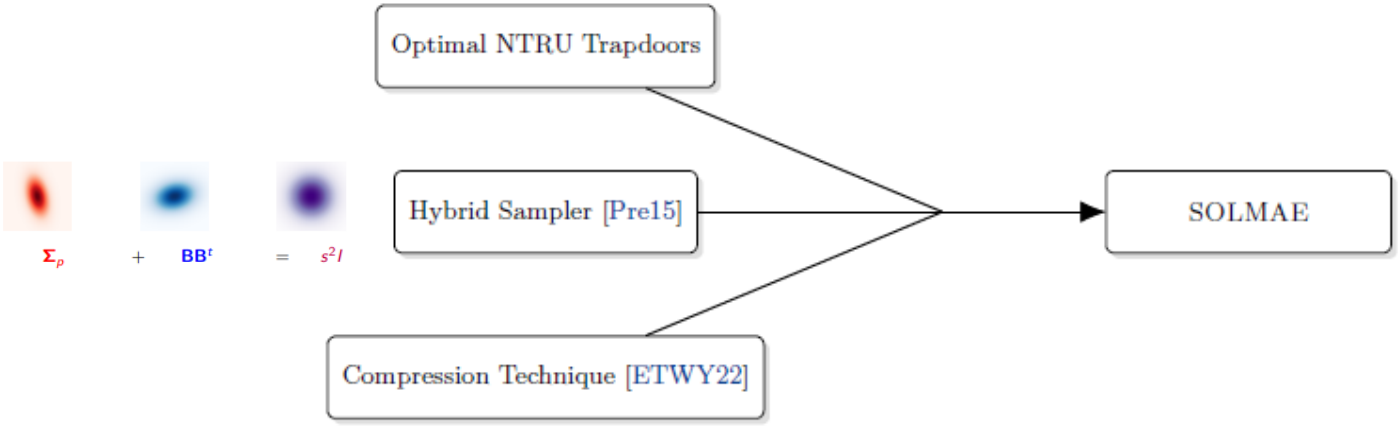


Fig. 1: Overview of SOLMAE

Overall, SOLMAE is summarized in Figure 1.

EFG+22. T. Espitau, P.-A. Fouque, F. Gérard, M. Rossi, A. Takahashi, M. Tibouchi, A. Wallet, and Y. Yu. Mitaka: A simpler, parallelizable, maskable variant of falcon. In *EUROCRYPT 2022, Part III*, vol. 13277 of *LNCS*, pp. 222–253. Springer, Heidelberg, 2022. 3, 5, 12, 13

ETWY22. T. Espitau, M. Tibouchi, A. Wallet, and Y. Yu. Shorter hash-and-sign lattice-based signatures. *IACR Cryptol. ePrint Arch., to appear at CRYPTO 2022*, p. 785, 2022. 3, 6, 11, 16

Pre15. T. Prest. *Gaussian Sampling in Lattice-Based Cryptography*. PhD thesis, École Normale Supérieure, Paris, France, 2015. 3, 5, 12, 13

Keygen(sk, pk)

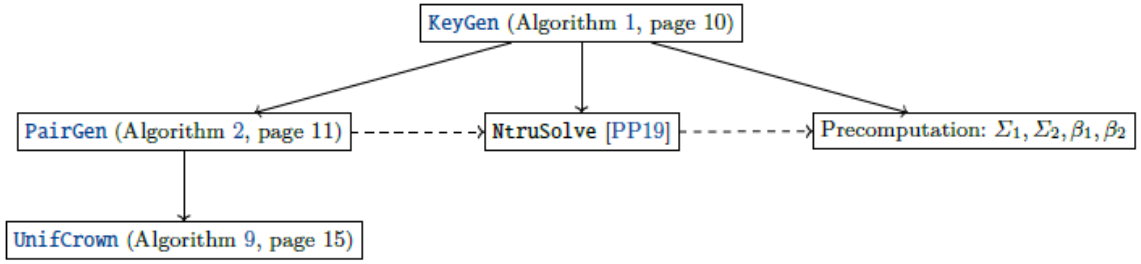


Fig. 2: Flowchart of KeyGen.

Algorithm 1: KeyGen

Input: A modulus q, a target quality parameter $1 < \alpha$, parameters $\sigma_{\text{sig}}, \eta > 0$
Output: A basis $((f,g),(F,G)) \in R^2$ of an NTRU lattice $\mathcal{L}_{\text{NTRU}}$ with $\mathcal{Q}(f,g) = \alpha$;

// Secret basis computation:
repeat
  $\mathbf{b}_1 := (f,g) \leftarrow \texttt{PairGen}(q, \alpha, R_-, R_+)$;
until f *is invertible modulo* q;
$\mathbf{b}_2 := (F,G) \leftarrow \texttt{NtruSolve}(q, f, g)$;

// Public key data computation:
$h \leftarrow g/f \bmod q$;
$\gamma \leftarrow 1.1 \cdot \sigma_{\text{sig}} \cdot \sqrt{2d}$; /* tolerance for signature length */

// Sampling data computation, in Fourier domain:
$\beta_1 \leftarrow \frac{1}{\langle \mathbf{b}_1, \mathbf{b}_1 \rangle_K} \cdot \mathbf{b}_1$;
$\Sigma_1 \leftarrow \sqrt{\frac{\sigma_{\text{sig}}^2}{\langle \mathbf{b}_1, \mathbf{b}_1 \rangle_K} - \eta^2}$;
$\widetilde{\mathbf{b}}_2 := (\widetilde{F}, \widetilde{G}) \leftarrow \mathbf{b}_2 - \langle \beta_1, \mathbf{b}_2 \rangle \cdot \mathbf{b}_1$;
$\beta_2 \leftarrow \frac{1}{\langle \widetilde{\mathbf{b}}_2, \widetilde{\mathbf{b}}_2 \rangle_K} \cdot \widetilde{\mathbf{b}}_2$;
$\Sigma_2 \leftarrow \sqrt{\frac{\sigma_{\text{sig}}^2}{\langle \widetilde{\mathbf{b}}_2, \widetilde{\mathbf{b}}_2 \rangle_K} - \eta^2}$;

$\mathsf{sk} \leftarrow (\mathbf{b}_1, \mathbf{b}_2, \widetilde{\mathbf{b}}_2, \Sigma_1, \Sigma_2, \beta_1, \beta_2)$;
$\mathsf{pk} \leftarrow (q, h, \sigma_{\text{sig}}, \eta, \gamma)$;
return sk, pk;

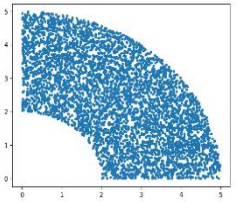
Algorithm 2: PairGen

Input: A modulus q, a target quality parameter $1 < \alpha$, two radii parameters $0 < R_- < R_+$
Output: A pair (f,g) with $\mathcal{Q}(f,g) = \alpha$
for $i = 1$ *to* $d/2$ **do**
  $x_i, y_i \leftarrow \texttt{UnifCrown}(R_-, R_+)$; /* see Algorithm 9 */
  $\theta_x, \theta_y \leftarrow \mathcal{U}(0,1)$;
  $\varphi_{f,i} \leftarrow |x_i| \cdot e^{2i\pi\theta_x}$;
  $\varphi_{g,i} \leftarrow |y_i| \cdot e^{2i\pi\theta_y}$;
end
$(f^{\mathbb{R}}, g^{\mathbb{R}}) \leftarrow \left(\mathsf{FFT}^{-1}((\varphi_{f,i})_{i \le d/2}), \mathsf{FFT}^{-1}((\varphi_{g,i})_{i \le d/2})\right)$;
$(\mathbf{f}, \mathbf{g}) \leftarrow (\lfloor f_i^{\mathbb{R}} \rceil)_{i \le d/2}, (\lfloor g_i^{\mathbb{R}} \rceil)_{i \le d/2}$;
$(\varphi(f), \varphi(g)) \leftarrow (\mathsf{FFT}(\mathbf{f}), \mathsf{FFT}(\mathbf{g}))$;
for $i = 1$ *to* $d/2$ **do**
  if $q/\alpha^2 > |\varphi_i(f)|^2 + |\varphi_i(g)|^2$ *or* $\alpha^2 q < |\varphi_i(f)|^2 + |\varphi_i(g)|^2$ **then**
    restart;
  end
end
return (f, g);

Algorithm 9: UnifCrown

Input: Parameters $0 < R_- < R_+$.
Output: A point (x,y) with uniform distribution in $A(R_-, R_+)$

$u_\rho, u_\theta \leftarrow \mathcal{U}(0,1)$;
$\rho \leftarrow \sqrt{R_-^2 + u_\rho(R_+^2 - R_-^2)}$;
$x \leftarrow \rho \cdot \cos(\frac{\pi}{2} u_\theta)$;
$y \leftarrow \rho \cdot \sin(\frac{\pi}{2} u_\theta)$;
return (x,y)



Sign(m, salt, sk)

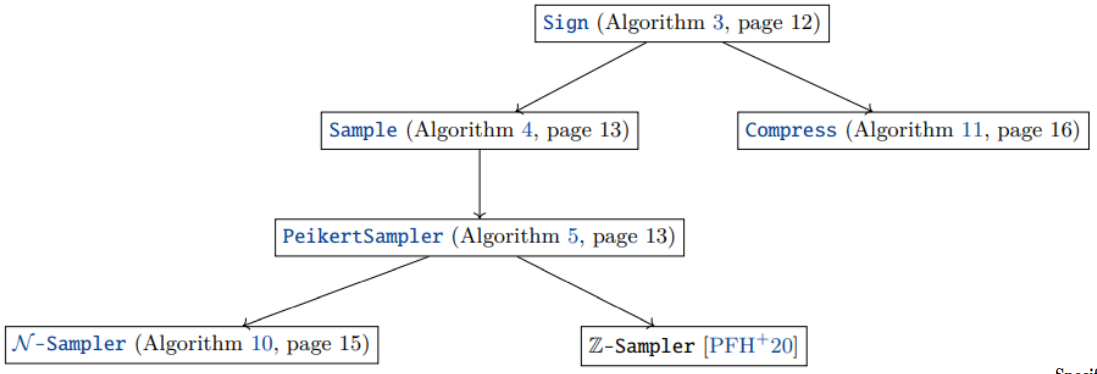


Fig. 3: Flowchart of Sign.

Algorithm 10: $\mathcal{N}$-Sampler

Input: The degree d of R.
Output: Two variables x, y with distribution $\mathcal{N}_d$

$u_\rho, u_\theta \leftarrow \mathcal{U}(0,1)$;
$\rho \leftarrow \sqrt{-2d \ln u_\rho}$;
$x \leftarrow \rho \cdot \cos(2\pi u_\theta)$;
$y \leftarrow \rho \cdot \sin(2\pi u_\theta)$;
return (x, y)

Specifications of $\mathbb{Z}$-Sampler: This step is surprisingly delicate. We reuse the ingenious method of FALCON, and refer to their documentation [PFH+20] for the details about the parameters. Below, we give only an informal description of the necessary steps based on [ZSS20, HPRR20].

Algorithm 3: Sign

Input: A message $M \in \{0,1\}^*$, a tuple $\mathbf{sk} = ((f,g),(F,G),(\widetilde{F},\widetilde{G}),\sigma_{\mathtt{sig}},\Sigma_1,\Sigma_2,\eta)$, a rejection parameter $\gamma > 0$.
Output: A pair $(r, \mathtt{Compress}(\mathbf{s}_1))$ with $r \in \{0,1\}^{320}$ and $\|(\mathbf{s}_1,\mathbf{s}_2)\| \leq \gamma$.

$r \leftarrow \mathcal{U}(\{0,1\}^{320})$;
$\mathbf{c} \leftarrow (0, \mathtt{H}(r||M))$;
$\hat{\mathbf{c}} \leftarrow \mathtt{FFT}(\mathbf{c})$;
repeat
  $(\hat{s}_1, \hat{s}_2) \leftarrow \hat{\mathbf{c}} - \mathtt{Sample}(\hat{\mathbf{c}}, \mathbf{sk})$;
  // $(\mathbf{s}_1,\mathbf{s}_2) \leftarrow D_{\mathcal{L}_{\mathtt{NTRU}},\mathbf{c},\sigma_{\mathtt{sig}}}$
until $\|(\mathtt{FFT}^{-1}(\hat{s}_1), \mathtt{FFT}^{-1}(\hat{s}_2))\|^2 \leq \gamma^2$;
$s_1 \leftarrow \mathtt{FFT}^{-1}(\hat{s}_1)$;
$s \leftarrow \mathtt{Compress}(s_1)$;
return (r, s);

The sampling of the signature vector in Algorithm 3 is a cascade of different sampling algorithms, Sample, PeikertSampler, and at the deepest level, $\mathbb{Z}$-Sampler. Their specifications follow.

Algorithm 4: Sample

Input: A target $\mathbf{c} = (0, \mathbf{c}') \in K_{\mathbb{R}}^2$, a tuple
  $\mathbf{sk} = (\mathbf{b}_1 = (f,g), \mathbf{b}_2 = (F,G), \widetilde{\mathbf{b}_2} = (\widetilde{F},\widetilde{G}), \sigma_{\mathtt{sig}}, \Sigma_1, \Sigma_2, \beta_1, \beta_2)$.
Output: A vector $\mathbf{v} \in \mathcal{L}_{\mathtt{NTRU}}$ with distribution statistically close to $D_{\mathcal{L}_{\mathtt{NTRU}},\mathbf{c},\sigma_{\mathtt{sig}}}$.

$\mathbf{t} \leftarrow \mathbf{c}, \mathbf{v} \leftarrow \mathbf{0}$;
for $i = 2$ *to* 1 **do**
  $t_i \leftarrow \langle \beta_i, \mathbf{t} \rangle_K$;
  $z_i \leftarrow \mathtt{PeikertSampler}(t_i, \Sigma_i, \eta)$; /* $z_i \leftarrow D_{R,t_i,\frac{\sigma_{\mathtt{sig}}}{\|\widetilde{\mathbf{b}}_i\|}}$ */
  $\mathbf{t} \leftarrow \mathbf{t} - z_i\mathbf{b}_i, \mathbf{v} \leftarrow \mathbf{v} + z_i\mathbf{b}_i$;
end
return $\mathbf{v}$;

Algorithm 5: PeikertSampler

Input: A target $t \in K_{\mathbb{R}}$, parameters $\Sigma, \eta \in K_{\mathbb{R}}^{++}$.
Output: A vector $\mathbf{v} \in R$ with distribution statistically close to $D_{R,t,\sigma}$, where $\sigma = \sqrt{\Sigma^2 + \eta^2}$.

$p \leftarrow \Sigma \cdot \mathcal{N}_1^{K_{\mathbb{R}}}$; /* $p \leftarrow \mathcal{N}_{\Sigma^2}^{K_{\mathbb{R}}}$, done with $\mathcal{N}$-Sampler (Algorithm 10) */
$(p_1, \ldots, p_d) \leftarrow \mathtt{FFT}^{-1}(p)$; /* $(p_i)_i \in \mathbb{R}^d$ */
$(t_1, \ldots, t_d) \leftarrow \mathtt{FFT}^{-1}(t)$; /* $(t_i)_i \in \mathbb{Z}^d$ */
for $i = 1$ *to* d **do**
  $x_i \leftarrow \mathbb{Z}\text{-}\mathtt{Sampler}(t_i - p_i, \eta)$;
end
return $\mathtt{FFT}(x_1, \ldots, x_d)$;

PFH+20. T. Prest, P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang. FALCON. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions. 12, 14

Verify(m, salt, sig, pk)

SOLMAE

Algorithm 6: Verif

Input: A signature (r, s) on M, a public key $\mathbf{pk} = h$, a bound γ.
Output: Accept or reject.

$s_1 \leftarrow \texttt{Decompress}(s)$;
$c \leftarrow \texttt{H}(r||M)$;
$s_2 \leftarrow c + hs_1 \bmod q$;
if $\|(\mathbf{s}_1, \mathbf{s}_2)\|^2 > \gamma^2$ **then**
  return Reject.
end
return Accept.

Algorithm 11: Compress

Input: A polynomial $s = \sum_{i=0}^{d-1} s_i X^i \in R = \mathbb{Z}[X]/(X^d + 1)$ and an integer *slen*.
Output: A compressed representation of *str* of s of bitsize *slen*, or $\perp$

$str \leftarrow \{\}$;
for $i = 0$ *to* $d - 1$ **do**
  $str \leftarrow (str \,||\, b)$ where $b = 1$ if $s_i < 0$, $b = 0$ otherwise;
  $str \leftarrow (str \,||\, b_6 b_5 \cdots b_0)$ where $b_j = (|s_i| \gg j)\&0x1$;
  $k \leftarrow |s_i| \gg 7$;
  $str \leftarrow (str \,||\, 0^k 1)$
end
if $|str| > slen$ **then**
  $str \leftarrow \perp$;
end
else
  $str \leftarrow (str \,||\, 0^{slen - |str|})$
end
return *str*

Algorithm 12: Decompress

Input: A bitstring *str* of bitsize *slen*
Output: A polynomial $s = \sum_{i=0}^{d-1} s_i X^i \in R = \mathbb{Z}[X]/(X^d + 1)$ or $\perp$

if $|str| \neq slen$ **then**
  return $\perp$;
end
for $i = 0$ *to* $d - 1$ **do**
  $s_i' \leftarrow \sum_{j=0}^{6} 2^{6-j} str[1 + j]$;
  $k \leftarrow 0$;
  while $str[8 + k] = 0$ **do**
    $k \leftarrow k + 1$
  end
  $s_i \leftarrow (-1)^{str[0]} \cdot (s_i' + 2^7 k)$;
  if $s_i = 0$ *and* $str[0] = 1$ **then**
    return $\perp$
  end
  $str \leftarrow str[9 + k :]$
end
if $|str| \neq 0^{|str|}$ **then**
  return $\perp$;
end
return $s = \sum_{i=0}^{d-1} s_i X^i$

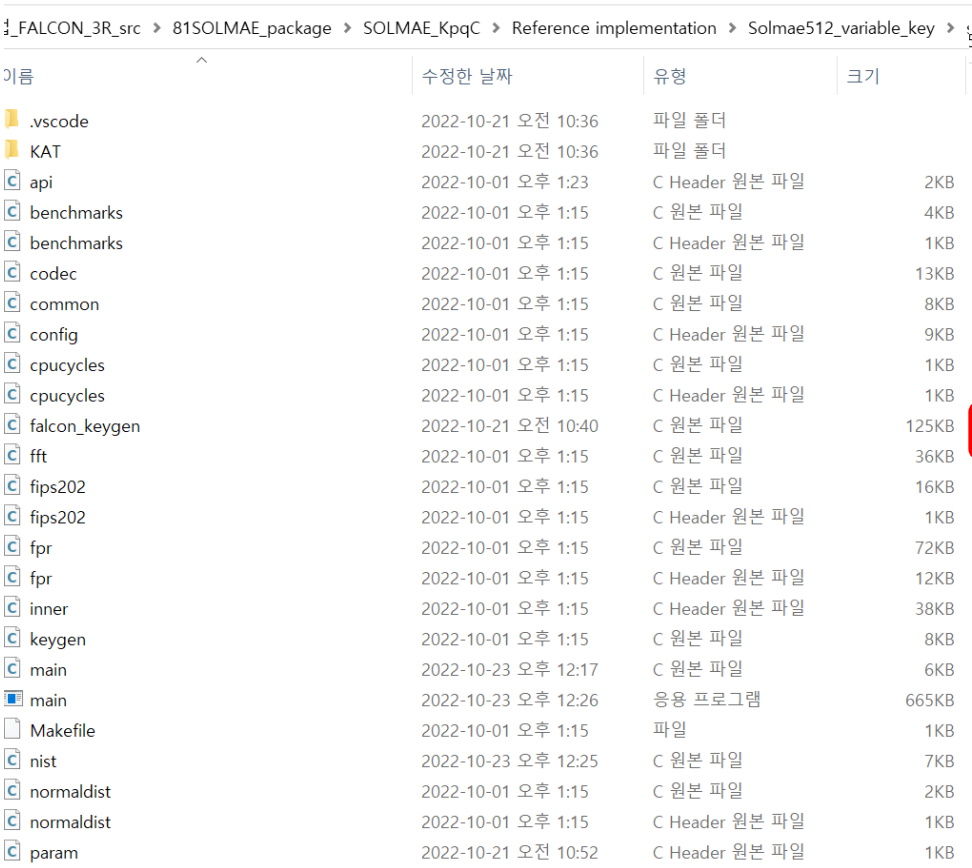
List of Parameters

Table 1: List of parameters for SOLMAE

	SOLMAE–512	SOLMAE–1024
ring degree d	512	1024
dimension $2d$	1024	2048
modulus q	12289	12289
salt length k	320	320
smoothing η	1.338	1.351
smoothness ϵ	2^{-41}	2^{-41}
quality α	1.17	1.64
correction δ	0.065	0.3
lower radius R_-	101.95	100.85
upper radius R_+	122.49	148.54
signature width σ_{sig}	173.54	245.62
slack τ	1.04	1.04
rejection bound γ^2	33870790	134150669

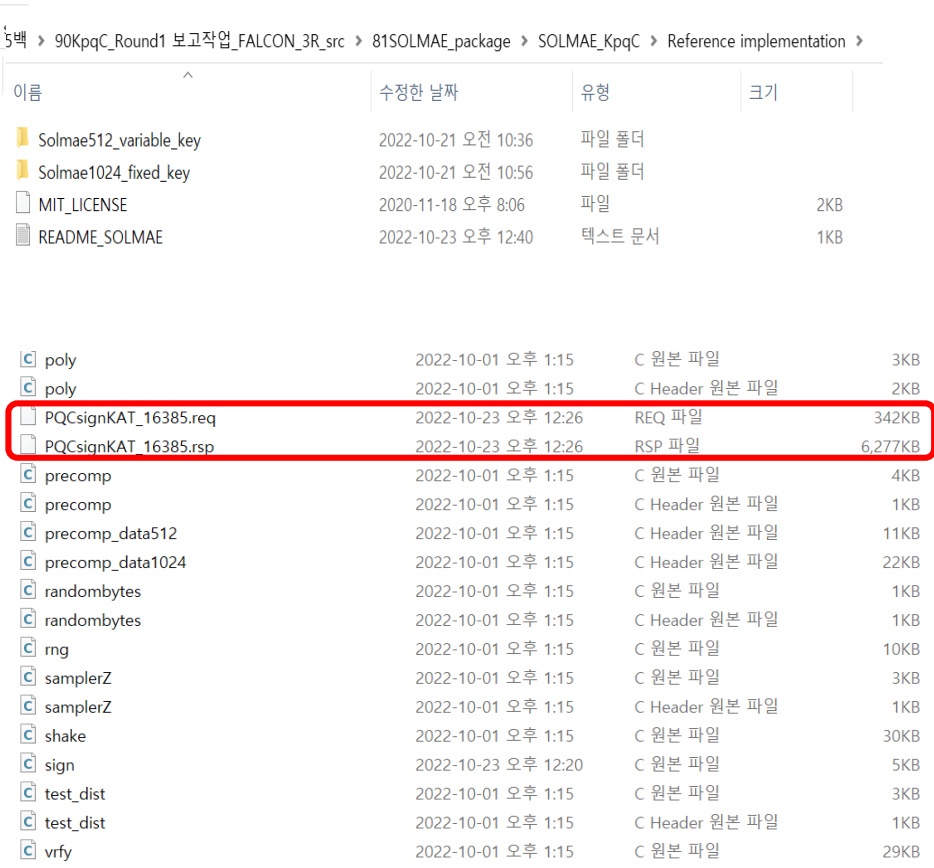
Note that the value of smoothing η can be used 1.320 for SOLMAE−512 and SOLMAE−1024 together. In practice, it doesn't make difference.

Reference Implementation in C, SOLMAE_512



_FALCON_3R_src › 81SOLMAE_package › SOLMAE_KpqC › Reference implementation › Solmae512_variable_key ›

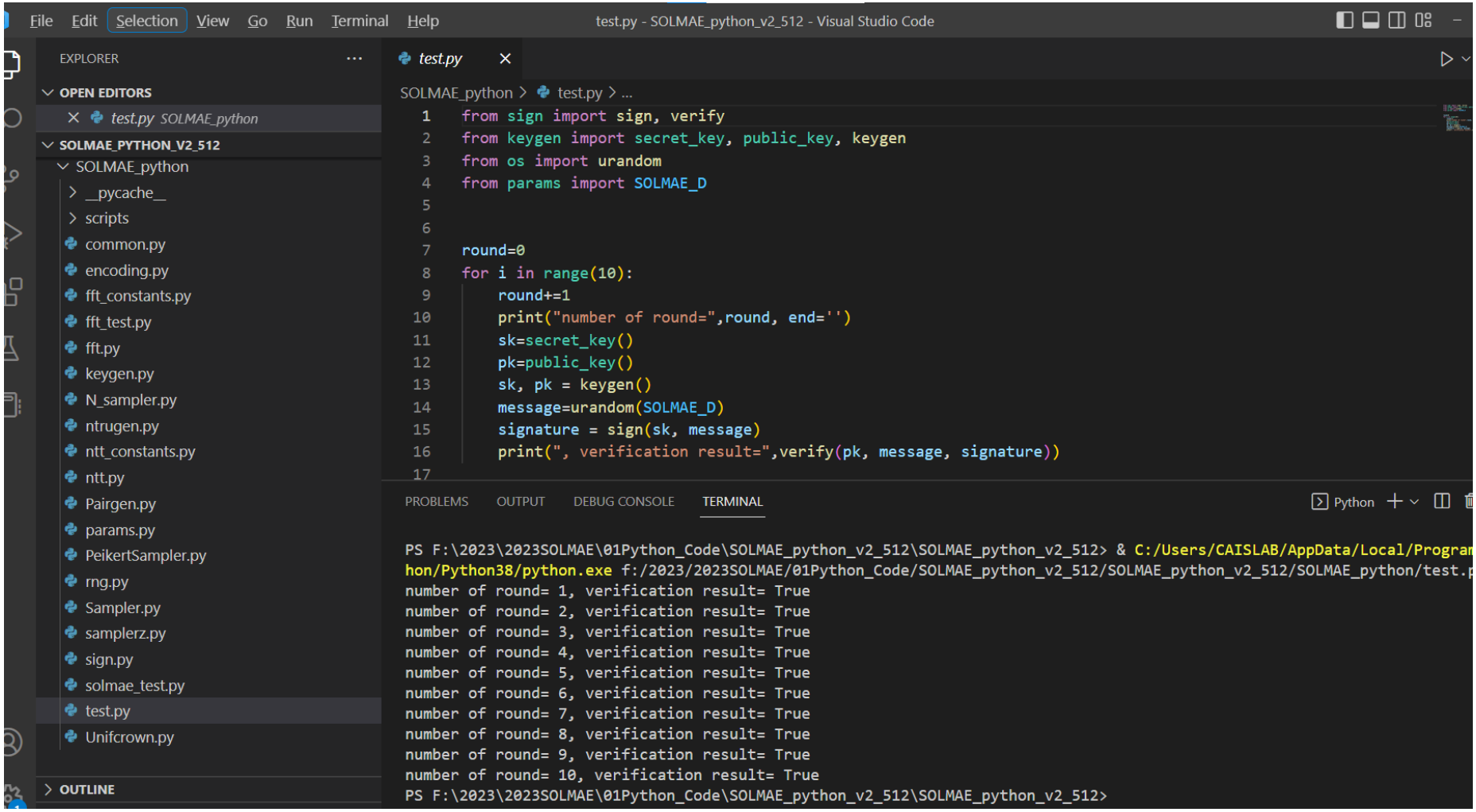
이름	수정한 날짜	유형	크기
.vscode	2022-10-21 오전 10:36	파일 폴더	
KAT	2022-10-21 오전 10:36	파일 폴더	
api	2022-10-01 오후 1:23	C Header 원본 파일	2KB
benchmarks	2022-10-01 오후 1:15	C 원본 파일	4KB
benchmarks	2022-10-01 오후 1:15	C Header 원본 파일	1KB
codec	2022-10-01 오후 1:15	C 원본 파일	13KB
common	2022-10-01 오후 1:15	C 원본 파일	8KB
config	2022-10-01 오후 1:15	C Header 원본 파일	9KB
cpucycles	2022-10-01 오후 1:15	C 원본 파일	1KB
cpucycles	2022-10-01 오후 1:15	C Header 원본 파일	1KB
falcon_keygen	2022-10-21 오전 10:40	C 원본 파일	125KB
fft	2022-10-01 오후 1:15	C 원본 파일	36KB
fips202	2022-10-01 오후 1:15	C 원본 파일	16KB
fips202	2022-10-01 오후 1:15	C Header 원본 파일	1KB
fpr	2022-10-01 오후 1:15	C 원본 파일	72KB
fpr	2022-10-01 오후 1:15	C Header 원본 파일	12KB
inner	2022-10-01 오후 1:15	C Header 원본 파일	38KB
keygen	2022-10-01 오후 1:15	C 원본 파일	8KB
main	2022-10-23 오후 12:17	C 원본 파일	6KB
main	2022-10-23 오후 12:26	응용 프로그램	665KB
Makefile	2022-10-01 오후 1:15	파일	1KB
nist	2022-10-23 오후 12:25	C 원본 파일	7KB
normaldist	2022-10-01 오후 1:15	C 원본 파일	2KB
normaldist	2022-10-01 오후 1:15	C Header 원본 파일	1KB
param	2022-10-21 오전 10:52	C Header 원본 파일	1KB



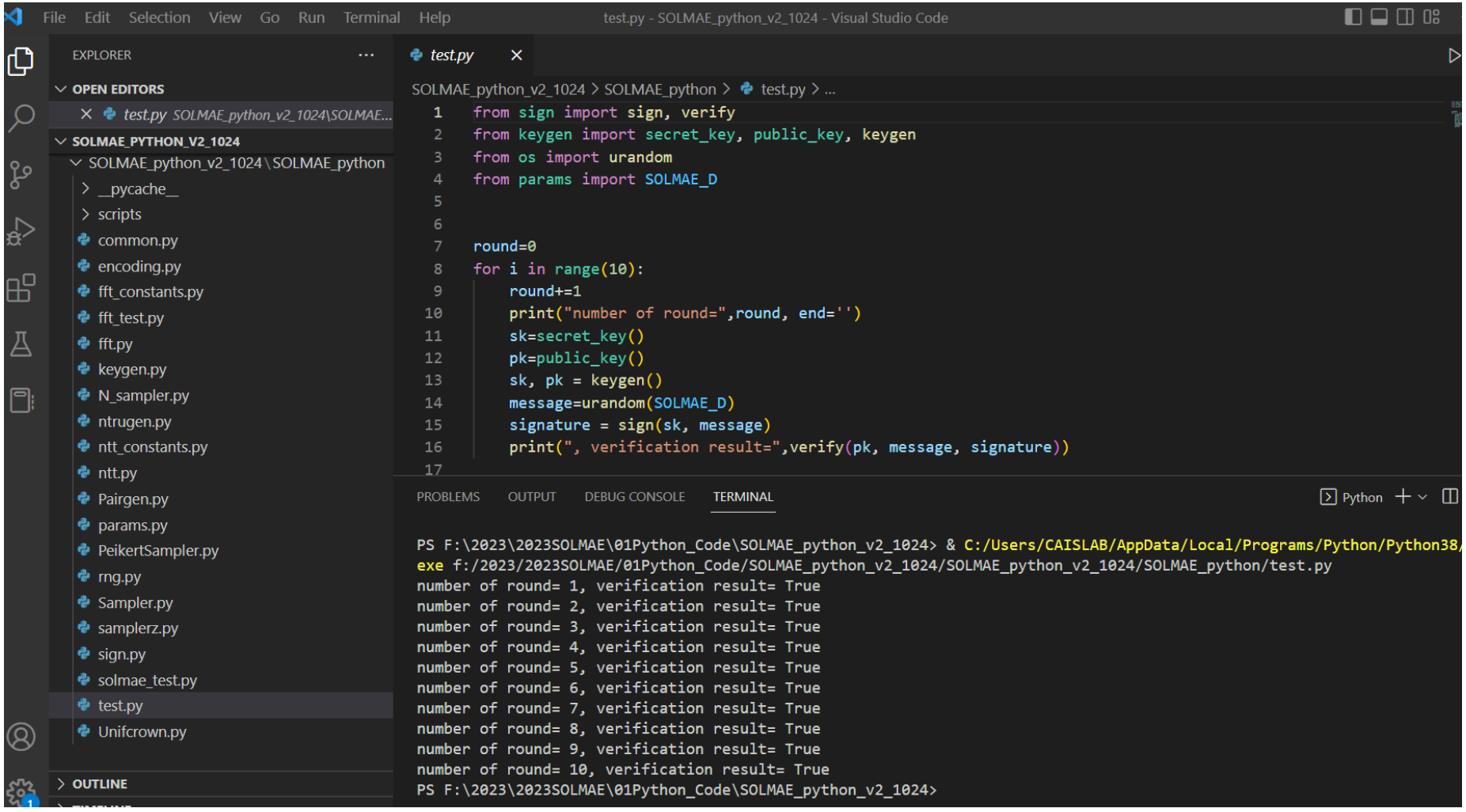
5백 › 90KpqC_Round1 보고작업_FALCON_3R_src › 81SOLMAE_package › SOLMAE_KpqC › Reference implementation ›

이름	수정한 날짜	유형	크기
Solmae512_variable_key	2022-10-21 오전 10:36	파일 폴더	
Solmae1024_fixed_key	2022-10-21 오전 10:56	파일 폴더	
MIT_LICENSE	2020-11-18 오후 8:06	파일	2KB
README_SOLMAE	2022-10-23 오후 12:40	텍스트 문서	1KB
poly	2022-10-01 오후 1:15	C 원본 파일	3KB
poly	2022-10-01 오후 1:15	C Header 원본 파일	2KB
PQCsignKAT_16385.req	2022-10-23 오후 12:26	REQ 파일	342KB
PQCsignKAT_16385.rsp	2022-10-23 오후 12:26	RSP 파일	6,277KB
precomp	2022-10-01 오후 1:15	C 원본 파일	4KB
precomp	2022-10-01 오후 1:15	C Header 원본 파일	1KB
precomp_data512	2022-10-01 오후 1:15	C Header 원본 파일	11KB
precomp_data1024	2022-10-01 오후 1:15	C Header 원본 파일	22KB
randombytes	2022-10-01 오후 1:15	C 원본 파일	1KB
randombytes	2022-10-01 오후 1:15	C Header 원본 파일	1KB
rng	2022-10-01 오후 1:15	C 원본 파일	10KB
samplerZ	2022-10-01 오후 1:15	C 원본 파일	3KB
samplerZ	2022-10-01 오후 1:15	C Header 원본 파일	1KB
shake	2022-10-01 오후 1:15	C 원본 파일	30KB
sign	2022-10-23 오후 12:20	C 원본 파일	5KB
test_dist	2022-10-01 오후 1:15	C 원본 파일	3KB
test_dist	2022-10-01 오후 1:15	C Header 원본 파일	1KB
vrfy	2022-10-01 오후 1:15	C 원본 파일	29KB

SOLMAE_512 in Python



SOLMAE_1024 in Python



Partial Output of SOLMAE_512 in Python

- Public-key h =
- [7595, 3060, 2018, 11009, 9207, 7481, 2090, 5533, 9452, 8619, 1936, 7892, 11702, 4719, 5185, 9817, 10798, 11062, 2182, 4514, 1497, 5813, 2522, 758, 8821, 10617, 3788, 7738, 12037, 9620, 813, 4350, 4282, 3696, 7276, 1474, 979, 9946, 1818, 4190, 4086, 8315, 3654, 8468, 10350, 8593, 2706, 9167, 11999, 2657, 10160, 6025, 6124, 12080, 4532, 706, 3876, 1371, 1689, 2491, 3558, 7278, 8561, 9356, 1671, 11087, 11528, 5591, 6828, 8283, 11694, 10064, 409, 2773, 6682, 3715, 5961, 6788, 10340, 11140, 11152, 1231, 5077, 4902, 11222, 10460, 1388, 4060, 7235, 2303, 5972, 6928, 1082, 3943, 1621, 4612, 5304, 2304, 345, 1089, 4102, 5689, 4796, 9446, 2706, 2607, 2736, 5784, 4977, 2925, 79, 8071, 8455, 9299, 6622, 857, 10260, 5614, 11659, 8678, 10311, 2754, 11893, 4712, 6392, 8764, 1251, 1856, 2911, 6932, 8744, 8432, 9497, 12243, 5960, 4446, 9503, 965, 12205, 10800, 6435, 1799, 10939, 7513, 4269, 7954, 2757, 12175, 9696, 4608, 8050, 4652, 11028, 8149, 11147, 7568, 8971, 9641, 12121, 10561, 12225, 8513, 3167, 10384, 9640, 10053, 416, 8720, 5096, 5777, 623, 12223, 1661, 8457, 3194, 6113, 1992, 5692, 3334, 11313, 10170, 1707, 4675, 11892, 10373, 8798, 8844, 2675, 9534, 3486, 1381, 2786, 10592, 4341, 9867, 7071, 7394, 11457, 7838, 8308, 8170, 1088, 1571, 6495, 1581, 2478, 7490, 9784, 6640, 5376, 3490, 1434, 1045, 4658, 9797, 8798, 10213, 2299, 998, 6745, 4470, 11995, 4444, 7276, 6326, 4623, 10425, 3729, 2674, 4921, 9651, 4586, 11847, 1582, 3004, 12187, 1111, 8902, 3284, 220, 6388, 473, 6889, 2331, 671, 9781, 10608, 3817, 7210, 3112, 4296, 7056, 4007, 12182, 5650, 9756, 5912, 1050, 9781, 7305, 8591, 10446, 2638, 6480, 7978, 286, 3655, 1099, 6349, 9072, 10948, 12113, 482, 8736, 10287, 814, 6937, 7491, 8802, 1304, 7089, 7347, 4218, 8422, 7015, 8511, 925, 4826, 8218, 1344, 2575, 2491, 7195, 7842, 8878, 5417, 11763, 4696, 8745, 8553, 10718, 3701, 3887, 10248, 10914, 11529, 12017, 4515, 1392, 4333, 3574, 118, 10284, 10731, 8031, 450, 11576, 3378, 690, 2984, 11354, 324, 10984, 3108, 3954, 6717, 10884, 2817, 6904, 4240, 10835, 10374, 6959, 8598, 2570, 7694, 11302, 10578, 5724, 7786, 1620, 689, 3660, 11632, 18, 10655, 10268, 6814, 1338, 4343, 1450, 2674, 8440, 8161, 11096, 3620, 358, 10935, 7805, 1682, 5310, 2138, 11256, 9233, 3334, 6928, 2723, 5477, 10003, 3512, 8309, 4988, 12144, 932, 7660, 4296, 11886, 4298, 9664, 8583, 3956, 9993, 2557, 9319, 10311, 9238, 11755, 1826, 11050, 11485, 10138, 1779, 8524, 4711, 838, 198, 10088, 3736, 3747, 8267, 33, 3160, 2431, 9344, 2734, 4184, 3199, 8342, 7839, 7249, 4107, 8003, 3879, 4686, 6767, 1620, 6546, 8652, 2547, 7633, 5645, 4708, 3856, 8871, 5582, 11643, 11758, 7824, 7736, 11293, 92, 7847, 183, 9203, 11131, 6604, 4148, 4336, 9193, 11755, 9963, 1196, 2403, 11150, 7280, 8290, 7006, 10286, 810, 1777, 6730, 6153, 8427, 10674, 2298, 8011, 5449, 8157, 10563, 2064, 6718, 6970, 11329, 2235, 43, 11625, 3013, 8074, 7554, 7429, 10327, 9991, 7267, 3770, 3818, 3539, 4295, 3391, 10722, 1425, 4643, 1, 11303, 8211, 625, 692, 1805, 419, 2527, 1324, 9262, 10291, 2713, 8679, 3324, 1403, 3665, 10457, 424, 10322, 3963, 9623, 2892, 10808, 10188, 9945, 9547, 5041, 10240, 4910, 11573, 7215]
- ==========
- Length of Message for SOLMAE_512 = 512
- Message is
fb5eb9fb77457536218a365564fb6bcb385b90a18ae3e6136e985dfc6c63bf0b381206ae17ec324174b2d8847117d1f108f9911d0e19f56da19ae565d3ab2be1d035053770c8fbe6d56b8c4055f6d2a64cffc372f29f84076f90e202d01632ac43b324981601ae5bd75ddebbc5d1f902a256b39060797dd1c1e1d8c8250b65697453542f96baad95bac7d375caf3b0c358ec2c93bafd25dac2ad1ee6f2dad8ded629e669293b43b91ab864909581475e0551c7ce0827433f6b1ec15a36be9906f48042c337ca870a27577ae4b88def60fb05ba1e5d0f8fda57228ec6cacdaeedecc993ed6de79f1c210fc6fab50b73c8eec1eba67e9e08db5cf35f029d61c277650c80a94f88891592948451cf410dc671023cb559eaf898887789c03c18d699559926fb60ff27be5096831309cfce3684d9a077fa8149bb9a99f44e287c34329fcf1bb679ebe4eadf911c3defaa0bcad23f0650d247aa916bb45a40ddf7b2bc2f14c8026a73b21b5c429902376d231985bc4588169194ea8245293baadce2f3bb34e03ba919b069fe221efb9886ebb36844511f53a3321cf458107e5182f9430868dd055e75382ddefaded7a7e63d6f73ca62524b757106e1310d978230e0d5cb68a3fc17470362bcaa1a82656aee93824f55bb8756e9b0c7430f03799e11f57b140b502805bb826f52c1303ed81ca75de639496eecfd99dbc835d80234da3
- ==========
- Length of Sig. = 666
- Sig is
39b9a9054a8fb4f777c7cac312d5822a2b33a9d5c094c90840ba285f28ed30c4a0f4464004b49af2910a29c81ad1f78d1d9d4a65d5a364098c650e5cb947ed94799c375955c2138c02d90d5ef310676a2afc654f7ebc7bff19883e4b9c8ff93a47d952de781c4ad4ee12c4dd9d06ece5dd6f31e242d2220ad42ccfe12ef089a2d25f641ad331aabeee2f2dc131e15f113200f964f02c334a206f22bfb19876b0451535efa6b39d8f6e89e2e553b70a8323fdc9733075ec55d3c7f32853c870bccb3105330bc0c0159d19c6640f1deaeeeae29add16e678f0edb354debf0a399c934445628351bbdd51966e0743b56b7970be77d591eabcee1cb12721851ad5644874f16285a0e755935280d6d3594a4601751a68c4522f82783ed18b2605f7658441022275be229ba8dc421eccfbc9c0ca7467e4be67918b569b59fb7d104dbe5b79445112215f641f28ebf3e0783d0df773e94604bfcba16450b3fdb5d06dadca3b268e53f7f5e5190f8c2769a2e1cdc9a36850d5e21ecd21f9ede2d7f6e6a538a871bf560f7b44c610fb2a012864b0e8f2b5dc741447f654739cbada16a3c819fcc6ceef8b1e0d3ea088b9c86519e9669eecec66646969bbaa778112c63952064f0e23ae6ea5ee8419089b052c3db5f2b2243804827b5cb929dc6d14ae17106f521207338a0bb38921f4f19c174a9481620446ffd4a43d9f4c22373861862c81fcee26af95043ad25cd3c6d0d49d47fa6d789eaa762731e981b639486ea3d1a3c28d889233b375ee0947f02c8bcf2b0e79106e0a9107e5fd5d58fd772d71b16204b94d502c749c295286a8c8ab7d30439cc3c6b8f6e095be353e1e7ef64b9ad7acc5bb3512c0224ccc410431a8e68e30d2b9140575e76173e9a5bb9f635dd546c937442cab84290e833a9eeb8e1d00000000000000000000
- ==========
- **Verification result = True**

Performance(1/2)

SOLMAE

4.1 Description of platform

Our implementation has been tested on various x86–64 platforms, and consistently outperforms FALCON in signing and verification in equal dimension, while key generation is slightly slower. Timings below have been collected on a single core of a Ryzen Threadripper Pro 5975WX @ 3.60 GHz workstation with hyperthreading and frequency scaling disabled.

Table 2: Performance comparison between SOLMAE and FALCON.

		SOLMAE–512	SOLMAE–1024	FALCON–512	FALCON–1024
KeyGen time	Mcycles	27	65	—	—
	time (ms)	7.5	18	5.0	15
pk size	Bytes	896	1792	896	1792
Sign time	kcycles	387	775	—	—
	time (μs)	108	216	220	441
sgn size	Bytes	666	1375	666	1280
Verif time	kcycles	40	84	—	—
	time (μs)	11	23	18	36

Performance(2/2)

SOLMAE

Under the low-speed computing environment, Intel(R) Core(TM) i7-8550U CPU@1.80GHz 8.00GB RAM, we have executed the performance check of our reference implementation without compression/decompression for SOLMAE–512 and SOLMAE–1024 whose C-src codes are attached in our submission package to KpqC competition.

For this test, the input messages are chosen 1,024 byte randomly per 10,000 times with each count using different key pairs. The average clock cycle and time (μs) during `KeyGen`, `Sign` and `Verif` using SOLMAE–512 and SOLMAE–1024 are shown in Table 3.

Table 3: Average performance per each step of SOLMAE–512 and SOLMAE–1024

	SOLMAE–512		SOLMAE–1024	
`KeyGen`	26,336,721.9	13,231.4	56,381,295.8	28,301.3
`Sign`	499,836.9	244.2	975,022.5	491.9
`Verif`	35,427.8	15.0	69,530.2	35.6

7. Security Evaluation

Way of attack

- Mathematical Attack for key-recovery
  - Lattice-basis attack (*e.g.*, LLL, BKZ, DBKZ, etc.)
  - Meet-in-the-middle attack
  - Hybrid attack, etc.

- Crypto Attack for signature forgery
  - Chosen-ciphertext attack
  - Decryption-failure attack
  - Complicated-padding systems, *etc*.

Security Level

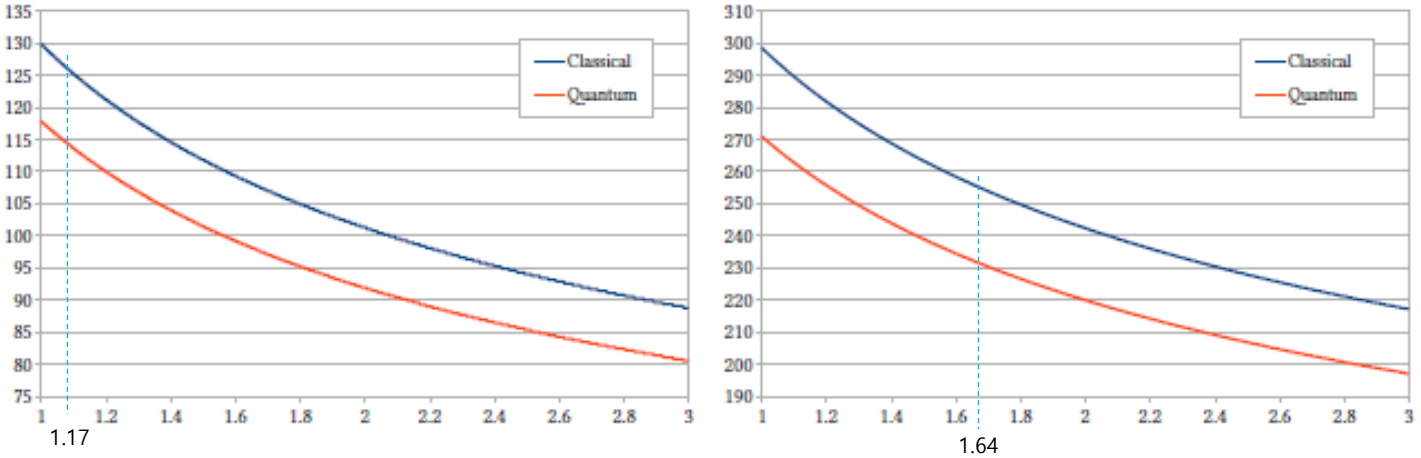


Fig. 4: Security (classical and quantum) against forgery as a function of the quality $1 \leq \alpha \leq 3$ of the lattice sampler (left: dimension 512 and right: dimension 1024).

Table 4: Security level for SOLMAE
(C is classical security, Q is quantum security.)

	SOLMAE−512	SOLMAE−1024
bit security (C/Q)	127/115	256/232
NIST equivalent	NIST-I	NIST-V

Summary in Security Level

- Key Recovery Attack -> SVP
- Signature Forgery -> app-CVP
  - ➔ Similar with the security of FALCON

- Other Attacks
  - Algebraic Attack : No known so far [KEP20]
  - Overstretched NTRU-type[KF17]: significantly improved
  - Hybrid Attack[How07] : not sufficient

KEF20. P. Kirchner, T. Espitau, and P.-A. Fouque. Fast reduction of algebraic lattices over cyclotomic fields. In *CRYPTO 2020, Part II*, vol. 12171 of *LNCS*, pp. 155–185. Springer, Heidelberg, 2020.

KF17. P. Kirchner and P.-A. Fouque. Revisiting lattice attacks on overstretched NTRU parameters. In *EUROCRYPT 2017, Part I*, vol. 10210 of *LNCS*, pp. 3–26. Springer, Heidelberg, 2017. 20

How07. N. Howgrave-Graham. A hybrid lattice-reduction and meet-in-the-middle attack against NTRU. In *CRYPTO 2007*, vol. 4622 of *LNCS*, pp. 150–169. Springer, Heidelberg, 2007. 20

Timing Attack by Secure-IC (1/2)

SOLMAE

SECURE-IC THE SECURITY SCIENCE COMPANY | METHOD

- § We analyse constant-timeness of the implementation of SOLMAE and HAETAE
- § Using reference C code of signature generation only
- § Considering to start with that only sk (private key) is the sensitive variable
  - There is venue for more refinements by considering the noise is sensitive as well
- § Leveraging the Catalyzr tool:
  - Analyzes propagation of depencies until reaching conditional branch & pointer dereference
- § The method can induce false positives, typically:
  - Rejection sampling
  - Variable which depends on the key throught its length (which is public)
  - Test of the 1st byte of sk which appears to be a known constant

Source: https://www.kpqc.or.kr/competition.html

IRCS 국제사이버보안연구원

KAIST

Timing Attack by Secure-IC (2/2)

SECURE-IC THE SECURITY SCIENCE COMPANY | CONCLUSIONS

§ SOLMAE signature generation is constant-time

§ HAETEA is not constant-time, owing to:

- Non constant time Barrett reduction
- Non constant time variable centering
- [...]
- Use of Gaussian noise through tables
- Use of double (floating) type

Source: https://www.kpqc.or.kr/competition.html

8. Concluding Remarks

RSA vs. SOLMAE

	RSA	SOLMAE	Cmt
Mathematics	Number Theory	Algebra over Cyclotomic Ring	
Trapdoor	e x d = 1 mod $\phi(n)$ Multiplicative inverse	B x A = 0 mod q over Z[x]/ψ(n) NTRU Trapdoor	
Computation	Modular exponentiation over n (= p x q)	Polynomial computation, FFT, NTT, etc.	
Comparison	Exact (=)	Bounded (<=)	
Random Sampling	Not necessary	Gaussian Sampling	
Security Problem	Integer factorization	Core SVP, u-SVP, CVP	
Digital/ Quantum Attack	N/Y	N/N	
Security Assumption	Worst-case No average-case	Average-case to Worst-case Reduction	

Pros and Cons of SOLMAE

- Pros
  - Modular Compactness
  - Simplicity and Efficiency: 2x faster than FALCON
  - Inherit all advantages of FALCON* except SCA**
    - ✓Small energy on FPGA[BKG22], Embedded 6.5K RAM (< 8k RAM)[GHK+21], TLS3.1[SKD20], DNSSEC[MdJvH+20][GS22]
  - Side Channel Resilience
  - Almost Achieved Trilemma(S+P+C)

- Cons
  - Reliance floating-point arithmetic
  - Algebraically structured security assumption
  - Need more formal proof

* P-A. Fouque et al., "FALCON : What's next?", NIST 4th PQC Standardization Conference, Nov. 29- Dec.1, 2022
** E. Karabulut and Aydin Aysu, "Breaking FALCON Post-Quantum Signature Scheme through Side-Channel Attacks, same as above.

SOLMAE Family (expected)

- ✓SOLMAE_ds : original SOLMAE/ SOLMAE_basic
- ✓eSOLMAE : enhanced SOLMAE against SCA
- ✓SOLMAE_km : KEM based on SOLMAE
- ✓SOLMAE_rg : Ring signature based on SOLMAE
- ✓SOLMAE_ag : Aggregated signature based on SOLMAE
- ✓SOLMAE_id : ID-based encryption based on SOLMAE
- ✓SOLMAE_bc : Block_Chain based on SOLMAE
- ✓SOLMAE_cc : Cryptocurrency based on SOLMAE
- ✓SOLMAE_pp : Privacy-Preserving based on SOLMAE
- ✓SOLMAE_fh : Fully-Homomorphic encryption based on SOLAME
- ✓and more

Typos in SOLMAE Specification

Location	Submitted	Corrected
lines 13 and 16 in **Algorithm 1**, p10	σ_{sig}	σ_{sig}^2
line 13 in **Algorithm 2**, p11	α^2/q	q/α^2
line 13 in **Algorithm 2**, p11 (2 places)	$\varphi(g)$	$\varphi_i(g)$
line 10 in **Algorithm 3**, p12	$\|(\hat{s_1}, \hat{s_2})\|$	$\|(FFT^{-1}(\hat{s_1}), FFT^{-1}(\hat{s_2}))\|$
line 7 in **Algorithm 4**, p13	σ_{sig}	σ_{sig}^2
line 8 from bottom in Def. of Σ_i, p13	σ_{sig}	σ_{sig}^2
line 2, p14	σ_{sig}	σ_{sig}^2
line 3 in **Algorithm 9**, p15	u_x, u_y	u_θ
line 5 in **Algorithm 9**, p15	$x \leftarrow \rho \cdot \cos(2\pi u_x)$	$x \leftarrow \rho \cdot \cos(\frac{\pi}{2} u_\theta)$
line 6 in **Algorithm 9**, p15	$y \leftarrow \rho \cdot \sin(2\pi u_y)$	$y \leftarrow \rho \cdot \sin(\frac{\pi}{2} u_\theta)$
unit of **sgn** size in Table 2, p18	kBytes	Bytes

Updated SOLMAE Spec. was posted at IRCS blog. Click here for details.

Further Challenges

Some challenges are left to do next:

- Implementation of intermediate NIST security level from II to IV (d=768,864,972)
- Implementation of compression and decompression to reduce the size of signature
- Optimized Implementation on various platform
- Backup documents to understand the underlying theory of SOLMAE, *etc.*

Appendix : Attacking Lattice

Lattice Basis Reduction

Good bases are helpful when solve problems on lattices.

E.g., Given an orthodox basis of a lattice, one can immediately compute the shortest vector.

Lattice Basis Reduction is a category of method of finding a "good" basis of the lattice given by a "bad" basis.

Famous basis reduction algorithms:

- **LLL Reduction**
- BKZ Reduction
- **DBKZ Reduction**
- HKZ Reduction
- **Slide Reduction**

Definition (LLL-reduced Basis)

Let $\mathbf{B} \in \mathbb{R}^{m \times n}$, we say $\mathbf{B}$ is ϵ-LLL-reduced, if it satisfies the following:

- Size Reduced: for all $i \neq j$, $|\mu_{i,j}| < \frac{1}{2}$
- Lovász's condition: For all $1 < i \leq n$, $\|\mathbf{b}_i^*\|^2 \leq (1+\epsilon)\|\mu_{i,i+1}\mathbf{b}_i^* + \mathbf{b}_{i+1}^*\|^2$.

Remark. We often set $\epsilon = \frac{1}{3}$.

Remark. We say $\mathbf{B}$ is a ϵ-LLL basis if it is ϵ-LLL-reduced.

We now introduce LLL-algorithm, that turns any lattice basis into a ϵ-LLL-reduced basis.

LLL Algorithm

Input : Lattice Basis $\mathbf{B} \in \mathbb{R}^{m \times n}$, real $\epsilon > 0$
Output: A ϵ-LLL bassis of $\mathcal{L}(\mathbf{B})$

Compute $\mathbf{b}_1^*, \cdots, \mathbf{b}_n^*$ **for** $i = 2$ **to** n **do**
  for $j = i - 1$ **to** *1* **do**
    $\mathbf{b}_i \leftarrow \mathbf{b}_i - c_{i,j}\mathbf{b}_j$ where
    $c_{i,j} = \lceil \langle \mathbf{b}_i, \mathbf{b}_j^* \rangle / \langle \mathbf{b}_j^*, \mathbf{b}_j^* \rangle \rfloor$
  end
end
if $\exists i$ *s.t.* $\|\mathbf{b}_i^*\|^2 > (1+\epsilon)\|\mu_{i,i+1}\mathbf{b}_i^* + \mathbf{b}_{i+1}^*\|$ **then**
  $\mathbf{b}_i \leftrightarrow \mathbf{b}_{i+1}$
  go to 1
end
return $\mathbf{b}_1, \cdots, \mathbf{b}_n$

Fact.

- For $\epsilon > 1/\text{poly}(n)$, the algorithm terminates in polynomial time
- $\|\mathbf{b}_n^*\|$ never decreases during the execution

LLL is just the beginning

Many more attacks

- Block Korkine-Zolotarev (BKZ)
  - Assumes we can solve SVP exactly in small dimension m.
  - Projects m vectors to smaller space, solves SVP there, lifts back.
  - Chains these in a way and interleaves with LLL to obtain short basis.
  - Quality depends heavily on m.
- Enumeration algorithms
  - Search for absolutely shortest, with some smart ideas.
  - Finds shortest vector.
  - Can balance time and quality of basis by stopping early/pruning.
- Sieving algorithms
  - Asymptotically faster than enumeration; better than BKZ.
  - Needs more space.
  - No guarantee that short vector found is shortest.
  - Balances time and quality of basis.

We cover enumeration. For sieving see slides 69 onwards of `http://thijs.com/docs/lec2.pdf` by Thijs Laarhoven.

(D)SVP Reduction

Define the following reduced basis.

- δ-SVP-reduced: A basis $\mathbf{B}$ is δ-SVP-reduced if $\|\mathbf{b}_1\| \leq \delta \cdot \lambda_1(\mathbf{B})$.
- δ-DSVP-Reduced: A basis $\mathbf{B}$ is δ-DSVP-reduced if $\mathbf{B}^{-s}$ is δ-SVP-reduced and $\mathbf{B}$ is $\frac{1}{3}$-LLL-reduced.

Given the access to δ-SVP oracle, δ-(D)SVP-reduce can be done efficiently:

- δ-SVP-reduce $\mathbf{B}$: Call δ-SVP oracle to get $\mathbf{z} \in \mathcal{L}(\mathbf{B})$ (s.t. $\|\mathbf{z}\| \leq \delta \cdot \lambda_1(\mathbf{B})$) and "substitute" the first vector of $\mathbf{B}$ with $\mathbf{z}$.
- δ-DSVP-Reduce $\mathbf{B}$: Work out $\mathbf{B}^{-s}$, and do δ-SVP-reduce on $\mathbf{B}^{-s}$. Then workout the new $\mathbf{B}$ with reduced $\mathbf{B}^{-s}$ and do $\frac{1}{3}$-LLL-reduce on the new $\mathbf{B}$. This procedure works fine since $\|\mathbf{b}_n^*\|$ never decreases during the LLL-reduction.

Remark. $\mathbf{B}$ is 1/3-LLL-reduced implies $\|\mathbf{b}_i\| \leq 4\|\mathbf{b}_{i+1}\|$.

DBKZ Algorithm(1/2)

The Self-Dual BKZ (DBKZ) Algorithm [MW16] proposed by Daniele Micciancio and Michael Walter is a algorithm that HSVP-reduce a lattice basis with given SVP-oracle of low dimension.

In the algorithm, N is set to

$$N := \lceil (2n^2/(k-1)^2) \cdot \log(n \log(5\|\mathbf{B}\|)/\epsilon) \rceil$$

for some $\epsilon \in [2^{-\text{poly}(n)}, 1]$.

Input: Lattice Basis $\mathbf{B} \in \mathbb{R}^{m \times n}$, real $\epsilon > 0$
Result: A new basis of $\mathcal{L}(\mathbf{B})$

for $\ell = 1$ **to** N **do**
  for $i = 1$ **to** $n - k$ **do**
    δ-SVP-reduce $\mathbf{B}_{[i,i+k-1]}$
  end
  for $j = n - k + 1$ **to** *1* **do**
    δ-DSVP-reduce $\mathbf{B}_{[j,j+k-1]}$
  end
end
δ-SVP-reduce $\mathbf{B}[1, k]$
return B

Borrowed from W. Liu presentation

DBKZ Algorithm(2/2)

Theorem

For approximation factor $1 \leq \delta leq 2^k$ *and an input basis* $\mathbf{B}_0$ *of* $\mathcal{L}$ *algorithm 2 outputs a basis* $\mathbf{B}$ *of* $\mathcal{L}$ *in polynomial time s.t.*

$$\|\mathbf{b}_1\| \leq (1+\epsilon)(\delta^2\gamma_k)^{\frac{n-1}{2(k-1)}} \mathrm{vol}(\mathcal{L})^{1/n}$$

by making $N \cdot (2n-2k+1)+1$ *calls to* δ*-SVP oracle for lattices with rank* k*.*

Proof. See [MW16].

Borrowed from W. Liu presentation