



Pengenalan *POST-QUANTUM CRYPTOGRAPHY* Persiapan dalam rangka migrasi ke *POST-QUANTUM CRYPTOGRAPHY* Versi 1.0

Deputi Bidang Strategi dan Kebijakan Keamanan Siber dan Sandi
Badan Siber dan Sandi Negara

2025

Pengenalan *POST-QUANTUM CRYPTOGRAPHY*

PERSIAPAN DALAM RANGKA MIGRASI KE *POST-QUANTUM CRYPTOGRAPHY* VERSI 1.0



Direktorat Kebijakan Teknologi Keamanan Siber dan Sandi
Badan Siber dan Sandi Negara (BSSN)
Jl. Cijayanti No. 70, Kec. Babakan Madang, Kabupaten Bogor
Website : <https://www.bssn.go.id>
Email : humas@bssn.go.id

TIM PERUMUS

Pengarah : Tjahjo Khurniawan

Penanggung Jawab : Soetedjo Joewono

Pereviu : Uha Isnaini, S.Si, M.Sc., Ph.D.
(Universitas Gadjah Mada)

Tim Penyusun :

1. Sari Agustini Hafman
2. Ratih Mumpuni Arti
3. Novita Angraini
4. Oky Jati Permana
5. Mohammad Rahmadi
6. Siti Hikmatul Hasanah
7. Dian Yuda Paripurna
8. Yhufi Swastantri Gustiviana

Penyunting : PROFESIONAL • ELABORATIF • RESPONSIF
INTEGRATIF • SOLID • AKUNTABEL • INOVATIF

1. Ratih Mumpuni Arti
2. Oky Jati Permana



PENGANTAR

Puji syukur kita panjatkan ke hadirat Tuhan Yang Maha Esa karena dokumen *Pengenalan Post-Quantum Cryptography – Persiapan dalam Rangka Migrasi ke Post-Quantum Cryptography Versi 1.0* dapat diselesaikan sebagai bagian dari agenda strategis nasional di bidang keamanan siber dan sandi. Penyusunan dokumen ini merupakan langkah awal untuk memastikan bahwa Indonesia memiliki arah kebijakan yang jelas dan terukur dalam menghadapi implikasi teknologi komputasi kuantum terhadap ketahanan sistem kriptografi nasional.

Perkembangan kemampuan komputasi kuantum berpotensi mengurangi tingkat keamanan algoritma kriptografi klasik yang saat ini digunakan secara luas pada berbagai layanan digital dan Infrastruktur Informasi Vital. Kondisi ini menuntut adanya kesiapan regulasi, standardisasi, dan tata kelola yang memungkinkan proses transisi menuju *Post-Quantum Cryptography* dilakukan secara sistematis, berbasis risiko, dan selaras dengan perkembangan standar internasional.

Dokumen ini disusun untuk memberikan kerangka konseptual yang diperlukan dalam memahami urgensi migrasi ke *Post-Quantum Cryptography*, serta memberikan gambaran awal mengenai aspek teknis dan kebijakan yang perlu dipertimbangkan dalam pengembangan strategi migrasi nasional. Informasi yang disajikan diharapkan dapat menjadi acuan awal bagi kementerian/lembaga, sektor industri, penyelenggara Infrastruktur Informasi Vital, serta pemangku kepentingan lain dalam menilai kesiapan organisasi dan merencanakan langkah-langkah adaptasi yang diperlukan.



Kami menyampaikan apresiasi kepada seluruh tim penyusun, para pakar, serta pemangku kepentingan yang telah berkontribusi dalam penyelesaian dokumen ini. Semoga dokumen ini menjadi pijakan penting dalam upaya membangun ekosistem kriptografi nasional yang adaptif, berstandar internasional, dan resilien terhadap ancaman di era pasca-kuantum.

Drs. Nugroho Sulistyo Budi, M.M., M.Han.

Kepala Badan Siber dan Sandi Negara)

Desember 2025



PENGANTAR DEPUTI I



Puji syukur kita panjatkan ke hadirat Tuhan Yang Maha Esa atas limpahan rahmat dan karunia-Nya, sehingga dokumen yang berjudul *Pengenalan Post-Quantum Cryptography – Persiapan dalam Rangka Migrasi ke Post-Quantum Cryptography Versi 1.0* dapat diselesaikan dengan baik. Dokumen ini disusun sebagai bagian dari upaya strategis dalam memperkuat kebijakan dan kesiapan nasional menghadapi era komputasi kuantum yang semakin dekat. Di tengah perkembangan teknologi yang begitu pesat, ancaman terhadap algoritma kriptografi klasik seperti RSA dan ECC menjadi semakin nyata, sehingga diperlukan langkah antisipatif untuk memastikan keberlanjutan perlindungan data dan komunikasi nasional.

Dokumen ini bertujuan untuk memberikan pengenalan yang komprehensif tentang *Post-Quantum Cryptography* agar para pemangku kepentingan, baik pemerintah, industri, akademisi, maupun penyelenggara Infrastruktur Informasi Vital, dapat mulai mempersiapkan organisasinya melakukan migrasi dari kriptografi klasik menuju *Post-Quantum Cryptography* secara terencana dan berstandar internasional. Dokumen ini mengulas perjalanan perkembangan kriptografi dari masa klasik hingga memasuki era kuantum, menjelaskan ancaman yang ditimbulkan oleh komputer kuantum, serta membahas berbagai algoritma *Post-Quantum Cryptography*, proses standardisasi global, dan sekilas mengenai strategi migrasi menuju sistem kriptografi pasca-kuantum. Kami berharap dokumen ini dapat menjadi rujukan awal yang bermanfaat bagi seluruh pihak yang terlibat dalam pengembangan, penerapan, dan kebijakan keamanan siber nasional.

Terima kasih dan apresiasi kami sampaikan kepada seluruh pakar, akademisi, dan tim penyusun yang telah memberikan kontribusi, masukan, serta dedikasi luar biasa sehingga dokumen ini dapat tersusun secara

komprehensif dan relevan. Semoga dokumen ini dapat menjadi pijakan awal bagi Indonesia dalam membangun ketahanan dan kemandirian kriptografi nasional, sekaligus memperkuat ekosistem keamanan siber yang adaptif dan berdaulat di era pasca-kuantum.

Tjahjo Khurniawan

Deputi Bidang Strategi dan Kebijakan Keamanan Siber dan Sandi

Desember 2025



RINGKASAN EKSEKUTIF

Dokumen Pengenalan *Post-Quantum Cryptography* ini disusun oleh Badan Siber dan Sandi Negara (BSSN) sebagai langkah strategis dalam memperkuat ketahanan dan keamanan siber nasional terhadap ancaman yang ditimbulkan oleh kemajuan komputasi kuantum. Seiring dengan meningkatnya kemampuan komputer kuantum, kebutuhan akan sistem kriptografi baru yang tahan terhadap serangan kuantum menjadi semakin mendesak. Oleh karena itu, dokumen ini memperkenalkan *Post-Quantum Cryptography* (PQC) sebagai cabang kriptografi yang dirancang untuk tetap aman terhadap serangan berbasis komputasi kuantum.

Secara keseluruhan, dokumen ini berfungsi sebagai rujukan untuk penyusunan kebijakan dan praktik keamanan informasi dengan kriptografi di Indonesia, memastikan perlindungan data dan komunikasi tetap aman di era pasca-kuantum, serta mendukung terwujudnya kemandirian dan kedaulatan teknologi kriptografi nasional. Dokumen ini memberikan pemahaman komprehensif mengenai konsep dasar PQC, jenis-jenis algoritma PQC, perkembangan standarisasi global yang dikeluarkan termasuk oleh NIST, ISO, dan ETSI, serta pengantar tentang migrasi dari sistem kriptografi klasik ke PQC, guna mendukung kesiapan Indonesia dalam melakukan transisi menuju era pasca-kuantum.

Dokumen ini ditujukan bagi para pemangku kepentingan keamanan siber dan sandi, meliputi pengambil kebijakan, perumus kebijakan teknis, profesional/praktisi keamanan siber, penyedia layanan keamanan TI, akademisi, dan peneliti. Melalui dokumen ini, diharapkan seluruh pihak terkait dapat memahami prinsip, standar, dan tahapan transisi menuju PQC dengan tepat, sehingga migrasi dapat dilakukan secara terencana, terukur, dan sejalan dengan standar keamanan internasional.

Apabila pengguna menemukan keraguan dalam dokumen ini, maka disarankan untuk melihat referensi asli yang terdapat pada daftar referensi.

DAFTAR ISI

PENGANTAR	iii
PENGANTAR DEPUTI I	v
RINGKASAN EKSEKUTIF	vii
DAFTAR ISI.....	viii
DAFTAR TABEL.....	ix
DAFTAR GAMBAR	x
BAB I PENDAHULUAN	1
A. Ruang Lingkup	2
B. Target Audiens	3
C. Sistematika Penyusunan Dokumen	4
BAB II PENGANTAR DAN KONSEP DASAR.....	6
A. Algoritma Kriptografi Klasik	7
B. Ancaman Komputasi Kuantum terhadap Kriptografi Klasik	20
C. Prinsip-Prinsip Dasar PQC	24
D. Dasar Matematika PQC	30
BAB III ALGORITMA <i>POST-QUANTUM CRYPTOGRAPHY</i>	40
A. Kriptografi Berbasis Latis (<i>Lattice-based cryptography</i>)	41
B. Kriptografi Berbasis Kode (<i>Code-based cryptography</i>).....	129
C. Kriptografi Kuadratik Multivariabel (<i>Multivariate-quadratic cryptography</i>).....	144
D. Kriptografi Berbasis Hash (<i>Hash-based cryptography</i>)	157
E. Kriptografi berbasis isogeni supersingular (<i>Supersingular Isogeny-based cryptography</i>).....	216
BAB IV STANDARDISASI <i>POST-QUANTUM CRYPTOGRAPHY</i>	240
A. NIST	242
B. ISO	248
C. ETSI	251
D. Amandemen <i>Executive Order</i> 14144 oleh Presiden USA.....	270
BAB V MIGRASI KE <i>POST-QUANTUM CRYPTOGRAPHY</i>	275
A. NIST SP 1800-38A, NIST SP 1800-38B dan NIST SP 1800C.....	276
B. ENISA PQC <i>INTEGRATION STUDY</i>	286
C. BSI JERMAN.....	289
D. AIVD – TNO – CWI Belanda.....	293
BAB VI PENUTUP.....	300
REFERENSI.....	302

DAFTAR TABEL

Tabel 3.1	Set parameter untuk skema enkapsulasi kunci berbasis LWE ..	58
Tabel 3.2	Set parameter untuk skema ML-KEM yang disetujui	69
Tabel 3.3	Ukuran (dalam byte) kunci dan ciphertext ML-KEM.....	69
Tabel 3.4	Rata rata kegagalan dekapsulasi pada ML-KEM	71
Tabel 3.5	Set parameter pada ML-DSA.....	117
Tabel 3.6	Fungsi Member untuk Addresses	172
Tabel 3.7	SLH-DSA parameter sets	214
Tabel 4.1	Daftar Dokumen terkait QSC dari ETSI	253
Tabel 4.2	Resume Standardisasi Algoritma PQC	273



DAFTAR GAMBAR

Gambar 3.1	Latis 2 dimensi.....	42
Gambar 3.2	Perbedaan struktural antara varian LWE, Ring-LWE dan Module-LWE.....	57
Gambar 3.3	Konsep sederhana Key establishment menggunakan KEM ..	61
Gambar 3.4	Komponen dari codeword c	131
Gambar 3.5	Alur kerja umum skema multivariat	147
Gambar 3.6	Tanda Tangan SLH-DSA.....	162
Gambar 3.7	Kunci publik SLH-DSA.....	163
Gambar 3.8	Kunci privat SLH-DSA.....	163
Gambar 3.9	Alamat (ADRS).....	168
Gambar 3.10	Alamat kompresi kunci publik WOTS+.....	169
Gambar 3.11	Alamat hash WOTS+	169
Gambar 3.12	Hash tree address	170
Gambar 3.13	Alamat FORS tree.....	170
Gambar 3.14	Alamat FORS tree roots compression	170
Gambar 3.15	Alamat pembangkitan kunci WOTS+.....	171
Gambar 3.16	Alamat pembangkitan kunci FORS	171
Gambar 3.17	Format data tanda tangan WOTS+	179
Gambar 3.18	Format data tanda tangan XMSS.....	182
Gambar 3.19	Merkle Hash Tree	185
Gambar 3.20	Format data Tanda tangan Hypertree	188
Gambar 3.21	Format data Tanda tangan FORS.....	192
Gambar 3.22	SLH-DSA Kunci Privat.....	199
Gambar 3.23	SLH-DSA Kunci Publik.....	200
Gambar 3.24	Format data Tanda tangan SLH-DSA.....	201
Gambar 3.25	Tiga strategi komputasi berukuran $el - 1 = 6$	232
Gambar 3.26	Representasi linear strategi pada simpul 6.	233

BAB I

PENDAHULUAN



BAB I

PENDAHULUAN

A. Ruang Lingkup

Di era digital yang terus berkembang, keamanan informasi menjadi salah satu pilar utama dalam mendukung berbagai aspek kehidupan modern. Dari transaksi keuangan hingga komunikasi pribadi, enkripsi memainkan peran penting dalam melindungi data dari akses yang tidak sah. Namun, dengan kemajuan pesat dalam komputasi kuantum, ancaman terhadap protokol kriptografi yang ada saat ini semakin nyata.

Komputasi kuantum, yang memanfaatkan prinsip-prinsip mekanika kuantum, memiliki potensi untuk memecahkan banyak algoritma kriptografi klasik yang digunakan secara luas, seperti *Rivest-Shamir-Adleman* (RSA), *Elliptic Curve Cryptography* (ECC), dan *Digital Signature Algorithm* (DSA). Kemampuan ini menimbulkan tantangan besar bagi sistem keamanan yang diandalkan saat ini. Untuk menjawab tantangan tersebut, dunia akademis dan industri teknologi telah mengalihkan perhatian pada bidang baru yang dikenal sebagai *Post-Quantum Cryptography* (PQC).

PQC adalah cabang kriptografi yang dirancang untuk tahan terhadap serangan dari komputer kuantum. Algoritma-algoritma PQC dirancang sedemikian rupa agar tetap aman dari serangan komputasi kuantum. Dokumen ini bertujuan untuk memberikan pengenalan yang komprehensif tentang PQC sehingga para pihak berkepentingan dapat mulai mempersiapkan organisasinya untuk bermigrasi dari kriptografi klasik ke PQC.

Dokumen ini memberikan penjelasan tentang:

1. Pengantar dan Konsep Dasar

Sebuah pengantar singkat tentang bagaimana cara kerja algoritma kriptografi klasik, bagaimana komputer kuantum bekerja dan mengapa mereka menjadi ancaman bagi kriptografi klasik, prinsip dasar dan landasan matematika PQC.

2. Algoritma PQC

Algoritma PQC dikelompokkan ke dalam beberapa kategori utama berdasarkan dasar matematis dan pendekatan kriptografinya, yaitu kriptografi berbasis latis, kriptografi berbasis kode, kriptografi kunci publik multivariat, kriptografi berbasis *hash*, dan kriptografi berbasis isogeni supersingular.

3. Standardisasi PQC

Tinjauan tentang upaya global, termasuk proses standardisasi PQC oleh *National Institute of Standards and Technology* (NIST), *International Organization for Standardization* (ISO), dan *European Telecommunications Standards Institute* (ETSI).

4. Migrasi ke PQC

Migrasi ke PQC merupakan proses peralihan dari sistem kriptografi klasik ke algoritma PQC yang tahan terhadap serangan oleh komputer kuantum. NIST telah mempublikasikan 3 (tiga) dokumen untuk panduan migrasi ini, yaitu NIST SP 1800-38A, NIST SP 1800-38B dan NIST SP 1800C. Selain NIST, beberapa organisasi yang telah mempublikasikan panduan migrasi ke PQC antara lain ENISA, *Bundesamt für Sicherheit in der Informationstechnik* (BSI) Jerman, dan *Algemene Inlichtingen- en Veiligheidsdienst* (AIVD) – *Nederlandse Organisatie voor Toegepast Natuurwetenschappelijk Onderzoek* (TNO) – *Centrum Wiskunde & Informatica* (CWI) Belanda.

B. Target Audiens

Dokumen ini ditujukan bagi para penyelenggara sistem elektronik, baik publik maupun privat, yang meliputi:

1. Pengambil kebijakan keamanan siber dan sandi yang memiliki pemahaman dasar tentang konsep kriptografi dan ingin memastikan transisi ke PQC dalam organisasi berjalan sesuai standar keamanan nasional dan internasional.
2. Perumus kebijakan keamanan siber dan sandi dengan latar belakang teknis atau bekerja sama dengan pakar teknis dalam penyusunan kebijakan yang selaras dengan perkembangan PQC.

3. Profesional/praktisi keamanan siber dan sandi yang memiliki pemahaman tentang kriptografi klasik serta dasar matematika terkait (misalnya, aljabar linier, teori bilangan, dan probabilitas) untuk mengimplementasikan dan mengevaluasi solusi PQC dalam sistem elektronik.
4. Penyedia layanan/produk keamanan TI yang memiliki kapabilitas teknis dalam mengembangkan dan mengadaptasi algoritma PQC ke dalam layanan atau produk keamanan sesuai standar internasional.
5. Para pihak yang terlibat dalam perumusan kebijakan keamanan siber dan sandi agar kebijakan yang dirumuskan selaras dengan perkembangan teknologi PQC.
6. Mahasiswa, akademisi, peneliti, dan pihak lain yang membutuhkan rujukan untuk menyusun kajian ilmiah, artikel, penelitian, maupun karya tulis terkait keamanan siber dan kriptografi.

C. Sistematika Penyusunan Dokumen

Dokumen ini terdiri atas 6 (enam) Bab, yang terdiri dari:

BAB I. PENDAHULUAN

Bab ini berisi tentang tujuan penyusunan dokumen, target audiens, rujukan dan sistematika penyusunan dokumen.

BAB II. PENGANTAR DAN KONSEP DASAR

Bab ini berisi tentang algoritma kriptografi klasik, ancaman komputasi kuantum, prinsip – prinsip PQC dan landasan matematika PQC.

BAB III. ALGORITMA *POST-QUANTUM CRYPTOGRAPHY*

Bab ini berisi tentang beberapa basis algoritma PQC dan contoh algoritma.

BAB IV. STANDARDISASI *POST-QUANTUM CRYPTOGRAPHY*

Bab ini berisi tentang beberapa standar PQC yang dikeluarkan NIST, ISO dan ENISA.

BAB V. MIGRASI KE *POST-QUANTUM CRYPTOGRAPHY*

Bab ini berisi tentang panduan migrasi ke PQC yang merujuk pada standar dari NIST, ENISA, BSI Jerman, dan AIVD-TNO-CWI Belanda.

BAB VI. PENUTUP

Bab ini berisi tentang penutup dan harapan dari penyusunan dokumen.



BAB II

PENGANTAR DAN KONSEP DASAR

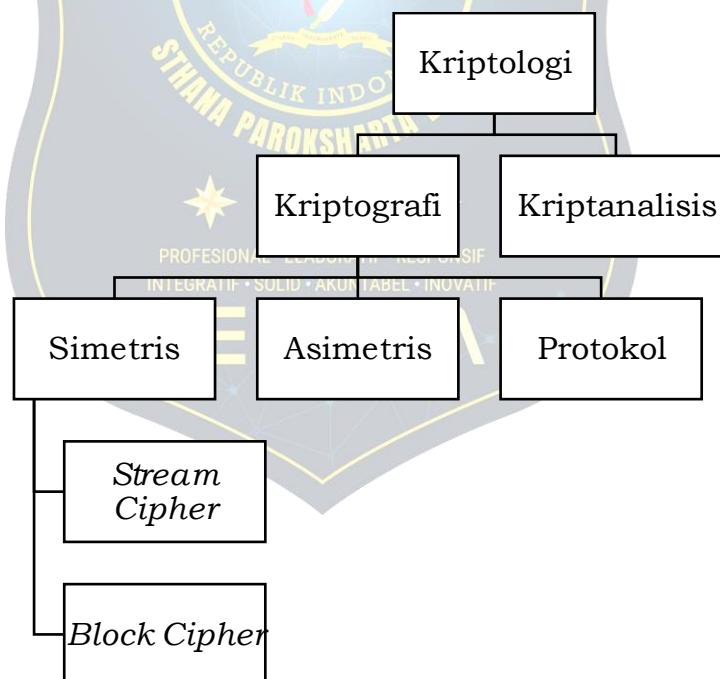


BAB II PENGANTAR DAN KONSEP DASAR

A. Algoritma Kriptografi Klasik

Kriptografi adalah ilmu tentang cara mengamankan komunikasi dari pihak yang tidak diinginkan [1]. Kriptografi mempelajari teknik matematika yang terkait dengan aspek keamanan informasi seperti konfidensialitas, integritas data, autentikasi entitas, dan autentikasi asal data [1]. Secara historis, tujuan utama kriptografi adalah untuk menyembunyikan makna suatu pesan. Seiring dengan perkembangan dunia digital, kriptografi menjadi alat yang sangat vital untuk melindungi data pribadi dan informasi sensitif.

Untuk dapat memahami konsep kriptografi, C. Paar et.al menjelaskannya melalui gambar berikut:



Gambar 2. 1 Pembagian bidang pada Kriptologi
digambar ulang dari [1].

Kriptografi merupakan bagian dari kriptologi. Kriptologi terbagi menjadi dua cabang yaitu kriptografi dan kriptanalisis. Kriptanalisis merupakan ilmu dan juga seni dalam memecahkan algoritma

kriptografi. Kriptanalisis menjadi hal yang sangat penting dalam kriptografi, karena tanpa adanya pihak yang mencoba memecahkan algoritma kriptografi, maka tak akan dapat diketahui apakah algoritma tersebut benar-benar aman atau tidak.

Kriptografi merupakan metode pengamanan informasi dengan cara mengenkripsi data menggunakan suatu algoritma dan kunci tertentu. Kriptografi sendiri dapat dibagi menjadi tiga bagian utama, yaitu kriptografi simetris, kriptografi asimetris dan protokol kriptografi. Ketiga bagian tersebut yang selanjutnya disebut sebagai kriptografi klasik memainkan peran penting dalam memastikan keamanan komunikasi, privasi, dan integritas data, baik di dunia maya maupun di kehidupan sehari-hari.

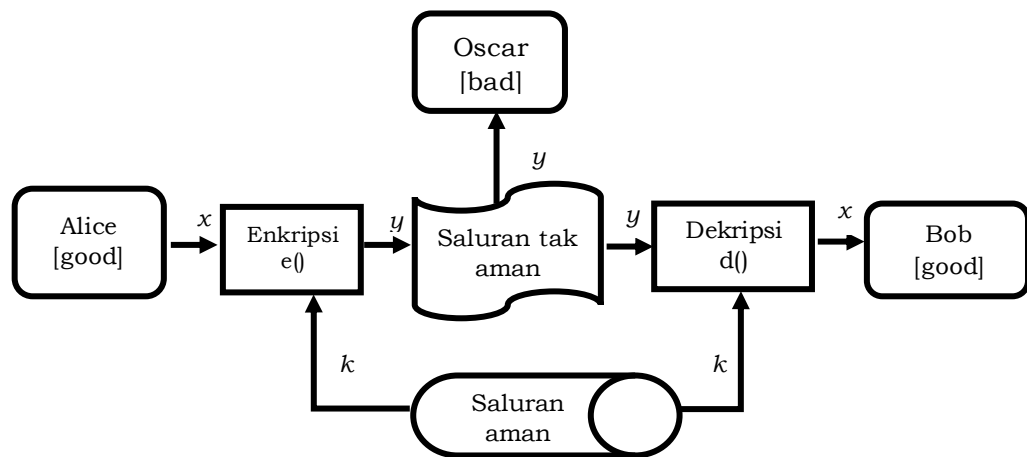
1. Kriptografi Simetris

Kriptografi simetris adalah jenis algoritma kriptografi di mana kunci yang sama digunakan untuk proses enkripsi dan dekripsi. Dalam skema ini, baik pengirim maupun penerima harus memiliki kunci yang sama dan menjaga kerahasiaan kunci tersebut. Keamanan kriptografi simetris bergantung pada kerahasiaan kunci dan panjang kunci yang digunakan.

Kriptografi simetris secara umum memiliki 2 ciri utama, yaitu:

- a. Proses enkripsi dan dekripsi menggunakan kunci yang sama;
- b. Fungsi untuk melakukan enkripsi dan dekripsi juga sama, bahkan identik.

Karena keamanan kriptografi simetris terletak pada kunci, maka proses distribusi kunci harus bersifat rahasia dan dijamin keamanannya. Siapapun yang memiliki kunci maka akan dapat membuka pesan yang dikomunikasikan. Oleh karena itu, selama komunikasi bersifat rahasia maka kunci juga harus tetap dirahasiakan. Konsep kriptografi simetris terdapat pada gambar 2.2.



Gambar 2. 2 Kriptografi simetris digambar ulang dari [1].

di mana :

- x adalah teks terang (*plaintext*)
- y adalah teks sandi (*ciphertext*)
- k adalah kunci

Kriptografi simetris dapat dijelaskan sebagai berikut : terdapat 2 orang, Alice dan Bob, yang ingin berkomunikasi melalui saluran komunikasi yang tidak aman. Masalah terjadi ketika ada Oscar sebagai orang jahat yang juga memiliki akses dengan cara meretas saluran komunikasi. Alice mengenkripsi pesan (*plaintext*) x menggunakan algoritma simetris, menghasilkan *ciphertext* y . Bob menerima *ciphertext* dan mendekripsi pesan tersebut. Dekripsi merupakan proses invers dari enkripsi. Apabila algoritma enkripsinya kuat, maka *ciphertext* akan terlihat seperti bit yang acak sehingga Oscar tidak akan dengan mudah mendapatkan informasi yang berguna dari teks tersebut.

Kriptografi simetris kemudian dibedakan menjadi 2 kelas yaitu, *stream cipher* dan *block cipher*. *Stream cipher* mengenkripsi data bit per bit atau *byte per byte* secara berurutan sedangkan *block cipher* mengenkripsi blok data sekaligus.

a. *Stream cipher*

Stream cipher adalah algoritma enkripsi simetris yang beroperasi pada *plaintext* secara kontinu dan menghasilkan *ciphertext* satu bit atau *byte* pada satu waktu secara berurutan. Setiap bit atau *byte plaintext* langsung dienkripsi menggunakan operasi berbasis *keystream*. *Keystream* yang digunakan untuk enkripsi memiliki sifat berubah dari waktu ke waktu, dipengaruhi oleh kunci rahasia dan terkadang nilai inisialisasi (*Initialization Vector – IV*).

Enkripsi dan Dekripsi *Stream Cipher*

Jika x_i adalah bit ke- i dari *plaintext*, y_i adalah *ciphertext* bit ke- i dan s_i adalah bit ke- i dari *keystream*, maka:

$$\text{Enkripsi : } y_i = x_i \oplus s_i \text{ mod } 2$$

$$\text{Dekripsi : } x_i = y_i \oplus s_i \text{ mod } 2$$

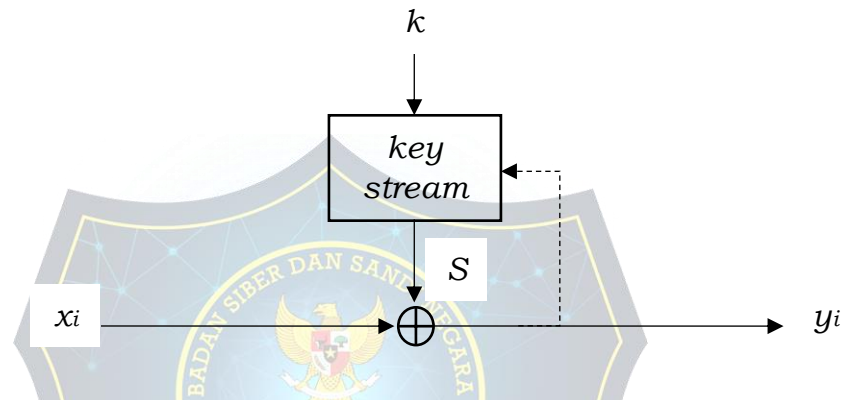
enkripsi dan dekripsi dapat dilihat pada gambar 2.3



Gambar 2. 3 Enkripsi dan Dekripsi pada *Stream Cipher*
digambar ulang dari [1].

Stream cipher dibedakan menjadi 2 (dua) jenis berdasarkan *keystream*, yaitu *synchronous stream cipher* dan *asynchronous stream cipher*. Pada *synchronous stream cipher*, *keystream* dihasilkan hanya berdasarkan kunci dan IV. Karena *keystream* dapat dihasilkan tanpa menunggu input, maka *synchronous stream cipher* efisien dalam hal kecepatan. Selain itu, jika terjadi kesalahan pada bit *ciphertext*, hanya bit itu saja yang rusak dalam dekripsinya. Namun kelemahannya adalah *keystream* tidak boleh berulang, karena akan menjadi

kerawanan yang akan digunakan oleh penyerang untuk menganalisis dan menghasilkan *plaintext*. Kelemahan lainnya adalah jika ada bit hilang atau rusak dalam transmisi, *keystream* dan *ciphertext* akan keluar sinkronisasi sehingga dekripsi menjadi tidak valid. Contoh algoritma *synchronous stream cipher* adalah ChaCha20.



Gambar 2. 4 *Synchronous dan Asynchronous Stream Cipher* digambar ulang dari [1].

Pada *asynchronous stream cipher*, *keystream* bergantung pada kunci dan *ciphertext* sebelumnya seperti digambarkan dengan garis putus-putus pada gambar 2.4. Dengan sistem kerja seperti itu, meskipun ada kesalahan bit atau kehilangan data, sistem dapat menyesuaikan kembali sinkronisasinya setelah menerima sejumlah bit *ciphertext* yang cukup. Keunggulan lainnya adalah lebih aman terhadap serangan karena *keystream* bergantung pada *ciphertext* sebelumnya, penyerang tidak bisa mengubah bit tertentu tanpa mempengaruhi *keystream* selanjutnya.

Asynchronous stream cipher juga memiliki beberapa kelemahan. Kelemahan pertama adalah *plaintext* atau *ciphertext* tidak dapat dienkripsi atau didekripsi secara paralel karena setiap bit *keystream* bergantung pada *plaintext* atau *ciphertext* sebelumnya, sehingga pemrosesan harus dilakukan secara berurutan. Kelemahan lainnya adalah jika satu bit

ciphertext rusak maka akan berdampak ke beberapa bit *plaintext* berikutnya sebelum teks kembali sinkron. Contoh algoritma *Asynchronous stream cipher* adalah AES-CFB dan AES-OFB.

b. *Block Cipher*

Block Cipher merupakan skema enkripsi yang memecah pesan *plaintext* yang akan dikirimkan menjadi sekelompok bit (disebut blok data) [1], bukan bit per bit seperti pada *stream cipher*. Untuk meningkatkan keamanannya, struktur *block cipher* biasanya terdiri dari beberapa putaran (*round*). Dalam praktiknya, sebagian besar *block cipher* memiliki panjang blok 128 bit (16 byte) seperti *Advanced Encryption Standard* (AES), atau panjang blok 64 bit (8 byte) seperti *Data Encryption Standard* (DES) atau *triple* DES (3DES).

Enkripsi dan Dekripsi *Block Cipher*

Secara formal, *block cipher* adalah fungsi yang mengambil *plaintext* x dengan panjang tetap dan mengubahnya menjadi *ciphertext* y dengan menggunakan kunci rahasia k maka :

$$E_k: \{0,1\}^n \rightarrow \{0,1\}^n$$

dengan :

$E_k = y$ adalah fungsi enkripsi dengan kunci k

$D_k = x$ adalah fungsi dekripsi, dengan sifat $D_k(E_k(x)) = x$

Artinya, apabila mengenkripsi *plaintext* x dengan kunci k , lalu mendekripsinya, maka harus didapatkan kembali nilai x .

Dua struktur umum yang digunakan dalam *block cipher* adalah:

1) *Feistel Network*

Feistel Network adalah teknik yang membagi blok *plaintext* menjadi dua bagian, L (kiri) dan R (kanan), lalu melakukan beberapa *round* transformasi menggunakan fungsi F .

$$L_{i+1} = R_i$$

$$R_{i+1} = L_i \oplus F(R_i, K_i)$$

Dimana: $F(R_i, K_i)$ adalah fungsi non linear yang tergantung pada subkunci *round* K_i

Pada model ini, dekripsi dapat dilakukan dengan langkah yang sama seperti enkripsi, hanya dengan membalik urutan subkunci *round*. Contoh algoritma *block cipher* yang menggunakan model ini adalah DES.

2) *Substitution – Permutation Network* (SPN)

SPN bekerja dengan mengganti dan merotasi bit melalui serangkaian langkah substitusi dan permutasi. Substitusi, melalui S-box, mengganti kelompok bit dengan bit lain berdasarkan tabel substitusi yang tetap. Dan permutasi, melalui P-box, mengacak posisi bit dalam blok untuk meningkatkan difusi. Proses ini dilakukan dalam beberapa *round* sehingga menghasilkan efek penyebaran (*avalanche effect*) dimana perubahan kecil pada input menyebabkan perubahan besar pada *output*.

Block cipher memproses pesan dengan blok berukuran tetap. Jika pesan yang diproses memiliki ukuran lebih panjang dari ukuran blok atau tidak berukuran kelipatan blok, maka digunakan mode operasi untuk memproses pesan tersebut. Beberapa mode operasi yang umum digunakan adalah:

- 1) *Electronic Code Book mode* (ECB), merupakan mode operasi yang memproses setiap blok *plaintext* secara independen. Kelemahan mode operasi ECB adalah tidak aman untuk data berulang.
- 2) *Cipher Block Chaining mode* (CBC), merupakan mode operasi yang memproses blok pesan dengan cara meng-XOR-kan setiap blok n dengan blok $n-1$ sebelum dilakukan enkripsi pada blok n tersebut.

- 3) *Cipher Feedback mode* (CFB), merupakan mode operasi yang menggunakan *output ciphertext* sebagai input berikutnya.
- 4) *Output Feedback mode* (OFB), adalah mode operasi yang menghasilkan *keystream* independen dari *plaintext* dan dapat menghindari propagasi kesalahan.
- 5) *Counter mode* (CTR), menggunakan penghitung atau *counter* yang dienkripsi untuk membuat *keystream*.
- 6) *XEX Tweakable Block Cipher with Ciphertext Stealing* (XTS) dikembangkan sebagai perbaikan dari ECB dan CBC, yang memiliki kelemahan dalam enkripsi data berbasis blok yang berulang. Mode ini digunakan untuk enkripsi *disk* dan media penyimpanan.

Keamanan *block cipher* dapat dibuktikan dengan memenuhi kriteria *Avalanche effect*, yaitu perubahan kecil pada input menghasilkan perubahan besar pada *output* dan memiliki *Key Sensitivity*, yaitu kunci yang berbeda harus menghasilkan *ciphertext* yang berbeda secara signifikan.

Kriptografi simetris umumnya lebih cepat dibandingkan dengan kriptografi asimetris, yang membuatnya lebih cocok untuk penggunaan skala besar, seperti enkripsi data dalam jaringan atau penyimpanan data. Karena proses enkripsi dan dekripsi hanya melibatkan algoritma matematis yang sederhana, kriptografi simetris lebih hemat sumber daya, menjadikannya pilihan utama dalam banyak aplikasi kriptografi.

Disamping keunggulan tersebut, kriptografi simetris juga memiliki beberapa kekurangan. Kekurangan tersebut antara lain:

a. Masalah distribusi kunci

Kriptografi simetris menggunakan kunci yang sama untuk enkripsi dan dekripsi, sehingga apabila penyerang berhasil mencuri kunci, maka seluruh komunikasi akan terbobol. Manajemen kunci pada kriptografi simetris menjadi sulit

karena setiap pasangan pengguna harus berbagi kunci melalui saluran yang aman.

b. Jumlah kunci

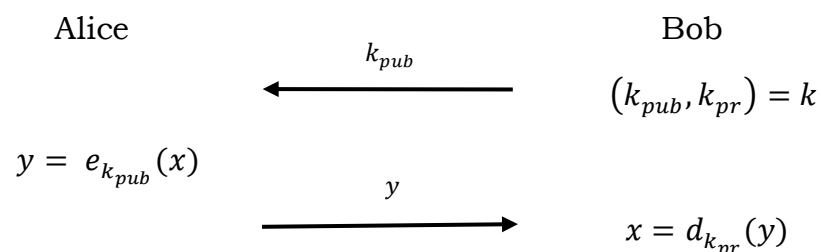
Ketika masalah distribusi kunci dapat diselesaikan, masalah lain yang mungkin timbul adalah kebutuhan jumlah kunci. Dalam jaringan n pengguna maka jumlah kunci yang dibutuhkan adalah $\frac{n \cdot (n-1)}{2}$ kunci, dan setiap pengguna harus menyimpan $n-1$ kunci secara aman.

c. Masalah nonrepudiasi

Kelemahan lain dari kriptografi simetris adalah akan sulit membuktikan siapa pengirim pesan karena kunci untuk enkripsi dan dekripsi sama, bahkan penyerang yang memiliki kunci dapat mengirimkan pesan palsu.

2. Kriptografi Asimetris

Kriptografi asimetris muncul dengan solusi untuk kerentanan pada kriptografi simetris. Kriptografi Asimetris, atau juga dikenal dengan Kriptografi Kunci Publik, pertama kali dikenalkan oleh Whitfield Diffie, Martin Hellman dan Ralph Merkle pada tahun 1976, dengan ide: setiap pengguna hanya memiliki satu pasang kunci (publik dan privat), sehingga tidak perlu menyimpan banyak kunci untuk setiap pasangan komunikasi. Protokol dasar untuk enkripsi pada kriptografi asimetris terdapat pada gambar 2.5.



Gambar 2.5 Protokol dasar untuk enkripsi pada kriptografi asimetris

Bob memiliki pasangan kunci k yang terdiri dari kunci publik dan kunci privat. Ketika Alice ingin mengirimkan pesan kepada Bob, Alice mengenkripsi pesan x menggunakan kunci publik milik Bob (k_{pub}) dan menghasilkan y . Alice mengirimkan y kepada Bob melalui jalur komunikasi yang mungkin tidak aman. Dengan menggunakan kunci privatnya, Bob dapat mendekripsi pesan y untuk mendapatkan kembali x .

Algoritma Asimetris dibagi menjadi 3 (tiga) kategori, yaitu:

a. Primitif Asimetris

Primitif asimetris merupakan algoritma asimetris yang digunakan sebagai pembangkit kunci publik dan kunci privat. Pasangan kunci ini digunakan di dalam skema enkripsi dan skema tanda tangan. Primitif asimetris dibagi menjadi 3 (tiga), yaitu algoritma dengan *Integer factorization problem* (masalah faktorisasi integer) contohnya algoritma RSA, *Discrete Logarithm Problem* (DLP-masalah logaritma diskrit) contohnya algoritma Diffie-Hellman, dan *Elliptic Curve Discrete Logarithm Problem based Algorithm* (masalah logaritma diskrit kurva eliptis) yang disingkat ECDLP, contohnya algoritma Brainpool, Curve dan Edward.

DLP telah lama menjadi fondasi bagi banyak skema kriptografi kunci publik. Untuk memecahkan DLP, penyerang harus menentukan bilangan bulat rahasia x sehingga $a^x \equiv \beta \bmod p$, dengan grup perkalian $\mathbb{Z}_q^*$ dari *field* prima, generator a , dan elemen target β di *field* terbatas. Secara informal, ditemukan masalah matematika sulit yang berkaitan dengan mengidentifikasi jumlah langkah diskrit x yang diperlukan untuk mencapai elemen target β mulai dari a . Dalam masalah ini, a sebagai basis atau generator. Kompleksitas yang dimiliki DLP menjadi inspirasi bagi pengembangan kriptografi tahan kuantum, khususnya yang berbasis latris.

Dalam konteks PQC, ide kesulitan ini ditransformasikan menjadi masalah-masalah baru pada latris, seperti *Short*

Integer Solution dan *Learning With Errors*, yang secara matematis lebih kompleks dan dirancang agar tetap aman bahkan terhadap serangan komputer kuantum.

b. Skema Enkripsi Asimetris

Skema enkripsi asimetris terdiri dari dua operasi utama, yaitu enkripsi dan dekripsi. Operasi enkripsi menghasilkan *ciphertext* dari sebuah *plaintext* menggunakan kunci publik penerima. Sebaliknya, operasi dekripsi mengembalikan *ciphertext* menjadi pesan asli menggunakan kunci privat milik penerima yang sesuai. Contoh algoritma asimetris skema enkripsi asimetris adalah RSA-OAEP dan RSA-KEM.

c. Skema Tanda Tangan Digital Asimetris

Tanda tangan digital dibangkitkan oleh penanda tangan (*signer*) menggunakan kunci privat dan diverifikasi oleh pemverifikasi (*verifier*) menggunakan kunci publik. Proses verifikasi dilakukan dengan membandingkan *digest* pesan yang telah dihitung dengan hasil dekripsi tanda tangan menggunakan kunci publik. Kunci privat bersifat rahasia dan hanya boleh diakses oleh pemilik sahnya untuk mencegah pemalsuan tanda tangan digital. Sementara itu, meskipun kunci publik tidak bersifat rahasia, integritasnya harus dijaga karena digunakan untuk memverifikasi keaslian tanda tangan melalui *digest* pesan.

Terdapat tiga teknik utama pada skema ini. Teknik pertama adalah algoritma tanda tangan digital RSA, yang menggunakan mekanisme kriptografi berbasis faktorisasi integer. Dua skema tanda tangan digital yang termasuk dalam teknik ini adalah RSASSA-PKCS1-v1 dan RSASSA-PSS. Teknik kedua adalah algoritma tanda tangan digital kurva eliptis (ECDSA), yang menggunakan mekanisme berbasis logaritma diskrit pada kurva eliptis, dengan skema ECDSA. Teknik ketiga adalah algoritma tanda tangan digital Kurva Edward

(EdDSA), yang menggunakan varian tanda tangan Schnorr berbasis kurva Edward terpuntir.

Mekanisme keamanan yang dapat disediakan oleh kriptografi asimetris antara lain:

a. Konfidensialitas

Konfidensialitas dapat dicapai dengan enkripsi pesan menggunakan algoritma seperti RSA atau Elgamal. Cara kerjanya yaitu pengirim mengenkripsi pesan menggunakan kunci publik penerima pesan dan hanya penerima pesan yang memiliki kunci privat yang dapat mendekripsi pesan.

b. Autentikasi

Mekanisme autentikasi bertujuan untuk memastikan bahwa pihak yang berkomunikasi adalah asli (bukan pihak yang menyamar). Metodenya adalah pengirim menandatangani pesan dengan kunci privatnya kemudian penerima dapat memverifikasi tanda tangan menggunakan kunci publik pengirim. Algoritma yang digunakan pada skema ini antara lain RSA dan ECDSA.

c. Integritas

Mekanisme integritas berfungsi agar pesan tidak mengalami perubahan selama transmisi. Kriptografi asimetris menyediakan integritas dengan cara pengirim membuat tanda tangan digital atas pesan menggunakan kunci privatnya kemudian tanda tangan ini diverifikasi oleh penerima menggunakan kunci publik pengirim. Contoh algoritma yang digunakan adalah RSA, DSA, dan ECDSA.

d. Nonrepudiasi

Nonrepudiasi merupakan mekanisme anti penyangkalan dimana pengirim tidak akan dapat menyangkal bahwa dia mengirimkan sebuah pesan. Nonrepudiasi dicapai dengan cara pengirim menandatangani pesan dengan kunci privatnya. Karena hanya pengirim yang memiliki kunci privat tersebut

maka dia tidak dapat menyangkal bahwa pesan tersebut berasal darinya.

e. *Key Exchange*

Mekanisme *key exchange* yang aman memungkinkan kedua belah pihak bertukar kunci enkripsi secara aman di saluran yang tidak aman. Skema kriptografi asimetris menyediakan *key exchange* dengan menggunakan protokol Diffie-Hellman *Key Exchange* (DHKE), *RSA key transport protocol* dan *Key Encapsulation Mechanisms*.

Disamping beberapa kelebihan dari kriptografi asimetris, sistem ini juga masih menyimpan beberapa kekurangan. Beberapa kekurangan dari kriptografi asimetris adalah:

a. Kinerja yang lebih lambat

Operasi matematika pada kriptografi asimetris seperti eksponensial modular seperti RSA membuatnya menjadi lebih lambat.

b. Autentikasi dari Kunci Publik

Dalam sistem skala besar, setiap pengguna memiliki sepasang kunci publik dan kunci privat yang harus dikelola, disimpan dan didistribusikan dengan aman. Kunci privat harus dijaga kerahasiaannya, dan kunci publik harus tersedia namun juga harus tetap asli (autentik) sehingga dapat dipercaya. Solusi umum yang digunakan selama ini adalah dengan adanya *Public Key Infrastructure* (PKI) yang berguna untuk mengelola sertifikat digital dan memastikan kunci publik berasal dari sumber terpercaya. *Certification authority* (CA) akan bertindak sebagai pihak ketiga terpercaya yang menandatangani kunci publik pengguna.

Keamanan kriptografi asimetris bergantung pada trust model yang digunakan untuk mendistribusikan kunci publik. CA dapat diserang atau terbobol. Ketika CA tidak dapat dipercaya maka seluruh sistem keamanan bisa runtuh.

- c. Ancaman dari komputer kuantum
Algoritma kriptografi asimetris yang ada saat ini, seperti RSA, DSA dan ECC rentan terhadap Algoritma Shor yang bisa menghancurkan keamanan mereka jika komputer kuantum yang cukup kuat berhasil dibuat. Ancaman dari komputer kuantum ini dapat diatasi dengan melakukan migrasi ke algoritma kriptografi *post-quantum*.

B. Ancaman Komputasi Kuantum terhadap Kriptografi Klasik

1. Sejarah singkat komputasi kuantum

Komputasi kuantum adalah cabang ilmu komputer yang memanfaatkan prinsip-prinsip mekanika kuantum untuk melakukan perhitungan yang jauh lebih efisien dibandingkan komputer klasik dalam beberapa jenis masalah. Konsep ini mulai muncul pada akhir abad ke-20 dan terus berkembang pesat hingga kini.

a. Awal Pemikiran

Pada 1981, fisikawan Richard Feynman dan David Deutsch mengusulkan ide bahwa komputer kuantum dapat meniru sistem fisik yang sangat kompleks, yang sulit atau bahkan mustahil disimulasikan menggunakan komputer klasik. Mereka menunjukkan bahwa mekanika kuantum memiliki potensi besar untuk mempercepat perhitungan dibandingkan model Turing klasik.

b. Model Komputasi Kuantum

David Deutsch pada 1985 merumuskan model komputer universal kuantum, yang menjadi dasar teori komputasi kuantum. Pada 1994, Peter Shor's memperkenalkan algoritma kuantum yang dapat memfaktorkan bilangan besar dengan sangat cepat, sebuah tugas yang sangat sulit bagi komputer klasik. Algoritma ini menunjukkan potensi praktis dari komputasi kuantum, terutama dalam bidang kriptografi.

c. Kemajuan Teknologi

Pada dekade 2000-an, perusahaan dan lembaga riset mulai mengembangkan prototipe komputer kuantum. Salah satu tonggak penting adalah pengumuman IBM pada 2001, yang berhasil menjalankan Algoritma Shor pada komputer kuantum kecil. Kemudian, teknologi ini terus berkembang dengan implementasi qubit berbasis ion terperangkap, superkonduktor, dan fotonik.

d. Era Modern

Komputasi kuantum kini memasuki tahap aplikasi praktis. Perusahaan seperti Google, IBM, dan D-Wave telah memperkenalkan komputer kuantum dengan puluhan hingga ratusan qubit. Pada 2019, Google mengklaim mencapai "keunggulan kuantum" (*quantum supremacy*), yakni menyelesaikan tugas yang tidak dapat dilakukan oleh komputer klasik dalam waktu wajar.

Komputasi kuantum memiliki potensi untuk merevolusi berbagai bidang, seperti kimia kuantum, pengoptimalan, kecerdasan buatan, dan kriptografi. Namun, masih ada tantangan besar, termasuk mengatasi *error correction* dan meningkatkan skala komputer kuantum menjadi lebih stabil dan efisien. Komputasi kuantum adalah salah satu inovasi teknologi yang paling menjanjikan di abad ke-21, membuka jalan bagi solusi baru terhadap masalah kompleks yang sebelumnya tidak terjangkau oleh teknologi klasik.

2. Algoritma Shor dan Algoritma Grover

Dengan munculnya komputer kuantum yang menggunakan prinsip mekanika kuantum, para ahli menyadari bahwa metode kriptografi klasik yang saat ini digunakan, seperti RSA, Diffie-Hellman, dan algoritma berbasis kurva eliptik, berisiko menjadi tidak aman. Komputer kuantum memiliki kemampuan untuk menjalankan algoritma tertentu yang bisa memecahkan masalah-

masalah yang sangat kompleks dengan jauh lebih cepat daripada komputer klasik.

Perhatian terhadap ancaman komputer kuantum dimulai pada tahun 1994 ketika Peter Shor's, seorang matematikawan, menemukan Algoritma Shor. Algoritma ini memungkinkan komputer kuantum untuk memfaktorkan bilangan besar secara eksponensial lebih cepat dibandingkan komputer klasik. Pemfaktoran bilangan besar adalah dasar keamanan dari algoritma RSA, yang umum digunakan dalam komunikasi terenkripsi di internet. Algoritma Shor menunjukkan bahwa komputer kuantum yang cukup kuat dapat memecahkan enkripsi RSA dalam waktu yang jauh lebih singkat, membahayakan sistem keamanan ini.

Algoritma Grover yang ditemukan pada tahun 1996 juga menunjukkan potensi komputer kuantum dalam mencari kunci algoritma kriptografi simetris. Algoritma ini memungkinkan peningkatan efisiensi serangan *brute-force* pada algoritma kriptografi simetris seperti AES, mengurangi kekuatan efektif kunci simetris menjadi setengahnya (misalnya, kunci AES-128 hanya memiliki kekuatan setara AES-64 di hadapan komputer kuantum). Walau ancamannya tidak sekuat dampak Shor's terhadap RSA, temuan ini tetap menunjukkan perlunya perombakan dalam pendekatan keamanan siber untuk menghadapi komputer kuantum.

3. Urgensi pengembangan PQC

Sejarah munculnya PQC berawal dari kebutuhan akan algoritma kriptografi yang tahan terhadap serangan komputer kuantum. Dengan munculnya komputer kuantum yang menggunakan prinsip mekanika kuantum, para ahli menyadari bahwa metode enkripsi klasik yang saat ini digunakan, seperti RSA, Diffie-Hellman, dan algoritma berbasis kurva eliptis, berisiko menjadi tidak aman. Komputer kuantum memiliki kemampuan untuk menjalankan algoritma tertentu yang bisa memecahkan

masalah-masalah yang sangat kompleks dengan jauh lebih cepat daripada komputer klasik.

Tabel 2.1 Sistem kriptografi yang banyak digunakan beserta tingkat keamanannya yang diasumsikan

Tabel dibuat ulang dari [3].

Table 1 Examples of widely deployed cryptographic systems and their conjectured security levels			
<i>Name</i>	<i>Function</i>	<i>Pre-quantum security level</i>	<i>Post-quantum security level</i>
Symmetric cryptography			
AES-128 ⁸	Symmetric encryption	128	64 (Grover)
AES-256 ⁸	Symmetric encryption	256	128 (Grover)
Salsa20 ⁵⁸	Symmetric encryption	256	128 (Grover)
GMAC ⁵⁹	MAC	128	128 (no impact)
Poly 1305 ⁶⁰	MAC	128	128 (no impact)
SHA-256 ⁶¹	Hash function	256	128 (Grover)
SHA3-256 ⁶²	Hash function	256	128 (Grover)
Public-key cryptography			
RSA-3072 ¹	Encryption	128	Broken (Shor)
RSA-3072 ¹	Signature	128	Broken (Shor)
DH-3072 ⁴²	Key exchange	128	Broken (Shor)
DSA-3072 ^{63,64}	Signature	128	Broken (Shor)
256-bit ECDH ^{4–6}	Key exchange	128	Broken (Shor)
256-bit ECDSA ^{66,67}	Signature	128	Broken (Shor)
Security levels shown are against the best pre-quantum and post-quantum attacks known. Security level b means that the best attacks use approximately 2^b operations. This optimization ignores parallelization requirements; see text for discussion of the impact of such requirements. For hash functions, ‘security’ in this table refers to pre-image security.			

Informasi tentang contoh sistem kriptografi yang banyak digunakan beserta tingkat keamanan yang diduga sebelum dan setelah munculnya komputer kuantum dapat dilihat pada tabel 2.1. Berdasarkan tabel 2.1 diketahui bahwa:

- a. Algoritma Grover hanya mengurangi *security strength* algoritma kunci simetris (AES-128, AES-256, Salsa 20), algoritma *Message Authentication Code*-MAC (GMAC, Poly1305) dan fungsi *hash* (SHA-256, SHA3-256) menjadi setengahnya.
- b. Algoritma Shor dapat memecahkan algoritma kriptografi kunci publik seperti RSA-3072, DH-3072, DSA-3072, 256 bit ECDH dan 256-bit ECDSA.

C. Prinsip-Prinsip Dasar PQC

1. Definisi

Menurut NIST, PQC adalah kriptografi yang mengacu pada pengembangan algoritma kriptografi yang aman terhadap serangan yang dilakukan oleh komputer kuantum sekaligus tetap efisien dan aman terhadap serangan dari komputer klasik. NIST mendefinisikan PQC sebagai algoritma yang ditujukan untuk menggantikan algoritma kriptografi konvensional seperti RSA, ECC, dan DSA yang dianggap rentan jika komputer kuantum berkembang menjadi lebih kuat [3].

PQC menurut Daniel J. Bernstein, Johannes Buchmann, dan Erik Dahmen dalam bukunya yang berjudul “*Introduction to Post-Quantum Cryptography*” [2] didefinisikan sebagai kumpulan metode dan algoritma kriptografi yang dirancang untuk tetap aman dari serangan oleh komputer kuantum. Definisi ini menekankan pada perlindungan kriptografi modern dari ancaman yang muncul dengan perkembangan komputer kuantum, yang memiliki potensi untuk merusak keamanan algoritma kriptografi klasik seperti RSA, DSA, dan ECC.

Menurut Uday S. Reddy dalam bukunya yang berjudul “*Quantum Computing and Post-Quantum Cryptography: Secure and Efficient Algorithms*”, PQC adalah cabang kriptografi yang berfokus pada pengembangan algoritma yang dapat mempertahankan keamanan data bahkan ketika komputer kuantum yang kuat tersedia [4]. PQC dirancang untuk melawan ancaman yang disebabkan oleh komputer kuantum, yang memiliki kemampuan untuk memecahkan algoritma kriptografi klasik yang saat ini digunakan secara luas seperti RSA, ECC, dan DSA.

2. Perbedaan antara kriptografi klasik dan PQC

Perbedaan antara kriptografi klasik dan PQC terletak pada dasar matematika yang mendasari keamanan dan ancaman yang dihadapi keduanya. Kriptografi klasik didasarkan pada asumsi bahwa beberapa masalah matematika sangat sulit dipecahkan oleh komputer klasik, seperti faktorisasi bilangan besar dalam algoritma RSA atau masalah logaritma diskrit dalam ECC. Masalah-masalah ini memerlukan waktu yang sangat lama untuk dipecahkan oleh komputer klasik dengan kekuatan pemrosesan yang ada. Sebagai hasilnya, algoritma-algoritma ini telah menjadi standar dalam banyak aplikasi keamanan digital saat ini. Namun, ancaman dari komputer kuantum, yang dapat memecahkan masalah-masalah tersebut jauh lebih cepat, telah mendorong pengembangan PQC, yang bertujuan untuk tetap aman dari serangan kuantum.

PQC dikembangkan dengan mempertimbangkan ancaman dari komputer kuantum, khususnya algoritma kuantum seperti Algoritma Shor, yang dapat memecahkan masalah faktorisasi dan logaritma diskrit dalam waktu polinomial. Algoritma kuantum ini membuat RSA dan ECC rentan terhadap serangan kuantum, sehingga PQC menggunakan masalah matematika yang berbeda yang diyakini sulit dipecahkan baik oleh komputer kuantum maupun klasik. Misalnya, *lattice-based cryptography* didasarkan pada masalah geometri latis yang sangat kompleks dan *code-based cryptography* didasarkan pada masalah *decoding* kode linear.

Algoritma-algoritma ini tahan terhadap serangan yang menggunakan komputer kuantum karena mereka tidak bergantung pada masalah yang dapat diserang oleh Algoritma Shor.

Ukuran kunci dan efisiensi adalah perbedaan signifikan antara kriptografi klasik dan PQC. Algoritma klasik seperti RSA atau ECC umumnya menggunakan ukuran kunci yang lebih kecil dan lebih efisien dalam hal komputasi dibandingkan dengan algoritma post-kuantum. Sebagai contoh, RSA dengan panjang kunci 2048 bit dapat dianggap cukup aman untuk saat ini, tetapi algoritma *lattice-based cryptography* seperti Kyber mungkin memerlukan ukuran kunci yang jauh lebih besar untuk mencapai tingkat keamanan yang setara. Meskipun PQC menawarkan keamanan yang lebih baik terhadap serangan kuantum, beberapa algoritma PQC dapat memiliki performa yang lebih lambat atau memerlukan memori lebih besar, yang menjadi tantangan dalam penerapannya secara luas di infrastruktur yang ada.

Perbedaan lain yang menonjol antara kriptografi klasik dan PQC adalah tantangan dalam standardisasi dan implementasi. Kriptografi klasik telah menjadi standar global dan digunakan di berbagai aplikasi, mulai dari komunikasi internet hingga enkripsi data pribadi. Protokol seperti TLS dan SSL bergantung pada algoritma klasik seperti RSA dan ECC. Sebaliknya, PQC masih dalam tahap pengembangan dan evaluasi. NIST sedang memimpin proyek untuk menstandarkan algoritma PQC terbaik melalui serangkaian kompetisi internasional, yang telah menghasilkan tiga dokumen standar algoritma di tahun 2024 dan beberapa draf standar yang sedang disusun tahun 2025. Standardisasi ini akan memungkinkan transisi dari algoritma klasik ke algoritma *post-quantum*, meskipun ini memerlukan waktu dan penyesuaian besar dalam infrastruktur teknologi.

Secara keseluruhan, perbedaan utama antara kriptografi klasik dan PQC terletak pada ancaman yang ditangani dan algoritma yang digunakan untuk menjamin keamanan. Kriptografi

klasik saat ini menghadapi ancaman serius dari komputer kuantum, sementara PQC dirancang untuk melindungi sistem dari ancaman ini. Namun, PQC menghadapi tantangan dari segi implementasi praktis karena ukuran kunci yang lebih besar dan efisiensi yang lebih rendah dibandingkan dengan algoritma klasik. Pengembangan PQC menjadi sangat penting untuk melindungi keamanan siber di masa depan, mengingat kemungkinan munculnya komputer kuantum yang mampu menyerang algoritma klasik. Dengan perkembangan teknologi kuantum yang semakin pesat, PQC akan menjadi pilar penting dalam menjaga keamanan informasi global.

3. Kriteria keamanan dalam PQC

Kriteria keamanan dalam PQC sangat penting karena tujuan utama PQC adalah menyediakan algoritma yang tetap aman ketika komputer kuantum tersedia dan mampu menyerang algoritma kriptografi klasik. Dengan kemajuan teknologi komputer kuantum, algoritma seperti RSA, ECC, dan DSA diperkirakan akan rentan terhadap serangan oleh algoritma kuantum seperti Algoritma Shor. Oleh karena itu, PQC harus memenuhi beberapa kriteria keamanan untuk memastikan keandalannya, baik dalam menghadapi serangan komputer kuantum maupun serangan klasik.

a. Keamanan terhadap Serangan Komputer Kuantum

Kriteria paling mendasar dalam PQC adalah ketahanan terhadap serangan komputer kuantum. Algoritma PQC harus aman dari serangan yang dapat dilakukan oleh komputer kuantum yang menjalankan Algoritma Shor atau Algoritma Grover. Algoritma Shor secara khusus menargetkan masalah faktorisasi bilangan bulat dan logaritma diskrit yang digunakan dalam algoritma klasik seperti RSA dan ECC.

Di sisi lain, Algoritma Grover dapat mempercepat pencarian *brute-force* pada kunci simetris, mengurangi kompleksitas pencarian dari 2^n menjadi $2^{n/2}$. Algoritma PQC menggunakan masalah matematika yang diyakini tidak dapat

diselesaikan secara efisien oleh komputer kuantum, seperti *Learning With Errors* dalam kriptografi berbasis latis (*lattice-based cryptography*) dan masalah *decoding* pada kode dalam *code-based cryptography*.

b. Keamanan Klasik

Selain ketahanan terhadap serangan kuantum, algoritma PQC juga harus tetap aman terhadap serangan yang menggunakan komputer klasik. Ini termasuk serangan *brute-force*, serangan analisis matematika, serta serangan oleh penyerang yang memiliki kemampuan komputasi konvensional yang kuat. Kriteria keamanan klasik ini mencakup kemampuan algoritma untuk mempertahankan keamanan meskipun komputasi klasik yang tersedia terus meningkat. Dengan demikian, PQC harus menawarkan perlindungan ganda yaitu aman dari serangan kuantum dan tetap tahan terhadap serangan klasik yang sudah dikenal.

c. Ketahanan terhadap Serangan Non-Kuantum

Algoritma PQC juga harus aman terhadap serangan non-kuantum yang lebih maju, seperti serangan kanal samping (*side-channel attacks*) dan serangan analisis kuantitatif waktu. Serangan kanal samping, misalnya, memanfaatkan informasi yang diperoleh dari implementasi fisik suatu algoritma (seperti konsumsi daya atau waktu pemrosesan), dan algoritma PQC harus mampu mempertahankan ketahanannya terhadap serangan ini. Selain itu, algoritma harus tahan terhadap serangan *fault injection*, di mana penyerang mencoba menyebabkan kesalahan dalam proses komputasi untuk mengekspos informasi sensitif.

d. Efisiensi dan Ukuran Kunci

Keamanan algoritma PQC tidak hanya didasarkan pada ketahanannya terhadap serangan, tetapi juga efisiensinya dalam hal ukuran kunci, waktu komputasi, dan penggunaan

memori. Algoritma PQC sering kali memerlukan ukuran kunci yang lebih besar dibandingkan dengan algoritma klasik. Misalnya, algoritma berbasis latis seperti Kyber dapat membutuhkan ukuran kunci yang mencapai ratusan kilobit, jauh lebih besar daripada ukuran kunci RSA atau ECC yang setara dalam konteks keamanan klasik. Meskipun ukuran kunci yang besar dapat meningkatkan keamanan, hal ini juga membawa tantangan dalam efisiensi dan penerapan praktis di berbagai perangkat, terutama yang memiliki sumber daya terbatas. Oleh karena itu, kriteria keamanan PQC juga mempertimbangkan keseimbangan antara ukuran kunci dan efisiensi komputasi.

e. Keamanan Jangka Panjang

Salah satu tujuan penting PQC adalah keamanan jangka panjang. Komputer kuantum yang mampu memecahkan algoritma klasik mungkin belum tersedia, tetapi pengembang harus merancang algoritma yang dapat diandalkan selama beberapa dekade mendatang. Oleh karena itu, algoritma post-kuantum harus cukup tangguh untuk menghadapi serangan di masa depan dengan mempertimbangkan peningkatan teknologi kuantum dan klasik. Sebagai bagian dari kriteria ini, para peneliti PQC juga harus mengantisipasi kemungkinan perkembangan algoritma baru atau temuan matematis yang dapat mengancam algoritma yang ada.

f. Ketahanan Terhadap Berbagai Jenis Serangan

Algoritma PQC harus diuji terhadap berbagai jenis serangan kriptografi, termasuk serangan adaptif, serangan pilihan *ciphertext*, dan serangan pemalsuan tanda tangan. Serangan adaptif memungkinkan penyerang untuk memilih input yang sesuai berdasarkan hasil dari operasi sebelumnya, yang dapat mengancam keamanan algoritma tertentu. Kriteria keamanan PQC mencakup uji ketahanan terhadap skenario-skenario ini, untuk memastikan bahwa algoritma tidak hanya

aman dalam kondisi ideal tetapi juga dalam situasi di mana penyerang memiliki kemampuan untuk memodifikasi input atau memperoleh sebagian informasi dari hasil operasinya.

D. Dasar Matematika PQC

1. Teori bilangan

Teori bilangan adalah salah satu cabang tertua dalam matematika, yang berfokus pada sifat-sifat bilangan bulat serta hubungan di antaranya. Dalam sejarahnya, teori ini sering kali diasosiasikan dengan studi keindahan abstrak matematika, namun di era modern, aplikasinya menjadi sangat penting, terutama dalam bidang kriptografi. Kriptografi, seni melindungi komunikasi rahasia, sangat bergantung pada konsep-konsep teori bilangan seperti bilangan prima, kongruensi, dan eksponensiasi modular.

Dengan kemajuan teknologi kuantum, ancaman terhadap algoritma kriptografi klasik semakin nyata. Misalnya, Algoritma Shor yang berjalan di komputer kuantum mampu memfaktorkan bilangan besar dan menyelesaikan logaritma diskret secara efisien, membuat sistem seperti RSA dan Diffie-Hellman rentan. Oleh karena itu, PQC yang dirancang untuk tetap aman terhadap serangan kuantum, sangat membutuhkan dasar teori bilangan yang kokoh. Konsep – konsep yang relevan pada teori bilangan yang digunakan sebagai dasar matematika PQC [5], yaitu:

a. Bilangan Prima

Bilangan prima adalah elemen fundamental dalam teori bilangan. Definisi formalnya adalah bilangan bulat lebih besar dari 1 yang tidak memiliki pembagi selain 1 dan dirinya sendiri. Misalnya, 2,3,5,7, dan 11 adalah bilangan prima. Sebaliknya, bilangan seperti 4,6,46, atau 88 disebut bilangan komposit karena memiliki pembagi selain 1 dan dirinya sendiri.

Salah satu teorema penting dalam teori bilangan adalah Teorema Dasar Aritmatika, yang menyatakan bahwa setiap

bilangan bulat positif dapat diuraikan secara unik menjadi hasil kali bilangan prima. Sebagai contoh:

$$60 = 2^2 \cdot 3 \cdot 5$$

Dalam algoritma seperti RSA, bilangan prima besar digunakan untuk membentuk kunci publik dan privat. Sifat-sifat unik bilangan prima membuat algoritma ini sulit dipecahkan tanpa pengetahuan tentang faktorisasi bilangan tersebut. Namun, dengan Algoritma Shor, komputer kuantum dapat memfaktorkan bilangan besar secara efisien, sehingga mengancam keamanan algoritma berbasis bilangan prima.

b. Kongruensi dan Operasi Modular

Kongruensi adalah konsep penting dalam teori bilangan. Jika a dan b memiliki sisa pembagian yang sama ketika dibagi dengan bilangan bulat positif n , maka dapat dikatakan bahwa a kongruen dengan b modulus n , yang ditulis:

$$a \equiv b \pmod{n}$$

Operasi Modular

Operasi modular menjaga bilangan tetap dalam rentang yang terbatas, membuatnya sangat efisien dalam perhitungan pada komputer. Sebagai contoh, $23 \bmod 7 = 2$, karena 23 dibagi 7 menyisakan 2.

Teorema Sisa Cina

Teorema Sisa Cina menyatakan bahwa ketika modulus dari suatu sistem kongruensi linear saling relatif prima (artinya, modulus-modulus tersebut tidak memiliki faktor pembagi yang sama selain 1), maka terdapat solusi unik untuk sistem tersebut modulus hasil perkalian modulus-modulus tersebut. Misalkan $n_1, n_2, \dots, n_r$ merupakan r bilangan bulat positif yang saling relatif prima. Misal $a_1, a_2, \dots, a_r$ adalah r bilangan bulat sembarang, maka sistem kongruensi :

$$x \equiv a_i \pmod{n_i}$$

Untuk $i = 1, 2, \dots, r$ mempunyai penyelesaian yang sama. Sebarang dua penyelesaian yang sama kongruen modulus $n = n_1, n_2, \dots, n_r$

Teorema Sisa Cina memungkinkan penyelesaian sistem persamaan modular dengan modulus yang berbeda-beda secara efisien. Dengan memecah masalah besar menjadi beberapa masalah kecil, kompleksitas perhitungan dapat dikurangi dan meningkatkan efisiensi, yang sangat berguna dalam kriptografi dan sistem modular lainnya.

c. Teorema Kecil Fermat

Teorema kecil Fermat adalah salah satu hasil paling penting dalam teori bilangan. Teorema ini menyatakan bahwa jika p adalah bilangan prima dan a adalah bilangan bulat yang tidak habis dibagi oleh p , maka:

$$a^{p-1} \equiv 1 \pmod{p}$$

Teorema ini digunakan dalam algoritma eksponensial modular, seperti yang diterapkan dalam pembuatan tanda tangan digital dan fungsi *hash*.

d. Algoritma Euclidean dan *Greatest Common Divisor* (GCD)

Untuk menentukan pembagi persekutuan terbesar GCD dari dua bilangan digunakan Algoritma Euclidean. Misalnya, terdapat dua bilangan a dan b , algoritma ini bekerja dengan menghitung sisa pembagian secara berulang hingga mencapai nol:

$$a = bq + r$$

di mana q adalah hasil bagi dan r adalah sisa pembagian. Ketika sisa pembagian menjadi nol, bilangan terakhir sebelum nol adalah GCD. Dalam Kriptografi, GCD digunakan untuk memastikan keabsahan kunci pada algoritma RSA dan juga muncul dalam *lattice-based cryptography*. PQC memanfaatkan berbagai konsep dari teori bilangan untuk membangun algoritma yang tahan terhadap serangan kuantum.

2. Aljabar linear

Salah satu elemen fundamental dalam PQC adalah penggunaan konsep aljabar linear, yang memungkinkan pengolahan data dalam ruang vektor, matriks, dan bilangan modular untuk membangun algoritma yang aman. Pada bagian ini memberikan landasan matematika berupa pengantar aljabar linear yang akan digunakan dalam pembahasan lebih lanjut terkait teknik-teknik PQC.

a. Vektor dan Ruang Vektor

Vektor adalah salah satu entitas utama yang menjadi dasar dalam aljabar linear. Secara umum, sebuah vektor adalah elemen dalam ruang vektor yang dapat direpresentasikan sebagai barisan bilangan. Dalam ruang berdimensi n (R^n), sebuah vektor dapat ditulis sebagai:

$$v = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

dengan:

- v menyatakan sebuah vektor,
- $v_1, v_2, \dots, v_n$ adalah komponen-komponen vektor yang merupakan elemen bilangan real,
- n adalah dimensi dari ruang vektor, yang menentukan jumlah komponen dalam vektor.

Konsep ruang vektor lebih umum dibandingkan sekadar vektor dalam R^n . Sebuah ruang vektor V adalah himpunan elemen-elemen (vektor) yang memenuhi dua operasi dasar, yaitu:

1) Penjumlahan vektor

Untuk $u, v \in V$, penjumlahan vektor menghasilkan elemen lain di V , dituliskan sebagai:

$$u + v = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} + \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} u_1 + v_1 \\ u_2 + v_2 \\ \vdots \\ u_n + v_n \end{bmatrix}$$

Operasi ini bersifat komutatif dan asosiatif, artinya hasilnya tidak tergantung pada urutan penjumlahan.

2) Perkalian skalar

Untuk $c \in R$ (bilangan real) dan $v \in V$ hasil perkalian skalar antara c dan v tetap berada di V , dinyatakan sebagai:

$$c \cdot v = c \cdot \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} c \cdot v_1 \\ c \cdot v_2 \\ \vdots \\ c \cdot v_n \end{bmatrix}$$

Ruang vektor juga memiliki elemen istimewa, yaitu vektor nol 0 , yang memiliki semua komponen bernilai nol:

$$0 = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Vektor nol memenuhi sifat bahwa untuk setiap $v \in V$:

$$v + 0 = v$$

Ruang vektor sering menjadi tempat di mana berbagai operasi lain seperti proyeksi, rotasi, atau transformasi linear dilakukan. Konsep ini sangat penting dalam membangun algoritma kriptografi modern, khususnya pada *lattice-based cryptography*, yang menggunakan vektor dalam ruang berdimensi tinggi untuk merepresentasikan masalah matematika kompleks.

b. Matriks dan Transformasi Linear

Matriks adalah struktur matematis yang berfungsi untuk merepresentasikan data atau operasi tertentu, terutama transformasi linear. Matriks adalah tabel berbentuk persegi panjang yang tersusun dari elemen-elemen dalam bentuk bilangan yang diatur dalam baris dan kolom. Matriks berukuran $m \times n$ dinyatakan sebagai:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

dengan:

- A adalah matriks
- a_{ij} adalah elemen matriks pada baris ke- i dan kolom ke- j
- m adalah jumlah baris
- n adalah jumlah kolom

Matriks memiliki banyak aplikasi, termasuk merepresentasikan transformasi linear. Sebuah transformasi linear adalah fungsi $T: V \rightarrow W$ yang memenuhi dua sifat:

- 1) Penjumlahan dipertahankan, Untuk semua $u, v \in V$:
- 2) Perkalian skalar dipertahankan, Untuk setiap $c \in R$ dan $v \in V$:

$$T(u + v) = T(u) + T(v)$$

$$T(c \cdot v) = c \cdot T(v)$$

Transformasi linear dapat direpresentasikan oleh matriks. Jika A adalah matriks transformasi linear $m \times n$, dan x adalah vektor input berdimensi n , maka hasil transformasi linear diberikan oleh:

$$y = A \cdot x$$

dengan y adalah vektor hasil transformasi berdimensi m .

Matriks digunakan secara luas dalam PQC untuk merepresentasikan relasi antara elemen-elemen kriptografi. Contohnya, pada *lattice-based cryptography*, matriks digunakan untuk menyusun masalah *lattice reduction*, sedangkan pada *code-based cryptography*, matriks digunakan untuk membangun kode koreksi kesalahan.

- c. Ruang *Null* dan *Rank* Matriks

- 1) Ruang *Null* ($Null(A)$) adalah himpunan semua vektor x yang memenuhi $A \cdot x = 0$. Ruang ini menunjukkan solusi vektor yang membuat hasil transformasi linear menjadi nol.
- 2) *Rank* Matriks adalah ukuran independensi linear dalam baris atau kolom sebuah matriks, yang mencerminkan jumlah informasi unik yang terkandung dalam matriks tersebut.

Konsep-konsep aljabar linear ini menjadi dasar dalam algoritma PQC, diantaranya *Lattice-Based Cryptography* yang menggunakan ruang vektor berdimensi tinggi untuk membentuk masalah *Shortest Vector Problem*, *Code-Based Cryptography* yang memanfaatkan matriks besar untuk mengkodekan dan melindungi data, dan *Multivariate Quadratic Cryptography* yang menggunakan sistem persamaan kuadrat multivariabel yang dapat direpresentasikan dengan matriks dan vektor.

3. Teori informasi

Teori informasi adalah cabang matematika yang berfokus pada pengukuran, representasi, dan transmisi informasi. Dikembangkan oleh Claude Shannon pada tahun 1948 dalam artikelnya "*A Mathematical Theory of Communication*", teori ini menjadi dasar bagi banyak sistem komunikasi modern, termasuk kriptografi. Dalam PQC, teori informasi digunakan untuk :

- a. Mengevaluasi keamanan algoritma terhadap ancaman serangan kuantum, khususnya dalam mengukur kerumitan algoritma melalui konsep entropi dan kapasitas saluran. Entropi digunakan untuk menilai seberapa sulit serangan *brute force* terhadap kunci kriptografi.
- b. Mendesain Protokol yang aman karena kapasitas saluran membantu menentukan batas maksimum keamanan dalam komunikasi terenkripsi.

- c. Teori informasi digunakan dalam merancang kode koreksi kesalahan (*Error Correction*), yang penting dalam algoritma berbasis kode seperti McEliece.

Pada bagian ini memberikan landasan Teori Informasi yang akan digunakan dalam pembahasan lebih lanjut terkait teknik-teknik dalam PQC.

a. Entropi

Entropi adalah ukuran matematis dari ketidakpastian atau informasi dalam suatu sistem. Dalam konteks sumber informasi diskret X yang menghasilkan simbol $\{x_1, x_2, \dots, x_n\}$ dengan probabilitas masing-masing $P(X = x_i) = p_i$, entropi $H(X)$ didefinisikan sebagai:

$$H(X) = - \sum_{i=1}^n p_i \log_2(p_i)$$

Definisi ini memberikan ukuran rata-rata jumlah informasi (dalam bit) yang diperoleh saat simbol diambil dari sumber X .

dimana:

- $H(X)$: Entropi dari sumber X , diukur dalam bit
- p_i : Probabilitas bahwa simbol x_i muncul, dengan $\sum_{i=1}^n p_i = 1$
- $\log_2$: Logaritma basis 2, digunakan karena informasi diukur dalam bit

Interpretasi:

- 1) Jika semua p_i sama besar ($p_i = \frac{1}{n}$), maka entropi $H(X)$ mencapai nilai maksimum $\log_2(n)$. Hal ini menggambarkan ketidakpastian maksimum, karena semua simbol memiliki peluang yang sama.
- 2) Jika salah satu $p_i = 1$ dan sisanya 0, maka $H(X) = 0$. Ini menunjukkan tidak ada ketidakpastian, karena simbol yang akan muncul sudah pasti.

Entropi sering digunakan dalam PQC untuk mengevaluasi kekuatan kunci kriptografi. Kunci dengan

entropi rendah lebih rentan terhadap serangan *brute force* dibandingkan dengan kunci yang memiliki entropi tinggi.

b. Informasi Bersyarat dan Informasi Bersama

Untuk dua variabel acak X dan Y , entropi bersyarat $H(X | Y)$ menggambarkan ketidakpastian dalam X setelah mengetahui Y .

$$H(X|Y) = \sum_{y \in Y} P(Y = y)H(X|Y = y)$$

Informasi bersama $I(X;Y)$ adalah ukuran ketergantungan antara X dan Y , didefinisikan sebagai:

$$I(X;Y) = H(X) - H(X | Y)$$

atau secara ekuivalen:

$$I(X;Y) = H(Y) - H(Y | X)$$

dimana:

- $H(X | Y)$: Entropi bersyarat X terhadap Y , menggambarkan ketidakpastian X setelah Y diketahui.
- $I(X;Y)$: Informasi bersama antara X dan Y , mengukur korelasi antara keduanya.

Informasi bersama relevan dalam PQC untuk mengevaluasi keefektifan protokol kriptografi dalam mengurangi ketidakpastian tentang kunci bagi pihak yang tidak berwenang.

c. Saluran Komunikasi

Saluran komunikasi adalah model yang merepresentasikan proses pengiriman pesan dari pengirim (X) ke penerima (Y). Probabilitas bersyarat $P(Y = y | X = x)$ menggambarkan kemungkinan bahwa y diterima ketika x dikirimkan. Kapasitas saluran adalah batas maksimum jumlah informasi yang dapat dikirimkan melalui saluran secara andal. Jika X adalah input dan Y *output* saluran, kapasitas saluran C dihitung sebagai:

$$C = \max_{P(X)} I(X;Y)$$

dengan:

- $P(X)$: Distribusi probabilitas input saluran.
- C : Kapasitas saluran, diukur dalam bit per simbol.



BAB III

ALGORITMA POST-QUANTUM CRYPTOGRAPHY



BAB III

ALGORITMA POST-QUANTUM CRYPTOGRAPHY

A. Kriptografi Berbasis Latis (*Lattice-based cryptography*)

1. Konsep latis dalam matematika

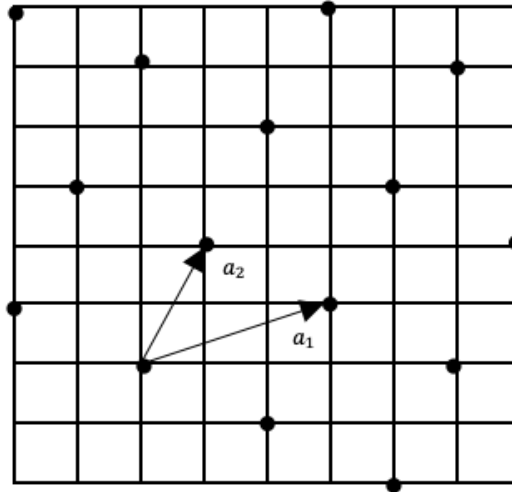
Lattice-based cryptography adalah jenis kriptografi yang keamanannya bergantung pada kesulitan menyelesaikan masalah matematika tertentu dalam latis. Menurut C.Paar, et.al, secara umum, latis adalah sekumpulan titik dalam ruang n -dimensi dengan struktur periodik [5]. Struktur periodik memunculkan nama "latis". Lebih tepatnya, latis L didefinisikan sebagai sekumpulan kombinasi integer linear dari sejumlah vektor independen $a_1, a_2, a_3, \dots, a_n$ yang dikenal sebagai vektor basis, dan koefisien integernya $x_i \in \mathbb{Z}$:

$$L = \{x_1 a_1 + x_2 a_2 + \dots + x_n a_n\}$$

Vektor basis a_1 memiliki dimensi m yaitu vektor-vektor tersebut berada dalam ruang berdimensi m . Dengan demikian, dapat dikatakan bahwa persamaan di atas mendefinisikan ruang latis berdimensi m . Perhatikan bahwa semua koefisien $x_i \in \mathbb{Z}$ adalah bilangan bulat. Jumlah vektor, n , disebut pangkat lattice n . Sebagai contoh, Gambar 3.1 menunjukkan latis dari dua vektor a_1 dan a_2 . Setiap titik padat pada gambar dapat dicapai dengan rumus latis:

$$L = \{x_1 a_1 + x_2 a_2\}$$

Dimana x_1 dan x_2 adalah bilangan bulat. Karena vektor berada pada ruang dimensi 2, maka latis memiliki dimensi $m = 2$. Dan karena latis tersebut direntangkan oleh dua vektor, pangkatnya juga $n = 2$.



Gambar 3.1 Latis 2 dimensi
Digambar ulang dari [1]

Seperti halnya skema kunci publik konvensional, kriptografi berbasis latis juga beroperasi pada angka dari himpunan terbatas, bukan pada angka riil. Untuk skema PQC, koordinat vektor basis dari ring bilangan bulat $\mathbb{Z}_q$. Untuk membangun kriptosistem berbasis latis yang efisien, yaitu cepat, bilangan bulat q umumnya dipilih sebagai bilangan prima atau pangkat dua. Perlu diingat bahwa ring tersebut terdiri dari bilangan bulat $\{0; 1; \dots; q - 1\}$ dan aritmatika dilakukan dengan operasi “mod q ”.

Dengan mengingat hal ini, dapat didefinisikan vektor a_1 dengan dimensi m dalam notasi kolom sebagai berikut:

$$a_1 = \begin{pmatrix} a_{0,i} \\ a_{1,i} \\ \vdots \\ a_{m-1,i} \end{pmatrix}, \text{ dimana } a_{j,i} \in \mathbb{Z}$$

Karena koordinatnya berasal dari ring bilangan bulat $\mathbb{Z}_q$ dan vektor memiliki dimensi m , akan digunakan notasi yang lebih pendek $a_i \in \mathbb{Z}_q^m$ dalam bentuk berikut. Himpunan n vektor basis a_i yang membentuk latis dapat dituliskan sebagai matriks:

$$A = \{a_1, a_2, \dots, a_n\}$$

Untuk latis dengan basis vektor n , digunakan notasi $A \in \mathbb{Z}_q^{m \times n}$

Selanjutnya akan dibahas teori-teori fundamental yang menjadi dasar bagi sebagian besar skema kriptografi modern

berbasis latis, yaitu *Short Integer Solution* (SIS) dan *Learning With Errors* (LWE). Selain itu, akan dibahas pula versi analog dari masalah-masalah tersebut yang diterapkan pada struktur *Ring* untuk meningkatkan efisiensi algoritma.

a. Masalah SIS

Masalah SIS menjadi dasar bagi berbagai fungsi kriptografi, seperti *hash* satu arah yang *collision-resistant*, skema identifikasi, tanda tangan digital, serta primitif kriptografi lainnya. Menurut Micciancio dan Peikert [6], SIS diparameterkan oleh bilangan bulat positif n dan q yang mendefinisikan grup $\mathbb{Z}_q^n$, sebuah bilangan riil positif β , dan sejumlah m elemen grup. Untuk konkritnya, n adalah parameter utama masalah (misalnya $n \geq 100$), dan $q > \beta$ keduanya paling sedikit merupakan suatu polinomial kecil dalam n . Semakin banyak vektor m , masalah menjadi lebih mudah karena ada lebih banyak kombinasi yang bisa dipilih. Namun semakin besar dimensi n masalah SIS akan semakin sulit.

Definisi SIS

Diberikan m vektor acak seragam $a_i \in \mathbb{Z}_q^n$ yang membentuk kolom-kolom dari sebuah matriks $A \in \mathbb{Z}_q^{n \times m}$, temukan vektor bilangan bulat tak nol $z \in \mathbb{Z}^m$ dengan norma $\|z\| \leq \beta$ sehingga:

$$f_A(z) := A_z = \sum_i a_i \cdot z_i = 0 \in \mathbb{Z}_q^n$$

Tanpa batasan panjang ($\|z\|$), SIS akan mudah diselesaikan dengan cara standar seperti eliminasi Gaussian. Karena itu, batas norma (β) harus lebih kecil dari modulus q . Apabila hal tersebut tidak dipenuhi maka selalu ada solusi trivial, misalnya vektor yang hanya berisi kelipatan q .

SIS dalam perspektif latis dapat dipandang sebagai permainan di dalam “latis q -ary”, yaitu kumpulan titik diskrit yang dibentuk oleh vektor – vektor solusi dari sistem $A_z =$

$0 \pmod{q}$). Masalah yang harus diselesaikan adalah bagaimana menemukan satu vektor pendek pada sistem linear tanpa *error*.

b. Masalah LWE

Setelah pemahaman mengenai SIS sebagai masalah mencari solusi pendek pada sistem linear tanpa *error*, pembahasan berlanjut ke LWE, yang memperkenalkan elemen baru berupa “*error* acak.” Apabila masalah SIS adalah bagaimana menemukan kombinasi bilangan bulat pendek yang menghasilkan nol, maka LWE adalah bagaimana untuk mempelajari sistem persamaan linear yang dipengaruhi oleh *error* kecil. Adanya *error* ini membuat masalah yang tadinya mudah (sistem linear biasa) berubah menjadi sangat sulit. Hal ini yang menjadikan LWE dasar utama bagi berbagai algoritma enkripsi dan pertukaran kunci dalam PQC.

Untuk masalah LWE, ide dasarnya adalah mengambil beberapa vektor basis $a_1, a_2, \dots, a_n \in \mathbb{Z}_q^m$ dan mengalikannya dengan koefisien integer rahasia $s_1, s_2, \dots, s_n$ sehingga mencapai titik target t pada latis:

$$s_1 a_1 + s_2 a_2 + \dots + s_n a_n = t$$

Sama seperti masalah DLP, seorang penyerang sekarang diberi titik awal (himpunan vektor basis) serta titik akhir t dan tujuannya adalah menemukan nilai yaitu, himpunan bilangan bulat $(s_1, s_2, \dots, s_n)$ yang harus digunakannya untuk mengalikan titik awal sehingga ia mencapai titik akhir.

Untuk menggambarkan masalah tersebut, dapat dibantu dengan matriks, sebagai berikut:

$$A \cdot s = t$$

Dimana, $s = (s_1, s_2, \dots, s_n)$ adalah himpunan koefisien bilangan bulat rahasia, matriks A berisi vektor basis dan t adalah vektor target. Berikut ini adalah contoh sederhana perkalian matriks-koefisien untuk mengembangkan masalah LWE.

Contoh

Terdapat latas yang didefinisikan di atas $\mathbb{Z}_{17}$ dengan pangkat $n = 4$ dan vektor memiliki dimensi $m = 7$, sehingga matriks memiliki bentuk $A \in \mathbb{Z}_{17}^{7 \times 4}$. Perlu diingat pula bahwa t juga vektor dengan dimensi 7, yang merupakan suatu titik di suatu tempat pada latas. Perhatikan persamaan berikut :

$$A \cdot s = \begin{pmatrix} 10 & 13 & 16 & 10 \\ 12 & 5 & 14 & 12 \\ 10 & 11 & 8 & 11 \\ 7 & 7 & 5 & 3 \\ 8 & 13 & 8 & 9 \\ 14 & 13 & 13 & 16 \\ 7 & 6 & 10 & 4 \end{pmatrix} \cdot \begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \end{pmatrix} \equiv \begin{pmatrix} 2 \\ 14 \\ 7 \\ 6 \\ 10 \\ 10 \\ 16 \end{pmatrix} = t \mod 17$$

Perlu diingat bahwa tujuan LWE adalah untuk menciptakan masalah yang sulit dipecahkan oleh penyerang. Namun contoh tersebut ternyata bukanlah masalah yang sulit karena persamaan di atas menggambarkan sistem persamaan linear. Perlu diingat bahwa teknik seperti eliminasi Gaussian akan menyelesaikan masalah ini dengan sangat cepat, bahkan untuk matriks yang sangat besar. Ini berarti dalam kasus latas, secara komputasi mudah untuk menemukan vektor koefisien integer (yang tidak diketahui) s_i yang jika dikalikan dengan vektor basis G mencapai titik t sehingga:

$$s = \begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \end{pmatrix} = \begin{pmatrix} 6 \\ 2 \\ 12 \\ 3 \end{pmatrix}$$

Masalah ini dapat diubah menjadi masalah komputasi yang sulit dengan memperkenalkan perubahan sederhana. Pertimbangkan t bukan sebagai titik pada latas tetapi sebagai titik yang dekat dengan titik pada latas. Perubahan ini dapat dijelaskan secara matematis dengan memperkenalkan vektor *error* e ke dalam persamaan, yang menghasilkan:

$$A \cdot s + e = t'$$

Vektor *error* memperkenalkan suatu perkiraan ke dalam masalah. Apabila ditinjau kembali contoh di atas, tetapi dengan vektor kesalahan tambahan e :

$$\begin{pmatrix} 10 & 13 & 16 & 10 \\ 12 & 5 & 14 & 12 \\ 10 & 11 & 8 & 11 \\ 7 & 7 & 5 & 3 \\ 8 & 13 & 8 & 9 \\ 14 & 13 & 13 & 16 \\ 7 & 6 & 10 & 4 \end{pmatrix} \cdot \begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \end{pmatrix} + \begin{pmatrix} e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \\ e_6 \\ e_7 \end{pmatrix} = \begin{pmatrix} 1 \\ 13 \\ 8 \\ 6 \\ 10 \\ 9 \\ 15 \end{pmatrix} \mod 17$$

Jika diberikan persamaan seperti itu tanpa menyertakan e atau s , penyerang dapat mencoba menggunakan eliminasi Gaussian atau teknik lain untuk menyelesaikan sistem persamaan linear, tetapi ia tidak akan dapat menghitung solusi yang benar karena t' bukan titik pada kisi. Jika membandingkan nilai-nilai di sisi kanan Persamaan di atas dan persamaan sebelumnya (yang belum menggunakan e), dapat diketahui bahwa vektor target t dan t' sebenarnya sangat berdekatan. Jadi, dapat dilihat bahwa bahkan penyimpangan kecil dari koefisien kisi, misalnya dengan nilai kesalahan $e_i \in \{-1, 0, 1\}$, cukup untuk membuat masalah yang *non-trivial*. Faktanya, vektor *error* dalam persamaan di atas adalah:

$$e = \begin{pmatrix} e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \\ e_6 \\ e_7 \end{pmatrix} = \begin{pmatrix} -1 \\ -1 \\ 1 \\ 0 \\ 0 \\ 1 \\ -1 \end{pmatrix} \equiv \begin{pmatrix} 16 \\ 16 \\ 1 \\ 0 \\ 0 \\ 1 \\ 16 \end{pmatrix} \mod 17$$

Dalam $\mathbb{Z}_{17}$ berlaku bahwa $-1 \equiv 16$. Untuk kriptosistem aktual, nilai kecil untuk e penting karena harus dipastikan bahwa penerima yang sah masih dapat memulihkan titik kisi t yang paling dekat dengan t' .

Singkatnya, dengan pengenalan vektor *error*, dapat tercapai tujuan bahwa teknik standar untuk memecahkan sistem persamaan linier (seperti eliminasi Gaussian) tidak

dapat digunakan untuk memulihkan variabel s dan e yang tidak diketahui secara efisien. Untuk sistem dunia nyata, pendekatan *brute-force* tidak berfungsi dengan memilih $rank\ n$ dan dimensi m latis yang cukup besar.

Hal yang menarik yaitu seseorang bahkan dapat menunjukkan bahwa memecahkan perkiraan sistem linier tersebut memang merupakan masalah sulit yang termasuk dalam kelas masalah NP-complete¹, dan yang dianggap sulit bahkan pada komputer kuantum skala besar. Oleh karena itu, definisi berikut tampaknya berguna untuk membangun sistem kriptografi yang tahan terhadap komputer kuantum.

Definisi Masalah LWE

Diberikan satu set n vektor basis $a_i \in \mathbb{Z}_q^m$ direpresentasikan oleh matriks A dan titik $t \in \mathbb{Z}_q^m$.

LWE merupakan masalah dalam menentukan himpunan koefisien rahasia $s = (s_1, s_2, \dots, s_n)$ dengan $s \in \mathbb{Z}_q$, sehingga:

$$A \cdot s + e = t \mod q$$

Dimana e adalah vektor error yang tidak diketahui yang terdiri dari bilangan bulat kecil modulus q .

Definisi LWE di atas dapat digunakan untuk membangun kriptosistem PQC tetapi untuk keamanan memerlukan parameter yang sangat besar. Untuk lebih efisien dalam praktik implementasinya, pada bagian selanjutnya akan dibahas mengenai masalah *Ring-LWE*.

c. Sistem enkripsi berbasis LWE simpel

Dengan masalah LWE, maka terdapat dasar untuk membangun kriptosistem berbasis latis simpel untuk mengenkripsi data. Sebelum membahas rinciannya, ada

¹ **NP-complete** secara garis besar merupakan kumpulan masalah yang memerlukan upaya eksponensial untuk diselesaikan pada sistem komputasi klasik dalam kasus umum. Namun, apabila sebuah solusi ditemukan, validitas solusi tersebut dapat diverifikasi dalam waktu polinomial. Contoh masalah NP-complete yang terkenal dalam ilmu komputer adalah *knapsack problem* dan *traveling salesman problem*

baiknya untuk membahas persyaratan penting dari skema tersebut.

Seperti halnya setiap skema enkripsi kunci publik, tujuannya adalah Alice mengenkripsi pesan dan mengirimkannya ke Bob, yang mendekripsinya. Alice akan mengkodekan pesannya dalam masalah LWE. Sebagai bagian dari proses ini, ia menambahkan *error* acak ke titik kisi. Karena masalah LWE sulit, Bob memerlukan beberapa informasi tambahan untuk mendekripsinya. Hal ini berbeda dari skema enkripsi kunci publik konvensional, yang sepenuhnya deterministik tanpa keacakan sama sekali seperti RSA. Namun, untuk kriptosistem berbasis LWE, *noise* tambahan disembunyikan dalam parameter t . Masalah *error* tersembunyi ini dapat diatasi oleh Alice dan Bob dengan menggunakan teknik pengkodean khusus.

Simple-LWE dibatasi dengan mengenkripsi hanya satu bit pesan pada satu waktu. Skema enkripsi yang dapat digunakan dalam praktik akan dibahas selanjutnya.

Pembangkitan kunci

Pembangkitan kunci LWE simpel menghasilkan matriks A dari vektor basis kisi dan vektor rahasia s dan menerapkan masalah LWE untuk menghasilkan kunci publik yang sesuai. Rinciannya adalah sebagai berikut:

Input :

q : modulus bilangan bulat

n : dimensi vektor

k : jumlah baris pada matriks A

Output :

- 1) Pilih n vektor acak $a_i \in \mathbb{Z}_q^k$ dan kombinasikan dalam sebuah matriks $A = a_1, a_2, \dots, a_n \in \mathbb{Z}_q^{k \times n}$.
- 2) Dipilih secara acak kunci rahasia s dari bilangan bulat “kecil”.

- 3) Dipilih secara acak vektor *error* acak e dari bilangan bulat “kecil”.
- 4) Hitunglah $t = A \cdot s + e$
- 5) Hasilnya, kunci publik : $k_{pub} = (t, A)$ dan kunci privat :
 $k_{pr} = s$

Kunci publik : $k_{pub} = (t, A)$ dengan $t \in \mathbb{Z}_q^k$ dan $A \in \mathbb{Z}_q^{k \times n}$

Kunci privat : $k_{pr} = s \in \mathbb{Z}_q^n$

Enkripsi

Input :

Kunci publik : $k_{pub} = (t, A)$, pesan $m \in \{0,1\}$

Output :

ciphertext $c = (c_{aux}, c_{msg})$ dengan $c_{aux} \in \mathbb{Z}_q^n$ dan $c_{msg} \in \mathbb{Z}_q$

- 1) Sampling bilangan bulat acak kecil menjadi vektor r , e_{aux} dan nilai e_{msg} .
- 2) Kodekan pesan m : $\bar{m} = enc(m) \in \mathbb{Z}_q$
- 3) Hitung $c_{aux} = A^T \cdot r + e_{aux}$
- 4) Hitung $c_{msg} = t^T \cdot r + e_{msg} + \bar{m}$
- 5) Hasilnya ciphertext $c = (c_{aux}, c_{msg})$

Dekripsi

Input:

Kunci privat : $k_{pr} = s \in \mathbb{Z}_q^n$

ciphertext : $c = (c_{aux}, c_{msg})$

Output

Pesan $m \in \{0,1\}$

- 1) Hasilnya Pesan $m = dec(c_{msg} - s^T \cdot c_{aux})$

Melalui varian LWE simpel, diperkenalkan skema kunci publik berbasis latis pertama yang dibangun atas asumsi LWE. Perlu diingat bahwa skema ini tidak dirancang untuk digunakan dalam aplikasi dunia nyata. Uraian berikut menjelaskan kekurangan dari LWE simpel serta fitur utama

yang perlu dimiliki oleh skema enkripsi berbasis latis agar dapat diterapkan secara praktis.

1) Enkripsi multi-bit

LWE simpel hanya mendukung enkripsi bit tunggal. Untuk kasus penggunaan praktis, misalnya, transportasi kunci simetris yang aman, ini tidak cukup dan skema perlu diperluas ke enkripsi multi-bit.

2) Ekspansi *ciphertext*

Ekspansi *ciphertext* dalam skema enkripsi berbasis LWE adalah salah satu isu penting karena berdampak langsung pada efisiensi transmisi data dan penyimpanan. Ekspansi ini terjadi karena ukuran *ciphertext* yang jauh lebih besar dibandingkan dengan *plaintext*.

3) Kunci yang besar

LWE memerlukan matriks dan vektor dengan dimensi tinggi, yang membuat ukuran kunci, *ciphertext*, dan operasi aritmetika menjadi besar. Masalah ini berdampak pada efisiensi transmisi data dan konsumsi memori, terutama dalam perangkat dengan sumber daya terbatas seperti IoT.

4) Distribusi error

Ketika melihat pembuatan nilai *error* e dan r dalam LWE simpel, terlihat bahwa dihindari untuk menentukan bagaimana tepatnya nilai integer "kecil" yang diinginkan dipilih. Faktanya, pengambilan sampel kesalahan untuk sistem enkripsi berbasis LWE bukanlah hal yang mudah dan telah mendapat perhatian signifikan dalam literatur ilmiah. Beberapa distribusi D yang dapat digunakan untuk menghasilkan vektor kesalahan tersebut mencakup, misalnya, pengambilan sampel dari distribusi Gaussian atau binomial diskrit.

Skema enkripsi berbasis LWE yang praktis harus mampu mengenkripsi pesan multi-bit tanpa ekspansi *ciphertext*

yang berlebihan atau ukuran kunci yang sangat besar. Untuk mengurangi kebutuhan penyimpanan, terdapat opsi untuk membangun masalah LWE pada ring polinomial, yang disebut sebagai masalah Ring-LWE. Ring-LWE memungkinkan penggunaan polinomial tunggal sebagai ganti matriks, yang diperlukan untuk skema latis standar.

Meskipun polinomial memiliki nilai yang besar, polinomial tersebut masih jauh lebih efisien dalam hal penyimpanan dan komputasi daripada skema yang menggunakan matriks A , yang diperlukan untuk kriptosistem berbasis LWE.

d. Masalah Ring-LWE

Untuk memahami Ring-LWE, yang menggantikan operasi matriks dengan perhitungan menggunakan polinomial, pertama-tama harus diperkenalkan gagasan tentang ring polinomial. Ruang lingkup akan dibatasi pada kasus yang paling umum digunakan dalam kriptografi latis, yaitu skema yang didasarkan pada ring polinomial yang didefinisikan dengan apa yang disebut polinomial siklotomik. Lebih khusus lagi, digunakan polinomial siklotomik khusus dalam bentuk $x^{2^i} + 1$.

Definisi Ring Polinomial

$$\text{Ring } R_q = \mathbb{Z}_q[x]/(x^n + 1)$$

Ring polinomial $\mathbb{Z}_q[x]/(x^n + 1)$ terdiri dari semua polinomial dengan derajat maksimum $n-1$ dengan koefisien dari $\mathbb{Z}_q$ dan n merupakan pangkat dua, yaitu $n = 2^i$.

Operasi penjumlahan, pengurangan, dan perkalian pada ring dilakukan sebagai aritmatika polinomial biasa, dengan hasil yang direduksi modulus polinomial siklotomik $x^n + 1$. Semua koefisien integer direduksi modulus q .

Ring polinomial $\mathbb{Z}_q[x]/(x^n + 1)$ memiliki semua properti dari sebuah ring. Jika hasil perkalian menghasilkan polinomial dengan derajat n atau lebih besar, maka polinomial tersebut direduksi modulus $x^n + 1$.

Dalam konteks ini, bentuk khusus $x^n + 1$ dari modulus bermanfaat karena reduksi sangatlah mudah. Perlu diingat bahwa $x^n + 1 \equiv 0 \text{ mod } x^n + 1$ berlaku. Hal ini dapat ditulis ulang sebagai:

$$x^n \equiv -1 \text{ mod } x^n + 1$$

Dengan kata lain, setiap kali x^n muncul, x^n dapat digantikan oleh -1 saat melakukan reduksi modulus. Dari pengamatan ini juga dapat disimpulkan bahwa $x^{n+1} \equiv -x$ dan $x^{n+2} \equiv -x^2$ dan seterusnya jika dilakukan aritmatika modulus x^{n+1} .

Untuk membangun kriptosistem berbasis latis, fakta penting adalah bahwa ada kesamaan antara matriks dan ring polinomial terkait kemampuan mereka untuk menghasilkan struktur aljabar seperti latis. Tanpa membahas detail matematika, sekarang akan didefinisikan masalah LWE yang didasarkan pada ring polinomial, bukan latis standar yang direntangkan oleh matriks.

Definisi Masalah Ring-LWE

Misalnya R_q menyatakan Ring $\mathbb{Z}_q[x]/(x^n + 1)$ di mana q adalah bilangan prima dan bilangan bulat positif n adalah pangkat dua. Diberikan polinomial $\mathbf{a}$ dan $\mathbf{t} \in R_q$.

Ring-LWE adalah masalah menentukan polinomial rahasia $s \in R_q$ sedemikian rupa sehingga:

$$\mathbf{a}(x).s(x) + e(x) = \mathbf{t}(x)$$

di mana vektor *error* e adalah polinomial dalam ring R_q dengan koefisien integer kecil yang diperoleh dari distribusi diskrit D .

Perhatikan bahwa huruf tebal digunakan untuk polinomial dengan nilai koefisien besar seperti $\mathbf{a}(x)$, $\mathbf{t}(x) \in$

R_q dan huruf biasa digunakan untuk polinomial seperti $e(x)$, $s(x)$ yang hanya memiliki nilai kecil.

Koefisien besar atau kecil bisa didefinisikan berdasarkan ukuran relatif terhadap modulus yang digunakan dalam operasi latis. Polinomial dengan koefisien besar merupakan sebuah polinomial $\mathbf{a}(x)$ atau $\mathbf{t}(x)$ dimana koefisiennya bisa lebih besar dan berada dalam rentang yang lebih besar misalnya $\mathbf{a}(x) \in R_q$, dengan nilai koefisien bisa mendekati nilai q yang digunakan dalam operasi modulus. Sedangkan, polinomial dengan koefisien kecil adalah polinomial seperti $e(x)$, $s(x)$ dimana koefisiennya hanya sedikit dan terbatas pada nilai kecil dalam ruang modulus yang lebih kecil.

Dengan membandingkan definisi masalah Ring-LWE dengan masalah LWE, dapat dengan mudah melihat kesamaannya. Namun, Ring-LWE hanya memerlukan polinomial satu dimensi $\mathbf{a}(x)$ daripada matriks persegi besar A . Hal ini mengurangi persyaratan komputasi dan memori, sementara masalah Ring-LWE diasumsikan aman karena kemiripannya dengan LWE. Kriptosistem yang didasarkan pada Ring-LWE secara komputasi lebih efisien daripada LWE, tetapi memiliki kerentanan karena memiliki lebih banyak struktur matematika yang berpotensi dapat dieksploitasi melalui kriptanalisis.

Sebagai solusi untuk kerentanan tersebut, kelas ketiga kriptosistem berbasis latis telah diusulkan, yang menggabungkan konsep LWE berbasis matriks dan Ring-LWE berbasis polinomial menjadi latis berbasis modul (*Module-LWE*). *Module-LWE* telah mendapatkan banyak perhatian dalam proses standardisasi. Namun, kriptosistem ini cukup rumit dan untuk menyederhanakannya akan tetap digunakan masalah Ring-LWE yang akan memungkinkan untuk membangun sistem enkripsi berbasis latis yang praktis.

e. Skema enkripsi Ring-LWE

Dengan definisi dari masalah Ring-LWE, maka selanjutnya dapat dibahas mengenai kriptosistem berbasis latis praktis yang merupakan peningkatan signifikan atas skema LWE simpel. Perlu diingat kembali bahwa skema enkripsi berbasis latis di dunia riil terlihat cukup mirip tetapi sering kali memiliki penyesuaian dan pengoptimalan tersendiri.

Sistem enkripsi Ring-LWE yang disajikan berikut ini awalnya diusulkan oleh Lyubashevski, Lindner, dan Peikert [7]. Walaupun teridentifikasi terdapat kesamaan antara Ring-LWE dengan LWE simpel, namun banyak keterbatasan LWE simpel yang tidak ada dalam skema Ring-LWE.

Pembangkitan kunci Ring-LWE

Input :

Polinomial $\mathbf{a}(x) \in R_q$ dari ring $R_q = \mathbb{Z}[x]_q/(x^n + 1)$

Polinomial (x) , $s(x) \in R_q$ dengan koefisien integer kecil yang diperoleh dari distribusi error diskrit D

Output :

kunci publik: $k_{pub} = (\mathbf{t}, \mathbf{a})$ dan kunci privat: $k_{pr} = s$

- 1) Pilih $\mathbf{a}(x) \in R_q$ dari ring $R_q = \mathbb{Z}[x]_q/(x^n + 1)$.
- 2) Pilih $e(x)$, $s(x) \in R_q$ dengan koefisien integer kecil yang diperoleh dari distribusi error diskrit D .
- 3) Hitung $\mathbf{t}(x) = \mathbf{a}(x).s(x) + e(x) \in R_q$
- 4) Hasilnya kunci publik $k_{pub} = (\mathbf{t}, \mathbf{a})$ dan kunci privat $k_{pr} = s$

Enkripsi Ring-LWE

Input:

kunci publik $k_{pub} = (\mathbf{t}, \mathbf{a})$, pesan $m \in \{0,1\}^n$

Output:

Cipher text $c = (c_{aux}, c_{msg})$

- 1) Pilih polinomial error $r(x)$, $e_{aux}(x)$, $e_{msg}(x)$ dengan koefisien integer kecil yang diperoleh dari distribusi error diskrit D
- 2) Tulis n bit pesan m sebagai polinomial pesan $m(x)$ dan hasilkan polinomial yang dikodekan: $\bar{m}(x) = enc(m(x))$.
- 3) Hitung $c_{aux}(x) = a(x).r(x) + e_{aux}(x)$
- 4) Hitung $c_{msg}(x) = t(x).r(x) + e_{msg}(x) + \bar{m}(x)$
- 5) Hasilnya *ciphertext* $c = (c_{aux}, c_{msg})$

Dekripsi Ring-LWE

Input:

Kunci privat $k_{pr} = s$, *cipher text* $c = (c_{aux}, c_{msg})$

Output:

Pesan m

- 1) Hitung $m'(x) = c_{msg}(x) - c_{aux}(x).s(x)$
- 2) Hasilnya pesan $m = dec(m'(x))$

Panjang bit yang dibutuhkan skema tersebut adalah koefisien polinomial "besar" dibatasi oleh bilangan bulat q , yang tidak sebesar modulus dalam kasus kriptosistem RSA atau Elgamal, di mana diperlukan setidaknya 2048 bit untuk skema yang aman. Sebaliknya, dalam skema Ring-LWE, q memiliki panjang 10 – 20 bit, yang berarti sekitar 4 – 8 digit desimal. Contohnya, $q = 12,289$ digunakan untuk kriptosistem *NewHope*. Polinomial kecil seperti $s, e, r, e_{aux}, e_{msg}$ yang juga merupakan elemen dari Ring R_q tapi koefisien polinomial dipilih dari distribusi error diskrit D modulus q , yang memiliki nilai kecil. Sebuah contoh distribusi error berpusat antara 0 dengan nilai kecil, contohnya

$$D_d = \{-3, -2, -1, 0, 1, 2, 3\}_q = \{58, 59, 60, 0, 1, 2, 3\}_q \text{ dimana } q = 61.$$

Perhatikan bahwa perhitungan yang melibatkan polinomial besar dan kecil, misalnya perkalian

$$c_{aux}(x).s(x)$$

akan memiliki polinomial produk yang juga memiliki koefisien besar.

f. Implementasi LWE

Dengan Ring-LWE telah diperkenalkan masalah berbasis latris generik dan skema enkripsi. Lalu, bagaimanakah parameter aktual yang harus dipilih agar skema tersebut dapat menahan penyerang yang kuat. Karena kriptografi berbasis latris merupakan bidang kriptografi yang tergolong baru, kriptanalisis konstruksi berbasis latris yang efisien dan aman masih merupakan bidang penelitian yang aktif.

Parameterisasi secara signifikan lebih kompleks dibandingkan dengan skema RSA atau logaritma diskrit seperti pertukaran kunci Diffie-Hellman atau enkripsi Elgamal di mana keamanannya sangat bergantung pada satu parameter, yaitu modulus. Sebaliknya, terdapat interaksi beberapa parameter untuk skema Ring-LWE, termasuk ukuran polinomial, modulus q , dan distribusi error. Contohnya adalah contoh Ring-LWE yang diusulkan oleh Regev dengan parameter $n = 128$ dan $q = 16411$ menggunakan distribusi error Dq berdasarkan distribusi Gaussian diskrit dengan varians yang dipilih secara cermat ($\sigma = 29,6$). Konfigurasi khusus ini diasumsikan pada awal tahun 2015 untuk memberikan keamanan yang setara dengan kriptografi simetris 71-bit. Namun, setelah dua tahun penelitian dengan hasil kriptanalisis baru, ternyata parameter ini hanya menyediakan tingkat keamanan 48 bit, artinya, contoh Ring-LWE ini dapat dipecahkan dengan mudah.

Setelah mempelajari bahwa substitusi varian matriks LWE dengan varian polinomial (Ring-LWE) memiliki manfaat implementasi yang signifikan, khususnya berkenaan dengan persyaratan penyimpanan, perlu diketahui bahwa terdapat kelemahan pada skema enkripsi berbasis Ring-LWE. Meskipun, Ring-LWE dianggap aman hingga saat ini, dengan

asumsi parameter keamanan yang dipilih dengan benar, bukti keamanan yang menunjukkan bahwa LWE merupakan masalah yang sulit tidak sepenuhnya berlaku untuk Ring-LWE. Struktur polinomial siklotomiknya dapat memungkinkan serangan yang lebih efisien di masa depan.

Sebagai tanggapan terhadap hal itu, versi perantara LWE yang disebut *Module-LWE* telah diusulkan yang bertujuan untuk menggabungkan kedua varian dengan menerapkan struktur matriks LWE dengan efisiensi polinomial terstruktur sebagai bagian dari elemen matriksnya. Gambaran umum struktural dari ketiga versi LWE yang berbeda digambarkan dalam Gambar 2.2. Ini juga menyoroti berbagai persyaratan penyimpanan A , dari skema paling ringkas Ring-LWE (4 elemen polinomial) hingga LWE sebagai skema yang paling memakan memori (16 elemen dalam matriks).

Standard -LWE
over $\mathbb{Z}_7$

$$\begin{array}{c} \mathbf{A} \\ \begin{bmatrix} 0 & 6 & 2 & 1 \\ 1 & 5 & 0 & 5 \\ 3 & 3 & 3 & 2 \\ 0 & 1 & 5 & 5 \end{bmatrix} \end{array} \cdot \begin{array}{c} \mathbf{s} \\ \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} \end{array} + \begin{array}{c} \mathbf{e} \\ \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} \end{array} = \begin{array}{c} \mathbf{t} \\ \begin{bmatrix} 3 \\ 4 \\ 1 \\ 5 \end{bmatrix} \end{array}$$

Ring - LWE
over $\mathbb{Z}_7[x]/x^4 + 1$

$$\begin{array}{c} x^3 \quad x^2 \quad x \quad 1 \\ \begin{bmatrix} 6 & 2 & 4 & 0 \end{bmatrix} \end{array} \cdot \begin{array}{c} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} x^3 \\ \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} x^2 \\ \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} x \\ \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \end{array} + \begin{array}{c} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} x^3 \\ \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} x^2 \\ \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} x \\ \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} \end{array} = \begin{array}{c} \begin{bmatrix} 1 \\ 2 \\ 5 \\ 4 \end{bmatrix} x^3 \\ \begin{bmatrix} 1 \\ 2 \\ 5 \\ 4 \end{bmatrix} x^2 \\ \begin{bmatrix} 1 \\ 2 \\ 5 \\ 4 \end{bmatrix} x \\ \begin{bmatrix} 1 \\ 2 \\ 5 \\ 4 \end{bmatrix} \end{array}$$

Module - LWE
over $\mathbb{Z}_7[x]/x^2 + 1$

$$\begin{array}{c} x \quad 1 \quad x \quad 1 \\ \begin{bmatrix} \begin{bmatrix} 4 & 1 \end{bmatrix} & \begin{bmatrix} 3 & 5 \end{bmatrix} \\ \begin{bmatrix} 3 & 5 \end{bmatrix} & \begin{bmatrix} 4 & 3 \end{bmatrix} \end{bmatrix} \end{array} \cdot \begin{array}{c} \begin{bmatrix} 1 \\ 0 \end{bmatrix} x \\ \begin{bmatrix} 1 \\ 1 \end{bmatrix} \end{array} + \begin{array}{c} \begin{bmatrix} 1 \\ 1 \end{bmatrix} x \\ \begin{bmatrix} 1 \\ 0 \end{bmatrix} \end{array} = \begin{array}{c} \begin{bmatrix} 6 \\ 0 \end{bmatrix} x \\ \begin{bmatrix} 6 \\ 3 \end{bmatrix} \end{array}$$

Gambar 3.2 Perbedaan struktural antara varian LWE, Ring-LWE dan *Module-LWE*
Digambar ulang dari [1].

Seperti semua skema kunci publik, setiap sistem kriptografi berbasis LWE juga lebih lambat beberapa kali lipat dibandingkan kriptografi simetris seperti AES. Oleh karena itu, aplikasi utama enkripsi PQC adalah pembentukan kunci antara dua pihak melalui saluran yang tidak aman (bukan untuk mengenkripsi data massal). Kunci tersebut kemudian digunakan untuk kriptografi simetris seperti AES. Dalam praktiknya, pembentukan kunci ini sering kali diwujudkan melalui mekanisme enkapsulasi kunci (*Key Encapsulation Mechanism*).

Tabel 3.1 Set parameter untuk skema enkapsulasi kunci berbasis LWE

Skema	Tipe	Keamanan yang Ekuivalen	n	k	q	δ
Kyber-512	Module-LWE	AES-128	256	2	3329	2^{-139}
Kyber-768	Module-LWE	AES-192	256	3	3329	2^{-164}
Kyber-1024	Module-LWE	AES-256	256	4	3329	2^{-174}
NewHope-512	Ring-LWE	AES-128	512	1	12289	2^{-213}
NewHope-1024	Ring-LWE	AES-256	1024	1	12289	2^{-216}
FrodoKEM-640	LWE	AES-128	640	1	2^{15}	$2^{-138,7}$
FrodoKEM-1340	LWE	AES-256	256	1	2^{16}	$2^{-252,5}$

Dengan:

- n menunjukkan jumlah koefisien dan derajat polinomial dari LWE.
- k merepresentasikan dimensi latas sebagai kelipatan n .

- q adalah modulus untuk ring $\mathbb{Z}_q$.
- δ adalah probabilitas kegagalan dekripsi karena kesalahan dalam proses *decoding*.

Set parameter dari Tabel 3.1 dirancang untuk menyediakan tingkat keamanan yang setara dengan AES dengan kunci 128, 192, dan 256-bit. Kriptosistem *NewHope* (Ring-LWE) dan Kyber (*Module-LWE*) keduanya memiliki banyak fitur yang sama dengan skema Ring-LWE dasar yang telah dibahas sebelumnya. Kedua skema tersebut menggunakan beberapa pengoptimalan komputasi untuk meningkatkan kinerja dan mengurangi ukuran *ciphertext*. Seperti dalam kasus skema kunci publik, kinerja selalu terkait dengan parameterisasi kriptosistem dan terdapat kompromi antara keamanan dan efisiensi. Skema enkripsi berbasis latis yang lebih konservatif FRODO-KEM didasarkan pada LWE.

2. Algoritma PQC berbasis Latis

Beberapa kriptosistem berbasis latis saat ini telah distandardisasi oleh NIST [8]. Kriptosistem ini populer karena lebih serbaguna daripada skema PQC lainnya dan memberikan keseimbangan yang baik antara argumen keamanan dan kekuatan kriptanalisis terhadap serangan klasik dan kuantum. Skema berbasis latis dan skema PQC lainnya yang mendukung enkripsi tidak dirancang untuk enkripsi data massal melainkan untuk pengangkutan kunci, sehingga umumnya hanya mendukung *Key Encapsulation Mechanism*. Algoritma PQC berbasis latis yang telah distandardisasi oleh NIST, yaitu FIPS 203 (ML-KEM) dan FIPS 204 (ML-DSA). Selain dua skema tersebut, NIST sedang dalam tahap perumusan FIPS 206 (Falcon) dan Hawk.

Secara teori, Hawk, yang kini berada pada round 2, memiliki keunggulan dibandingkan dengan ML-DSA. Keunggulannya terletak pada ukuran kunci dan tanda tangan Hawk yang lebih kecil dibandingkan dengan ML-DSA, dan Hawk juga membutuhkan sumber daya yang lebih rendah. Oleh karena itu, Hawk lebih cepat

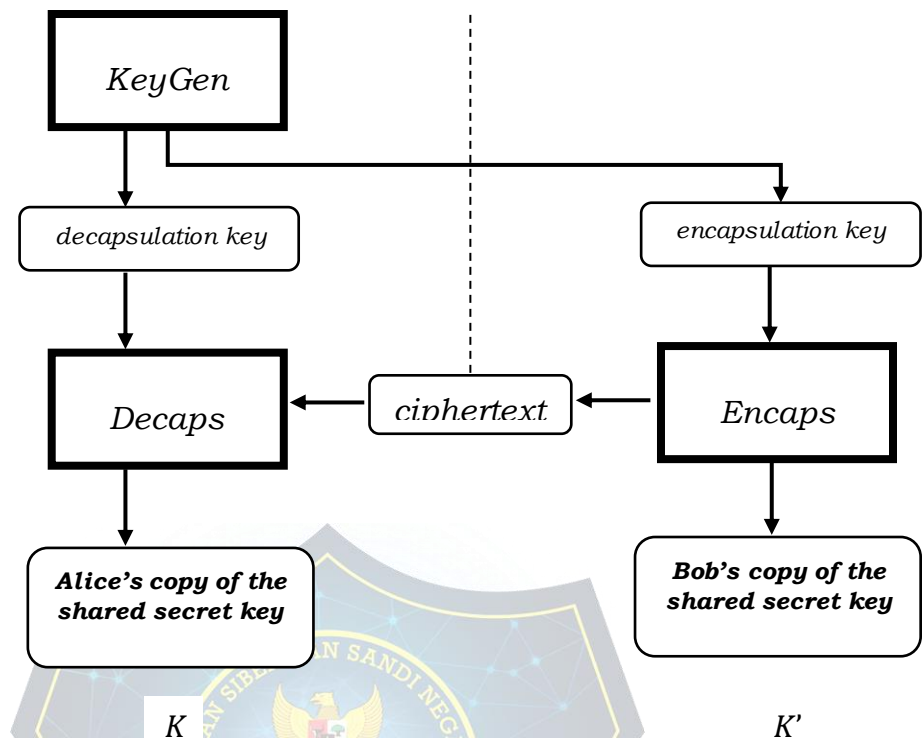
dan lebih tepat digunakan pada perangkat dengan kapasitas yang memiliki keterbatasan dalam hal daya pemrosesan, memori atau *bandwidth*. Meskipun demikian, kompatibilitas Hawk belum seluas ML-DSA, yang telah lebih banyak diterima dan diimplementasikan pada berbagai platform dan perangkat, terutama yang memiliki kapasitas lebih besar.

a. ML-KEM

1) KEM

KEM adalah suatu skema kriptografi yang dapat digunakan untuk membentuk sebuah kunci rahasia bersama (*shared secret key*) antara dua pihak yang berkomunikasi. *Shared secret key* tersebut kemudian dapat digunakan untuk kriptografi kunci simetris. ML-KEM mengadopsi pendekatan KEM yang terdiri dari tiga algoritma utama dan kumpulan set parameter [9]. Tiga algoritma tersebut, yaitu:

- a) algoritma probabilistik untuk pembangkitan pasangan kunci (kunci publik dan kunci privat) dinotasikan dengan *KeyGen*.
- b) algoritma probabilistik untuk pembungkusan kunci (*encapsulation*), yang menghasilkan *ciphertext* dan *shared secret key*, dinotasikan dengan *Encaps*.
- c) algoritma deterministik untuk pembukaan kunci (*decapsulation*), yang mengekstraksi *shared secret key* dari *ciphertext* menggunakan kunci privat, dinotasikan dengan *Decaps*.



Gambar 3.3 Konsep sederhana *Key establishment* menggunakan KEM
Digambar ulang dari [9].

Pada aplikasinya, KEM digunakan untuk menyediakan *shared-secret key* antara dua pihak, pada gambar 3.3 adalah Alice dan Bob, dengan alur sebagai berikut:

a) *Key Generation* (ML-KEM.KeyGen)

Alice memulai dengan menjalankan KeyGen untuk menghasilkan pasangan kunci, yaitu:

- 1) kunci enkapsulasi (*encapsulation key*) yang bisa dibagikan;
- 2) kunci dekapsulasi (*decapsulation key*) yang dirahasiakan.

b) *Encapsulation* (ML-KEM.Encaps)

Setelah memperoleh kunci enkapsulasi Alice, Bob menjalankan algoritma Encaps, yang menghasilkan salinan K milik Bob dari kunci rahasia bersama

(*shared secret key*) beserta *ciphertext* terkait. Bob mengirimkan *ciphertext* tersebut kepada Alice.

c) *Decapsulation* (ML-KEM.Decaps)

Alice menyelesaikan proses tersebut dengan menjalankan algoritma Decaps menggunakan kunci dekapsulasi dan *ciphertext* miliknya. Langkah terakhir ini menghasilkan salinan K' milik Alice dari kunci rahasia bersama.

Setelah menyelesaikan proses ini, Alice dan Bob ingin menyimpulkan bahwa *output* mereka memenuhi $K' = K$ dan bahwa nilai ini adalah kunci rahasia bersama yang aman dan acak. Namun, sifat-sifat ini hanya berlaku jika kondisi penting tertentu terpenuhi, seperti yang dibahas dalam SP 800-227.

Contoh:

ML-KEM beroperasi dalam ring polinomial $R_q = \mathbb{Z}[X]/(X^n + 1)$ dengan parameter $q = 3329$ dan $n = 256$. Setiap elemen dalam R_q direpresentasikan sebagai *array* dari koefisien polynomial modulus q .

a) *Key-Generation*

Pilih polinomial acak $a \in R_q$ yang akan digunakan dalam perhitungan kunci publik

Pilih polinomial acak $s, e \in R_q$ dengan distribusi kesalahan

Hitung kunci publik $pk = a \cdot s + e$

Kunci privat adalah pasangan (a, s)

b) *Key-Encapsulation*

Pilih polinomial acak $r \in R_q$ dan polynomial kesalahan $e' \in R_q$

Hitung *ciphertext* $c = (c_1, c_2)$ dengan

$$c_1 = a \cdot r + e', c_2 = r \cdot pk + e'$$

Kunci Bersama adalah $k' = r$

c) *Key-Decapsulation*

Hitung $u = c_1 \cdot s$

Hitung kunci Bersama $k' = c_2 - u$

Jika k' valid maka $k' = k$

2) Skema ML-KEM

ML-KEM adalah algoritma KEM berbasis *Module-LWE* yang memungkinkan dua pihak membentuk sebuah *shared-secret key* melalui saluran publik. Berikut ini merupakan deskripsi singkat dan tidak formal mengenai asumsi komputasional yang mendasari ML-KEM serta cara skema ML-KEM dibangun.

a) Asumsi Komputasional

Keamanan ML-KEM didasarkan pada kesulitan komputasional dari permasalahan yang disebut *Module-LWE*. Kesulitan dari permasalahan *Module-LWE* sendiri bergantung pada kesulitan permasalahan komputasional tertentu dalam *module* latis. Hal inilah yang menjadi dasar penamaan skema ini sebagai *Module Lattice-Based Key Encapsulation Mechanism* (ML-KEM).

Dalam permasalahan *LWE*, input adalah sekumpulan persamaan linear acak “*error*” yang melibatkan beberapa variabel rahasia $x \in \mathbb{Z}_q^n$ dan tugas yang harus dilakukan adalah memulihkan nilai x tersebut. *Error* dalam persamaan ini dirancang sedemikian rupa sehingga algoritma standar seperti eliminasi Gaussian menjadi tidak dapat diselesaikan secara efisien. Permasalahan *LWE* ini secara alami dapat digunakan untuk aplikasi kriptografi. Sebagai contoh, jika x dianggap sebagai kunci rahasia, maka seseorang dapat

mengenklripsi sebuah 1 bit *plaintext* dengan cara berikut:

- jika *plaintext* bernilai nol, maka diambil persamaan linear yang mendekati benar,
- jika *plaintext* bernilai satu, maka diambil persamaan linear yang jauh dari benar.

Dengan demikian, secara intuitif, hanya pihak yang memiliki kunci rahasia x yang dapat membedakan antara kedua jenis persamaan tersebut. Selanjutnya, proses enkripsi dapat didelegasikan kepada pihak lain dengan cara mempublikasikan sekumpulan besar persamaan linear *error*, yang dapat dikombinasikan secara tepat oleh pihak yang melakukan enkripsi. Hasil akhirnya adalah sebuah skema enkripsi asimetris (*public-key encryption scheme*).

Permasalahan *Module-LWE* memiliki kemiripan dengan permasalahan *LWE*, namun terdapat perbedaan penting. Dalam *Module-LWE*, $\mathbb{Z}_q^n$ ruang digantikan oleh R_q^n yang dibentuk dengan mengambil produk Kartesius sebanyak k kali dari ring polinomial R_q tertentu. Dengan demikian, rahasia dalam permasalahan *Module-LWE* adalah sebuah elemen x dari modul R_q^k .

b) Konstruksi ML-KEM

Secara garis besar, konstruksi skema ML-KEM dilakukan melalui dua tahap utama. Pertama, ide-ide yang telah dijelaskan sebelumnya digunakan untuk membangun skema enkripsi kunci publik (*Public-Key Encryption* / PKE) yang didasarkan pada permasalahan *Module-LWE*. Kedua, skema PKE tersebut kemudian diubah menjadi mekanisme pembungkusan kunci (*Key-Encapsulation*

Mechanism/KEM) dengan menggunakan transformasi Fujisaki–Okamoto (*FO transform*). Karena properti tertentu dari *FO transform*, KEM yang dihasilkan memiliki tingkat keamanan yang lebih tinggi dan tahan terhadap model serangan yang jauh lebih umum dibandingkan dengan skema PKE. Hasilnya, ML-KEM diyakini memenuhi keamanan tingkat IND-CCA2².

Spesifikasi algoritma ML-KEM dalam FIPS 203 mengikuti pola tersebut. FIPS 203 terlebih dahulu mendeskripsikan skema enkripsi kunci publik yang disebut K-PKE dan kemudian menggunakan algoritma-algoritma dari K-PKE sebagai subrutin dalam menjelaskan algoritma ML-KEM. Transformasi kriptografi dari K-PKE ke ML-KEM merupakan elemen krusial untuk mencapai keamanan IND-CCA2. Skema K-PKE tidak memiliki keamanan IND-CCA2 dan tidak boleh digunakan sebagai skema yang berdiri sendiri.

Fitur penting lainnya dari ML-KEM adalah penggunaan transformasi bilangan teoretis atau *Number-Theoretic Transform* (NTT). Transformasi NTT mengubah suatu polinomial $f \in R_q$ menjadi representasi alternatif berupa vektor f yang terdiri atas polinomial-polinomial linear. Bekerja menggunakan representasi NTT memungkinkan perkalian polinomial dilakukan jauh lebih cepat. Sementara itu, operasi lain seperti penjumlahan, pembulatan (*rounding*), dan sampling dapat dilakukan baik pada representasi asli maupun

² IND-CCA2 adalah tingkat keamanan yang memastikan bahwa meskipun seorang penyerang bisa memilih *ciphertext* dan mendapatkan informasi dari hasil dekripsi, ia tetap tidak bisa mempelajari informasi rahasia yang terkandung dalam sistem

representasi NTT, tergantung kebutuhan implementasi.

ML-KEM memenuhi properti esensial KEM, yaitu ketepatan (*correctness*). Artinya, selama tidak terjadi korupsi data atau interferensi, maka proses seperti yang digambarkan pada Gambar 3.3 akan menghasilkan $K' = K$ dengan probabilitas yang sangat tinggi. Selain itu, ML-KEM memiliki bukti keamanan teoretis asimtotik dalam model heuristik tertentu. Setiap set parameter ML-KEM (ML-KEM-512, ML-KEM-768, dan ML-KEM-1024) memiliki tingkat kekuatan keamanan yang diasosiasikan berdasarkan kriptanalisis terkini.

c) Set Parameter

Skema ML-KEM disertai dengan tiga set parameter, yaitu:

- 1) ML-KEM-512 – (kategori keamanan 1)
- 2) ML-KEM-768 – (kategori keamanan 3)
- 3) ML-KEM-1024 – (kategori keamanan 5)

Kategori keamanan 1 s.d. 5 mengacu pada definisi dalam NIST *Special Publication* 800-57 Bagian 1, yang memberikan pemetaan antara tingkat keamanan kriptografi asimetris dan simetris.

Skema ML-KEM dilengkapi dengan tiga Set Parameter yang masing-masing terdiri dari lima parameter variable yaitu k , η_1 , η_2 , du , dan dv . Selain itu, terdapat dua konstanta tetap $n = 256$ dan $q = 3329$. Nilai dari kelima variabel tersebut untuk masing-masing set parameter tercantum pada Tabel 3.2.

Berikut adalah penjelasan singkat mengenai peran masing-masing parameter variabel dalam algoritma K-PKE dan ML-KEM.

a) k – ukuran kunci

Parameter k menentukan panjang kunci bersama yang dihasilkan oleh algoritma. Dalam ML-KEM, nilai k biasanya adalah 256 bit. Ukuran k yang lebih besar meningkatkan keamanan, namun juga mempengaruhi kinerja algoritma.

Parameter k menentukan dimensi dari matriks $\hat{A}$ yang digunakan dalam algoritma K-PKE.KeyGen dan K-PKE.Encrypt. Parameter ini juga menentukan dimensi vector s dan e dalam K-PKE.KeyGen, serta dimensi vector y dan e_1 dalam K-PKE.Encrypt.

b) η_1 dan η_2 – parameter distribusi error

Parameter η_1 dan η_2 mengontrol distribusi error dalam polinomial yang digunakan dalam proses enkripsi dan dekripsi. Distribusi ini penting untuk memastikan bahwa masalah MLWE tetap sulit dipecahkan.

Parameter η_1 digunakan untuk menentukan distribusi probabilitas yang digunakan dalam pembangkitan vector s dan e pada K-PKE.KeyGen, serta vektor y pada K-PKE.Encrypt.

Parameter η_2 digunakan untuk menentukan distribusi probabilitas bagi pembangkitan vector e_1 dan e_2 dalam K-PKE Encrypt.

c) d_u dan d_v – parameter ukuran matriks

Parameter d_u dan d_v menentukan ukuran matriks dalam operasi linier yang digunakan

dalam algoritma. Ukuran matriks ini mempengaruhi efisiensi komputasi dan keamanan algoritma.

Parameter ini digunakan sebagai input untuk fungsi-fungsi Compress, Decompress, ByteEncode, dan ByteDecode yang terdapat pada algoritma K-PKE.Encrypt dan K-PKE.Decrypt.

d) n – derajat polinomial

Parameter n merujuk pada derajat polinomial dalam ring $R_q = \mathbb{Z}[X]/(X^n + 1)$. Dalam ML-KEM, nilai n adalah 256, sehingga polinomial dalam R_q memiliki derajat hingga 255. Pemilihan n yang tepat mempengaruhi efisiensi komputasi dan keamanan algoritma.

e) q – modulus

Parameter q adalah bilangan prima besar yang digunakan sebagai modulus dalam operasi polinomial. Nilai q yang umum digunakan dalam ML-KEM adalah 3329. Pemilihan nilai q mempengaruhi ukuran ciphertext dan tingkat keamanan terhadap serangan.

FIPS 203 menyetujui set parameter sebagaimana terdapat pada tabel 3.2. Setiap set parameter dikaitkan dengan tingkat kekuatan keamanan tertentu yang diperlukan untuk pembangkitan keacakan. Ukuran kunci ML-KEM dan *ciphertext* untuk setiap set parameter terdapat pada tabel 3.3.

Tabel 3.2 Set parameter untuk skema ML-KEM yang disetujui

	n	q	k	η_1	η_2	du	dv	Kekuatan RBG yang dipersyaratkan (bit)
ML-KEM-512	256	3329	2	3	2	10	4	128
ML-KEM-768	256	3329	3	2	2	10	4	192
ML-KEM-1024	256	3329	4	2	2	11	5	256

Tabel 3.3 Ukuran (dalam *byte*) kunci dan *ciphertext* ML-KEM

	Kunci enkapsulasi	Kunci Dekapsulasi	Ciphertext	Shared Secret Key
ML-KEM-512	800	1632	768	32
ML-KEM-768	1184	2400	1088	32
ML-KEM-1024	1568	3168	1568	32

Nama suatu set parameter juga dapat digunakan untuk menyatakan sebuah KEM tanpa parameter tambahan. Secara khusus, ML-KEM- x dapat digunakan untuk merujuk pada KEM tanpa parameter yang dihasilkan dari instansiasi skema ML-KEM dengan set parameter ML-KEM- x .

Tiga set parameter yang tercantum dalam Tabel 3.2 dirancang untuk memenuhi kategori kekuatan keamanan tertentu yang ditetapkan oleh NIST dalam *Call for Proposals* aslinya. Dalam pendekatan ini, kekuatan keamanan tidak dinyatakan dengan satu

nilai numerik tunggal, seperti “128 bit keamanan.” Sebaliknya, setiap set parameter ML-KEM diklaim memiliki tingkat keamanan paling sedikit setara dengan *Block cipher* umum dengan ukuran kunci tertentu, atau fungsi *hash* umum dengan panjang *output* tertentu.

Secara lebih tepat, diklaim bahwa sumber daya komputasi yang diperlukan untuk memecahkan ML-KEM adalah lebih besar atau sama dengan sumber daya komputasi yang diperlukan untuk memecahkan *block cipher* atau fungsi *hash* pembandingan, jika diestimasi menggunakan model komputasi yang realistis.

Perlu dicatat bahwa model-model komputasi yang berbeda dapat memiliki tingkat realisme yang berbeda pula, sehingga dapat menghasilkan estimasi kekuatan keamanan yang lebih akurat atau kurang akurat. Beberapa model komputasi yang umum dibahas terdapat dalam NIST IR 8413.

Pemilihan Set Parameter yang Tepat

Ketika pertama kali menetapkan proteksi kriptografi terhadap data, set parameter terkuat yang tersedia sebaiknya digunakan. Pendekatan ini memberikan sejumlah keuntungan, antara lain:

- a) Mengurangi kemungkinan kebutuhan migrasi mahal ke himpunan parameter dengan tingkat keamanan lebih tinggi di masa depan, dan
- b) Menjamin margin keamanan yang lebih luas terhadap serangan masa depan, termasuk serangan kuantum. Namun, perlu diperhatikan bahwa penggunaan set parameter dengan tingkat keamanan lebih tinggi juga dapat berdampak pada kinerja sistem, misalnya

proses algoritma menjadi lebih lambat, atau ukuran objek kriptografi (misalnya kunci dan *ciphertext*) menjadi lebih besar dari yang dapat diterima oleh aplikasi.

NIST merekomendasikan penggunaan ML-KEM-768 sebagai set parameter standar, karena menyediakan margin keamanan yang besar dengan biaya kinerja yang masih wajar. Namun, dalam kondisi tertentu jika kinerja menjadi kendala utama, maka dapat digunakan ML-KEM-512. Jika diperlukan keamanan yang lebih tinggi (misalnya untuk data jangka panjang atau aplikasi kritikal), maka dapat digunakan ML-KEM-1024.

- d) **Kegagalan Dekapsulasi**
 Probabilitas kegagalan dekapsulasi terjadi apabila seluruh input sudah lulus pengecekan dan pembangkitan keacakan juga sukses namun menghasilkan output $K' \neq K$. Probabilitas kegagalan dekapsulasi dapat dilihat pada tabel 3.4.

Tabel 3.4 Rata rata kegagalan dekapsulasi pada ML-KEM

Set parameter	Rata – rata kegagalan dekapsulasi
ML-KEM-512	$2^{-138.8}$
ML-KEM-768	$2^{-164.8}$
ML-KEM-1024	$2^{-174.8}$

- e) **Terminologi untuk Kunci**
 KEM melibatkan tiga jenis kunci yang berbeda yaitu kunci enkapsulasi, kunci dekapsulasi, dan kunci rahasia bersama (*shared-secret key*). ML-KEM

dibangun di atas skema enkripsi kunci publik komponen K-PKE, yang memiliki dua jenis kunci tambahan yaitu kunci enkripsi dan kunci dekripsi. Dalam literatur, kunci enkapsulasi dan kunci enkripsi terkadang disebut sebagai "kunci publik", sedangkan kunci dekapsulasi dan kunci dekripsi terkadang disebut sebagai "kunci privat". Untuk mengurangi kebingungan, pada FIPS 203 tidak akan menggunakan istilah "kunci publik" atau "kunci privat". Sebaliknya, kunci akan dirujuk hanya menggunakan istilah yang lebih spesifik, yaitu, salah satu dari "kunci enkapsulasi", "kunci dekapsulasi", "kunci enkripsi", "kunci dekripsi", dan "*shared-secret key*".

3) Persyaratan untuk implementasi ML-KEM

a) K-PKE hanyalah sebuah komponen

Skema enkripsi kunci publik K-PKE yang dijelaskan pada tidak boleh digunakan sebagai skema kriptografi yang berdiri sendiri (*stand-alone cryptographic scheme*). Sebaliknya, algoritma-algoritma yang membentuk K-PKE hanya boleh digunakan sebagai subrutin di dalam algoritma-algoritma milik ML-KEM. Secara khusus, algoritma K-PKE.KeyGen, K-PKE.Encrypt dan K-PKE.Decrypt tidak disetujui untuk digunakan sebagai skema PKE yang terpisah.

b) Akses terkontrol ke fungsi internal

Mekanisme enkapsulasi kunci ML-KEM menggunakan fungsi-fungsi internal "*derandomized*", yaitu ML-KEM.KeyGen_internal, ML-KEM.Encaps_internal, dan ML-KEM.Decaps_internal. Antarmuka untuk fungsi-fungsi internal tersebut tidak boleh disediakan

untuk aplikasi lain, kecuali untuk keperluan pengujian. Secara khusus, proses pengambilan nilai acak yang diperlukan untuk pembangkitan kunci sebagaimana ditentukan dalam ML-KEM.KeyGen, dan enkapsulasi kunci sebagaimana ditentukan dalam ML-KEM.Encaps, harus dilakukan oleh modul kriptografi.

Ketentuan ini memastikan bahwa sumber acak yang digunakan dalam proses pembangkitan dan enkapsulasi kunci tidak dikontrol oleh aplikasi eksternal, melainkan dikelola secara aman oleh modul kriptografi bersertifikat (misalnya bersertifikat FIPS 140-3). Hal ini penting untuk mencegah kelemahan pada keamanan implementasi akibat *randomness reuse* atau bias dalam sumber acak.

c) Implementasi ekuivalen

Untuk setiap algoritma yang ditetapkan dalam FIPS 203, suatu implementasi yang sesuai dapat mengganti urutan langkah-langkah yang diberikan dengan urutan langkah-langkah lain yang secara matematis ekuivalen. Dengan kata lain, algoritma yang ditentukan dalam FIPS 203 dapat digantikan oleh prosedur berbeda sepanjang prosedur tersebut menghasilkan *output* yang benar untuk setiap input (istilah “input” mencakup masukan yang ditentukan secara eksplisit, serta seluruh nilai parameter dan semua nilai acak yang digunakan dalam proses).

d) Penggunaan yang disetujui untuk *shared-secret key*

Apabila proses pembangkitan bilangan acak (*randomness generation*) berhasil, maka nilai K yang dihasilkan oleh ML-KEM.Encaps dan nilai K' yang dihasilkan oleh ML-KEM.Decaps selalu berupa nilai 256-bit. Dalam kondisi yang sesuai, kedua nilai

tersebut akan sama dan dapat digunakan langsung sebagai *shared secret key* untuk kriptografi simetris. Apabila diperlukan proses *key derivation* lebih lanjut maka kunci simetris yang final harus diturunkan dari *shared secret key* 256-bit ini dengan cara yang telah disetujui sebagaimana ditentukan dalam NIST SP 800-108 dan NIST SP 800-56C.

Sebagaimana dibahas sebelumnya, ML-KEM merupakan KEM yang memiliki keamanan IND-CCA2. Namun demikian, apabila ML-KEM digabungkan dengan mekanisme lain dalam suatu KEM gabungan maka tingkat keamanan IND-CCA2 tidak selalu terjamin. Oleh karena itu, developer dan implementor harus mengevaluasi tingkat keamanan dari setiap prosedur apabila metode derivasi kunci sebagaimana ditetapkan dalam NIST SP 800-56C diterapkan terhadap ML-KEM dalam kombinasi dengan mekanisme *key establishment* lainnya. Panduan tambahan mengenai KEM gabungan dapat ditemukan dalam NIST SP 800-227.

e) **Pembangkitan keacakan**

Terdapat dua algoritma dalam FIPS 203 yang memerlukan pembangkitan bilangan acak sebagai langkah internal, yaitu ML-KEM.KeyGen, dan ML-KEM.Encaps. Dalam *pseudocode*, proses pembangkitan bilangan acak ini dinyatakan dengan pernyataan dalam bentuk: $m \xleftarrow{\$} \mathbb{B}$. Untuk setiap pemanggilan (*invocation*) pernyataan tersebut, harus dihasilkan *string* baru berisi *byte* acak (*fresh string of random bytes*). Byte acak ini harus dihasilkan menggunakan *Random Bit Generator* (RBG) yang disetujui sebagaimana ditetapkan dalam NIST SP 800-90A, SP 800-90B, dan SP 800-90C. Selain itu,

RBG yang digunakan harus memiliki security strength paling sedikit 128 bit untuk ML-KEM-512, 192 bit untuk ML-KEM-768, dan 256 bit untuk ML-KEM-1024.

f) Pemeriksaan input

Algoritma ML-KEM.Encaps dan ML-KEM.Decaps memerlukan pemeriksaan input. Implementor wajib memastikan bahwa ML-KEM.Encaps dan ML-KEM.Decaps hanya dijalankan terhadap input yang telah diperiksa.

g) Dekstruksi nilai *intermediate*

Data yang digunakan dalam tahapan komputasi antara (*intermediate computation steps*) dari algoritma KEM dapat dimanfaatkan oleh pihak lawan untuk mengompromikan keamanan. Oleh karena itu, implementor wajib memastikan bahwa data *intermediate* dimusnahkan segera setelah tidak lagi diperlukan. Secara khusus, untuk algoritma ML-KEM.KeyGen, ML-KEM.Encaps, dan ML-KEM.Decaps, hanya output yang ditetapkan secara resmi yang boleh disimpan di memori setelah algoritma selesai dijalankan. Seluruh data lainnya harus dimusnahkan sebelum algoritma berakhir.

Terdapat dua pengecualian terhadap aturan tersebut:

- *Seed* (d, z) yang dihasilkan pada Langkah 1 dan 2 dari ML-KEM.KeyGen dapat disimpan untuk digunakan kembali di kemudian hari melalui ML-KEM.KeyGen_internal. Karena *seed* ini dapat digunakan untuk menghitung kunci dekapsulasi maka *seed* tersebut termasuk data sensitif dan harus diperlakukan dengan tingkat

perlindungan yang sama seperti kunci dekapsulasi.

- Matriks $\hat{A}$ yang dihasilkan pada Langkah 3–7 dari K-PKE.KeyGen (sebagai subrutin dari ML-KEM.KeyGen) dapat disimpan agar tidak perlu dihitung ulang pada operasi berikutnya (misalnya dalam ML-KEM.Decaps). Matriks $\hat{A}$ yang sama juga dihasilkan pada Langkah 4–8 dari K-PKE.Encrypt (sebagai subrutin dari ML-KEM.Encaps atau ML-KEM.Decaps) dan juga dapat disimpan. Dalam kedua kasus tersebut, Matriks $\hat{A}$ merupakan data yang dapat dihitung dengan mudah dari kunci enkapsulasi dan tidak memerlukan perlindungan khusus.

h) Larangan penggunaan aritmatika *floating-point*
Implementasi ML-KEM tidak boleh menggunakan aritmatika *floating-point*, karena pada kondisi tertentu kesalahan pembulatan pada operasi aritmatika *floating-point* dapat menyebabkan hasil yang tidak benar

4) Algoritma Pendukung

a) Fungsi kriptografis

Algoritma-algoritma yang ditetapkan dalam FIPS 203 memerlukan penggunaan beberapa fungsi kriptografi. Setiap fungsi tersebut harus diinstansiasi menggunakan fungsi *hash* yang disetujui atau XOF yang disetujui sebagaimana ditetapkan di bawah ini. Fungsi *hash* dan XOF yang relevan dijelaskan secara rinci dalam FIPS 202.

Fungsi Hash

SHA3-256 dan SHA3-512 merupakan fungsi *hash* yang memiliki satu input dengan panjang variabel dan satu *output* dengan panjang tetap (*fixed-length output*). Pemanggilan kedua fungsi tersebut terhadap suatu input M akan dinyatakan sebagai $\text{SHA3-256}(M)$ dan $\text{SHA3-512}(M)$. Input dan output untuk SHA3-256 dan SHA3-512 selalu berupa *byte arrays*. Spesifikasi pada ML-KEM menggunakan tiga fungsi *hash*, yaitu H , J , dan G , yang didefinisikan sebagai berikut.

Fungsi H dan J masing-masing menerima satu input dengan panjang variabel dan menghasilkan satu *output* sepanjang 32 *byte*. Kedua fungsi ini dinotasikan dengan $H: \mathbb{B}^* \rightarrow \mathbb{B}^{32}$ dan $J: \mathbb{B}^* \rightarrow \mathbb{B}^{32}$, masing-masing dan diinstansiasi sebagai

$$H(s) := \text{SHA3-256}(s) \text{ dan } J(s) := \text{SHAKE256}(s, 8 \cdot 32)$$

dengan $s \in \mathbb{B}^*$.

Fungsi G menerima satu input dengan panjang variabel dan menghasilkan dua *output*, masing-masing sepanjang 32 *byte*. Fungsi ini dinotasikan dengan $G: \mathbb{B}^* \rightarrow \mathbb{B}^{32} \times \mathbb{B}^{32}$. Dua *output* dari fungsi G dilambangkan sebagai $(a, b) \leftarrow G(c)$, dengan $a, b \in \mathbb{B}^{32}$, $c \in \mathbb{B}^*$ dan $G(c) = a \parallel b$. Fungsi G diinstansiasi sebagai berikut:

$$G(c) := \text{SHA3-512}(c).$$

Fungsi *eXtendable-Output Function* (XOF)

SHAKE128 dan SHAKE256 adalah fungsi XOF yang masing-masing memiliki satu input dengan panjang variabel dan satu *output* dengan panjang variabel. Standar ini mengikuti konvensi berikut sebagaimana diuraikan dalam FIPS 202, yaitu:

- Input dan output untuk SHAKE128 dan SHAKE256 selalu berupa *byte arrays*
- Saat memanggil fungsi SHAKE128 atau SHAKE256, panjang output yang diinginkan selalu ditentukan dalam satuan bit.

Sebagai contoh, ekspresi berikut:

$$r := \text{SHAKE128}(M, 8 \cdot 64)$$

menunjukkan bahwa M adalah *array byte*, dan r adalah *array* berisi 64 *byte*.

FIPS 203 menggunakan XOF *wrapper* yang didefinisikan berdasarkan antarmuka pemrograman bertahap (*incremental API*) untuk SHAKE128 sebagaimana dijelaskan dalam NIST SP 800-185. API SHAKE128 tersebut terdiri atas tiga fungsi, yaitu:

- $\text{ctx} \leftarrow \text{SHAKE128.Init}()$
Menginisialisasi konteks XOF (*context*) dengan nama ctx .
- $\text{ctx} \leftarrow \text{SHAKE128.Absorb}(\text{ctx}, \text{str})$
Menginjeksi data yang akan digunakan pada fase *absorbing* dari SHAKE128, dan memperbarui konteks ctx sesuai dengan data yang dimasukkan.
- $(\text{ctx}, B) \leftarrow \text{SHAKE128.Squeeze}(\text{ctx}, 8 \cdot z)$
Mengekstraksi z *byte output* yang dihasilkan selama fase *squeezing* dari SHAKE128, serta memperbarui konteks ctx sesuai dengan output yang dihasilkan.

Walaupun fungsi-fungsi di atas dibangun menggunakan permutasi Keccak- f , dan bukan secara langsung dari XOF SHAKE128, fungsi-fungsi tersebut didefinisikan sedemikian sehingga satu

pemanggilan SHAKE128 secara tunggal dalam bentuk:

$$\text{output} \leftarrow \text{SHAKE128}(str_1 \parallel \dots \parallel str_m, 8b_1 + \dots + 8b_\ell)$$

ekuivalen dengan menjalankan Algoritma 2 dalam FIPS 203. Ekuivalensi ini tetap berlaku, terlepas dari apakah panjang $|str_i|$ dan nilai b_j merupakan kelipatan dari panjang blok SHAKE128 ataupun tidak.

Fungsi-fungsi yang disebutkan di atas memainkan berbagai peran dalam algoritma yang ditentukan dalam FIPS 203, dan hanya akan dipanggil melalui *wrapper functions*. Hal ini dilakukan untuk menghindari potensi kebingungan antara “array byte” dan “bit-length”, dengan memastikan bahwa seluruh pemrosesan hanya dilakukan pada level *byte* dan menggunakan panjang dalam satuan *array byte*.

Pseudorandom function (PRF)

Fungsi PRF menerima parameter $\eta \in \{2,3\}$, satu input sepanjang 32 *byte* dan satu input sepanjang 1 *byte*. Fungsi ini menghasilkan output sepanjang $(64 \cdot \eta)$ *byte*. Fungsi ini dinotasikan sebagai berikut:

$$\text{PRF} : \{2,3\} \times \mathbb{B}^{32} \times \mathbb{B} \rightarrow \mathbb{B}^{64\eta}$$

dan harus diinstasiasi dengan

$$\text{PRF}_\eta(s, b) := \text{SHAKE256}(s \parallel , 8.64.\eta)$$

dengan $\eta \in \{2,3\}$, $s \in \mathbb{B}^{32}$, dan $b \in \mathbb{B}$

Perlu dicatat bahwa parameter η hanya digunakan untuk menentukan panjang output yang diinginkan, dan tidak digunakan untuk pemisahan domain.

b) Algoritma umum

Bagian ini menjelaskan bagaimana data diubah dan direpresentasikan di dalam ML-KEM dari bit, *byte*,

hingga integer yang dihitung dalam ruang modular $q = 3329$. Tujuannya adalah untuk memastikan interoperabilitas, efisiensi ukuran data, dan keakuratan komputasi di seluruh tahapan algoritma.

Algoritma konversi

Bagian ini menetapkan beberapa algoritma untuk melakukan konversi antara *array bit*, *array byte* dan *array* integer modulus m . Algoritma BitsToBytes dan BytesToBits digunakan untuk mengubah data antara bentuk *array bit* dan *array byte*. BitsToBytes mengubah setiap 8 bit menjadi 1 *byte* dalam urutan *little-endian* (bit terkecil di posisi awal). Sedangkan BytesToBits memecah setiap *byte* menjadi 8 bit. Contohnya, *string* 8-bit 11010001 dikonversi menjadi *byte* bernilai $2^0 + 2^1 + 2^3 + 2^7 = 139$

Kompresi dan dekompresi

Bagian ini juga menetapkan suatu operasi untuk melakukan kompresi terhadap bilangan bulat modulus q , serta operasi dekompresi yang bersesuaian. Karena $q = 3329$, dan Panjang bit dari q adalah 12. Maka untuk $d < 12$, dilakukan algoritma Compress_d untuk memperkecil representasi bilangan menjadi hanya d bit dan Decompress_d untuk mengembalikannya ke bentuk semula dengan perkiraan yang sangat dekat.

Input dan output dari fungsi ini adalah integer module m . Pembagian dan pembulatan pada komputasi fungsi ini dilakukan pada himpunan bilangan rasional dan komputasi *floating-point* tidak diperbolehkan.

Algoritma kompresi dan dekompresi harus memenuhi 2 properti penting. Pertama, jika dilakukan dekompresi lalu kompresi maka hasilnya

selalu sama. Jika d cukup besar (mendekati 12), nilai hasil dengan kompresi yang diikuti dekompresi hampir identik dengan nilai aslinya.

Encoding dan Decoding

Algoritma ByteEncode dan ByteDecode digunakan untuk mengonversi antara integer modulus m dan *bytes*. Algoritma ByteEncode mengubah *array* yang berisi $n=256$ integer modulus m menjadi *array byte* yang bersesuaian. Algoritma ByteDecode kebalikannya, mengonversi kembali *array byte* menjadi *array* integer modulus m .

Untuk penjelasan berikut ini, akan lebih mudah apabila ByteDecode dan ByteEncode dipandang sebagai proses konversi antara integer dan bit. Konversi antara bit dan *byte* bersifat langsung dan dilakukan dengan menggunakan BitsToBytes dan BytesToBits. Rentang nilai yang valid untuk parameter d adalah $1 \leq d \leq 12$. *Array* bit dibagi menjadi segmen-segmen berukuran d bit. Operasi dilakukan dengan dua cara berbeda, bergantung pada nilai d :

- Untuk nilai d yang memenuhi $1 \leq d \leq 11$, konversi bersifat satu-ke-satu. ByteDecode_d mengonversi setiap segmen d -bit dari inputnya menjadi satu integer modulus 2^d sedangkan ByteEncode_d melakukan operasi inversnya.
- Untuk nilai $d = 12$, ByteDecode_{12} menghasilkan integer modulus q sebagai output, sedangkan ByteEncode_{12} menerima integer modulus q sebagai input. Secara khusus, ByteDecode_{12} mengonversi setiap segmen sepanjang 12 bit dari inputnya menjadi integer modulus $2^{12} =$

4096 dan kemudian hasilnya direduksi terhadap modulus q . Proses ini tidak lagi bersifat satu-ke-satu karena beberapa segmen 12 bit dapat merepresentasikan bilangan yang lebih besar daripada $q - 1 = 3328$ namun lebih kecil dari 4096. Namun demikian, kondisi tersebut tidak dapat terjadi untuk *array* yang dihasilkan oleh ByteEncode_{12} . Aspek-aspek ini dari ByteDecode_{12} dan ByteEncode_{12} menjadi penting ketika melakukan pemeriksaan terhadap kunci enkapsulasi ML-KEM.

Algoritma sampling

Algoritma-algoritma dalam ML-KEM memerlukan dua subrutin sampling. Kedua algoritma ini dapat digunakan untuk mengonversi aliran *byte* acak yang terdistribusi *uniform* (*uniformly random bytes*) menjadi sampel dari suatu distribusi tertentu yang diinginkan (*desired distribution*). Dalam FIPS 203, kedua algoritma tersebut akan dipanggil dengan input berupa aliran *byte pseudorandom*. Akibatnya, output yang dihasilkan merupakan sampel dari distribusi yang secara komputasional tidak dapat dibedakan (*computationally indistinguishable*) dari distribusi yang diinginkan.

Sampling *uniform* pada representasi NTT

Algoritma SampleNTT mengonversi sebuah *seed* acak bersama dengan dua *indexing bytes* menjadi sebuah polinomial dalam domain NTT. Apabila *seed* acak tersebut terdistribusi secara *uniform*, maka polinomial yang dihasilkan akan diambil dari *computationally indistinguishable* dari distribusi *uniform* pada T_q . Output dari SampleNTT adalah

sebuah *array* $\mathbb{Z}_q^{256}$ yang berisi koefisien-koefisien dari elemen yang disampling dalam ruang T_q .

Sampling dari distribusi binomial terpusat

ML-KEM menggunakan suatu distribusi khusus yang disebut $\mathcal{D}_\eta(R_q)$ yaitu distribusi polinomial dalam R_q dengan koefisien kecil. Polinomial-polinomial tersebut terkadang disebut sebagai “error” atau “noise”. Distribusi ini diparameterisasi oleh sebuah integer $\eta \in \{2,3\}$.

c) The *Number-Theoretic Transforms* (NTT)

NTT adalah bentuk khusus dari *Discrete Fourier Transform* (DFT). Dalam ML-KEM, NTT digunakan untuk meningkatkan efisiensi operasi perkalian dalam Ring R_q . Perlu diingat bahwa R_q adalah ring $R_q = \mathbb{Z}_q[X]/(X^n + 1)$ yang terdiri atas polinomial dengan bentuk $f = f_0 + f_1X + \dots + f_{255}X^{255}$ dimana setiap $f_j \in \mathbb{Z}_q$ untuk seluruh j , dan operasi dalam ring ini didefinisikan berdasarkan aritmetika modulus $X^n + 1$.

Ring R_q isomorfis terhadap Ring T_q , yang merupakan penjumlahan langsung dari 128 ekstensi kuadratik dari $\mathbb{Z}_q$. NTT adalah isomorfisme yang efisien secara komputasi antara kedua ring tersebut. Ketika polinomial $f \in R_q$ menjadi input, maka output NTT adalah elemen $\hat{f} := \text{NTT}(f)$ dari Ring T_q yang disebut sebagai representasi NTT dari f . Sifat isomorfisme ini menyatakan bahwa:

$$f \times_{R_q} g = \text{NTT}^{-1}(\hat{f} \times_{T_q} \hat{g})$$

dengan $\times_{R_q}$ dan $\times_{T_q}$ masing-masing menyatakan operasi perkalian dalam R_q dan T_q .

Karena T_q merupakan hasil kali dari 128 ring yang masing-masing terdiri atas polinomial berderajat paling tinggi satu, maka operasi $\times T_q$ jauh lebih efisien dibandingkan dengan operasi $\times R_q$. Oleh sebab itu, NTT dianggap sebagai komponen integral dari ML-KEM, bukan sekadar optimisasi.

5) Skema komponen K-PKE

K-PKE terdiri atas tiga algoritma utama, yaitu:

- a) Pembangkitan kunci ($K\text{-PKE.KeyGen}$)
- b) Enkripsi ($K\text{-PKE.Encrypt}$)
- c) Dekripsi ($K\text{-PKE.Decrypt}$)

Ketika K-PKE digunakan sebagai bagian dari ML-KEM, maka K-PKE mewarisi parameter set yang dipilih untuk ML-KEM. Setiap set parameter menentukan nilai numerik untuk setiap parameter. Meskipun nilai n selalu 256 dan nilai q selalu 3329, nilai dari parameter lainnya yaitu k, η_1, η_2, du , dan dv berbeda-beda pada ketiga set parameter. Parameter-parameter tersebut, beserta set parameternya, dijelaskan secara rinci pada bagian Set Parameter.

Algoritma yang menjadi bagian dari skema komponen K-PKE tidak melakukan *input checking*, karena algoritma tersebut hanya digunakan sebagai subrutin dari algoritma utama ML-KEM. Adapun algoritma-algoritma utama ML-KEM melakukan *input checking* sesuai kebutuhan. Setiap algoritma dalam K-PKE disertai dengan uraian singkat dan informal dalam bentuk teks untuk memudahkan pemahaman. Untuk kesederhanaan, uraian ini ditulis dalam bentuk vektor dan matriks yang elemennya merupakan elemen-elemen dari Ring R_q . Namun, dalam implementasi sebenarnya, sebagian besar komputasi dilakukan dalam domain NTT guna

meningkatkan efisiensi perkalian. Dengan demikian, vektor dan matriks yang relevan memiliki elemen-elemen dalam T_q . Selain itu, kunci enkripsi dan kunci dekripsi dari K-PKE juga disimpan dalam bentuk representasi NTT (NTT form).

Berikut ini penjelasan dari setiap algoritma pada skema komponen K-PKE.

a) Pembangkitan kunci K-PKE

Algoritma pembangkitan kunci K-PKE.KeyGen menerima sebuah *seed* sebagai input dan menghasilkan dua *output* yaitu kunci enkripsi ek_{PKE} dan kunci dekripsi dk_{PKE} . Sebagaimana halnya pada sistem enkripsi kunci publik (*public-key encryption*), kunci enkripsi dapat dipublikasikan secara bebas, sedangkan kunci dekripsi dan *seed* (nilai acak/*randomness*) harus dijaga kerahasiaannya. Khusus dalam konteks ML-KEM, Kunci enkripsi K-PKE akan digunakan sebagai kunci enkapsulasi pada ML-KEM. Kunci dekripsi serta *seed* yang digunakan oleh K-PKE.KeyGen harus tetap bersifat rahasia, karena keduanya dapat digunakan untuk melakukan dekapsulasi pada ML-KEM.

Algorithm 13 K-PKE.KeyGen (d)

Uses randomness to generate an encryption key and a corresponding decryption key.

Input: randomness $d \in B^{32}$.

Output : encryption key $ek_{PKE} \in B^{384k+32}$.

Output : decryption key $dk_{PKE} \in B^{384k}$.

Output: vector $\mathbf{y} \in \mathbb{F}^n$ with $\mathcal{F}(\mathbf{y}) = \mathbf{x}$.

$((\rho, \sigma) \leftarrow G(d \parallel k) \triangleright$ expand 32+1 bytes to two pseudorandom 32-byte seeds

$N \leftarrow 0$

for ($i \leftarrow 0; i < k; i++$) $\triangleright$ generate matrix $\hat{A} \in (\mathbb{Z}_q^{256})^{k \times k}$

for ($j \leftarrow 0; j < k; j++$)

$\hat{A}[i, j] \leftarrow \text{SampleNTT}(p \parallel j \parallel i) \triangleright j$ and i are bytes 33 and 34 of the input

end for

```

end for

for ( $i \leftarrow 0; i < k; i++$ ) ▷ generate matrix  $\hat{A} \in (\mathbb{Z}_q^{256})^k$ 
 $s[i] \leftarrow \text{SamplePolyCBD}_{n_1}(\text{PRF}_{n_1}(\sigma, N))$  ▷  $e[i] \in (\mathbb{Z}_q^{256})$  sampled from CBD
 $N \leftarrow N + 1$ 
end for

 $\hat{s} \leftarrow \text{NTT}(s)$  ▷ run NTT  $k$  times (once for each coordinate of  $s$ )
 $\hat{e} \leftarrow \text{NTT}(e)$  ▷ run NTT  $k$  times
 $\hat{t} \leftarrow \hat{A} \cdot \hat{s} + \hat{e}$  ▷ noisy linear system in NTT domain
 $ek_{\text{PKE}} \leftarrow \text{ByteEncode}_{12}(\hat{t}) \parallel \rho$  ▷ run  $\text{ByteEncode}_{12}$   $k$  times, then append  $\hat{A}$ -seed
 $dk_{\text{PKE}} \leftarrow \text{ByteEncode}_{12}(\hat{s})$  ▷ run  $\text{ByteEncode}_{12}$   $k$  times
return ( $ek_{\text{PKE}}, dk_{\text{PKE}}$ )

```

Uraian Informal

Kunci dekripsi pada K-PKE.KeyGen merupakan vektor sepanjang k , yang elemennya merupakan elemen-elemen dari Ring R_q . Secara konseptual, s merupakan sekumpulan variabel rahasia sedangkan kunci enkripsi merupakan sekumpulan persamaan linear "dengan error" dalam bentuk $(A, As + e)$ dengan variabel rahasia s . Baris-baris dari matriks A berfungsi sebagai koefisien persamaan. Matriks ini dibangkitkan secara *pseudorandom* menggunakan fungsi XOF hanya berdasarkan *seed* yang disimpan di dalam kunci enkripsi. Vektor rahasia s dan vektor error e disampling dari distribusi binomial terpusat menggunakan keacakan yang diekspansi dari *seed* lain σ melalui fungsi PRF. Setelah A , s , dan e dihasilkan, maka bagian terakhir dari kunci enkripsi dihitung sebagai $t = As + e$. Hasil perhitungan kemudian dikodekan ke dalam *array byte* dan dijadikan output dari algoritma.

Dalam algoritma K-PKE.KeyGen, pemilihan set parameter memengaruhi panjang vektor rahasia s (melalui parameter k), ukuran vektor *error* e serta ukuran matriks *pseudorandom* A . Selain itu, pemilihan set parameter juga menentukan

distribusi *error* melalui parameter η_1 yang digunakan untuk men-*sample* entri-entri dari s , dan e .

b) Enkripsi K-PKE

Algoritma enkripsi K-PKE.Encrypt dari K-PKE menerima tiga input, yaitu kunci enkripsi ek_{PKE} , plaintext m sepanjang 32 *byte* dan nilai acak r dan menghasilkan satu output yaitu *ciphertext* c . Matriks $\hat{A}$ dapat disimpan sebagaimana dijelaskan dalam bagian persyaratan untuk implementasi ML-KEM. Penyimpanan ini memungkinkan operasi berikutnya untuk menggunakan $\hat{A}$ secara langsung, tanpa perlu melakukan ekspansi ulang dari *seed* publik ρ .

Algorithm 14 K-PKE.Decrypt(ek_{PKE}, m, r) Uses the encryption key to encrypt a plaintext message using the randomness r. Input: encryption key $ek_{PKE} \in \mathbb{B}^{384k+32}$. Input: message $m \in \mathbb{B}^{32}$. Input: randomness $r \in \mathbb{B}^{32}$. Output: ciphertext $c \in \mathbb{B}^{32(d_u+k+d_v)}$. $N \leftarrow 0$ $\hat{t} \leftarrow \text{ByteDecode}_{12}(ek_{PKE}[0:384k]) \triangleright$ run ByteDecode_{12} k times to decode $\hat{t} \in (\mathbb{Z}_q^{256})^k$ $\rho \leftarrow ek_{PKE}[384k:384k+32] \triangleright$ extract 32-byte seed from ek_{PKE} for ($i \leftarrow 0; i < k; i++$) $\triangleright$ re-generate matrix $\hat{A} \in (\mathbb{Z}_q^{256})^{k \times k}$ samples in Alg. 13 for ($j \leftarrow 0; j < k; j++$) $\hat{A}[i, j] \leftarrow \text{SampleNTT}(\rho j i) \triangleright j$ and i are bytes 33 and 34 of the input end for end for for ($i \leftarrow 0; i < k; i++$) $\triangleright$ generate $y \in (\mathbb{Z}_q^{256})^k$

```

 $y[i] \leftarrow \text{SamplePolyCBD}_{\eta_1}(\text{PRF}_{\eta_1}(r, N)) \triangleright y[i] \in (\mathbb{Z}_q^{256})^k \text{ sampled from}$ 
CBD
 $N \leftarrow N + 1$ 
end for
for  $(i \leftarrow 0; i < k; i++)$   $\triangleright$  generate  $e_1 \in (\mathbb{Z}_q^{256})^k$ 
 $e_1[i] \leftarrow \text{SamplePolyCBD}_{\eta_2}(\text{PRF}_{\eta_2}(r, N)) \triangleright e_1[i] \in (\mathbb{Z}_q^{256})^k \text{ sampled from}$ 
CBD
 $N \leftarrow N + 1$ 
end for
 $e_2 \leftarrow \text{SamplePolyCBD}_{\eta_2}(\text{PRF}_{\eta_2}(r, N)) \triangleright e_2 \in (\mathbb{Z}_q^{256})^k \text{ sampled from}$ 
CBD
 $\hat{y} \leftarrow \text{NTT}(y)$   $\triangleright$  run NTT  $k$  times
 $u \leftarrow \text{NTT}^{-1}(\hat{A}^T \cdot \hat{y}) + e_1$   $\triangleright$  run  $\text{NTT}^{-1}$   $k$  times
 $\mu \leftarrow \text{Decompress}_1(\text{ByteDecode}_1(m))$ 
 $v \leftarrow \text{NTT}^{-1}(\hat{A}^T \cdot \hat{y}) + e_2 + \mu$   $\triangleright$  encode plaintext  $m$  into polynomial  $v$ 
 $c_1 \leftarrow \text{ByteEncode}_{d_u}(\text{Compress}_{d_u}(u))$   $\triangleright$  run  $\text{ByteEncode}_{d_u}$  and  $\text{Compress}_{d_u}$   $k$  times
 $c_1 \leftarrow \text{ByteEncode}_{d_u}(\text{Compress}_{d_u}(v))$ 
return  $c \leftarrow (c_1 || c_2)$ 

```

Uraian Informal

Algoritma K-PKE.Encrypt dimulai dengan mengekstraksi vektor t dan $seed$ dari kunci enkripsi. $Seed$ tersebut kemudian diekspansi ulang untuk membangkitkan kembali matriks A dengan cara yang sama seperti pada K-PKE.KeyGen. Apabila t dan A diturunkan secara benar dari kunci enkripsi yang dihasilkan oleh K-PKE.KeyGen, maka keduanya akan identik dengan nilai-nilai yang sesuai pada proses K-PKE.KeyGen. Sebagaimana dijelaskan pada bagian pembangkitan kunci, pasangan $(A, t = As + e)$ dapat dianggap sebagai sistem persamaan linear dengan error dalam variabel rahasia s .

Seseorang dapat membangkitkan persamaan linear dengan error tambahan dalam variabel rahasia yang sama — tanpa mengetahui s dengan cara memilih kombinasi linear acak dari persamaan-persamaan dengan error di dalam sistem (A, t) . Kemudian, informasi dapat dikodekan pada “konstanta” dari persamaan gabungan tersebut yaitu, pada entri yang merupakan kombinasi linear dari entri-entri t . Informasi ini kemudian dapat didekripsi oleh pihak yang memiliki s . Sebagai contoh, seseorang dapat mengodekan satu bit dengan menentukan apakah konstanta tersebut diubah secara signifikan atau tidak. Jika tidak diubah secara signifikan, maka persamaan menjadi hampir benar (*nearly correct*) dan akan didekripsi sebagai bit 0. Jika diubah secara signifikan, maka persamaan menjadi jauh dari benar (*far-from-correct*) dan akan didekripsi sebagai bit 1. Dalam kasus K-PKE, konstanta tersebut bukan sekadar sebuah bilangan, melainkan sebuah polinomial dengan 256 koefisien, sehingga memungkinkan untuk mengodekan lebih banyak informasi satu bit pada setiap koefisien.

Untuk tujuan tersebut, K-PKE.Encrypt melakukan langkah-langkah berikut:

1. Membangkitkan vektor acak $y \in R_q^k$
2. Membangkitkan *error* $e_1 \in R_q^k$ dan $e_2 \in R_q^k$
Keduanya disampling dari distribusi binomial terpusat menggunakan *pseudorandomness* yang diekspansi melalui PRF dari input keacakan r .
3. Menghitung “persamaan dengan *error* baru” $A^T y + e_1, t^T y + e_2$
4. Menambahkan hasil pengkodean pesan μ dari *plaintext* m ke suku terakhir dari pasangan tersebut.
5. Pasangan hasil perhitungan, yaitu (u, v) , kemudian dikompresi, disusun ulang ke dalam *array byte*, dan dihasilkan sebagai *ciphertext*.

c) Dekripsi K-PKE

Algoritma K-PKE.Decrypt menerima kunci dekripsi dk_{PKE} dan *ciphertext* c sebagai input, tidak memerlukan nilai acak, serta menghasilkan *output* berupa *plaintext* m .

Algorithm 15 K-PKE.Decrypt(dk_{PKE}, c)
Uses the decryption key to decrypt a ciphertext
Input: decryption key $dk_{PKE} \in \mathbb{B}^{384k}$.
Input: ciphertext $c \in \mathbb{B}^{32(d_u k + d_v)}$.
Output: message $m \in \mathbb{B}^{32}$.
$c_1 \leftarrow c[0 : 32d_u k]$ $c_2 \leftarrow c[32d_u k : 32(d_u k + d_v)]$ $u' \leftarrow \text{Decompress}_{d_u}(\text{ByteDecode}_{d_u}(c_1)) \triangleright \text{run Decompress}_{d_u} \text{ and } \text{ByteDecode}_{d_u} k \text{ times}$ $v' \leftarrow \text{Decompress}_{d_v}(\text{ByteDecode}_{d_v}(c_2))$ $\hat{s} \leftarrow \text{ByteDecode}_{12}(dk_{PKE}) \triangleright \text{run ByteDecode}_{12} k \text{ times}$ $w \leftarrow v' - \text{NTT}^{-1}(\hat{s}^T \circ \text{NTT}(u')) \triangleright \text{run NTT } k \text{ times; run } \text{NTT}^{-1} \text{ once}$ $m \leftarrow \text{ByteEncode}_1(\text{Compress}_1(w)) \triangleright \text{decode plaintext } m \text{ from polynomial } v$ return m

Uraian Informal

Algoritma K-PKE.Decrypt dimulai dengan mengambil kembali pasangan (u', v') dari *ciphertext* c . Dalam konteks ini u' dapat dipandang sebagai koefisien-koefisien persamaan sedangkan v' merupakan konstanta dari persamaan tersebut. Kunci dekripsi dk_{PKE} mengandung vektor variabel rahasia s . Dengan demikian, algoritma dekripsi dapat menggunakan kunci ini untuk menghitung konstanta sebenarnya $v = s^T u'$ dan menghitung $v' - v$. Tahapan akhir dari algoritma adalah mendekode pesan *plaintext* m dari hasil selisih $v' - v$., kemudian mengeluarkan m sebagai hasil dekripsi.

6) Algoritma Internal Utama

Bagian ini menetapkan tiga algoritma berikut:

- a) ML-KEM.KeyGen_internal
- b) ML-KEM.Encaps_internal
- c) ML-KEM.Decaps_internal

Ketiga algoritma tersebut bersifat deterministik, yang berarti output sepenuhnya ditentukan oleh input. Tidak ada proses pengambilan nilai acak (*randomness sampling*) yang dilakukan di dalam algoritma-algoritma ini. Ketiga algoritma ini akan digunakan untuk membangun skema ML-KEM. Algoritma dalam bagian ini menggunakan parameter-parameter berikut n, q, k, du dan dv . Selain itu, subrutin yang dipanggil oleh algoritma-algoritma ini juga menggunakan parameter tambahan η_1 dan η_2 . Nilai n selalu 256 dan nilai q selalu 3329, sementara nilai parameter lainnya berbeda-beda tergantung pada set parameter.

Antarmuka yang ditetapkan dalam bagian ini akan digunakan untuk pengujian implementasi ML-KEM melalui *Cryptographic Algorithm Validation Program* (CAVP). Fungsi pembangkitan kunci yang dijelaskan di bagian ini juga dapat digunakan untuk mengembangkan kembali (*re-expand*) kunci dari sebuah *seed*, termasuk dalam konteks pembuktian kepemilikan kunci privat melalui regenerasi. Antarmuka yang dijelaskan di bagian ini tidak boleh tersedia untuk aplikasi umum, selain untuk tujuan pengujian. Selain itu, *seed* acak (sebagaimana ditentukan dalam *ML-KEM.KeyGen* dan *ML-KEM.Encaps*) harus dibangkitkan oleh modul kriptografi.

- a) Pembangkitan kunci internal

Algoritma ML-KEM.KeyGen_internal menerima dua buah *seed* acak sebagai input dan menghasilkan kunci enkapsulasi serta kunci dekapsulasi.

Uraian Informal

Subrutin inti dari ML-KEM.KeyGen_internal adalah algoritma pembangkitan kunci dari K-PKE. Kunci enkapsulasi secara langsung merupakan kunci enkripsi dari K-PKE. Kunci dekapsulasi terdiri atas kunci dekripsi dari K-PKE, Kunci enkapsulasi, *hash* dari kunci enkapsulasi, dan nilai acak sepanjang 32 *byte*. Nilai acak sepanjang 32 *byte* tersebut akan digunakan dalam mekanisme “*implicit rejection*” pada algoritma internal dekapsulasi.

Algorithm 16 ML-KEM.KeyGen_internal(d, z)	
Uses randomness to generate an encapsulation key and a corresponding decapsulation key.	
Input: randomness $d \in \mathbb{B}^{32}$.	
Input: randomness $z \in \mathbb{B}^{32}$.	
Output: encapsulation key $ek \in \mathbb{B}^{384k+32}$.	
Output: decapsulation key $dk \in \mathbb{B}^{768k+96}$.	
$(ek_{PKE}, dk_{PKE}) \leftarrow K\text{-}PKE.\text{KeyGen}(d)$ $\triangleright$ run key generation for K-PKE	
$ek \leftarrow ek_{PKE}$ $\triangleright$ KEM encaps key is just the PKE encryption key	
$dk \leftarrow (dk_{PKE} ek H(ek) z)$ $\triangleright$ KEM decaps key includes PKE decryption key	
return (ek,dk)	

b) Enkapsulasi internal

Algoritma ML-KEM.Encaps_internal menerima kunci enkapsulasi dan *array byte* acak sebagai input, serta menghasilkan dua output yaitu *ciphertext* dan *shared secret key*.

Algorithm 17 ML-KEM.Encaps_internal(ek, m)

Uses the encapsulation key and randomness to generate a key and an associated ciphertext.

Input: encapsulation key $ek \in \mathbb{B}^{384k+32}$.

Input: randomness $m \in \mathbb{B}^{32}$.

Output: shared secret key $K \in \mathbb{B}^{32}$.

Output: ciphertext $c \in \mathbb{B}^{32(d_u k + d_v)}$.

$(K, r) \leftarrow G(m || H(ek)) \triangleright$ derive shared secret key K and randomness r

$c \leftarrow K - \text{PKE.Encrypt}(ek, m, r) \triangleright$ encrypt m using K-PKE with randomness r

return (K, c)

Uraian informal

Subrutin inti dari ML-KEM.Encaps_internal adalah algoritma enkripsi dari K-PKE, yang digunakan untuk mengenkripsi nilai acak m menjadi sebuah *ciphertext* c . Salinan dari *shared secret key* K dan nilai acak yang digunakan selama proses enkripsi diturunkan dari m dan kunci enkapsulasi ek melalui proses *hashing*. Secara khusus, kunci enkapsulasi ek di-hash dengan Fungsi hash H dan hasil dari fungsi hash tersebut kemudian di-concat dengan nilai acak m . Gabungan hasil tersebut kemudian di-hash menggunakan fungsi hash G . Akhirnya, algoritma menghasilkan dua output, yaitu *shared secret key* K , dan *ciphertext* c .

c) Dekapsulasi internal

Algoritma ML-KEM.Decaps_internal menerima kunci dekapsulasi dan *ciphertext* sebagai input, tidak menggunakan nilai acak apa pun dan menghasilkan output berupa *shared secret key*.

Algorithm 18 ML-KEM.Decaps_internal(dk, c)

Uses the decapsulation key to produce a shared secret key from a ciphertext.

Input: decapsulation key $dk \in \mathbb{B}^{768k+96}$.

Input: ciphertext $c \in \mathbb{B}^{32(d_u k + d_v)}$.

Output: shared secret key $K \in \mathbb{B}^{32}$.

```

 $dk_{PKE} \leftarrow dk[0 : 384k]$   $\triangleright$  extract (from KEM decaps key) the PKE decryption
key
 $ek_{PKE} \leftarrow dk[384k : 768k + 32]$   $\triangleright$  extract PKE encryption
key
 $h \leftarrow dk[768k + 32 : 768k + 64]$   $\triangleright$  extract hash of PKE encryption
key
 $z \leftarrow dk[768k + 64 : 768k + 96]$   $\triangleright$  extract implicit rejection value
 $m' \leftarrow \text{K-PKE.Decrypt}(dk_{PKE}, c)$   $\triangleright$  decrypt ciphertext
 $(K', r') \leftarrow G(m' \parallel h)$ 
 $\bar{K} \leftarrow (z \parallel c)$ 
 $c' \leftarrow \text{K-PKE.Encrypt}(ek_{PKE}, m', r')$   $\triangleright$  re-encrypt using the derived randomness
 $r'$ 
if  $c \neq c'$  then
     $K' \leftarrow \bar{K}$   $\triangleright$  if ciphertext do not match, "implicitly
    reject"
end if
return  $K'$ 

```

Uraian Informal

Algoritma ML-KEM.Decaps_internal dimulai dengan memisahkan komponen-komponen kunci dekapsulasi dk milik ML-KEM. Komponen-komponen tersebut adalah pasangan kunci (kunci enkripsi dan kunci dekripsi) dari K-PKE, nilai *hash* h , dan nilai acak z . Selanjutnya, kunci dekripsi dari K-PKE digunakan untuk mendekripsi *ciphertext* input c guna memperoleh *plaintext* m' .

Kemudian, algoritma dekapsulasi melakukan proses enkripsi ulang (*re-encryption*) terhadap m' , dan menghitung calon kunci rahasia bersama (*candidate shared secret key*) K' dengan cara yang sama sebagaimana dilakukan dalam proses enkapsulasi.

Secara khusus nilai K' dan nilai acak enkripsi r' dihitung melalui proses *hashing* terhadap m' dan kunci enkripsi K-PKE. *Ciphertext* baru c' kemudian dihasilkan dengan mengenkripsi m' menggunakan nilai acak r' . Pada tahap akhir, algoritma dekapsulasi memeriksa apakah *ciphertext* yang dihasilkan c' sama dengan *ciphertext* yang diberikan c . Jika keduanya tidak sama, algoritma melakukan mekanisme "*implicit rejection*" nilai K' diganti dengan hasil *hash* dari c yang di-*concat* dengan nilai acak z yang tersimpan dalam kunci rahasia ML-KEM. (Lihat pembahasan tentang kegagalan dekapsulasi).

Dalam kedua kasus — baik cocok maupun tidak — algoritma akan mengeluarkan output *shared secret key* hasil akhir K' .

Bendera status "*implicit reject*", yang dihitung pada langkah ke-9 (melalui perbandingan antara c dan c'), merupakan *secret intermediate data*. Sesuai dengan ketentuan yang dijelaskan pada sebelumnya, bendera ini harus dimusnahkan sebelum algoritma ML-KEM.Decaps internal selesai dijalankan. Secara khusus, pengembalian nilai bendera tersebut dalam bentuk apa pun sebagai output tidak diperbolehkan.

7) Mekanisme Enkapsulasi Kunci ML-KEM

Bagian ini menjelaskan tiga algoritma utama dari skema ML-KEM, yaitu:

- a) Pembangkitan Kunci (ML-KEM.KeyGen)
- b) Enkapsulasi (ML-KEM.Encaps)
- c) Dekapsulasi (ML-KEM.Decaps)

Untuk menginstansiasi ML-KEM, harus dipilih satu set parameter. Setiap set parameter dikaitkan dengan tingkat

keseimbangan tertentu antara keamanan dan kinerja. Tiga himpunan parameter yang tersedia yaitu ML-KEM-512, ML-KEM-768 dan ML-KEM-1024. Ketiga himpunan parameter tersebut dijelaskan secara rinci dalam bagian set parameter. Setiap himpunan parameter memberikan nilai numerik tertentu untuk parameter-parameter berikut, k , η_1 , η_2 , du , dan dv . Nilai n selalu 256, dan nilai q selalu 3329, sedangkan nilai parameter lainnya berbeda-beda tergantung pada set parameter yang digunakan.

Implementor harus memastikan bahwa ketiga algoritma ML-KEM yang disebutkan di atas hanya dijalankan dengan menggunakan set parameter yang valid dan bahwa set parameter tersebut dipilih dengan tepat sesuai dengan kebutuhan aplikasi. Selain itu, implementor juga wajib memastikan bahwa set parameter yang digunakan dalam setiap pemanggilan fungsi ML-KEM.Encaps atau ML-KEM.Decaps harus sesuai dengan set parameter yang terkait dengan input yang diberikan.

a) **Pembangkitan kunci ML-KEM**

Algoritma ML-KEM.KeyGen dari skema ML-KEM tidak menerima input apa pun, menghasilkan nilai acak secara internal, dan menghasilkan dua output, yaitu kunci enkapsulasi dan kunci dekapsulasi. Kunci enkapsulasi dapat dijadikan informasi publik, sedangkan kunci dekapsulasi harus tetap bersifat rahasia.

Seed (d , z) yang dihasilkan pada langkah 1 dan 2 dari ML-KEM.KeyGen dapat disimpan untuk digunakan kemudian dalam proses ekspansi ulang menggunakan ML-KEM.KeyGen_internal.

Karena *seed* ini dapat digunakan untuk menghitung ulang kunci dekapsulasi, maka *seed* tersebut merupakan data sensitif dan harus dilindungi dengan cara yang sama seperti kunci dekapsulasi, sesuai ketentuan dalam SP 800-227.

Keamanan dalam pembentukan kunci bersama (*secure key establishment*) bergantung pada penggunaan pasangan kunci yang telah dihasilkan dengan benar melalui ML-KEM.KeyGen. Apabila pemilik pasangan kunci KEM tidak menghasilkan kunci tersebut sendiri, tetapi menerimanya dari pihak ketiga tepercaya atau sumber lain, pemilik dapat — secara opsional — melakukan beberapa pemeriksaan terhadap pasangan kunci tersebut. Pemeriksaan ini dapat mendeteksi bentuk korupsi tertentu, namun tidak menjamin sepenuhnya bahwa pasangan kunci tersebut dihasilkan dengan benar oleh ML-KEM.KeyGen.

Algorithm 19 ML-KEM.KeyGen()

Generates an encapsulation key and a corresponding decapsulation key.

Output: encapsulation key $ek \in \mathbb{B}^{384k+32}$.

Output: decapsulation key $dk \in \mathbb{B}^{768k+96}$.

$d \xleftarrow{\$} \mathbb{B}^{32}$ $\triangleright d$ is 32 random bytes

$z \xleftarrow{\$} \mathbb{B}^{32}$ $\triangleright z$ is 32 random bytes

if $d == \text{NULL}$ **or** $z == \text{NULL}$ **then**

return $\perp$ $\triangleright$ return an error indicator if random bit generation failed

end if

$(ek, dk) \leftarrow \text{ML-KEM.KeyGen_internal}(d, z)$ $\triangleright$ run internal key generation algorithm

return (ek, dk)

Pemeriksaan pasangan kunci

Untuk memeriksa kandidat pasangan kunci $\overline{ek}, \overline{dk}$, dilakukan pemeriksaan berikut:

1. (Konsistensi *seed*)

Jika tersedia *seed* (d, z) , jalankan ML-KEM.KeyGen_internal (d, z) dan verifikasi bahwa output sama dengan pasangan $\overline{ek}, \overline{dk}$

2. (Pemeriksaan kunci enkapsulasi) periksa $\overline{ek}$ sebagaimana ditentukan pada bagian enkapsulasi ML-KEM
3. (Pemeriksaan kunci dekapsulasi) periksa $\overline{dk}$ sebagaimana ditentukan pada bagian dekapsulasi ML-KEM
4. (Konsistensi pasangan)
 - i. Bangkitkan *array* berisi 32 *byte* acak dengan melakukan $m \xleftarrow{\$} \mathbb{B}$
 - ii. Jalankan $(K, c) \leftarrow \text{ML-KEM.Encaps_internal}(\overline{ek}, m)$
 - iii. Jalankan $K' \leftarrow \text{ML-KEM.Decaps_internal}(\overline{dk}, c)$.
 - iv. Tolak, kecuali jika $K == K'$

Penting untuk dicatat bahwa proses pemeriksaan ini tidak menjamin bahwa pasangan kunci tersebut merupakan output dari ML-KEM.KeyGen.

b) Enkapsulasi ML-KEM

Algoritma enkapsulasi ML-KEM atau dikenal dengan ML-KEM.Encaps menerima ek sebagai input, membangkitkan keacakan secara internal dan menghasilkan *ciphertext* dan *shared secret* $\star$ *key* $\star$ untuk komunikasi. Algoritma ini mempersyaratkan *input checking*, sebagai berikut:

Pemeriksaan kunci enkapsulasi

Pemeriksaan ini bertujuan untuk memastikan bahwa kandidat kunci enkapsulasi $\overline{ek}$ yang digunakan dalam algoritma ML-KEM.Encaps memiliki bentuk yang benar dan berada dalam rentang nilai yang valid, dengan memenuhi:

- (Pemeriksaan jenis)

Jika $\overline{ek}$ bukan berupa *array byte* dengan panjang $384k+32$ (sesuai dengan nilai k yang ditentukan dalam set parameter yang digunakan), maka pemeriksaan input dianggap gagal.
- (Pemeriksaan modulus),

dilakukan dengan komputasi:

$$\text{test} \leftarrow \text{ByteEncode}_{12}(\text{ByteDecode}_{12}(\overline{ek}[0:384k]))$$

apabila $\text{test} \neq (\overline{ek}[0:384k])$, maka pemeriksaan input dianggap gagal. Perhitungan ini digunakan untuk memastikan bahwa integer yang di-*encoding* dalam kunci publik berada dalam rentang nilai yang benar, yaitu $[0, q - 1]$.

Apabila pada kedua pemeriksaan di atas lulus, maka ML-KEM.Encaps dapat dijalankan dengan input $ek := \overline{ek}$. Namun, proses pemeriksaan ini tidak memberikan jaminan bahwa ek adalah output yang diproduksi dari ML-KEM.KeyGen. ML-KEM.Encaps tidak dapat dijalankan dengan kunci enkapsulasi yang belum diperiksa dengan pemeriksaan di atas. Namun, pemeriksaan ini tidak harus dilakukan setiap kali ek digunakan, melainkan bisa dilakukan sekali saja atau oleh pihak lain yang dapat dipercaya. Berikut adalah algoritma enkapsulasi kunci.

Algorithm 20 ML-KEM.Encaps(ek)

Uses the encapsulation key to generate a shared secret key and an associated ciphertext.

Checked input: encapsulation key $ek \in \mathbb{B}^{384k+32}$

Output: shared secret key $K \in \mathbb{B}^{32}$.

Output: ciphertext $c \in \mathbb{B}^{32(d_u k + d_v)}$.

$m \xleftarrow{\$} \mathbb{B}^{32}$

▷ m is 32 random bytes

if $m == \text{NULL}$ then

return $\perp$ ▷ return an error indication if random bit generation failed

end if

$(K, c) \leftarrow \text{ML-KEM.Encaps_internal}(ek, m)$ ▷ run internal encapsulation algorithm

return (K, c)

Langkah-Langkah Algoritma:

➤ Pengecekan input

Algoritma memeriksa apakah ek valid sebelum melanjutkan proses. Hal ini penting untuk mencegah kesalahan atau serangan berbasis input yang tidak sah.

➤ Pembangkitan bilangan acak

Algoritma menghasilkan nilai acak m secara internal, yang membuat setiap eksekusi unik dan meningkatkan keamanan terhadap serangan. Bilangan acak m adalah 32 *byte* bilangan acak yang diambil dari sumber entropi kriptografis. Jika m bernilai NULL (artinya gagal menghasilkan bilangan acak), algoritma mengembalikan error (1) dan berhenti.

➤ Enkapsulasi kunci

Algoritma memanggil ML-KEM.Encaps_internal (ek , m) untuk melakukan proses enkapsulasi menggunakan ek dan bilangan acak m . Fungsi ini menghasilkan: *Shared secret key* (K) yang akan digunakan untuk komunikasi aman menggunakan enkripsi simetris dan *Ciphertext* (c) nantinya dikirim ke penerima untuk proses dekripsi.

c) Dekapsulasi ML-KEM

ML-KEM.Decaps adalah algoritma dekapulasi yang digunakan untuk mendapatkan *shared secret key* (K) dari *ciphertext* ($\bar{c}$) menggunakan *decapsulation key* ($\bar{dk}$).

Pemeriksaan Input untuk Dekapsulasi

Sebelum menjalankan algoritma ML-KEM.Decaps, penting untuk memastikan bahwa $\bar{dk}$ dan $\bar{c}$ memiliki format yang benar dan sesuai dengan spesifikasi. Pemeriksaan ini bertujuan untuk mencegah kesalahan dalam proses dekapulasi serta mengurangi potensi serangan.

➤ (Pemeriksaan jenis *ciphertext*)

Pastikan bahwa $\bar{c}$ memiliki panjang yang benar sesuai dengan spesifikasi. Jika $\bar{c}$ bukan *array byte* dengan

panjang 32 ($d_u k + d_v$) untuk nilai d_u , d_v , dan k yang ditentukan oleh set parameter yang relevan, maka pemeriksaan input telah gagal. Jika panjang atau struktur $\bar{c}$ tidak valid, maka proses dekapsulasi tidak akan dilakukan.

➤ (Pemeriksaan jenis kunci dekapsulasi)

Pastikan bahwa $\overline{dk}$ memiliki panjang yang sesuai dengan parameter yang digunakan. Jika $\overline{dk}$ bukan *array byte* dengan panjang $768k + 96$ untuk nilai k yang ditentukan oleh set parameter yang relevan, maka pemeriksaan input telah gagal. Jika $\overline{dk}$ tidak memenuhi format yang benar, proses dekapsulasi dianggap gagal.

➤ (Pemeriksaan *hash*)

Memverifikasi bahwa $\bar{c}$ tidak mengandung nilai yang berada di luar rentang yang diperbolehkan.

Lakukan perhitungan berikut:

$$\text{test} \leftarrow H(\overline{dk} [384k : 768k + 32])$$

Jika $\text{test} \neq \overline{dk} [768k + 32 : 768k + 64]$, maka pemeriksaan input telah gagal. Jika terdapat data yang tidak valid dalam $\bar{c}$, proses dekapsulasi dianggap gagal.

Jika semua pemeriksaan di atas lolos, maka ML-KEM.Decaps dapat dijalankan dengan input $dk := \overline{dk}$ dan $c := \bar{c}$. ML-KEM.Decaps tidak boleh dijalankan dengan kunci dekapsulasi atau *ciphertext*, kecuali keduanya telah diperiksa. Namun pemeriksaan kunci dekapsulasi tidak perlu dilakukan oleh pihak yang melakukan dekapsulasi, ataupun pada setiap eksekusi ML-KEM.Decaps. Jaminan bahwa pemeriksaan ini telah dilakukan dapat diperoleh melalui cara yang didefinisikan pada NIST SP 800-227. Pemeriksaan *ciphertext* harus dilakukan setiap eksekusi ML-KEM.Decaps.

Berikut ini algoritma dekapsulasi:

Algorithm 21 ML-KEM.Decaps(dk, c)

Uses the decapsulation key to produce a shared secret key from a ciphertext.

Checked input : decapsulation key $dk \in \mathbb{B}^{768k+96}$.

Checked input : ciphertext $c \in \mathbb{B}^{32(d_u k + d_v)}$.

Output: shared secret key $K \in \mathbb{B}^{32}$.

$K' \leftarrow \text{ML-KEM.Decaps_internal}(dk, c) \triangleright$ run internal decapsulation algorithm

return (K')

Langkah – langkah algoritma:

- Panggil fungsi internal ML-KEM.Decaps_internal(dk, c) untuk menjalankan algoritma dekapsulasi dan menghasilkan nilai K'
- Kembalikan nilai K' sebagai hasil akhir (*shared secret key*)

8) Perbedaan antara CRYSTAL-Kyber dan FIPS 203 *Initial Public Draft*

- a) Pada spesifikasi putaran ketiga, *shared secret key* diperlakukan sebagai nilai dengan panjang variabel, yang bergantung pada bagaimana kunci tersebut digunakan dalam aplikasi terkait. Dalam spesifikasi ini, panjang *shared secret key* ditetapkan sebesar 256 bit. Kunci ini dapat digunakan secara langsung dalam aplikasi sebagai kunci simetris, atau dapat menjadi bahan dasar untuk derivasi kunci simetris lainnya.
- b) Algoritma ML-KEM.Encaps dan ML-KEM.Decaps dalam spesifikasi ini menggunakan varian yang berbeda dari transformasi Fujisaki–Okamoto dibandingkan dengan spesifikasi putaran ketiga. Secara khusus, ML-

KEM.Encaps tidak lagi menyertakan *hash* dari *ciphertext* dalam proses derivasi *shared secret*, dan ML-KEM.Decaps telah disesuaikan agar konsisten dengan perubahan tersebut.

- c) Dalam spesifikasi putaran ketiga nilai acak awal m pada algoritma ML-KEM.Encaps terlebih dahulu dihash sebelum digunakan. Secara spesifik, antara baris ke-1 dan ke-2 pada ML-KEM.Encaps, terdapat langkah tambahan yang melakukan operasi $m \leftarrow H(m)$. Tujuan langkah ini adalah untuk memberikan perlindungan terhadap kemungkinan kegagalan proses pembangkitan nilai acak. Karena dalam standar ini diwajibkan penggunaan mekanisme pembangkitan acak yang telah disetujui oleh NIST, langkah tersebut tidak lagi diperlukan dan tidak diterapkan dalam ML-KEM.
- d) Spesifikasi ini menyertakan langkah-langkah pemeriksaan input eksplisit yang tidak terdapat pada spesifikasi putaran ketiga. Misalnya, ML-KEM.Encaps mewajibkan bahwa *array byte* yang memuat kunci enkapsulasi harus terdecode dengan benar menjadi *array* bilangan bulat modulus q tanpa memerlukan reduksi modulus tambahan.

b. ML-DSA

1) Deskripsi Algoritma

ML-DSA adalah skema tanda tangan digital berbasis CRYSTAL-Dilithium. Skema ML-DSA menggunakan konstruksi Fiat Shamir *with aborts* [10] yang dikembangkan oleh Lyubashevsky. Teknik ini menggunakan *rejection sampling* untuk membuat skema Fiat-Shamir berbasis latis menjadi lebih ringkas dan aman. Skema dengan ukuran tanda tangan terkecil yang menggunakan pendekatan ini dikembangkan oleh Ducas, Durmus, Lepoint, dan Lyubashevsky. Skema tersebut

berbasis pada asumsi NTRU dan menggunakan Gaussian *sampling* untuk membuat tanda tangan. Namun, karena Gaussian *sampling* sulit diimplementasikan dengan aman dan efisien, Dilithium memilih untuk hanya menggunakan distribusi *uniform* sebagai alternatif yang lebih sederhana dan lebih mudah diimplementasikan.

Dilithium meningkatkan efisiensi skema berbasis distribusi uniform yang sebelumnya dikembangkan oleh Bai dan Galbraith. Salah satu perbaikan utama adalah teknik baru yang mengurangi ukuran kunci publik. Dilithium memiliki kombinasi ukuran kunci publik dan tanda tangan terkecil di antara semua skema tanda tangan berbasis latis yang hanya menggunakan *uniform sampling*.

Skema ML-DSA terdiri dari tiga algoritma utama, yaitu : ML-DSA.KeyGen, ML-DSA.Sign dan ML-DSA.Verify. ML-DSA distandardisasi oleh NIST pada dokumen FIPS 204. FIPS 204 juga mendefinisikan skema tanda tangan yang terkait erat namun terpisah domain, yaitu HashML-DSA. HashML-DSA berbeda dari ML-DSA karena mencakup langkah *pra-hashing* tambahan sebelum penandatanganan. Skema HashML-DSA terdiri dari tiga algoritma utama, yaitu : ML-DSA.KeyGen (algoritma yang sama dengan yang digunakan untuk ML-DSA), HashML-DSA.Sign dan HashML-DSA.Verify.

ML-DSA didesain agar memiliki keamanan yang kuat terhadap pemalsuan eksistensial dalam serangan dengan pesan yang dipilih (*Strongly existentially unforgeable under chosen message attack* - SUF-CMA). Sehingga meskipun seorang penyerang dapat meminta pihak yang sah untuk menandatangani pesan apa pun yang diinginkan, mereka tetap tidak dapat menghasilkan tambahan tanda tangan yang valid berdasarkan kunci

publik dari *signer*. Ini mencakup skenario di mana penyerang mencoba membuat tanda tangan baru untuk pesan yang sudah pernah ditandatangani oleh pihak yang sah [11].

2) Skema Algoritma

a) Asumsi Komputasional

Keamanan skema tanda tangan digital berbasis latis biasanya terkait dengan dua masalah matematis yang sulit, yaitu masalah *LWE* dan Masalah *Short Integer Solution* (SIS). Jika dalam *LWE* dan *SIS* struktur $\mathbb{Z}_q^n$ digantikan dengan modul R_q^k dengan mengambil produk kartesius k -lipat dari sebuah ring polinomial R_q maka permasalahan ini berubah menjadi *Module-LWE* dan *Module-SIS*. Keamanan *ML-DSA* didasarkan pada masalah *Module-LWE* di atas R_q serta varian tidak standar dari *Module-SIS* yang disebut *SelfTargetMSIS*.

b) Konstruksi *ML-DSA*

ML-DSA adalah skema tanda tangan digital berbasis latis yang menyerupai *Schnorr signature*, dengan beberapa optimasi. *Schnorr signature* berasal dari *Fiat-Shamir heuristic* pada protokol interaktif antara *verifier* yang mengetahui g dan nilai $y = gx$ dan *prover* yang mengetahui g dan x . Protokol interaktif tersebut terdiri dari tiga Langkah, yaitu:

➤ *Commitment*

Pembuktian dimulai dengan *prover* yang memilih secara acak bilangan bulat positif r yang lebih kecil dari orde elemen g . Kemudian *prover* membuat komitmen terhadap nilai tersebut dengan mengirimkan hasil g^r kepada *verifier*.

➤ *Challenge*

Verifier mengirimkan bilangan bulat positif c yang lebih kecil dari orde elemen g kepada *Prover*.

➤ *Response*

Prover mengembalikan $s = r - cx$ setelah direduksi modulus orde g dan *Verifier* memeriksa $g^s \cdot y^c = g^r$, jika benar maka *verifier* menerima pembuktian. Proses ini memastikan bahwa *prover* memiliki pengetahuan tentang x tanpa harus mengungkapkannya secara langsung.

Protokol ini dibuat noninteraktif dan diubah menjadi skema tanda tangan dengan mengganti pilihan acak *verifier* atas c pada langkah 2 dengan proses deterministik yang secara pseudorandom menurunkan c dari *hash* komitmen g^r yang digabungkan dengan pesan yang akan ditandatangani. Untuk skema tanda tangan ini, y adalah kunci publik, dan x adalah kunci privat.

Prinsip Dasar ML-DSA adalah membangun skema tanda tangan dari sebuah protokol interaktif analog. Dalam skema ini, ingin dibuktikan bahwa *Prover* mengetahui matriks $A \in \mathbb{Z}_q^{K \times L}$, $S_1 \in \mathbb{Z}_q^{L \times n}$, dan $S_2 \in \mathbb{Z}_q^{K \times n}$ dengan koefisien kecil untuk S_1 dan S_2 . *Prover* menunjukkan pengetahuan tentang matriks ini kepada *Verifier* yang hanya mengetahui matriks A dan $T \in \mathbb{Z}_q^{K \times n}$ dengan hubungan: $T = AS_1 + S_2$. Protokol interaktif tersebut akan berjalan sebagai berikut:

➤ *Commitment*

Prover menghasilkan $y \in \mathbb{Z}_q^L$ dengan koefisien kecil dan berkomitmen pada nilainya dengan

mengirimkan $w_{Approx} = Ay + y_2$ ke *verifier*, di mana $y_2 \in \mathbb{Z}_q^K$ adalah vektor dengan koefisien kecil.

➤ *Challenge*

Verifier mengirimkan vektor $c \in \mathbb{Z}_q^n$ dengan koefisien kecil kepada *Prover*.

➤ *Response*

Prover mengembalikan $z = y + S_1c$, dan *Verifier* memeriksa z , memiliki koefisien kecil dan bahwa $Az - Tc \approx w_{Approx}$.

Dalam bentuk awalnya, protokol interaktif di atas memiliki celah keamanan, yaitu:

➤ Bias dalam respons z

Nilai respon $z = y + S_1c$ dapat memiliki pola yang bergantung pada nilai privat S_1 . Jika seorang penyerang dapat mengamati banyak tanda tangan, mereka mungkin dapat mengungkapkan informasi tentang S_1 karena pola ini.

➤ Bias dalam respons r

Nilai r dihitung sebagai

$$r = w_{approx} - A_z + T_c = y + S_2c$$

Karena r mengandung S_2c , itu juga bisa menunjukkan pola terkait S_2 , yang bisa dimanfaatkan oleh penyerang.

Untuk mengatasi masalah bias ini, skema tanda tangan menggunakan modifikasi berbasis Fiat-Shamir (seperti dalam tanda tangan Schnorr), dengan dua langkah utama *challenge* tidak berasal dari *verifier* dan menggunakan *Rejection Sampling* untuk menghilangkan bias. Pemeriksaan ini analog dengan yang dilakukan pada Baris 28 Algoritma 7.

Dalam tanda tangan Fiat-Shamir With Aborts yang disederhanakan, kunci publiknya adalah (A, T) , dan kunci privatnya adalah (S_1, S_2) . Dalam standar ML-DSA, sejumlah penyempurnaan dan modifikasi ditambahkan ke kerangka dasar ini demi alasan keamanan atau efisiensi:

- Untuk mengurangi ukuran kunci dan tanda tangan serta untuk menggunakan perkalian polinomial berbasis NTT yang cepat, ML-DSA menggunakan matriks berstruktur modul. Dibandingkan dengan skema dasar yang dijelaskan di atas, ML-DSA menggantikan blok matriks berdimensi- $n \times n$ dan blok vektor berdimensi- n dengan polinomial dalam ring R_q .
- Untuk lebih mengurangi ukuran kunci publik, matriks A diturunkan secara pseudorandom dari *seed* publik (ρ) 256 bit, yang disertakan dalam kunci publik ML-DSA sebagai pengganti A .
- Untuk pengurangan lebih lanjut dalam ukuran kunci publik, kunci publik ML-DSA mengganti nilai terkompresi t_1 untuk t , yang menghilangkan d bit orde rendah dari setiap koefisien.
- Untuk memperoleh properti *beyond unforgeability* (BUFF), ML-DSA tidak secara langsung menandatangani pesan M , melainkan menandatangani pesan representatif μ yang diperoleh dengan melakukan *hashing* terhadap penggabungan *hash* kunci publik dan M .
- Untuk mengurangi ukuran tanda tangan, daripada menggunakan *commitment* $w_{Approx} =$

$\mathbf{A}\mathbf{y} + y_2$ dalam tanda tangan, tanda tangan ML-DSA menggunakan versi bulat dari $\mathbf{w} = \mathbf{A}\mathbf{y}$ sebagai komitmen w_1 dan hanya menyertakan hash $\tilde{c}$ dari $w_1 \parallel \mu$.

- Untuk memastikan bahwa w_1 dapat direkonstruksi oleh *verifier* dari $\mathbf{z}$ dan nilai terkompresi t_1 , tanda tangan juga harus menyertakan petunjuk $\mathbf{h} \in R_2^k$ yang dihitung oleh *signer* menggunakan kunci privat *signer*.

Singkatan ML-DSA-44, ML-DSA-65, dan ML-DSA-87 digunakan untuk merujuk pada pilihan parameter ML-DSA. Dalam singkatan-singkatan ini, akhiran numerik merujuk pada dimensi matriks $\mathbf{A}$. Misalnya, dalam ML-DSA-65, matriks $\mathbf{A}$ adalah matriks 6×5 atas R_q .

- c) Penandatanganan Deterministik dan *Hedged*
- Untuk menjaga keamanan ML-DSA, nilai komitmen *signer* ($\mathbf{y}$) tidak boleh digunakan lebih dari sekali untuk menandatangani pesan yang berbeda dan tidak boleh mudah ditebak oleh penyerang. Oleh karena itu, diperlukan elemen keacakan (*randomness*). Secara prinsip, keacakan yang digunakan untuk menghasilkan $\mathbf{y}$ dapat diperoleh dengan dua cara, yaitu:

- Menggunakan keacakan baru setiap kali melakukan tanda tangan.
- Menghasilkan nilai keacakan secara pseudorandom dari pesan yang ditandatangani serta nilai acak pra-komputasi yang disimpan dalam kunci privat *signer*.

Secara default, FIPS 204 menetapkan bahwa algoritma penandatanganan harus menggunakan dua jenis keacakan. Pendekatan ini disebut sebagai

varian terlindung (*hedged*) dalam prosedur penandatanganan. Penggunaan keacakan baru (*fresh randomness*) saat penandatanganan membantu mengurangi risiko serangan *side-channel*, seperti serangan analisis daya atau gangguan elektromagnetik. Penggunaan keacakan yang telah dihitung sebelumnya (*precomputed randomness*) melindungi dari kemungkinan kelemahan atau kegagalan dalam generator bilangan acak yang digunakan oleh *signer* pada saat proses tanda tangan.

FIPS 204 juga memungkinkan varian penandatanganan yang sepenuhnya deterministik, yang dapat digunakan jika *signer* tidak memiliki akses ke sumber keacakan baru saat proses tanda tangan. Namun, tidak adanya keacakan dalam varian deterministik meningkatkan risiko serangan *side-channel*, terutama *fault attacks* (serangan yang mengeksploitasi kesalahan perangkat keras atau sistem). Hal ini membuat mitigasi serangan menjadi lebih sulit. Oleh karena itu, varian deterministik tidak boleh digunakan pada platform yang rentan terhadap serangan *side-channel*, terutama jika tidak ada cara lain untuk mengurangi risiko serangan tersebut.

Hanya mengimplementasikan varian *hedged* sudah cukup untuk memastikan interoperabilitas. Algoritma verifikasi yang sama dapat digunakan untuk memverifikasi tanda tangan dari kedua varian. Mengimplementasikan varian deterministik tidak meningkatkan interoperabilitas, sehingga tidak ada keuntungan tambahan dari sisi kompatibilitas sistem.

d) Penggunaan tanda tangan digital

Manajemen kunci yang aman merupakan persyaratan penting dalam penggunaan tanda tangan digital. Aspek ini bergantung pada konteks penggunaannya dan cakupannya lebih dari algoritma pembangkitan kunci, penandatanganan, dan verifikasi tanda tangan. Panduan lebih lanjut tentang manajemen kunci tersedia dalam seri NIST SP 800-57.

Tanda tangan digital bermanfaat ketika terkait dengan identitas tertentu. Untuk menghubungkan kunci publik dengan identitas seseorang, diperlukan bukti kepemilikan kunci privat. Dalam infrastruktur kunci publik (PKI), penerbitan sertifikat kunci publik melibatkan jaminan identitas dan bukti kepemilikan kunci privat. Namun, jika sertifikat kunci publik tidak tersedia, pengguna tanda tangan digital perlu mempertimbangkan apakah kunci publik harus dikaitkan dengan identitas. Metode untuk memperoleh jaminan identitas dan bukti kepemilikan kunci privat tersedia dalam NIST SP 800-89.

e) Persyaratan Tambahan

- Pembangkitan keacakan

Algoritma ML-DSA.Keygen menggunakan *Random Bit Generator* (RBG) untuk menghasilkan *seed* acak (ξ) 256 bit. *Seed* ξ harus unik dan baru serta dihasilkan menggunakan RBG yang telah disetujui, sesuai dengan standar SP 800-90A, SP 800-90B, dan SP 800-90C. Selain itu, RBG yang digunakan harus memiliki kekuatan keamanan setidaknya

192 bit untuk ML-DSA-65 dan 256 bit untuk ML-DSA-87. Untuk ML-DSA-44, RBG sebaiknya memiliki kekuatan keamanan setidaknya 192 bit dan wajib memiliki kekuatan keamanan minimal 128 bit. Jika RBG yang disetujui dengan kekuatan keamanan minimal 128 bit tetapi kurang dari 192 bit digunakan maka klaim kekuatan keamanan ML-DSA-44 diturunkan dari kategori 2 ke kategori 1.

Nilai *rnd* juga dihasilkan oleh RBG dalam varian "*hedged*" untuk ML-DSA dan HashML-DSA. *rnd* berfungsi untuk melindungi dari serangan *side-channel* dan *fault attacks*. Walaupun idealnya *rnd* dihasilkan oleh RBG yang telah disetujui, metode lain juga dapat digunakan untuk mendapatkan nilai acak yang baru. Pada varian penandatanganan deterministik, kebocoran informasi dapat terjadi melalui *side-channel timing attacks*. Jika kerahasiaan pesan yang ditandatangani penting, varian "*hedged*" harus digunakan. Bahkan RBG yang lemah lebih baik dibandingkan menggunakan varian deterministik sepenuhnya untuk mengurangi risiko kebocoran informasi.

- Pemeriksaan panjang kunci publik dan tanda tangan

Algoritma verifikasi ML-DSA dan Algoritma verifikasi HashML-DSA menentukan panjang tanda tangan (σ) dan kunci publik (pk) berdasarkan parameter. Implementasi tidak boleh menerima σ atau pk dengan panjang yang berbeda dari standar. Jika panjangnya berbeda, sistem harus mengembalikan *false* (gagal

verifikasi). Tidak memeriksa panjang kunci atau tanda tangan bisa mengganggu keamanan ML-DSA, termasuk kemampuan untuk mencegah pemalsuan tanda tangan.

- Nilai *intermediate*

Data yang digunakan secara internal dalam langkah perhitungan *intermediate* saat pembuatan kunci atau penandatanganan bisa digunakan oleh penyerang untuk mendapatkan informasi tentang kunci privat, sehingga berpotensi membahayakan keamanan. Data *intermediate* dalam algoritma verifikasi juga bisa menjadi sensitif dalam beberapa aplikasi, seperti verifikasi tanda tangan yang digunakan sebagai *bearer token* (misalnya untuk autentikasi) atau verifikasi tanda tangan pada pesan *plaintext* yang seharusnya tetap rahasia. Nilai *intermediate* dalam algoritma verifikasi dapat mengungkapkan informasi tentang inputnya (pesan, tanda tangan, atau kunci publik). Dalam beberapa aplikasi (misalnya pada penandatanganan deterministik serta verifikasi pesan dan tanda tangan yang bersifat rahasia), diperlukan satu atau lebih input bersifat rahasia. Oleh karena itu, implementasi ML-DSA harus memastikan bahwa setiap nilai *intermediate* yang berpotensi sensitif dihancurkan segera setelah tidak lagi diperlukan.

- Tidak menggunakan aritmetika *floating point*
Implementasi ML-DSA tidak boleh menggunakan aritmetika *floating-point* karena kesalahan pembulatan dalam operasi *floating-*

point dapat menyebabkan hasil yang tidak akurat dalam beberapa kasus.

f) Penggunaan kriptografi simetris

FIPS 204 menggunakan fungsi SHAKE256 dan SHAKE128, sebagaimana didefinisikan dalam FIPS 202. Meskipun FIPS 202 menetapkan fungsi-fungsi ini sebagai *string* bit input dan *output*, sebagian besar implementasi menggunakan input dan output sebagai *string byte*.

ML-DSA akan selalu memanggil fungsi-fungsi ini dengan panjang *output* kelipatan delapan bit dan memperlakukan *output* fungsi-fungsi ini sebagai *string byte*, yang akan menjadi *string byte* yang sama dengan yang dihasilkan dari pengambilan *string* bit yang diharapkan dari pembacaan literal FIPS 202 dan memprosesnya dengan `BitsToBytes`. Namun, untuk memungkinkan penandatanganan pesan yang bukan bilangan bulat *byte*, dokumen ini akan membebani SHAKE256 sehingga inputnya mungkin berupa *string* bit tetapi biasanya berupa *string byte*. Kesetaraan berikut berlaku untuk setiap *string byte* str dan integer $\ell \geq 1$:

$$\text{SHAKE256}(str, 8\ell) = \text{SHAKE256}(\text{BytesToBits}(str), 8\ell).$$

Selain menggunakan varian API yang sebagian besar berorientasi *byte* yang didefinisikan dalam FIPS 202 untuk SHAKE256 dan SHAKE128, standar ini terkadang menggunakan API inkremental yang didefinisikan dalam SP 800-185. API ini terdiri dari tiga fungsi untuk setiap varian SHAKE. Fungsi-fungsi ini dapat digunakan untuk menyerap dan mengompresi urutan *string* dengan panjang sembarang. Fungsi-fungsi ini melakukan *buffering* untuk menangani blok data yang tidak lengkap saat

menyerap atau mengkompresi. Misalnya, untuk SHAKE256:

- $ctx \leftarrow \text{SHAKE256.Init}()$
Menginisialisasi konteks fungsi hash.
- $ctx \leftarrow \text{SHAKE256.Absorb}(ctx, str)$
Menyuntikkan data yang akan digunakan dalam fase penyerapan SHAKE256 dan memperbarui konteks ctx .
- $(ctx, out) \leftarrow \text{SHAKE256.Squeeze}(ctx, \ell)$
Mengekstrak byte $output$ ℓ yang dihasilkan selama fase squeezing SHAKE256 dan memperbarui ctx konteks.

Meskipun fungsi-fungsi di atas ditentukan berdasarkan permutasi Keccak- f , bukan Fungsi Output eXtendable (XOF), SHAKE256, fungsi-fungsi tersebut didefinisikan sedemikian rupa sehingga rangkaian perintah apa pun dengan bentuk berikut:

1. $ctx \leftarrow \text{SHAKE256.Init}()$
2. Untuk $i = 1$ hingga m
 $ctx \leftarrow \text{SHAKE256.Absorb}(ctx, stri)$
3. Untuk $j = 1$ hingga k
 $(ctx, outj) \leftarrow \text{SHAKE256.Squeeze}(ctx, 8bj)$
4. $output \leftarrow out1 || \dots || outk$

akan menghasilkan $output$ yang sama dengan satu panggilan SHAKE256:

$$output \leftarrow \text{SHAKE256}(str1 || \dots || str_m, 8b_1 + \dots + 8b_k).$$

Kesetaraan ini berlaku baik $|stri|$ dan b_j merupakan kelipatan panjang blok SHAKE256 maupun tidak.

Karena semua $output$ SHAKE128 dan SHAKE256 dalam FIPS 204 menghasilkan bilangan bulat $byte$, fungsi pembungkus H dan G didefinisikan sebagai berikut:

1. $H(str, \ell) = \text{SHAKE256}(str, 8\ell)$

2. $G(str, \ell) = \text{SHAKE128}(str, 8\ell)$
3. $H.Init() = \text{SHAKE256}.Init()$
4. $G.Init() = \text{SHAKE128}.Init()$
5. $H.Absorb(ctx, str) = \text{SHAKE256}.Absorb(ctx, str)$
6. $G.Absorb(ctx, str) = \text{SHAKE128}.Absorb(ctx, str)$
7. $H.Squeeze(ctx, \ell) = \text{SHAKE256}.Squeeze(ctx, 8\ell)$
8. $G.Squeeze(ctx, \ell) = \text{SHAKE128}.Squeeze(ctx, 8\ell)$

Selain SHAKE128 dan SHAKE256, HashML-DSA.Sign dan HashML-DSA.Verify dapat memanggil fungsi *hash* lain yang telah disetujui untuk pre-hashing. *Pseudocode* dalam standar ini juga memperlakukan fungsi-fungsi ini sebagai *output* yang mengembalikan *string byte*, sementara mendukung *string bit* atau *string byte* sebagai input. Perlu dicatat bahwa fungsi *hash* yang didefinisikan dalam menggunakan aturan yang berbeda (yaitu, pengurutan *big-endian*) untuk menghubungkan bit, *byte*, dan *word*.

g) **Set Parameter**

ML-DSA memiliki tiga set parameter yang ditentukan dalam Tabel 3.5, di mana setiap set menentukan nilai untuk berbagai parameter dalam proses pembangkitan kunci, penandatanganan dan verifikasi. Set parameter ini dirancang untuk memenuhi kategori kekuatan keamanan tertentu yang ditetapkan oleh NIST dalam Panggilan Proposal aslinya. Secara konkret, set parameter ML-DSA-44 diklaim berada dalam kategori kekuatan keamanan 2, ML-DSA-65 diklaim berada dalam kategori 3, dan ML-DSA-87 diklaim berada dalam kategori 5.

Tabel 3.5 Set parameter pada ML-DSA

Parameter (see Section 6.1 and 6.2 of FIPS 204)	Values assigned by each parameter set		
	ML-DSA-44	ML-DSA-65	ML-DSA-87
q - modulus [see 6.1]	8380417	8380417	8380417
ζ a 512th root of unity in $\mathbb{Z}_q$ [see 7.5]	1753	1753	1753
d - # of dropped bits from $\mathbf{t}$ [see 6.1]	13	13	13
τ - # of ± 1 's in polynomial c [see 6.2]	39	49	60
λ - collision strength of c [see 6.2]	128	192	256
γ_1 - coefficient range of y [see 6.2]	2^{17}	2^{19}	2^{19}
γ_2 - low-order rounding range of $\mathbf{y}$ [see 6.2]	$(q - 1) / 88$	$(q - 1) / 32$	$(q - 1) / 32$
(k, ℓ) dimensions of $\mathbf{A}$ [see 6.1]	(4,4)	(6,5)	(8,7)
η private key range [see 6.2]	2	4	2
$\beta = \tau \cdot \eta$ [see 6.2]	78	196	120
ω - max # of 1's in the hint $\mathbf{h}$ [see 6.2]	80	55	75
Challenge entropy $\log_2\left(\frac{2^{256}}{\tau}\right) + \tau$ [see 6.2]	192	225	257
Repetitions (see explanation below)	4,25	5,1	3,85
Claimed security strength	Category 2	Category 3	Category 5

3) Fungsi Eksternal

Fungsi penandatanganan, verifikasi, dan pembangkitan kunci dapat dibagi menjadi komponen "eksternal" dan "internal" untuk menyederhanakan pengujian API dan Program Validasi Algoritma Kriptografi (*Cryptographic Algorithm Validation Program/CAVP*). Komponen eksternal menghasilkan keacakan dan melakukan berbagai pemeriksaan sebelum memanggil komponen internalnya. Komponen internal bersifat deterministik dan dapat mengasumsikan bahwa komponen eksternal tidak mengalami kondisi kesalahan.

Perbedaan antara fungsi eksternal dan internal juga menyederhanakan penyajian algoritma untuk penandatanganan dan verifikasi dengan mengelompokkan operasi yang digunakan bersama antara ML-DSA.Sign dan HashML-DSA.Sign dalam ML-DSA.Sign_internal dan mengelompokkan operasi yang digunakan bersama antara ML-DSA.Verify dan HashML-DSA.Verify dalam ML-DSA.Verify_internal

a) Pembangkitan kunci ML-DSA

Algoritma ML-DSA.KeyGen tidak menerima input dan menghasilkan output kunci publik dan kunci privat, yang keduanya dikodekan dalam *string byte*. Algoritma ini menggunakan RBG yang disetujui untuk menghasilkan benih acak ξ 256-bit (32-byte) yang diberikan sebagai input ke ML-DSA.KeyGen_internal, yang menghasilkan kunci publik dan privat.

Algorithm ML-DSA.KeyGen ()

Generates a public-private key pair.

Output: Public key $pk \in \mathbb{B}^{32+32k \cdot (\text{bitlen}(q-1)-d)}$

private key $sk \in \mathbb{B}^{32+32+64+32 \cdot ((\ell+k) \cdot \text{bitlen}(2\eta)+dk)}$.

$\xi \leftarrow \mathbb{B}^{32}$

▷choose random seed

if $\xi = \text{NULL}$ **then**

return $\perp$ ▷return an error indication if random bit generation failed

end if

return ML-DSA.KeyGen_internal (ξ)

b) Penandatanganan ML-DSA

ML-DSA.Sign menerima tiga input utama yaitu kunci privat, pesan dan *context string*. Secara *default*, *context string* adalah *string* kosong. Namun, aplikasi tertentu dapat menentukan *context* non-kosong untuk menambahkan informasi tambahan dalam tanda tangan. Algoritma ini mendukung dua mode utama, yaitu *Hedged* dan Deterministik.

Untuk versi "*hedged*" default dari penandatanganan ML-DSA, algoritma (pada baris 5) menggunakan RBG yang disetujui untuk menghasilkan *seed* acak 256-bit (32-byte) *rnd*. Jika varian deterministik

diinginkan, maka *rnd* diatur ke *string* nol tetap $\{0\}^{32}$. Nilai *rnd*, kunci privat, dan pesan yang dikodekan dimasukkan ke ML-DSA.Sign_internal, yang menghasilkan tanda tangan.

Algorithm ML-DSA.Sign (<i>sk</i> , <i>M</i> , <i>ctx</i>)
Generates an ML-DSA signature. Input: Private key $sk \in \mathbb{B}^{32+32+64+32 \cdot ((\ell+k) \cdot \text{bitlen}(2n) + dk)}$, message $M \in \{0,1\}^*$, context <i>string</i> <i>ctx</i> (a byte <i>string</i> of 255 or fewer bytes). Output: Signature $\sigma \in \mathbb{B}^{\lambda/4 + \ell \cdot 32 \cdot (1 + \text{bitlen}(\gamma_1 - 1)) + \omega + k}$ <pre> if <i>ctx</i> > 255 then return $\perp$ ▷ return an error indication if the context <i>string</i> is too long end if <i>rnd</i> $\leftarrow \mathbb{B}^{32}$ ▷ for the optional deterministic variant, substitute <i>rnd</i> $\leftarrow \{0\}^{32}$ if <i>rnd</i> = NULL then return $\perp$ ▷ return an error indication if random bit generation failed end if <i>M'</i> $\leftarrow \text{BytesToBits}(\text{IntegerToBytes}(0,1) \parallel$ $\text{IntegerToBytes}(\text{ctx} ,1) \parallel \text{ctx}) \parallel M$ <i>σ</i> $\leftarrow \text{ML-DSA.Sign_internal}(sk, M', rnd)$ return <i>σ</i> </pre>

c) Verifikasi ML-DSA

Algoritma ML-DSA.Verify digunakan untuk memverifikasi keabsahan tanda tangan digital dengan memeriksa apakah tanda tangan cocok dengan pesan dan kunci publik yang diberikan. Algoritma ini menerima empat input utama, yaitu Kunci publik, Pesan, Tanda tangan (*signature*) dan *Context string*. Kunci publik, tanda tangan, dan *context string* dikodekan dalam format *byte string*, sedangkan pesan dalam *bit string*. Keputusan

verifikasi berupa Boolean: *true* (valid) atau *false* (tidak valid).

Algorithm ML-DSA.Verify (pk, M, σ, ctx)
Verifies a signature σ for a message M. Input: Public key $pk \in \mathbb{B}^{32+32k \cdot (\text{bitlen}(q-1)-d)}$, message $M \in \{0,1\}^*$, signature $\sigma \in \mathbb{B}^{\lambda/4+\ell \cdot 32 \cdot (1+\text{bitlen}(\gamma_1-1))+\omega+k}$, context <i>string</i> ctx (a byte <i>string</i> of 255 or fewer bytes). Output: Boolean. <pre> if ctx > 255 then return 1 ▷ return an error indication if the context <i>string</i> is too long end if $M' \leftarrow \text{BytesToBits}(\text{IntegerToBytes}(0,1) \parallel$ $\text{IntegerToBytes}(ctx ,1) \parallel ctx) \parallel M$ return ML-DSA.Verify_internal(pk, M', σ) </pre>

- d) HashML-DSA
- Untuk beberapa modul kriptografi yang menghasilkan tanda tangan ML-DSA, *hashing* pesan pada langkah 6 dari ML-DSA.Sign_internal dapat menghasilkan kinerja yang tidak dapat diterima jika M pesan berukuran besar. Misalnya, platform mungkin memerlukan dukungan perangkat keras untuk hashing agar mencapai kinerja yang dapat diterima, tetapi tidak memiliki dukungan perangkat keras khusus untuk SHAKE256. Untuk beberapa *use-case*, hal ini dapat diatasi dengan menandatangani intisari pesan beserta beberapa informasi pemisahan domain, daripada menandatangani pesan secara langsung. Versi ML-DSA ini dikenal sebagai ML-DSA "*pra-hash*" atau HashML-DSA. Secara umum, versi ML-DSA "*murni*" lebih disukai.

Meskipun pembangkitan kunci untuk HashML-DSA sama dengan ML-DSA, hal ini berbeda untuk algoritma penandatanganan HashML-DSA.Sign atau algoritma verifikasi HashML-DSA.Verify. Seperti ML-DSA, algoritma penandatanganan HashML-DSA mengambil konten yang akan ditandatangani, kunci privat, dan konteks sebagai input, serta fungsi *hash* atau XOF yang akan digunakan untuk melakukan *pra-hash* pada konten yang akan ditandatangani. *Context string* memiliki panjang maksimum 255 byte.

- Penandatanganan HashML-DSA

Dalam versi HashML-DSA, input pesan ke ML-DSA.Sign_internal merupakan hasil penerapan fungsi *hash* atau XOF pada konten yang akan ditandatangani. *Output* fungsi *hash* atau XOF diawali dengan pemisah domain satu *byte*, yang menunjukkan panjang *string* konteks, *context string*, dan pengodean aturan pengkodean khusus (DER) dari fungsi *hash* atau OID XOF. Pemisah domain bernilai satu untuk penandatanganan "*pra-hash*". Pengodean DER dari OID mencakup tag dan panjangnya. Algoritma menunjukkan pengodean DER dari OID untuk SHA-256, SHA-512, dan SHAKE128. Namun, ini dapat digunakan dengan fungsi *hash* atau XOF lainnya.

Algorithm HashML-DSA.Sign (sk, M, ctx, PH)

Generate a "pre-hash" ML-DSA signature.

Input: Private key $sk \in \mathbb{B}^{32+32+64+32 \cdot ((\ell+k) \cdot \text{bitlen}(2\eta) + dk)}$,
message $M \in \{0,1\}^*$,
context string ctx (a byte string of 255 or fewer bytes),
pre-hash function PH .

Output: ML-DSA signature $\sigma \in \mathbb{B}^{\lambda/4 + \ell \cdot 32 \cdot (1 + \text{bitlen}(\gamma_1 - 1)) + \omega + k}$

```

if |ctx| > 255 then
    return  $\perp$   $\triangleright$  return an error indication if the
    context
    string is too long
end if

rnd  $\leftarrow \mathbb{B}^{32}$   $\triangleright$  for the optional deterministic
variant,
    substitute rnd  $\leftarrow \{0\}^{32}$ 
if rnd = NULL then
    return  $\perp$   $\triangleright$  return an error indication if random
    bit
    generation failed
end if

switch PH do
    case SHA-256:
        OID  $\leftarrow$ 
        0x06,0x09,0x60,0x86,0x48,0x01,0x65,0x03,0x04,0x02,0x01
         $\triangleright$  2.16.840.1.101.3.4.2.1
        PHM  $\leftarrow$  SHA256(M)
    case SHA-512:
        OID  $\leftarrow$ 
        0x06,0x09,0x60,0x86,0x48,0x01,0x65,0x03,0x04,0x02,0x03
         $\triangleright$  2.16.840.1.101.3.4.2.3
        PHM  $\leftarrow$  SHA512(M)
    case SHAKE128:
        OID  $\leftarrow$ 
        0x06,0x09,0x60,0x86,0x48,0x01,0x65,0x03,0x04,0x02,0x0B
         $\triangleright$  2.16.840.1.101.3.4.2.11
        PHM  $\leftarrow$  SHAKE128(M, 256)
    case ...
    ...
end switch
M'  $\leftarrow$  BytesToBits (IntegerToBytes (1, 1) || IntegerToBytes
(|ctx|, 1) || ctx) || OID || PHM)
 $\sigma \leftarrow$  ML-DSA.Sign_internal(sk, M', rnd)
return  $\sigma$ 

```

- Verifikasi HashML-DSA

Fungsi ini membangun M' dengan cara yang sama seperti Algoritma 4 dan meneruskan M'

yang dihasilkan ke Algoritma ML-DSA.Verify_internal untuk verifikasi. Seperti halnya pembuatan tanda tangan *pra-hash*, M' dapat dibangun di luar modul kriptografi yang menjalankan ML-DSA.Verify_internal. Namun, dalam kasus HashML-DSA, *hash* atau XOF konten harus dihitung dalam modul kriptografi yang divalidasi FIPS 140, yang mungkin merupakan modul kriptografi yang berbeda dari modul kriptografi yang menjalankan ML-DSA.Verify_internal. *Identifier* yang terkait dengan tanda tangan menunjukkan apakah ML-DSA atau versi *pra-hash* HashML-DSA dari verifikasi tanda tangan harus digunakan, serta fungsi *hash* atau XOF yang akan digunakan untuk menghitung *pra-hash*. *Context string* yang tidak kosong harus digunakan dalam verifikasi jika *string* tersebut ditentukan baik dalam pengidentifikasi tanda tangan maupun oleh aplikasi yang menggunakan tanda tangan tersebut.

Algorithm HashML-DSA.Verify (pk, M, σ, ctx, PH)

Verifies a pre-hash HashML-DSA signature.

Input: Public key $pk \in \mathbb{B}^{32+32k \cdot (\text{bitlen}(q-1)-d)}$,
message $M \in \{0,1\}^*$,
signature $\sigma \in B^{\lambda/4+\ell \cdot 32 \cdot (1+\text{bitlen}(\gamma_1-1))+\omega+k}$,
context string ctx (a byte string of 255 or fewer bytes),
pre-hash function PH.

Output: Boolean.

```
if |ctx| > 255 then
    return false
end if
```

```
switch PH do
    case SHA-256:
```

```

     $OID \leftarrow$ 
    0x06,0x09,0x60,0x86,0x48,0x01,0x65,0x03,0x04,0x02,0x01
    ▷ 2.16.840.1.101.3.4.2.1

     $PH_M \leftarrow SHA256(M)$ 
    case SHA-512:
         $OID \leftarrow$ 
        0x06,0x09,0x60,0x86,0x48,0x01,0x65,0x03,0x04,0x02,0x03
        ▷ 2.16.840.1.101.3.4.2.3

         $PH_M \leftarrow SHA512(M)$ 
    case SHAKE128:
         $OID \leftarrow$ 
        0x06,0x09,0x60,0x86,0x48,0x01,0x65,0x03,0x04,0x02,0x0B
        ▷ 2.16.840.1.101.3.4.2.11

         $PH_M \leftarrow SHAKE128(M, 256)$ 
    case ...
    ...
end switch
 $M' \leftarrow \text{BytesToBits}(\text{IntegerToBytes}(1, 1) \parallel \text{IntegerToBytes}(|\text{ctx}|, 1) \parallel \text{ctx} \parallel OID \parallel PH_M)$ 
return ML-DSA.Verify_internal( $pk, M', \sigma$ )

```

4) Fungsi Internal

a) ML-DSA.KeyGen_internal

Algoritma ML-DSA.KeyGen_internal mengambil *seed* acak 32 *byte* sebagai input dan menghasilkan kunci publik dan kunci privat yang keduanya dikodekan sebagai *string byte*. *Seed* ξ diekspansi sesuai kebutuhan menggunakan XOF (yaitu, varian byte dari SHAKE256) yang dilambangkan dengan H untuk menghasilkan nilai acak lainnya. Pengodean byte tunggal dari parameter k dan ℓ disertakan dalam input XOF untuk diskon domain.

Algorithm ML-DSA.KeyGen_internal(ξ)
Generates a public-private key pair from a seed.
Input: Seed $\xi \in \mathbb{B}^{32}$
Output: Public key $pk \in \mathbb{B}^{32+32k \cdot (\text{bitlen}(q-1)-d)}$ and private key $sk \in \mathbb{B}^{32+32+64+32 \cdot ((\ell+k) \cdot \text{bitlen}(2\eta)+dk)}$

```

 $(\rho, \rho', K) \in \mathbb{B}^{32} \times \mathbb{B}^{64} \times \mathbb{B}^{32} \leftarrow H(\xi \parallel \text{IntegerToBytes}(k, 1) \parallel \text{IntegerToBytes}(\ell, 1), 128)$ 
 $\triangleright$  expand seed
 $\hat{A} \leftarrow \text{ExpandA}(\rho)$   $\triangleright \mathbf{A}$  is generated and stored in NTT representation as  $\hat{A}$ 
 $(s_1, s_2) \leftarrow \text{ExpandS}(\rho')$ 
 $t \leftarrow \text{NTT}^{-1}(\hat{A} \circ \text{NTT}(s_1)) + s_2$   $\triangleright$  compute  $t = As_1 + s_2$ 
 $(t_1, t_0) \leftarrow \text{Power2Round}(t)$   $\triangleright$  PowerTwoRound is applied componentwise
 $pk \leftarrow \text{pkEncode}(\rho, t_1)$ 
 $tr \leftarrow H(pk, 64)$ 
 $sk \leftarrow \text{skEncode}(\rho, K, tr, s_1, s_2, t_0)$   $\triangleright K$  and  $tr$  are for use in signing
return  $(pk, sk)$ 

```

b) ML-DSA.Sign_internal

ML-DSA.Sign_internal menghasilkan tanda tangan yang dikodekan sebagai *string byte*. Algoritma ini menggunakan kunci privat sk (*string byte*), pesan terformat M' (*string bit*), dan *string 32byte rnd* sebagai input. Ada dua cara algoritma penandatanganan dapat menggunakan ML-DSA.Sign_internal: "hedged" dan "deterministik." Varian "hedged" default dari ML-DSA.Sign dan HashML-DSA.Sign menggunakan nilai acak baru untuk rnd , sedangkan varian deterministik opsional menggunakan *string byte* konstan $\{0\}_{32}$. Dalam kedua varian, *signer* pertama-tama mengekstrak hal berikut dari kunci privat: *seed* acak publik ρ , *seed* acak privat 32 *byte* K , *hash* 64 *byte* dari kunci publik tr , vektor polinomial rahasia s_1 dan s_2 , dan vektor polinomial t_0 yang mengodekan d bit signifikan terkecil dari setiap koefisien polinomial kunci publik t yang tidak

terkompresi. ρ kemudian diekspansi ke matriks A yang sama seperti pada pembangkitan kunci.

Sebelum pesan M ditandatangani, pesan tersebut digabungkan dengan *hash* kunci publik tr dan di-*hash* menjadi representasi pesan 64-byte μ menggunakan H .

Signer menghasilkan benih 64 byte tambahan ρ'' untuk keacakan privat selama setiap operasi penandatanganan. ρ'' dihitung sebagai $\rho'' \leftarrow H(K || rnd || \mu, 64)$. Dalam varian *hedged* nilai default, rnd adalah *output* dari RBG, sedangkan dalam varian deterministik, rnd adalah *string* 32 *byte* yang seluruhnya terdiri dari nol. Inilah satu-satunya perbedaan antara varian deterministik dan *hedged* nilai dari ML-DSA.Sign.

Bagian utama dari ML-DSA.Sign_internal terdiri dari *loop* pengambilan *rejection sample* dengan setiap iterasi *loop* menghasilkan tanda tangan yang valid atau *tanda tangan yang tidak valid yang pelepasannya akan membocorkan informasi tentang kunci privat. Loop diulang hingga tanda tangan yang valid dihasilkan, yang kemudian dapat dikodekan sebagai *string byte* dan *output*. Implementasi dapat membatasi jumlah iterasi dalam *loop* ini agar tidak melebihi nilai maksimum yang terbatas. Jika opsi ini digunakan dan jumlah iterasi maksimum terlampaui tanpa menghasilkan tanda tangan yang valid, ML-DSA.Sign_internal akan mengembalikan konstanta yang merepresentasikan kesalahan dan tidak ada *output* lain, sehingga menghancurkan hasil dari upaya penandatanganan yang gagal.

Loop pengambilan *rejection sample* mengikuti paradigma Fiat-Shamir *with abort* dan (selain

langkah penolakan) strukturnya mirip dengan tanda tangan Schnorr (misalnya, EdDSA).

Algorithm ML-DSA.Sign_internal (sk, M', rnd)

Deterministic algorithm to generate a signature for a formatted message M' .

Input: Private key $sk \in \mathbb{B}^{32+32+64+32 \cdot ((\ell+k) \cdot \text{bitlen}(2\eta) + dk)}$,
formatted message $M' \in \{0,1\}^*$,
and per message randomness or dummy variable $rnd \in \mathbb{B}^{32}$.

Output: Signature $\sigma \in \mathbb{B}^{\lambda/4 + \ell \cdot 32 \cdot (1 + \text{bitlen}(\gamma_1 - 1)) + \omega + k}$

$(\rho, K, tr, s_1, s_2, t_0) \leftarrow \text{skDecode}(sk)$

$\hat{s}_1 \leftarrow \text{NTT}(s_1)$

$\hat{s}_2 \leftarrow \text{NTT}(s_2)$

$\hat{t}_0 \leftarrow \text{NTT}(t_0)$

$\hat{A} \leftarrow \text{ExpandA}(\rho)$

NTT

$\triangleright A$ is generated and stored in

representation as $\hat{A}$

$\mu \leftarrow H(\text{BytesToBits}(tr) \parallel M', 64)$

$\triangleright$ message representative that may optionally be computed in a different cryptographic module

$\rho'' \leftarrow H(K \parallel rnd \parallel \mu, 64)$

$\triangleright$ compute private random seed

$\kappa \leftarrow 0$

PROFESIONAL • ELABORATIF • RESPONSIF
INTEGRATIF • SOLID • AKUNTABEL • INOVATIF

$\triangleright$ initialize counter κ

$(z, h) \leftarrow \perp$

while $(z, h) = \perp$ **do**

$\triangleright$ rejection sampling loop

$y \in R_q^l \leftarrow \text{ExpandMask}(\rho'', \kappa)$

$w \leftarrow \text{NTT}^{-1}(\hat{A} \circ \text{NTT}(y))$

$w_1 \leftarrow \text{HighBits}(w)$

$\triangleright$ signer's commitment

$\triangleright$ HighBits is applied

componentwise

$\tilde{c} \leftarrow H(\mu \parallel w_1 \text{Encode}(w_1), \lambda/4)$

$\triangleright$ commitment hash

$c \in R_q \leftarrow \text{SampleInBall}(\tilde{c})$

$\triangleright$ verifier's challenge

$\hat{c} \leftarrow \text{NTT}(c)$

$\langle \hat{c}s_1 \rangle \leftarrow \text{NTT}^{-1}(\hat{c} \circ \hat{s}_1)$

$\langle \hat{c}s_2 \rangle \leftarrow \text{NTT}^{-1}(\hat{c} \circ \hat{s}_2)$

$z \leftarrow y + \langle \hat{c}s_1 \rangle$

$\triangleright$ signer's response

$r_0 \leftarrow \text{LowBits}(w - \langle \hat{c}s_2 \rangle)$

▷ **LowBits** is applied componentwise

```
if  $\|z\|_{\infty} \geq \gamma_1 - \beta$  or  $\|r_0\|_{\infty} \geq \gamma_2 - \beta$  then  $(z, h) \leftarrow \perp$ 
                                     ▷ validity checks
else
   $\langle\langle \hat{c}t_0 \rangle\rangle \leftarrow NTT^{-1}(\hat{c} \circ \hat{t}_0)$ 
   $h \leftarrow MakeHint(w - \langle\langle \hat{c}s_2 \rangle\rangle + \langle\langle \hat{c}t_0 \rangle\rangle)$  ▷ Signer's Hint
```

▷ **MakeHint** is applied componentwise

```
if  $\|\langle\langle \hat{c}t_0 \rangle\rangle\|_{\infty} \geq \gamma_2$  or the number of 1's in h is
greater
  than  $\omega$ , then  $(z, h) \leftarrow \perp$ 
end if
end if
 $\kappa \leftarrow \kappa + \ell$  ▷ increment
counter
end while
 $\sigma \leftarrow sigEncode(\tilde{c}, z \bmod^{\pm} q, h)$ 
return  $\sigma$ 
```

c) ML-DSA.Verify_internal

Algoritma **ML-DSA.Verify_internal** menggunakan kunci publik pk (*string byte*), pesan M (*string bit*), dan tanda tangan σ (*string byte*) sebagai input. ML-DSA.Verify_internal tidak memerlukan keacakan. ML-DSA.Verify_internal menghasilkan nilai Boolean (yaitu, nilai yang benar jika tanda tangan valid terhadap pesan dan kunci publik dan salah jika tanda tangan tidak valid) sebagai output. ML-DSA.Verify_internal menentukan panjang tanda tangan σ dan kunci publik pk berdasarkan parameter yang dijelaskan sebelumnya. Jika implementasi ML-DSA.Verify_internal dapat menerima input untuk σ atau pk dengan panjang berapa pun, ia akan mengembalikan nilai *false* setiap kali panjang salah satu input ini berbeda dari panjang yang ditentukan.

Algorithm ML-DSA.Verify_internal(pk, M', σ)

Internal function to verify a signature σ for a formatted message M' .

Input: Public key $pk \in \mathbb{B}^{32+32k(\text{bitlen}(q-1)-d)}$
 message $M' \in \{0,1\}^*$
 Signature $\sigma \in \mathbb{B}^{\lambda/4+\ell \cdot 32 \cdot (1+\text{bitlen}(\gamma_1-1))+\omega+k}$

Output: Boolean

$(\rho, t_1) \leftarrow \text{pkDecode}(pk)$

$(\tilde{c}, z, h) \leftarrow \text{sigDecode}(\sigma)$

▷ signer's commitment hash

$\tilde{c},$

response z , and hint h

if $h = \perp$ **then return** false ▷ hint was not properly encoded

end if

$\hat{A} \leftarrow \text{ExpandA}(\rho)$

▷ A is generated and stored in NTT representation as

$\hat{A}$

$tr \leftarrow H(pk, 64)$

$\mu \leftarrow H(\text{BytesToBits}(tr) \parallel M', 64)$ ▷ message representative; may optionally be computed

in another cryptographic module

$c \leftarrow \text{SampleInBall}(\tilde{c})$

▷ compute verifier's challenge from $\tilde{c}$

$w'_{\text{Approx}} \leftarrow \text{NTT}^{-1}(\hat{A} \circ \text{NTT}(z)) - \text{NTT}(c) \circ \text{NTT}(t_1 \cdot 2^d)$

▷ $w'_{\text{Approx}} = Az - ct_1 \cdot 2^d$

$w'_1 \leftarrow \text{UseHint}(h, w'_{\text{Approx}})$

▷ reconstruction of signer's commitment

commitment

▷ UseHint applied componentwise

$\tilde{c}' \leftarrow H(\mu \parallel w_1 \text{Encode}(w'_1), \lambda/4)$ ▷ hash it – this should match

$\tilde{c}$

return $[\|z\|_\infty < \gamma_1 - \beta]$ and $[\tilde{c}' = \tilde{c}]$

B. Kriptografi Berbasis Kode (Code-based cryptography)

1. Deskripsi

Algoritma PQC *code-based cryptography* memanfaatkan struktur matematis pengkodeaan error (*error-correcting codes*). *Code-based cryptography* yang awalnya dirancang untuk melindungi data dari gangguan dalam transmisi, digunakan untuk

membangun skema kriptografi yang tangguh terhadap serangan baik oleh komputer klasik maupun kuantum.

a. Teori kode koreksi kesalahan

Teori kode koreksi kesalahan (*theory of error-correcting codes*) adalah salah satu fondasi utama dalam PQC. Teori ini dikembangkan untuk meningkatkan keandalan komunikasi digital dengan mendeteksi dan memperbaiki kesalahan pada data yang ditransmisikan melalui jalur komunikasi yang rentan terhadap gangguan. Namun, dalam beberapa dekade terakhir, teori ini telah dimanfaatkan dalam pengembangan algoritma kriptografi yang aman terhadap ancaman komputer kuantum. Algoritma berbasis *theory of error-correcting codes* seperti McEliece dan Niederreiter memanfaatkan sifat matematis kode koreksi kesalahan untuk menciptakan masalah yang sulit dipecahkan, baik oleh komputer klasik maupun kuantum. Berikut ini beberapa konsep dasar dari Algoritma berbasis *theory of error-correcting codes*.

1) *Linear Error Correction Coding*

Linear error correction coding adalah metode fundamental dalam pengkodean koreksi kesalahan yang menggunakan prinsip-prinsip matematis untuk mendeteksi dan memperbaiki kesalahan yang terjadi selama transmisi data. Kode ini bekerja dengan menambahkan bit redundan ke pesan asli sehingga memungkinkan pendeteksian dan perbaikan kesalahan. Proses pengkodean ini mencakup tiga tahap utama: *encoding*, *transmission*, dan *decoding*.

a) *Encoding*

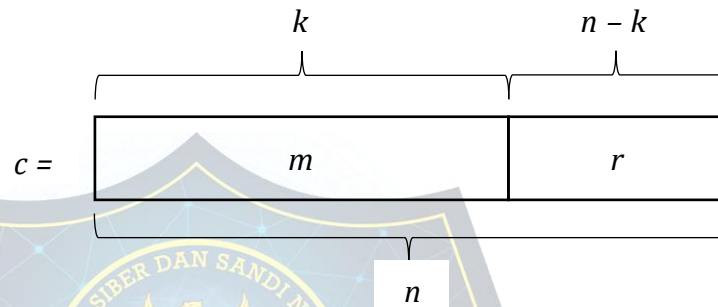
Proses pengkodean dimulai dengan mengubah pesan m menjadi *codeword* c . Pesan ini dikalikan dengan matriks generator G untuk menghasilkan *codeword*

yang menggabungkan pesan asli dengan bit redundan:

$$c = m \cdot G$$

dengan:

- m adalah vektor pesan dengan panjang k ,
- G adalah matriks generator berukuran $k \times n$,
- c adalah *codeword* dengan panjang n .



Gambar 3.4 Komponen dari *codeword* c

Codeword c terdiri atas dua bagian yaitu bagian pesan m yang memuat informasi dan bagian redundan r yang memungkinkan pendeteksian kesalahan. Dengan demikian, *codeword* dapat direpresentasikan sebagai:

$$c = (m|r)$$

b) *Transmission*

Selama transmisi melalui saluran komunikasi, *codeword* c dapat mengalami noise, yang menyebabkan kesalahan (*bit flips*). Hal ini menghasilkan *codeword* yang diterima c' , yang dimodelkan secara matematis sebagai:

$$c' = c + e$$

dengan:

- e adalah vektor kesalahan,
- Operasi $+$ di sini adalah operasi XOR bit-wise

Jika e memiliki bit "1" pada suatu posisi, maka bit yang sesuai pada c akan berubah (*flip*). Jika e memiliki bit "0", bit pada c tetap tidak berubah.

Contoh:

Jika kode kata asli adalah $c = [1,0,1,1,1,0,1]$ dan vektor kesalahan $e = [0,0,0,1,0,0,0]$, maka kode kata yang diterima menjadi:

$$c' = [1,0,1,0,1,0,1]$$

c) *Decoding*

Tahap *decoding* bertujuan untuk memverifikasi kode kata c' yang diterima, mendeteksi kesalahan, dan memperbaikinya jika memungkinkan. Proses ini dimulai dengan menghitung *syndrome* s menggunakan matriks paritas H :

$$s = H \cdot (c')^T$$

dengan:

- H adalah matriks paritas dengan ukuran $(n - k) \times n$,
- $(c')^T$ adalah transposisi dari kode kata yang diterima.

Makna dari s :

- Jika $s = 0$: Tidak ada kesalahan yang terdeteksi, kode kata valid.
- Jika $s \neq 0$: Kesalahan terdeteksi, dan informasi dalam s dapat digunakan untuk memperbaiki kesalahan, selama jumlah bit yang salah berada dalam kapasitas koreksi kode.

Penjelasan Matematisnya sebagai berikut:

Karena *codeword* asli c memenuhi $H \cdot (c')^T = 0$, maka:

$$s = H \cdot (c' + e)^T = H \cdot c^T + H \cdot e^T$$

karena $H \cdot c^T = 0$, maka:

$$s = H \cdot e^T$$

Dengan kata lain, *syndrome* s sepenuhnya bergantung pada vektor kesalahan e , sehingga dapat digunakan untuk mengidentifikasi dan memperbaiki kesalahan.

Contoh Aplikasi Linear Error Correction Coding

Misalkan sebuah pesan m dengan panjang $k = 4$ bit harus diencode menjadi *codeword* c dengan panjang $n = 7$ bit, di mana $n - k = 3$ bit adalah redundansi.

Matriks paritas H diberikan sebagai berikut:

$$H = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Codeword c harus memenuhi:

$$H \cdot c^T = 0$$

Jika pesan asli adalah $m = [1,0,1,1]$, *codeword* dapat diencode menjadi $c = [1,0,1,1,1,0,1]$, dengan menambahkan bit redundansi $r = [1,0,1]$.

2) *The Syndrome Decoding Problem*

The Syndrome Decoding Problem adalah salah satu masalah utama dalam *theory of error-correcting codes*, yang bertujuan untuk menemukan vektor kesalahan e berdasarkan nilai *syndrome* s . Dalam sistem berbasis *linear code*, *syndrome* dihitung menggunakan matriks paritas H dan kode kata yang diterima c' . Masalah ini sangat sulit secara komputasi, terutama untuk kode yang panjang, dan menjadi dasar keamanan pada algoritma kriptografi berbasis kode seperti McEliece dan Niederreiter.

Diberikan *linear code* dengan matriks paritas H berukuran $(n - k) \times n$ atas lapangan biner $GF(2)$, sebuah *syndrome* s yang terdiri dari $n - k$ bit, dan kemampuan koreksi kesalahan maksimum t . Tugas untuk menemukan vektor e sepanjang n bit yang memenuhi

syarat Hamming Weight $HW(e) \leq t$ dan $H \cdot e^T = s$ disebut sebagai *syndrome decoding problem*. Batasan $HW(e) \leq t$ menjadikan masalah ini sangat sulit dalam kasus umum. Bahkan, masalah ini memiliki kompleksitas eksponensial dalam waktu eksekusi pada kasus terburuk.

Walaupun *syndrome decoding problem* menjanjikan untuk digunakan dalam kriptografi, namun *syndrome* ini tidak cocok untuk *error-correcting code* yang tujuan utama adalah menemukan vektor kesalahan yang merusak data secara efisien. Oleh karena itu, sejak tahun 1960-an, dikembangkan banyak *error-correcting code* yang memungkinkan proses *decoding* yang efisien. Prinsip dasarnya adalah memperkenalkan struktur matematis ke dalam kode sehingga *decoding* menjadi dapat dilakukan. Algoritma *decoding* terkenal yang sering digunakan dalam praktik termasuk algoritma Berlekamp-Massey dan Patterson.

2. Skema Algoritma

Skema PQC berbasis kode adalah metode enkripsi yang memanfaatkan kompleksitas matematika untuk mengamankan data. Cara kerjanya mirip seperti membuat teka-teki yang sangat sulit dipecahkan. Pengirim pesan (Alice) akan menyamarkan pesannya menggunakan kode rahasia yang hanya diketahui oleh penerima (Bob). Selain itu, Alice juga sengaja menambahkan beberapa kesalahan kecil pada pesan tersebut. Bob, yang memiliki kunci untuk memecahkan kode, dapat dengan mudah memperbaiki kesalahan dan membaca pesan asli. Namun, bagi pihak luar yang tidak memiliki kunci rahasia, pesan tersebut akan tampak seperti kumpulan data acak yang tidak berarti. Keamanan sistem ini bergantung pada kesulitan memecahkan masalah matematika yang sangat kompleks, sehingga membuat pesan tetap aman meskipun ada upaya penyadapan.

a. Algoritma McEliece

Algoritma McEliece adalah skema kriptografi asimetris berbasis kode koreksi kesalahan, pertama kali diperkenalkan oleh Robert McEliece pada tahun 1978. Algoritma ini menggunakan kode Goppa biner, jenis kode koreksi kesalahan yang efisien untuk *decoding*. Keamanan algoritma didasarkan pada kesulitan menyelesaikan *syndrome decoding problem*, yang diketahui sangat sulit secara komputasi, bahkan untuk komputer kuantum.

Prinsip utama algoritma McEliece adalah menyembunyikan struktur kode asli (kode Goppa) sehingga penyerang tidak dapat memanfaatkan efisiensi *decoding* dari kode tersebut. Bob, sebagai pemilik kunci privat, dapat mendekripsi pesan dengan efisien menggunakan kode asli, sementara penyerang harus menghadapi masalah decoding yang sangat kompleks.

Pada tahun 2017, algoritma McEliece diajukan sebagai kandidat pada NIST PQC *Standardization Project*. McEliece memulai perjalanannya dalam proses seleksi NIST PQC pada putaran pertama (2017–2019). Algoritma ini lolos ke putaran kedua berkat desain yang stabil dan rekam jejak penelitian yang sudah lebih dari 40 tahun tanpa adanya serangan efektif terhadap keamanannya. Pada putaran kedua (2019–2020), meski tetap dianggap kuat, isu utama yang dihadapi adalah ukuran kunci publik yang sangat besar dibandingkan dengan kandidat lain, seperti algoritma berbasis latis. Namun, algoritma McEliece tetap dipertahankan sebagai finalis pada putaran ketiga (2020–2022), khususnya untuk kategori *public-key encryption* dan KEM, berkat kekuatannya yang teruji dan kestabilannya dalam jangka panjang.

Pada Agustus 2024, NIST mengumumkan pemenang standar PQC. Meskipun McEliece tidak terpilih sebagai standar, algoritma ini tetap dipertahankan sebagai “*alternate*

finalist". NIST menekankan bahwa algoritma McEliece memiliki tingkat keamanan yang sangat tinggi dan rekam jejak yang solid tanpa serangan yang berhasil, menjadikannya kandidat kuat untuk standarisasi di masa depan. Hingga saat ini, meskipun McEliece masih dipertimbangkan karena keamanan yang teruji, namun ukuran kunci publik yang besar tetap menjadi kendala utama yang menghalangi algoritma ini untuk diprioritaskan dibandingkan dengan kandidat PQC lainnya.

Berikut ini adalah proses dalam algoritma McEliece yang terdiri dari *Key Generation*, Enkripsi dan Dekripsi.

1) *Key Generation*

Proses ini bertujuan untuk membuat pasangan kunci publik dan privat:

a) Kunci Privat

- Kunci Privat terdiri dari G, S, P .
- Matriks generator G untuk kode Goppa dengan parameter $[n, k, t]_2$,
di mana:

n : Panjang kode

k : jumlah simbol Informasi

t : kemampuan koreksi kesalahan (jumlah maksimum bit yang dapat dikoreksi)

- Matriks invertibel S berukuran $k \times k$, yang digunakan untuk melakukan transformasi linear pada kode.
- Matriks permutasi P berukuran $n \times n$, yang digunakan untuk mengacak posisi bit.

b) Kunci Publik

Matriks $\hat{G}$ diterbitkan sebagai kunci publik, sementara struktur kode asli tetap tersembunyi dengan:

$$\hat{G} = S \cdot G \cdot P$$

2) Enkripsi

Pengirim pesan menggunakan kunci publik $\hat{G}$ untuk mengenkripsi pesan m :

$$c = m \cdot \hat{G} + e$$

di mana:

- m : vektor pesan asli dengan Panjang k ,
- $\hat{G}$: matriks generator publik $k \times n$,
- e : vektor kesalahan acak dengan Hamming Weight $HW(e) \leq t$, yang memastikan pesan dapat didekode oleh penerima

3) Dekripsi

Penerima pesan menerima *ciphertext* c dan menggunakan kunci privatnya untuk memulihkan pesan asli, dengan langkah-langkah berikut:

a) Hilangkan permutasi (P^{-1})

Ciphertext c yang diterima adalah kombinasi pesan terenkripsi dengan permutasi dan tambahan kesalahan:

$$c = m \cdot \hat{G} + e$$

dengan $\hat{G} = S \cdot G \cdot P$.

Menggunakan invers matriks permutasi P^{-1} untuk membalikkan efek permutasi pada *ciphertext* c :

$$c' = c \cdot P^{-1}$$

Hasil c' adalah kode kata dalam bentuk asli kode Goppa dengan kesalahan e yang masih ada.

b) *Decoding* kode Goppa

Setelah permutasi dibalik, dengan menggunakan algoritma *decoding* efisien untuk kode Goppa untuk memulihkan pesan sementara m' . Proses ini

melibatkan langkah mengidentifikasi dan menghapus vektor kesalahan e :

$$m' = \text{Decode}(c', \text{Goppa}).$$

Karena diketahui struktur kode Goppa, ia dapat menggunakan algoritma *decoding* khusus, seperti algoritma Patterson, untuk menemukan m' dari kode kata c' dan mengoreksi kesalahan.

- c) Hilangkan transformasi linear (S^{-1})

Hasil dari *decoding* adalah pesan sementara m' , yang merupakan hasil dari transformasi linear pada pesan asli m :

$$m' = m \cdot S$$

Untuk mendapatkan pesan asli, balikkan transformasi ini menggunakan inversi matriks S^{-1} :

$$m = m' \cdot S^{-1}$$

Hasil akhirnya adalah pesan asli m

- 4) Contoh

Misalkan Bob dan Alice ingin menggunakan algoritma McEliece untuk mengenkripsi dan mendekripsi pesan dengan parameter panjang kode (n)= 7; panjang pesan (k)= 4; kemampuan koreksi kesalahan (t)=1.

- a) Langkah 1: Pembangkitan Kunci

Bob sebagai penerima pesan membuat pasangan kunci publik dan privat.

- Matriks Generator (G)

Matriks generator untuk kode Goppa dipilih sebagai berikut:

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

- Matriks Permutasi (P)

Matriks permutasi dipilih untuk mengacak posisi bit:

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

- Matriks Transformasi Linear (S)

Matriks invertibel berukuran $k \times k$:

$$S = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Matriks Generator Publik ($\hat{G}$)

Bob menghitung kunci publik:

$$\hat{G} = S \cdot G \cdot P$$

Setelah perhitungan, diperoleh matriks publik $\hat{G}$:

$$\hat{G} = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Bob menyimpan G , S , dan P sebagai kunci privat dan menerbitkan $\hat{G}$ sebagai kunci publik.

b) Langkah 2: Enkripsi

Alice, pengirim pesan, menggunakan kunci publik $\hat{G}$ untuk mengenkripsi pesan m .

- Pesan Asli (m)

Alice ingin mengirimkan pesan:

$$m = [1, 0, 1, 1]$$

- Vektor Kesalahan (e)

Alice memilih vektor kesalahan acak dengan Hamming Weight $HW(e) = 1$

$$e = [0,0,0,1,0,0,0]$$

- Menghitung *Ciphertext* (c)

Alice mengenkripsi pesan menggunakan kunci publik $\hat{G}$:

$$c = m \cdot \hat{G} + e$$

Langkah pertama adalah menghitung $m \cdot \hat{G}$:

$$m \cdot \hat{G} = [1,0,1,1] \cdot \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Hasilnya:

$$m \cdot \hat{G} = [1,1,0,0,1,1,1]$$

kemudian tambahkan vektor kesalahan e :

$$c = [1,1,0,0,1,1,1] + [0,0,0,1,0,0,0] = [1,1,0,1,1,1,1]$$

Alice mengirimkan *ciphertext* $c = [1,1,0,1,1,1,1]$ kepada Bob.

c) Langkah 3: Dekripsi

- Menghilangkan permutasi (P^{-1})

Bob menggunakan invers matriks P untuk membalikkan permutasi:

$$c' = c \cdot P^{-1}$$

Dengan menghitung c' , hasilnya adalah:

$$c' = [1,0,1,1,1,1,1]$$

- *Decoding* Kode Goppa

Untuk $c' = [1,0,1,1,1,1,1]$ Bob menggunakan algoritma *decoding* untuk kode Goppa untuk menghapus kesalahan $e = [0,0,0,1,0,0,0]$.

$$m' = [1,0,1,1]$$

- Menghilangkan Transformasi Linear (S^{-1})

Bob membalikkan transformasi linear menggunakan S^{-1} .

$$m = m' \cdot S^{-1}$$

Karena S adalah matriks identitas dalam contoh ini, $m = m' = [1,0,1,1]$.

Bob berhasil memulihkan pesan asli $m = [1,0,1,1]$.

b. Algoritma Niederreiter

Algoritma Niederreiter merupakan varian dari skema McEliece yang diperkenalkan oleh Harald Niederreiter pada tahun 1986. Skema ini menggunakan *parity-check matrix* (H) sebagai pengganti generator *matrix* (G) dalam proses enkripsi, sehingga memiliki keunggulan berupa proses enkripsi yang lebih sederhana serta *overhead* yang lebih rendah dibandingkan dengan McEliece. Algoritma ini banyak digunakan dalam PQC karena kemampuannya dalam transportasi kunci rahasia dan ketahanan terhadap serangan berbasis komputer kuantum.

1) Key Generation

Output:

kunci publik $k_{pub} = (\hat{H})$ dan kunci privat $k_{pr} = (S, H, P)$

Proses:

- Pilih kode linier $C[n, k, d]$ dengan kemampuan *error correcting* maksimum t , serta matriks *parity-check* H dan fungsi `decode()`.
- Buat kunci privat $k_{pr} = (S, H, P)$, di mana S adalah matriks invertible acak berukuran $(n - k) \times (n - k)$, sedangkan P adalah matriks permutasi acak berukuran $n \times n$.
- Hitung kunci publik sebagai: $k_{pub} = \hat{H} = S \times H \times P$

2) Enkripsi

Proses:

- Konversikan pesan m menjadi error vektor e dengan panjang bit n dan Hamming weight maksimum t .
- Hitung ciphertext dengan: $c = \hat{H} \cdot e^T$

3) Dekripsi

Proses:

- a) Hitung nilai antara:

$$u = S^{-1} \cdot c$$

- b) Dekode nilai antara tersebut menggunakan fungsi

$$v = \text{decode}()$$

- c) Hitung kembali error vektor dengan: $e^T = P^{-1} \cdot v^T$.
error vektor ini mengandung pesan asli yang dikirimkan.

4) Contoh

- a) Pembuatan Kunci (*Key Generation*)

- Langkah 1: Pilih Matriks *Parity-Check* H
Misalkan Bob menggunakan kode linier dengan *parity-check* matriks berikut:

$$H = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Matriks H ini akan digunakan dalam proses enkripsi dan memiliki dimensi 3×7 .

- Langkah 2: Pilih Matriks Acak S dan P
Bob memilih dua matriks acak:

S : Matriks *invertible* acak ukuran 3×3

$$S = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

P : Matriks permutasi ukuran 7×7

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

- Langkah 3: Hitung Kunci Publik
Kunci publik dihitung dengan:

$$\hat{H} = S \times H \times P$$

$$\hat{H} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \times P$$

$$\hat{H} = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$\hat{H} = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Sehingga didapatkan kunci publik $\hat{H}$ dan kunci privat S, H, P .

b) Enkripsi

Misalkan Alice ingin mengirim pesan kepada Bob. Dalam skema Niederreiter, pesan diubah menjadi *error* vektor.

- Langkah 1: Ubah Pesan Menjadi *Error* Vektor
Misalkan pesan yang ingin dikirim adalah bit ke-3 yang aktif, sehingga vektor kesalahan e menjadi:

$$e = [0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0]$$

- Langkah 2: Hitung *Ciphertext*

Alice menghitung *ciphertext* c dengan:

$$c = \hat{H} \cdot e^T$$

$$c = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$c = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

Ciphertext yang dikirimkan Alice ke Bob adalah:

$$c = (1,0,0)$$

c) Dekripsi

Bob menerima *ciphertext* c dan ingin mendekripsinya untuk mendapatkan kembali pesan.

- Langkah 1: Hilangkan pengaruh S dengan menghitung

Bob menghitung nilai antara u :

$$u = S^{-1} \cdot c$$

Menggunakan invers dari S :

$$S^{-1} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$u = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

- Langkah 2: Decode u untuk Mendapatkan v
Menggunakan tabel *lookup* atau *decoding*, Bob mendapatkan v :

$$v = [0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0]$$

- Langkah 3: Hitung Pesan Awal

Bob menghitung:

$$e^T = P^{-1} \cdot v^T$$

$$e^T = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Pesan asli telah berhasil dienkripsi sebagai bit ke-3 aktif.

C. Kriptografi Kuadratik Multivariabel (*Multivariate-quadratic cryptography*)

1. Deskripsi

Dalam *Multivariate Public Key Cryptosystem* (MPKC), kunci publik diberikan dalam bentuk himpunan polinomial multivariat non-linear pada suatu *finite field*. Ide kriptografi multivariat dimulai

pada tahun 1980-an, dan banyak kriptografer terkemuka (seperti Ong, Schnorr, Matsumoto, Imai, Harashima, Diffie, Fell, Miyagawa, Tsujii, Kurosawa, Fujioka, dan lainnya) mencoba membangun berbagai jenis MPKC. Namun, serangan *Linearization Equations* yang dilakukan oleh Patarin terhadap sistem kriptografi Matsumoto-Imai memberikan dorongan besar terhadap perkembangan MPKC.

Secara umum, kunci publik dari MPKC adalah sistem polinomial kuadratik multivariat:

$$\begin{aligned} p^{(1)}(x_1, \dots, x_n) &= \sum_{i=1}^n \sum_{j=1}^n p_{i,j}^{(1)} \cdot x_i x_j + \sum_{i=1}^n p_i^{(1)} \cdot x_i + p_0^{(1)} \\ p^{(2)}(x_1, \dots, x_n) &= \sum_{i=1}^n \sum_{j=1}^n p_{i,j}^{(2)} \cdot x_i x_j + \sum_{i=1}^n p_i^{(2)} \cdot x_i + p_0^{(2)} \\ &\vdots \\ p^{(m)}(x_1, \dots, x_n) &= \sum_{i=1}^n \sum_{j=1}^n p_{i,j}^{(m)} \cdot x_i x_j + \sum_{i=1}^n p_i^{(m)} \cdot x_i + p_0^{(m)}. \end{aligned}$$

Semua koefisien dan variabel berasal dari $\mathbb{F}_q$ sebuah *finite field* dengan q elemen.

Himpunan polinomial publik:

$$\mathcal{P}(x_1, \dots, x_n) = (p^{(1)}(x_1, \dots, x_n), \dots, p^{(m)}(x_1, \dots, x_n))$$

secara matematis merupakan pemetaan dari $\mathbb{F}_q^n$ ke $\mathbb{F}_q^m$.

Operasi publik seperti mengenkripsi pesan atau memverifikasi tanda tangan hanya melibatkan evaluasi $\mathcal{P}(x_1, \dots, x_n)$. Proses mendekripsi ciphertext atau membangkitkan tanda tangan berhubungan dengan “membalik” pemetaan $\mathcal{P}(x_1, \dots, x_n)$, yang setara dengan menyelesaikan suatu kasus dari masalah berikut:

Masalah *Multivariate Quadratic* (MQ)

Diberikan suatu sistem $\mathcal{P}$ yang terdiri dari m polinomial kuadratik multivariat $p^{(1)}(x), \dots, p^{(m)}(x)$ dalam n variabel $x_1, \dots, x_n$, temukan suatu vektor $\bar{x} = (\bar{x}_1, \dots, \bar{x}_n)$ sedemikian sehingga:

$$\mathcal{P}(\bar{x}_1, \dots, \bar{x}_n) = (0, \dots, 0),$$

atau

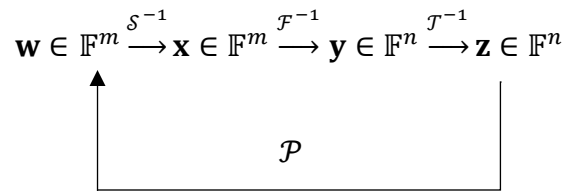
$$p^{(1)}(\bar{x}) = \dots = p^{(m)}(\bar{x}) = 0$$

Secara umum, masalah MQ terbukti bersifat *Nondeterministic Polynomial-hard* (NP-hard) pada setiap *finite field* $GF(q)$. Buktinya sangat sederhana dan langsung untuk $q = 2$ [12]. Kasus-kasus paling sulit dari MQ umumnya diperoleh ketika m dan n memiliki ordo besaran (*order of magnitude*) yang sama (ketika m jauh lebih besar dari n , atau n jauh lebih besar dari m , terdapat beberapa algoritma yang diketahui bekerja secara efisien). Sebagai catatan, bahwa sering kali algoritma terbaik yang diketahui pada MQ memiliki kompleksitas yang serupa baik untuk kasus terburuk maupun kasus acak. Hal ini juga berlaku untuk algoritma kuantum. Hingga saat ini, belum ada algoritma kuantum yang dapat menyelesaikan masalah MQ secara efisien. Untuk melakukan dekripsi dan pembuatan tanda tangan yang efisien, perlu dibangun suatu *trapdoor* rahasia untuk pemetaan $\mathcal{P}(x_1, \dots, x_n)$.

Dalam konstruksi standar MPKC, seseorang memilih suatu sistem $\mathcal{F}$ yang terdiri dari m polinomial kuadrat dalam n variabel, yang dapat dibalik dengan mudah (pemetaan pusat- *central map*). Setelah itu, dua pemetaan *Affine* invertibel $\mathcal{S}$ dan $\mathcal{T}$ dipilih untuk menyembunyikan struktur pemetaan pusat $\mathcal{F}$ dalam kunci publik. Kunci publik dari sistem kriptografi adalah pemetaan kuadrat terkomposisi $\mathcal{P} = \mathcal{S} \circ \mathcal{F} \circ \mathcal{T}$, yang dianggap sulit dibalik. Kunci privat terdiri dari $\mathcal{S}, \mathcal{F}$ dan $\mathcal{T}$, sehingga memungkinkan pembalikan $\mathcal{P}$. Pemetaan $\mathcal{S}$ dan $\mathcal{T}$ digunakan untuk melindungi pemetaan $\mathcal{F}$.

Proses standar untuk enkripsi dan dekripsi, atau pembuatan dan verifikasi tanda tangan, bekerja sebagai berikut (lihat Gambar 3.5).

Decryption/Signature Generation



Encryption/Signature Verification

Gambar 3.5 Alur kerja umum skema multivariat

Digambar ulang dari [17].

2. Skema Algoritma

a) Skema Enkripsi ($m \geq n$)

- Enkripsi

Untuk mengenkripsi pesan $\mathbf{z} \in \mathbb{F}^n$, cukup menghitung

$$\mathbf{w} = \mathcal{P}(\mathbf{z}).$$

Ciphertext dari pesan $\mathbf{z}$ adalah $\mathbf{w} \in \mathbb{F}^m$.

- Dekripsi

Untuk mendekripsi *ciphertext* $\mathbf{w} \in \mathbb{F}^m$, secara berurutan hitung:

$$\mathbf{x} = \mathcal{S}^{-1}(\mathbf{w}), \mathbf{y} = \mathcal{F}^{-1}(\mathbf{x}) \text{ dan}$$

$$\mathbf{z} = \mathcal{T}^{-1}(\mathbf{y}).$$

$\mathbf{z} \in \mathbb{F}^n$ adalah *plaintext* yang sesuai dengan *ciphertext* $\mathbf{w}$.

Karena $m \geq n$, *pre-image* dari $\mathbf{x}$ di bawah $\mathcal{F}$ dan *plaintext* yang dihasilkan unik.

b) Skema Tanda Tangan ($m \leq n$)

- Pembangkitan Tanda Tangan

Untuk menandatangani dokumen d , gunakan fungsi *hash*

$\mathcal{H}: \{0,1\}^* \rightarrow \mathbb{F}^m$ untuk menghitung nilai $\mathbf{w} = \mathcal{H}(d) \in \mathbb{F}^m$.

Kemudian hitung $\mathbf{x} = \mathcal{S}^{-1}(\mathbf{w}), \mathbf{y} = \mathcal{F}^{-1}(\mathbf{x})$ dan $\mathbf{z} = \mathcal{T}^{-1}(\mathbf{y})$.

Tanda tangan dokumen d adalah $\mathbf{z} \in \mathbb{F}^n$. $\mathcal{F}^{-1}(\mathbf{x})$ berarti menemukan satu dari banyak kemungkinan *pre-image* $\mathbf{x}$ dibawah pemetaan pusat $\mathcal{F}$. Karena $n \geq m$ maka yakin bahwa *pre-image* seperti itu ada.

- Verifikasi

Untuk memverifikasi keaslian tanda tangan $\mathbf{z}$, cukup hitung $\mathbf{w}' = \mathcal{P}(\mathbf{z})$ dan nilai hash $\mathbf{w} = \mathcal{H}(d)$ dari dokumen. Jika $\mathbf{w}' = \mathbf{w}$ maka tanda tangan diterima, jika tidak ditolak.

3. Skema tanda tangan Rainbow

Skema tanda tangan Rainbow termasuk dalam keluarga MPKC, salah satu famili kriptosistem *post quantum*. Sejarah skema tanda tangan Rainbow dapat ditelusuri kembali ke serangan *linearization equations* yang dilakukan oleh Patarin terhadap sistem kriptografi Matsumoto-Imai pada tahun 1995 [13]. Setahun kemudian, pada tahun 1996, Patarin mengembangkan skema tanda tangan *Oil and Vinegar* (OV) [14] berdasarkan serangan tersebut. Setelah versi *balanced* dari skema ini diretas oleh serangan *invariant subspace* pada tahun 1998 [15], Kipnis, Patarin, dan Goubin mengusulkan pada tahun 1999 *Unbalanced Oil and Vinegar Signature Scheme* (UOV) [16] yang hingga saat ini diyakini aman. Untuk meningkatkan efisiensi dari skema ini, Ding dan Schmidt pada tahun 2005 mengusulkan skema tanda tangan Rainbow [17], yang dapat dianggap sebagai versi berlapis dari UOV.

Skema tanda tangan Rainbow dengan u lapisan dapat dijelaskan sebagai berikut:

- Misalkan $\mathbb{F}_q$ menjadi *finite field* dengan q elemen dan $v_1 < v_2 < \dots < v_u < v_{u+1} = n$ sebagai integer. Tetapkan $V_i = \{1, \dots, v_i\}$, $o_i = v_{i+1} - v_i$, dan $O_i = \{v_i, \dots, v_{i+1}\}$ untuk $i = 1, \dots, u$. Sehingga diperoleh $|V_i| = v_i$ dan $|O_i| = o_i$ untuk $i = 1, \dots, u$.
- Pemetaan pusat $\mathcal{F}$ dari Rainbow terdiri dari $m = n - v_1$ polinomial kuadratik multivariat $f^{(v_1+1)}, \dots, f^{(n)}$ dengan bentuk sebagai berikut:

$$f^{(k)}(\mathbf{x}) = \sum_{i,j \in V_\ell} \alpha_{ij}^{(k)} x_i x_j + \sum_{i \in V_\ell, j \in O_\ell} \beta_{ij}^{(k)} x_i x_j + \sum_{i \in V_\ell \cup O_\ell} \gamma_i^{(k)} x_i + \eta^{(k)}$$

dengan $\ell \in \{1, \dots, u\}$ adalah satu-satunya integer sehingga $k \in O_\ell$.

Perhatikan bahwa dalam setiap polinomial $f^{(k)}$ dengan $k \in O_\ell$ tidak ada suku kuadratik $x_i x_j$ di mana i dan j keduanya berada di O_ℓ . Fakta ini nantinya akan digunakan untuk menghasilkan tanda tangan. Polinomial semacam ini disebut polinomial *Oil-Vinegar* ketika skema OV diperkenalkan.

- c. Untuk menyembunyikan struktur $\mathcal{F}$ dalam kunci publik, $\mathcal{F}$ dikomposisikan dengan dua pemetaan linier atau *Affine* yang invertibel $\mathcal{S}: \mathbb{F}^m \rightarrow \mathbb{F}^m$ dan $\mathcal{T}: \mathbb{F}^n \rightarrow \mathbb{F}^n$. Oleh karena itu, kunci publik Rainbow memiliki bentuk $\mathcal{P} = \mathcal{S} \circ \mathcal{F} \circ \mathcal{T}: \mathbb{F}^n \rightarrow \mathbb{F}^m$, sementara kunci privat terdiri dari tiga pemetaan $\mathcal{S}, \mathcal{F}$ dan $\mathcal{T}$, sehingga memungkinkan pembalikan pemetaan kunci publik.

d. **Pembangkitan Tanda Tangan**

Untuk membuat tanda tangan suatu pesan (atau nilai *hash*) $\mathbf{w} \in \mathbb{F}^m$, dilakukan tiga langkah berikut:

- 1) Hitung $\mathbf{x} = \mathcal{S}^{-1}(\mathbf{w}) \in \mathbb{F}^m$.
- 2) Hitung pre-image $\mathbf{y}$ dari $\mathbf{x}$ di bawah pemetaan pusat $\mathcal{F}$ menggunakan algoritma inversi pemetaan pusat (*inversion of the Rainbow central map*).
- 3) Hitung tanda tangan $\mathbf{z} \in \mathbb{F}^n$ dengan $\mathbf{z} = \mathcal{T}^{-1}(\mathbf{y})$.

Algorithm Inversion of the Rainbow central map

Input: Rainbow central map $\mathcal{F} = (f^{(v_1+1)}, \dots, f^{(n)})$, vector $\mathbf{x} \in \mathbb{F}^m$.

Output: vector $\mathbf{y} \in \mathbb{F}^n$ with $\mathcal{F}(\mathbf{y}) = \mathbf{x}$.

Choose random values for the variables $y_1, \dots, y_{v_1}$ and substitute these values into the polynomials

$f^{(i)}(i = v_1 + 1, \dots, n)$.

for $\ell = 1$ to u **do**

 Perform Gaussian Elimination on the polynomials

$f^{(i)}(i \in O_\ell)$ to get the values of the variables

$y_i(i \in O_\ell)$

 Substitute the values of $y_i(i \in O_\ell)$ into polynomials

$f^{(i)}(i = v_\ell + 1, \dots, n)$.

```

end for
return  $\mathbf{y} = (y_1, \dots, y_n)$ 

```

e. Verifikasi Tanda Tangan

Untuk memeriksa apakah $\mathbf{z} \in \mathbb{F}^n$ adalah tanda tangan yang valid untuk suatu pesan $\mathbf{w} \in \mathbb{F}^m$, cukup hitung $\mathbf{w}' = \mathcal{P}(\mathbf{z})$. Jika $\mathbf{w}' = \mathbf{w}$, tanda tangan diterima; jika tidak, ditolak.

Berikut adalah penjelasan secara mendetail mengenai Rainbow [18] sesuai dengan proposal yang diajukan pada NIST PQC putaran ke-3 beserta serangan yang dilakukan terhadap Rainbow yang menyebabkan Rainbow tidak menjadi kandidat algoritma yang distandarkan oleh NIST.

a. Spesifikasi Rainbow

1) Parameter

- *Finite field*: $\mathbb{F} = \mathbb{F}_q$, dengan q elemen.
- Integer: $0 < v_1 < \dots < v_u < v_{u+1} = n$.
- Himpunan indeks $V_i = \{1, \dots, v_i\}$, $O_i = \{v_{i+1}, \dots, v_{i+1}\}$ untuk $i = 1, \dots, u$.

Catatan: setiap $k \in \{v_{i+1}, \dots, v_{i+1}\}$ hanya termasuk dalam satu dari himpunan O_i .

- Ukuran kumpulan indeks: $|V_i| = v_i$ dan himpunan $o_i = |O_i|$ untuk $i = 1, \dots, u$.
- Jumlah persamaan: $m = n - v_1$.
- Jumlah variabel: n .

2) Pembangkitan kunci

Kunci Privat:

Kunci privat terdiri dari:

- Dua pemetaan *affine* invertibel: $\mathcal{S}: \mathbb{F}^m \rightarrow \mathbb{F}^m$ dan $\mathcal{T}: \mathbb{F}^n \rightarrow \mathbb{F}^n$
- Pemetaan pusat kuadrat $\mathcal{F}: \mathbb{F}^n \rightarrow \mathbb{F}^n$, yang terdiri dari m polinomial multivariat $f^{(v_1+1)}, \dots, f^{(n)}$.

Catatan: Berdasarkan definisi himpunan O_i , terdapat tepat satu $\ell \in \{1, \dots, u\}$ untuk setiap $k \in \{v_{i+1}, \dots, n\}$ sedemikian sehingga $k \in O_\ell$.

Polinomial $f^{(k)}$ untuk setiap $k = v_{i+1}, \dots, n$ memiliki bentuk:

$$f^{(k)}(x_1, \dots, x_n) = \sum_{i,j \in V_\ell, i \leq j} \alpha_{ij}^{(k)} x_i x_j + \sum_{i \in V_\ell, j \in O_\ell} \beta_{ij}^{(k)} x_i x_j + \sum_{i \in V_\ell \cup O_\ell} \gamma_i^{(k)} x_i + \delta^{(k)}$$

dengan $\ell \in \{1, \dots, u\}$ adalah satu-satunya integer sedemikian sehingga $k \in O_\ell$. Koefisien $\alpha_{ij}^{(k)}, \beta_{ij}^{(k)}, \gamma_i^{(k)}$, dan $\delta^{(k)}$ adalah elemen yang dipilih secara acak dari $\mathbb{F}$.

Ukuran kunci privat adalah:

$$\underbrace{m \cdot (m+1)}_{\text{pemetaan Affine}} + \underbrace{n \cdot (n+1)}_{\text{pemetaan Affine}} + \underbrace{\sum_{i=1}^u \left(\frac{v_i \cdot (v_i+1)}{2} + v_i \cdot o_i + v_i + o_i + 1 \right)}_{\text{pemetaan pusat } \mathcal{F}}$$

Kunci Publik:

Kunci publik adalah pemetaan terkomposisi:

$$\mathcal{P} = \mathcal{S} \circ \mathcal{F} \circ \mathcal{T}: \mathbb{F}^n \rightarrow \mathbb{F}^m$$

dan terdiri dari m polinomial kuadratik dalam $\mathcal{P} = \mathcal{S} \circ \mathcal{F} \circ \mathcal{T}: \mathbb{F}^n \rightarrow \mathbb{F}[x_1, \dots, x_n]$.

Ukuran kunci publik adalah:

$$m \cdot \frac{(n+1) \cdot (n+2)}{2}$$

3) Pembangkitan tanda tangan

Diberikan suatu dokumen d yang akan ditandatangani, menggunakan fungsi *hash* $\mathcal{H}: \{0,1\}^* \rightarrow \mathbb{F}^m$ untuk menghitung nilai *hash* $\mathcal{H}(d) \in \mathbb{F}^m$. Tanda tangan $\mathbf{z} \in \mathbb{F}^n$ dari dokumen d kemudian dihitung sebagai berikut:

- Hitung $\mathbf{x} = \mathcal{S}^{-1}(\mathbf{h}) \in \mathbb{F}^m$.

- Hitung pre-image $\mathbf{y} \in \mathbb{F}^n$ dari $\mathbf{x}$ di bawah pemetaan pusat $\mathcal{F}$. Hal ini dilakukan seperti yang dijelaskan pada Algoritma 1 di bawah.
- Hitung tanda tangan $\mathbf{z} \in \mathbb{F}^n$ dengan $\mathbf{z} = \mathcal{T}^{-1}(\mathbf{y})$

Algorithm 1 Inversion of the Rainbow central map

Input: Rainbom central map $\mathcal{F} = (f^{(v_1+1)}, \dots, f^{(n)})$, vector $\mathbf{x} \in \mathbb{F}^m$.

Output: vector $\mathbf{y} \in \mathbb{F}^n$ with $\mathcal{F}(\mathbf{y}) = \mathbf{x}$.

Choose random values for the variables $y_1, \dots, y_{v_1}$ and substitute these values into the polynomials

$f^{(i)}(i = v_1 + 1, \dots, n)$.

for $\ell = 1$ to u **do**

Perform Gaussian Elimination on the polynomials

$f^{(i)}(i \in O_\ell)$ to get the values of the variables

$y_i(i \in O_\ell)$

Substitute the values of $y_i(i \in O_\ell)$ into polynomials

$f^{(i)}(i = v_\ell + 1, \dots, n)$.

end for

return $\mathbf{y} = (y_1, \dots, y_n)$

4) Verifikasi Tanda Tangan

Diberikan sebuah dokumen d dan sebuah tanda tangan $\mathbf{z} \in \mathbb{F}^n$, keaslian tanda tangan dapat diverifikasi dengan langkah-langkah berikut:

- Gunakan fungsi hash $\mathcal{H}$ untuk menghitung nilai hash $\mathbf{h} = \mathcal{H}(d) \in \mathbb{F}^m$.
- Hitung $\mathbf{h}' = \mathcal{P}(\mathbf{z}) \in \mathbb{F}^n$.

Jika $\mathbf{h}' = \mathbf{h}$, tanda tangan $\mathbf{h}$ dianggap sah dan diterima. Jika tidak, tanda tangan ditolak.

Skema tanda tangan Rainbow dapat didefinisikan untuk sejumlah lapisan u . Untuk $u = 1$, skema ini menjadi versi sederhana yang dikenal sebagai UOV. Namun, memilih $u = 2$ memberikan

skema dengan perhitungan yang lebih efisien dan ukuran kunci yang lebih kecil pada tingkat keamanan yang sama. Memilih $u > 2$, memberikan sedikit keuntungan dalam hal kinerja, tetapi membutuhkan kunci yang lebih besar untuk mencapai tingkat keamanan yang sama. Oleh karena itu, demi kemudahan implementasi, pilihan $u = 2$ sering digunakan sebagai jumlah lapisan Rainbow.

Algoritma berikut menggambarkan proses pembangkitan kunci (RainbowKeyGen), pembangkitan tanda tangan (RainbowSign), dan verifikasi tanda tangan (RainbowVer) dalam bentuk algoritmik.

Algorithm 2 RainbowKeyGen: Key Generation of Rainbow
Input: Rainbow parameters q, v_1, o_1, o_2 Output: Rainbow key pair (sk, pk) $m \leftarrow o_1 + o_2$ $n \leftarrow m + v_1$ repeat $M_S \leftarrow \text{Matrix}(q, m, n)$ until $\text{IsInvertible}(M_S) == \text{TRUE}$ $c_S \leftarrow_R \mathbb{F}^m$ $S \leftarrow \text{Aff}(M_S, c_S)$ $\text{Inv}S \leftarrow M_S^{-1}$ repeat $M_T \leftarrow \text{Matrix}(q, n, n)$ until $\text{IsInvertible}(M_T) == \text{TRUE}$ $c_T \leftarrow_R \mathbb{F}^n$ $T \leftarrow \text{Aff}(M_T, c_T)$ $\text{Inv}T \leftarrow M_T^{-1}$ $\mathcal{F} \leftarrow \text{Rainbowmap}(q, v_1, o_1, o_2)$ $\mathcal{P} \leftarrow S \circ \mathcal{F} \circ T$ $sk \leftarrow (\text{Inv}S, c_S, \mathcal{F}, \text{Inv}T, c_T)$ $pk \leftarrow \mathcal{P}$

return (sk, pk)

Algorithm 3 RainbowSign: Signature generation process of Rainbow

Input: Rainbow private key $(InvS, c_s, \mathcal{F}, InvT, c_T)$, document d

Output: signature $\mathbf{z} \in \mathbb{F}^n$ such that $\mathcal{P}(\mathbf{z}) = \mathcal{H}(d)$

$\mathbf{h} \leftarrow \mathcal{H}(d)$

$\mathbf{x} \leftarrow InvS \cdot (\mathbf{h} - c_s)$

$\mathbf{y} \leftarrow InvF(\mathcal{F}, \mathbf{x})$

$\mathbf{z} \leftarrow InvT \cdot (\mathbf{y} - c_T)$

return $\mathbf{z}$

Algorithm 4 InvF: Inversion of the Rainbow central map

Input: Rainbow central map $\mathcal{F} = (f^{(v_1+1)}, \dots, f^{(n)})$, vector $\mathbf{x} \in \mathbb{F}^m$.

Output: vector $\mathbf{y} \in \mathbb{F}^n$ with $\mathcal{F}(\mathbf{y}) = \mathbf{x}$.

repeat

$y_1, \dots, y_{v_1} \leftarrow_R \mathbb{F}$

$\hat{f}^{(v_1+1)}, \dots, \hat{f}^{(n)} \leftarrow f^{(v_1+1)}(y_1, \dots, y_{v_1}), \dots, f^{(n)}(y_1, \dots, y_{v_1})$

$t, (y_{v_1+1}, \dots, y_{v_2}) \leftarrow \text{Gauss}(\hat{f}^{(v_1+1)} = x_{v_1+1}, \dots, \hat{f}^{(v_2)} = x_{v_2})$

if $t == \text{TRUE}$ **then**

$\hat{f}^{(v_2+1)}, \dots, \hat{f}^{(n)} \leftarrow \hat{f}^{(v_2+1)}(y_{v_1+1}, \dots, y_{v_2}), \dots, \hat{f}^{(n)}(y_{v_1+1}, \dots, y_{v_2})$

$t, (y_{v_2+1}, \dots, y_n) \leftarrow \text{Gauss}(\hat{f}^{(v_2+1)} = x_{v_2+1}, \dots, \hat{f}^{(n)} = x_n)$

end if

until $t == \text{TRUE}$

return $\mathbf{y} = (y_1, \dots, y_n)$

Algorithm 5 RainbowVer: Signature verification of Rainbow

Input: document d , signature $\mathbf{z} \in \mathbb{F}^n$

Output: **TRUE** or **FALSE**

```

h ←  $\mathcal{H}(d)$ 
h' ←  $\mathcal{P}(z)$ 
if h' == h then
    return TRUE
else
    return FALSE
end if

```

b. Serangan terhadap Rainbow

Pada Oktober 2020, Ward Beullens [19] memperkenalkan dua serangan *key recovery* baru, yaitu serangan *Intersection* dan serangan *Rectangular MinRank* yang diklaim dapat menyerang Rainbow pada tingkat keamanan SL1. Secara khusus, Beullens memberikan bukti komputasional bahwa serangan *key recovery* barunya tersebut berhasil dan berkinerja seperti yang diharapkan secara teoretis pada parameter keamanan Rainbow di NIST PQC *round* ke-2 di tingkat SL1, dan parameter Rainbow di NIST PQC *round* ke-3 yang sedikit lebih besar.

Penjelasannya adalah sebagai berikut:

- 1) Serangan *key recovery* untuk memulihkan kunci privat
Serangan ini memperkenalkan teknik baru untuk menemukan vektor di ruang kunci privat ($\mathcal{O}_2$) menggunakan pemetaan diferensial dari kunci publik Rainbow.

Untuk parameter SL 1 (kategori keamanan terendah), serangan ini berhasil memulihkan kunci privat hanya dalam waktu rata-rata 53 jam menggunakan laptop dengan prosesor Intel i9-10885H. Setiap percobaan memiliki probabilitas keberhasilan sekitar 1/15,06. Serangan ini memerlukan rata-rata 15,06 percobaan untuk berhasil.

2) Serangan Kombinasi (*combine attack*)

Serangan kombinasi ini menggabungkan teknik baru dengan serangan *Rectangular MinRank*. Serangan ini dirancang untuk parameter SL 3 dan SL 5 (kategori keamanan lebih tinggi). Serangan kombinasi ini mengurangi jumlah matriks yang diperlukan untuk menyelesaikan masalah *MinRank*, sehingga meningkatkan efisiensi serangan. Kompleksitas serangan untuk parameter SL 5 adalah sekitar $2^{164,1}$ operasi aritmetika di finite field.

Berdasarkan hal tersebut maka NIST **tidak memilih Rainbow** untuk distandardisasi.

4. Algoritma Kriptografi Kuadrat Multivariabel lainnya

Pada *round* berikutnya, masih terdapat beberapa algoritma Kriptografi Kuadrat Multivariabel yang masih dalam proses seleksi NIST, yaitu:

a. *Multivariate Quadratic Signature Scheme* (MAYO)

MAYO adalah skema tanda tangan berbasis persamaan kuadrat multivariat yang merupakan varian dari skema OV yang diajukan oleh Patarin pada tahun 1997. Skema ini didasarkan pada kesulitan menyelesaikan sistem persamaan kuadrat multivariat, yang merupakan masalah NP-complete.

b. *Simple Noncommutative unbalanced Oil and Vinegar* (SNOVA)

SNOVA adalah varian dari skema UOV yang dirancang untuk beroperasi di atas ring nonkomutatif guna meningkatkan efisiensi dan mengurangi ukuran kunci publik. Skema ini bertujuan untuk mengatasi masalah ukuran kunci publik yang besar yang sering dikaitkan dengan skema UOV tradisional

c. *Quasi-Random Unbalanced Oil and Vinegar* (QR-UOV)

QR-UOV adalah skema tanda tangan berbasis UOV yang menggunakan parameter kuasi-acak untuk meningkatkan

keamanan dan efisiensi. Skema ini mempertimbangkan trade-off antara ukuran kunci dan tingkat keamanan untuk berbagai level keamanan pasca-kuantum.

d. UOV

UOV adalah skema tanda tangan berbasis persamaan polinomial kuadrat multivariat. Meskipun dianggap aman, skema ini masih menghadapi tantangan terkait dengan ukuran kunci publik yang besar.

D. Kriptografi Berbasis Hash (*Hash-based cryptography*)

1. Fungsi *hash* kriptografi

Fungsi *hash* adalah komponen kriptografi fundamental yang berfungsi mengonversi input data dengan panjang variabel menjadi *output* dengan panjang tetap, yang dikenal sebagai nilai *hash* atau *digest*. Dalam proses ini, fungsi *hash* bertindak seperti "sidik jari" digital, memberikan representasi unik dari data asli dengan panjang yang jauh lebih kecil. Salah satu karakteristik utama fungsi *hash* adalah sifat deterministiknya, yang memastikan bahwa input yang sama akan selalu menghasilkan *output hash* yang sama.

Fungsi *hash* memainkan peran kunci dalam berbagai aplikasi, dari menjaga integritas data hingga mendukung algoritma kriptografi yang lebih kompleks, seperti tanda tangan digital [2]. Karena hanya menghasilkan representasi, fungsi *hash* tidak dirancang untuk memungkinkan pemulihan data asli dari nilai *hash*, menjadikannya alat ideal untuk aplikasi keamanan tertentu.

Fungsi *hash* memiliki beberapa sifat penting, yaitu:

a. *Preimage Resistance*

Preimage resistance, atau ketahanan terhadap inversi, adalah sifat yang memastikan bahwa jika nilai *hash* $H(x)$ diketahui, sangat sulit untuk menemukan input x yang menghasilkan nilai *hash* tersebut. Dalam konteks ini, "sulit" berarti bahwa upaya untuk menemukan x memerlukan waktu yang tidak praktis, bahkan dengan kemampuan komputasi

modern. Sifat ini menjadikan fungsi *hash* sangat aman untuk aplikasi seperti *password hashing*, di mana nilai *hash* dari *password* digunakan untuk membandingkan kredensial pengguna tanpa menyimpan *password* aslinya. Dengan *preimage resistance*, bahkan jika nilai *hash* terekspos, penyerang tidak dapat dengan mudah memulihkan data sensitif yang terkait dengannya.

b. *Second Preimage Resistance*

Sifat kedua ini, sering disebut sebagai ketahanan terhadap *second preimage*, memastikan bahwa jika input x dan nilai hash $H(x)$ diketahui, sangat sulit untuk menemukan input lain x' (di mana $x \neq x'$) yang menghasilkan nilai *hash* yang sama, yaitu $H(x') = H(x)$. Ketahanan ini melindungi fungsi *hash* dari manipulasi data yang dapat melewati mekanisme autentikasi. Dalam aplikasi tanda tangan digital, misalnya, jika *second preimage resistance* dilanggar, penyerang dapat mengganti pesan yang ditandatangani tanpa mengubah tanda tangan digital, yang berpotensi menghancurkan keamanan sistem.

c. *Collision Resistance*

Collision resistance adalah sifat yang memastikan bahwa sangat sulit untuk menemukan dua input berbeda x dan x' yang menghasilkan nilai *hash* yang sama, yaitu $H(x') = H(x)$. Kolisi ini, meskipun secara teoritis dapat terjadi karena sifat deterministik fungsi *hash*, dirancang agar membutuhkan upaya komputasi yang luar biasa besar untuk ditemukan. Resistensi terhadap kolisi sangat penting dalam aplikasi yang melibatkan integritas data, seperti sistem kontrol versi dan *blockchain*, di mana kolisi dapat memungkinkan perubahan data tanpa terdeteksi. *Handbook of Applied Cryptography* menekankan bahwa algoritma hash yang kuat, seperti SHA-256, dirancang untuk meminimalkan kemungkinan kolisi dalam skala komputasi realistis.

2. Konstruksi tanda tangan digital berbasis *hash*

Tanda tangan digital berbasis *hash*, juga dikenal sebagai *hash-based signature schemes*, adalah skema tanda tangan digital yang keamanannya bergantung pada keamanan fungsi *hash* kriptografi. Skema ini berbeda dengan skema tanda tangan digital yang lebih umum, seperti RSA dan ECDSA, yang keamanannya bergantung pada masalah matematika seperti faktorisasi bilangan bulat atau logaritma diskrit. Konstruksi ini bergantung pada kekuatan fungsi *hash* kriptografi untuk memastikan integritas dan keaslian pesan. Tanda tangan digital berbasis *hash* banyak digunakan dalam PQC karena tidak rentan terhadap serangan komputer kuantum seperti Algoritma Shor.

Pada tanggal 13 Agustus 2024, NIST menerbitkan standar terkait PQC, salah satunya adalah FIPS 205 *Stateless Hash-Based Digital Signature* (SLH-DSA). Standar ini menetapkan algoritma tanda tangan digital berbasis *stateless-hash* (*Stateless Hash-Based Digital Signature* -SLH-DSA).

3. Algoritma SLH-DSA

SLH-DSA berbasis algoritma SPHINCS+ yang distandarisasi dalam NIST FIPS 205 merupakan algoritma ini dirancang untuk memberikan keamanan terhadap serangan komputer kuantum, sekaligus menawarkan alternatif untuk algoritma tanda tangan digital klasik seperti RSA dan ECDSA.

a. Prinsip Kerja

SLH-DSA didasarkan pada konstruksi *hash-based signature*. Secara sederhana, algoritma ini menggunakan fungsi *hash* untuk menghasilkan tanda tangan digital. Keamanan SLH-DSA bergantung pada kesulitan dalam menemukan *collision* (dua input berbeda yang menghasilkan output *hash* yang sama) untuk fungsi *hash* yang digunakan.

1) *Stateless*

Salah satu fitur utama SLH-DSA adalah sifatnya yang "*stateless*", artinya algoritma ini tidak perlu

menyimpan informasi status antara pembuatan tanda tangan. Ini berbeda dengan algoritma tanda tangan digital klasik yang membutuhkan penyimpanan informasi status, seperti misalnya *nonce*, hasil perhitungan hash sebelumnya dan variabel atau parameter internal lainnya, yang dapat menjadi rentan terhadap serangan jika status tersebut bocor atau dimodifikasi.

Pada SLH-DSA, tidak ada informasi atau status yang perlu disimpan antara pembuatan tanda tangan, sehingga tidak ada risiko jika data tersebut bocor atau dimanipulasi. Setiap tanda tangan dibuat secara independen, tanpa tergantung pada status atau informasi sebelumnya, menjadikannya lebih aman terhadap jenis serangan yang dapat mengeksploitasi penyimpanan status.

2) Komponen Utama

SLH-DSA terdiri dari dua komponen utama:

a) FORS

Merupakan skema tanda tangan "beberapa kali" (*few-time signature scheme*). FORS digunakan untuk menandatangani sejumlah kecil pesan dengan satu pasangan kunci. SLH-DSA menghasilkan pasangan kunci publik dan privat yang mencakup sejumlah besar pasangan kunci FORS yang dihasilkan secara pseudorandom dari satu nilai *seed*.

b) XMSS

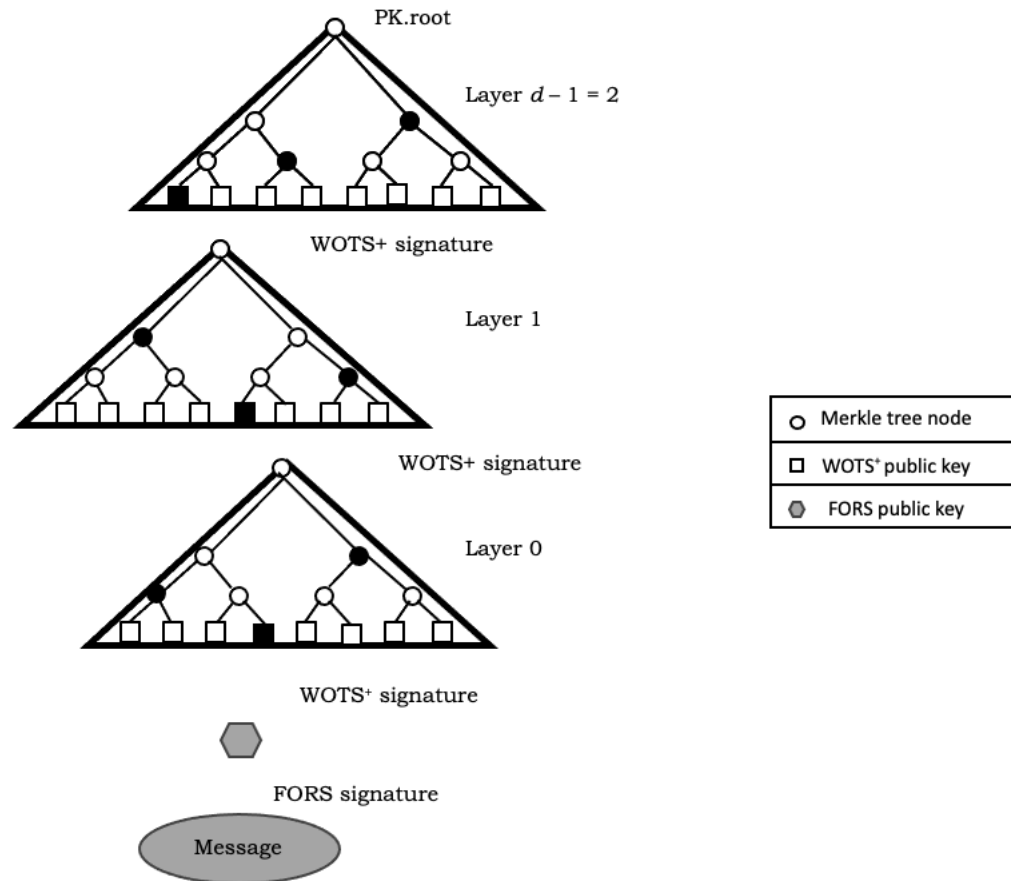
Merupakan skema tanda tangan "banyak kali" (*multi-time signature scheme*). XMSS menggunakan tanda tangan sekali pakai (*one-time signature*) WOTS+ dan *Merkle hash tree*. XMSS digunakan untuk mengautentikasi kunci publik FORS dengan membentuk struktur hierarkis yang disebut *hypertree*.

Secara konseptual, pasangan kunci SLH-DSA terdiri dari sekumpulan pasangan kunci FORS yang sangat besar. Untuk set parameter dalam standar ini, pasangan kunci SLH-DSA berisi 2^{63} , 2^{64} , 2^{66} atau 2^{68} kunci FORS, yang dihasilkan secara pseudorandom dari *single seed*. Skema tanda tangan beberapa kali FORS memungkinkan setiap pasangan kunci untuk menandatangani sejumlah kecil pesan dengan aman. Tanda tangan SLH-DSA dibuat dengan menghitung *hash* acak dari pesan, menggunakan bagian dari *digest* pesan yang dihasilkan untuk memilih kunci FORS secara acak, dan menandatangani bagian yang tersisa dari *digest* pesan dengan kunci tersebut. Tanda tangan SLH-DSA terdiri dari tanda tangan FORS dan informasi yang mengautentikasi kunci publik FORS. Informasi autentikasi dibuat menggunakan tanda tangan XMSS.

XMSS adalah skema tanda tangan multiwaktu yang dibuat menggunakan kombinasi WOTS+ *one time signature* dan *Merkle hash tree*. Kunci XMSS terdiri dari $2h'$ kunci WOTS+ dan dapat menandatangani $2h'$ pesan. Kunci publik WOTS+ dibentuk menjadi *Merkle hash tree*, dan akar pohon tersebut adalah kunci publik XMSS. (*Merkle hash tree* yang dibentuk dari kunci WOTS+ juga disebut sebagai XMSS *tree*). Tanda tangan XMSS terdiri dari tanda tangan WOTS+ dan jalur autentikasi dalam *Merkle hash tree* untuk kunci publik WOTS+.

Pada Gambar 3.6, segitiga mewakili XMSS *tree*, persegi mewakili kunci publik WOTS+, dan lingkaran mewakili simpul interior *hash tree*. Di dalam XMSS *tree*, kotak dan lingkaran yang diisi mewakili jalur

autentikasi untuk kunci publik WOTS+ yang diperlukan untuk memverifikasi tanda tangan.



Gambar 3.6 Tanda Tangan SLH-DSA

Informasi autentikasi untuk kunci publik FORS adalah tanda tangan *hypertree*. *Hypertree* adalah pohon pohon XMSS, seperti yang digambarkan dalam Gambar 3.6. Pohon tersebut terdiri dari d lapisan di mana lapisan teratas (lapisan $d-1$) terdiri dari satu pohon XMSS, lapisan berikutnya (lapisan $d-2$) terdiri dari $2h'$ pohon XMSS, dan lapisan terendah (lapisan 0) terdiri dari $2(d-1)h'$ pohon XMSS.

Untuk set parameter dalam standar ini, d adalah 7, 8, 17, atau 22. Kunci publik setiap kunci XMSS pada lapisan 0 hingga $d-2$ ditandatangani oleh kunci XMSS

pada lapisan yang lebih tinggi berikutnya. Kunci XMSS pada lapisan 0 secara kolektif memiliki $2dh' = 2h$ kunci WOTS+, yang digunakan untuk menandatangani $2h$ kunci publik FORS pada pasangan kunci SLH-DSA. Urutan tanda tangan XMSS d yang diperlukan untuk mengautentikasi kunci publik FORS saat dimulai dengan kunci publik kunci XMSS pada lapisan $d - 1$ adalah tanda tangan hypertree. Tanda tangan SLH-DSA terdiri dari tanda tangan FORS beserta tanda tangan *hypertree*.

Kunci publik SLH-DSA berisi dua komponen n -byte: (1) **PK.root**, yang merupakan kunci publik dari kunci XMSS pada lapisan $d - 1$, dan (2) **PK.seed**, yang digunakan untuk menyediakan pemisahan domain antara pasangan kunci SLH-DSA yang berbeda.

Kunci privat SLH-DSA terdiri dari n -byte seed **SK.seed** yang digunakan untuk menghasilkan secara acak semua nilai rahasia untuk kunci WOTS+ dan FORS dan kunci n -byte **SK.prf** yang digunakan dalam pembuatan *hash* acak dari pesan. Kunci privat SLH-DSA juga mencakup salinan **PK.root** dan **PK.seed**, karena nilai-nilai ini diperlukan selama pembuatan tanda tangan dan verifikasi tanda tangan.

SK.seed	n bytes
SK.prf	n bytes
PK.seed	n bytes
PK.root	n bytes

Gambar 3.8 Kunci privat SLH-DSA

PK.seed	n bytes
PK.root	n bytes

Gambar 3.7 Kunci publik SLH-DSA

Skema WOTS+, XMSS, *hypertree*, dan FORS yang dijelaskan dalam dokumen FIPS 205 [20] tidak dimaksudkan untuk digunakan sebagai skema tanda tangan mandiri, hanya menjelaskan skema komponen yang diperlukan untuk mengimplementasikan SLH-DSA. Secara khusus, bagian ini tidak menyertakan fungsi untuk pembuatan pasangan kunci, dan fungsi verifikasi tanda tangan hanya ditetapkan untuk tanda tangan *hypertree*.

Saat digunakan dalam standar [20], tanda tangan WOTS+, XMSS, dan FORS diverifikasi secara implisit menggunakan fungsi untuk menghasilkan kunci publik dari pesan dan tanda tangan. Saat memverifikasi tanda tangan SLH-DSA, *hash* acak dari pesan dan tanda tangan FORS digunakan untuk menghitung kandidat kunci publik FORS. Kandidat kunci publik FORS dan tanda tangan WOTS+ dari kunci XMSS lapisan 0 digunakan untuk menghitung kandidat kunci publik WOTS+, yang kemudian digunakan bersama dengan jalur autentikasi terkait untuk menghitung kandidat kunci publik XMSS. Kunci publik XMSS lapisan 0 kandidat digunakan bersama dengan tanda tangan XMSS lapisan 1 untuk menghitung kunci publik XMSS lapisan 1 kandidat. Proses ini diulang hingga kunci publik lapisan $d - 1$ kandidat telah dihitung. Verifikasi tanda tangan SLH-DSA berhasil jika kunci publik XMSS lapisan $d - 1$ kandidat yang dihitung sama dengan akar kunci publik SLH-DSA **PK.root**.

b. Persyaratan Tambahan pada SLH-DSA

1) *Randomness Generation*

- a) Proses pembuatan kunci SLH-DSA memerlukan tiga nilai acak masing-masing sepanjang n byte: PK.seed, SK.seed, dan SK.prf, di mana n dapat bernilai 16, 24,

atau 32 byte tergantung pada set parameter yang digunakan.

- b) Setiap nilai ini harus dihasilkan secara unik untuk setiap pembuatan kunci menggunakan Random Bit Generator (RBG) yang disetujui, sesuai dengan pedoman yang ditetapkan dalam NIST SP 800-90A, SP 800-90B, dan SP 800-90C.
 - c) RBG yang digunakan harus memiliki kekuatan keamanan minimal sebesar $8n$ bit.
- 2) Penghancuran Data Sensitif (*Destruction of Sensitive Data*)
- a) Data yang digunakan secara internal oleh algoritma pembuatan kunci dan penandatanganan dalam langkah-langkah perhitungan menengah dapat dimanfaatkan oleh pihak yang tidak berwenang untuk mendapatkan informasi tentang kunci privat, yang dapat membahayakan keamanan.
 - b) Implementasi SLH-DSA harus memastikan bahwa salinan lokal dari input dan data menengah yang sensitif dihancurkan segera setelah tidak lagi diperlukan.
- 3) Pemeriksaan Kunci (*Key Checks*)
- a) Validasi Kunci Publik: Implementasi harus memverifikasi bahwa kunci publik memiliki panjang $2n$ byte.
 - b) Pemeriksaan Kunci Privat: Pemilik kunci privat harus memastikan bahwa kunci privat memiliki panjang $4n$ byte dan menggunakan SK.seed serta PK.seed untuk menghitung ulang PK.root, kemudian membandingkannya dengan nilai yang ada untuk memastikan konsistensi dan keabsahan kunci privat.

4) Aritmatika *Floating Point*

Implementasi SLH-DSA tidak boleh menggunakan aritmatika *floating-point*, karena kesalahan pembulatan dalam operasi *floating point* dapat menyebabkan hasil yang salah dalam beberapa kasus. Dalam semua *pseudocode* dalam standar ini [20] di mana pembagian dilakukan (misalnya, x/y) dan y tidak membagi x , baik $\lfloor x/y \rfloor$ maupun $\lceil x/y \rceil$ digunakan. Keduanya dapat dihitung tanpa aritmatika *floating-point*, seperti pembagian integer biasa x/y menghitung $\lfloor x/y \rfloor$, dan $\lceil x/y \rceil = \lfloor (x + y - 1)/y \rfloor$ untuk integer non-negatif x dan integer positif y .

Algorithm 1 $\text{gen_len}_2(n, \lg w)$

Computes len_2 (Equation 5.3).

Input: Security parameter n , bits per hash chain $\lg w$.

Output: len_2 .

$w \leftarrow 2^{\lg w}$

$\text{len}_1 \leftarrow \left\lfloor \frac{8n + \lg w - 1}{\lg w} \right\rfloor$

▷Equation

5.1 and ▷Equation 5.2

$\text{max_checksum} = \text{len}_1 \cdot (w - 1)$

▷maximum possible checksum value

$\text{len}_2 \leftarrow 1$

signed using

$\text{capacity} \leftarrow w$
 $= \text{capacity} - 1$

▷maximum value that may be

▷ len_2 hash chains is $w^{\text{len}_2} - 1$

while $\text{capacity} \leq \text{max_checksum}$ **do**

$\text{len}_2 \leftarrow \text{len}_2 + 1$

$\text{capacity} \leftarrow \text{capacity} \cdot w$

end while

return len_2

Meskipun nilai len_2 (pada deskripsi WOTS+) dapat dihitung tanpa menggunakan aritmatika *floating-point* (lihat Algoritma 1), sebaiknya nilai ini dihitung terlebih dahulu. Untuk semua set parameter dalam standar ini, len_2 adalah 3.

c. Fungsi dan Addressing

Spesifikasi SLH-DSA menggunakan enam fungsi — **PRF***msg*, **H***msg*, **PRF**, **T**_ℓ, **H**, dan **F** — yang semuanya diimplementasikan menggunakan fungsi hash atau *eXtendable-Output Function* (XOF) dengan panjang *output* tetap. Input dan output dari setiap fungsi adalah *string byte*. Dalam definisi berikut, $\mathbb{B} = \{0, \dots, 255\}$ menunjukkan himpunan semua byte, $\mathbb{B}^n$ menunjukkan himpunan *string byte* dengan panjang n byte, dan $\mathbb{B}^*$ menunjukkan himpunan semua *string byte*.

- 1) **PRF***msg* (**SK**.prf, *opt_rand*, *M*) ($\mathbb{B}^n \times \mathbb{B}^n \times \mathbb{B}^* \rightarrow \mathbb{B}^n$) adalah fungsi *pseudorandom* (PRF) yang menghasilkan pengacakan (*R*) untuk *hashing* acak dari pesan yang akan ditandatangani.
- 2) **H***msg* (*R*, **PK**.seed, **PK**.root, *M*) ($\mathbb{B}^n \times \mathbb{B}^n \times \mathbb{B}^n \times \mathbb{B}^* \rightarrow \mathbb{B}^m$) digunakan untuk menghasilkan *digest* pesan yang akan ditandatangani.
- 3) **PRF** (**PK**.seed, **SK**.seed, **ADRS**) ($\mathbb{B}^n \times \mathbb{B}^n \times \mathbb{B}^{32} \rightarrow \mathbb{B}^n$) adalah PRF yang digunakan untuk menghasilkan nilai rahasia dalam kunci privat WOTS+ dan FORS.
- 4) **T**_ℓ (**PK**.seed, **ADRS**, *M*_ℓ) ($\mathbb{B}^n \times \mathbb{B}^{32} \times \mathbb{B}^{\ell n} \rightarrow \mathbb{B}^n$) adalah fungsi *hash* yang memetakan pesan ℓn -byte ke pesan n -byte.
- 5) **H** (**PK**.seed, **ADRS**, *M*₂) ($\mathbb{B}^n \times \mathbb{B}^{32} \times \mathbb{B}^{2n} \rightarrow \mathbb{B}^n$) adalah kasus khusus **T**_ℓ yang mengambil pesan $2n$ -byte sebagai input.
- 6) **F** (**PK**.seed, **ADRS**, *M*₁) ($\mathbb{B}^n \times \mathbb{B}^{32} \times \mathbb{B}^n \rightarrow \mathbb{B}^n$) adalah fungsi *hash* yang mengambil pesan n -byte sebagai input dan menghasilkan output n -byte.

Empat fungsi yang dijelaskan di atas mengambil alamat 32-byte (yaitu **ADRS**) sebagai input. **ADRS** terdiri dari nilai publik yang menunjukkan posisi nilai yang sedang dihitung oleh fungsi. Nilai **ADRS** yang berbeda digunakan untuk setiap

panggilan ke setiap fungsi. Dalam kasus **PRF**, ini digunakan untuk menghasilkan sejumlah besar nilai rahasia yang berbeda dari *single seed*. Dalam kasus **T_ℓ**, **H**, dan **F**, ini digunakan untuk mengurangi serangan multi-target. Dalam pseudocode, di mana alamat dilewatkan sebagai parameter, alamat tersebut dapat dilewatkan baik dengan referensi maupun dengan nilai.

Struktur **ADRS** sesuai dengan batas kata, dengan setiap kata memiliki panjang 4 *byte* dan nilai dikodekan sebagai bilangan bulat tak bertanda dalam urutan *byte big-endian*. Kata pertama **ADRS** menentukan alamat lapisan, yang merupakan tinggi pohon XMSS dalam *hypertree*. Pohon pada lapisan bawah memiliki tinggi nol dan pohon XMSS tunggal di bagian atas memiliki tinggi $d - 1$ (lihat Gambar 3.6). Tiga kata ADRS berikutnya menentukan alamat pohon, yang merupakan posisi pohon XMSS dalam lapisan *hypertree*. Pohon XMSS paling kiri dalam lapisan memiliki alamat pohon nol, dan pohon XMSS paling kanan dalam lapisan L memiliki alamat pohon $2(d-1-L)h' - 1$. Kata berikutnya digunakan untuk menentukan jenis alamat, yang berbeda-beda tergantung pada kasus penggunaan.

Layer address	4 <i>bytes</i>
Tree address	12 <i>bytes</i>
<i>type</i>	4 <i>bytes</i>
	12 <i>bytes</i>

Gambar 3.9 Alamat (ADRS)

Ada tujuh jenis alamat yang berbeda yang digunakan dalam SLH-DSA. Tipe alamat menentukan bagaimana 12 *byte* terakhir dari alamat akan ditafsirkan. Algoritma dalam standar

ini ditulis berdasarkan asumsi bahwa setiap kali tipe dalam **ADRS** diubah, 12 *byte* terakhir dari alamat diinisialisasi ke nol.

Tipe untuk **WOTS_HASH** (yaitu, *type* = 0) untuk alamat *hash* **WOTS+**, yang digunakan saat menghitung rantai *hash* di **WOTS+**. Saat tipe adalah **WOTS_HASH**, kata berikutnya mengodekan alamat pasangan kunci, yang merupakan indeks pasangan kunci **WOTS+** dalam pohon **XMSS** yang ditentukan oleh lapisan dan alamat pohon, dengan kunci **WOTS+** paling kiri memiliki indeks nol dan kunci **WOTS+** paling kanan memiliki indeks $2^{h'} - 1$. Berikutnya adalah alamat rantai, yang mengodekan indeks rantai dalam **WOTS+**, diikuti oleh alamat *hash*, yang mengodekan alamat fungsi *hash* dalam rantai.

layer address		layer address	
tree address		tree address	
<i>type</i> = 0 (WOTS_HASH)		<i>type</i> = 1 (WOTS_PK)	
key pair address	4 <i>bytes</i>	key pair address	4 <i>bytes</i>
chain address	4 <i>bytes</i>	<i>padding</i> = 0	8 <i>bytes</i>
hash address	4 <i>bytes</i>		

Gambar 3.11 Alamat *hash* **WOTS+**

Gambar 3.10 Alamat kompresi kunci publik **WOTS+**

Tipe untuk **WOTS_PK** (yaitu, *type* = 1) saat mengompresi kunci publik **WOTS+**. Seperti saat tipenya adalah **WOTS_HASH**, kata berikutnya mengodekan indeks pasangan kunci **WOTS+** dalam pohon **XMSS** yang ditentukan oleh lapisan dan alamat pohon. Dua kata **ADRS** yang tersisa tidak diperlukan dan ditetapkan ke nol.

Tipe untuk **TREE** (yaitu, *type* = 2) saat menghitung *hash* dalam pohon **XMSS**. Untuk tipe alamat ini, kata berikutnya selalu ditetapkan ke nol. Kata berikutnya mengodekan tinggi

simpul dalam sedang dan kata mengodekan simpul pada tersebut.	layer address	4 bytes	pohon yang dihitung, terakhir indeks tinggi
	tree address		
	<i>type</i> = 2 (TREE)		
	<i>padding</i> = 0		
	tree height		
	tree index		

Gambar 3.12 Hash tree address

Tipe ditetapkan ke FORS_TREE (yaitu, *type* = 3) saat menghitung *hash* dalam pohon FORS. Kata berikutnya adalah alamat pasangan kunci, yang mengodekan kunci FORS yang digunakan. Nilainya sama dengan alamat pasangan kunci untuk kunci WOTS+ yang digunakan untuk menandatangani kunci FORS. Dua kata berikutnya — tinggi pohon dan indeks pohon — mengodekan simpul dalam pohon FORS yang sedang dihitung. Tinggi pohon dimulai dengan nol untuk simpul daun. Indeks pohon dihitung terus-menerus di seluruh *k* pohon FORS yang berbeda. Node paling kiri pada pohon paling kiri memiliki indeks nol, dan node paling kanan pada pohon paling kanan pada level *j* memiliki indeks $k \cdot 2^{(a-j)} - 1$, di mana *a* adalah tinggi pohon.

layer address = 0	4 bytes
tree address	
<i>type</i> = 3 (FORS_TREE)	
key pair address	
tree height	
tree index	

Gambar 3.13 Alamat FORS tree

layer address = 0	4 bytes
tree address	
<i>type</i> = 4 (FORS_ROOTS)	
key pair address	
<i>Padding</i> = 0	
	8 bytes

Gambar 3.14 Alamat FORS tree roots compression

Tipe untuk FORS_ROOTS (yaitu, *type* = 4) saat mengompresi akar pohon *k* FORS. Kata berikutnya adalah alamat pasangan kunci, yang memiliki arti yang sama seperti yang ada di alamat FORS_TREE. Dua kata ADRS yang tersisa tidak diperlukan dan ditetapkan ke nol.

Tipe untuk WOTS_PRF (yaitu, *type* = 5) saat menghasilkan nilai rahasia untuk kunci WOTS+. Nilai untuk kata lain di alamat ditetapkan ke nilai yang sama seperti untuk alamat WOTS_HASH yang digunakan untuk rantai. Alamat hash selalu ditetapkan ke nol.

layer address	
tree address	
<i>type</i> = 5 (WOTS_PRF)	
key pair address	4 bytes
chain address	4 bytes
hash address = 0	4 bytes

Gambar 3.15 Alamat pembangkitan kunci WOTS+

layer address = 0	
tree address	
<i>type</i> = 6 (FORS_PRF)	
key pair address	4 bytes
Tree height = 0	4 bytes
tree index	4 bytes

Gambar 3.16 Alamat pembangkitan kunci FORS

Tipe untuk FORS_PRF (yaitu, *type* = 6) saat menghasilkan nilai rahasia untuk kunci FORS (lihat Gambar 3.16). Nilai untuk kata lain dalam alamat ditetapkan ke nilai yang sama seperti untuk alamat FORS_TREE (Gambar 3.13) yang digunakan untuk simpul daun yang sama.

Algoritma dalam standar FIPS 205 menggunakan fungsi member. Jika struktur data kompleks (misalnya, **ADRS**) berisi komponen *X*, maka **ADRS**.getX() mengembalikan nilai *X*, dan **ADRS**.setX(*Y*) menetapkan komponen *X* dalam **ADRS** ke nilai yang dimiliki oleh *Y*. Jika struktur data *s* berisi beberapa instans *X*, maka *s*.getX(*i*) mengembalikan nilai instans ke-*i* dari *X* dalam *s*. Misalnya, jika *s* adalah tanda tangan FORS, maka

$s.getAUTH(i)$ mengembalikan jalur autentikasi untuk pohon ke- i . Setiap kali jenis dalam alamat berubah, 12 *byte* terakhir dari alamat diinisialisasi ke nol. Fungsi anggota **ADRS**.setTypeAndClear(Y) untuk alamat menetapkan *type* dari **ADRS** ke Y dan menetapkan 12 *byte* terakhir ADRS ke nol. Tabel 3.6 menunjukkan notasi alternatif untuk setiap fungsi anggota yang beroperasi pada alamat.

Tabel 3.6 Fungsi Member untuk *Addresses*

Member function	Expanded notation
ADRS .setLayerAddress(l)	ADRS $\leftarrow$ toByte(l , 4) $\parallel$ ADRS [4:32]
ADRS .setTreeAddress(t)	ADRS $\leftarrow$ ADRS [0:4] $\parallel$ toByte(t , 12) $\parallel$ ADRS [16:32]
ADRS .setTypeAndClear(Y)	ADRS $\leftarrow$ ADRS [0:16] $\parallel$ toByte(Y , 4) $\parallel$ toByte(0, 12)
ADRS .setKeyPairAddress(i)	ADRS $\leftarrow$ ADRS [0:20] $\parallel$ toByte(i , 4) $\parallel$ ADRS [24:32]
ADRS .setChainAddress(i) ADRS .setTreeHeight(i)	ADRS $\leftarrow$ ADRS [0:24] $\parallel$ toByte(i , 4) $\parallel$ ADRS [28:32]
ADRS .setHashAddress(i) ADRS .setTreeIndex(i)	ADRS $\leftarrow$ ADRS [0:28] $\parallel$ toByte(i , 4)
$i \leftarrow$ ADRS .getKeyPairAddress(i)	$i \leftarrow$ toInt(ADRS [20:24], 4)
$i \leftarrow$ ADRS .getTreeIndex()	$i \leftarrow$ toInt(ADRS [28:32], 4)

d. Arrays, *Byte String*, dan Integer

Jika X adalah *string byte* dengan panjang n , maka $X[i]$ untuk $i \in \{0, \dots, n-1\}$ akan merujuk ke elemen ke- i dalam *string* X . Jika X adalah sebuah *array* yang berukuran m yang masing-masing berisi n *string byte*, maka $X[i]$ untuk $i \in \{0, \dots, m-1\}$ akan merujuk ke *string byte* ke- i dalam X , dan X akan merujuk ke *string byte* ke- $m \cdot n$ $X[0] \parallel X[1] \parallel \dots \parallel X[m-1]$. *String byte* dapat diartikan sebagai representasi *big-endian* dari sebuah integer. Dalam kasus seperti itu, *string byte* X dengan panjang n diubah menjadi integer:

$$X[0] \cdot 256^{n-1} + X[1] \cdot 256^{n-2} + \dots + X[n-2] \cdot 256 + X[n-1].$$

Demikian pula, bilangan bulat x dapat diubah menjadi *string byte* dengan panjang n dengan mencari koefisien

$$x_0, x_1, \dots, x_{n-2}, x_{n-1} \in \{0, \dots, 255\} \text{ sehingga}$$

$$x = x_0 \cdot 256^{n-1} + x_1 \cdot 256^{n-2} + \dots + x_{n-2} \cdot 256 + x_{n-1}$$

lalu menetapkan *string byte* menjadi $x_0x_1 \dots x_{n-2}x_{n-1}$. Algoritma 2 merupakan fungsi yang mengubah *string byte* X dengan panjang n menjadi integer, dan Algoritma 3 merupakan fungsi yang mengubah integer x menjadi *string byte* dengan panjang n .

Algorithm 2 toInt(X, n)

Converts a byte string to an integer.

Input: n -byte string X .

Output: Integer value of X .

```
total ← 0
for i from 0 to n - 1 do
    total ← 256 · total + X[i]
end for
return total
```

Algorithm 3 toByte(x, n)

Converts an integer to a byte string.

Input: Integer x , string length n .

Output: Byte string of length n containing binary representation of x in big-endian byte-order.

```
total ← x
for i from 0 to n - 1 do
    S[n - 1 - i] ← total mod 256    ▷ least
    significant 8 bits of total
    total ← total >> 8
end for
return S
```

Untuk skema WOTS+ dan FORS, pesan yang akan ditandatangani perlu dipecah menjadi urutan *string* b -bit, dengan setiap *string* b -bit diinterpretasikan sebagai bilangan bulat antara 0 dan $2^b - 1$, b akan menjadi nilai lg_w saat fungsi

$\text{base_}2^b$ digunakan dalam WOTS, dan b akan menjadi nilai a saat fungsi $\text{base_}2^b$ digunakan dalam FORS. Untuk set parameter dalam standar ini, lg_w adalah 4, dan a adalah 6, 8, 9, 12, atau 14.

Ini setara dengan membuat representasi $\text{base_}2^b$ dari pesan tersebut. Fungsi $\text{base_}2^b$ (Algoritma 4) mengambil *string* byte X , panjang *string* bit b , dan panjang output out_len sebagai input dan mengembalikan *array* bilangan bulat basis- $2b$ yang mewakili $out_len \cdot b$ bit pertama dari X jika *byte* individual dalam X dikodekan sebagai *string* 8-bit dalam urutan bit *big – endian*. X harus setidaknya $\lceil out_len \cdot b / 8 \rceil$ *byte* panjangnya. Karena nilai bits tidak akan pernah melebihi $b + 7$, bilangan bulat tak bertanda $b + 7$ -bit sudah cukup untuk menyimpan total (yaitu, total dapat disimpan sebagai bilangan bulat tak bertanda 32-bit).

Algorithm 4 $\text{base_}2^b(X, b, out_len)$

Computes the base 2^b representation of X .

Input: Byte *string* X of length at least $\lceil \frac{out_len \cdot b}{8} \rceil$, integer b , output length out_len .

Output: Array of out_len integers in the range $[0, \dots, 2^b - 1]$.

$in \leftarrow 0$

$bits \leftarrow 0$

$total \leftarrow 0$

for out **from** 0 **to** $out_len - 1$ **do**

while $bits < b$ **do**

$total \leftarrow (total \ll 8) + X[in]$

$in \leftarrow in + 1$

$bits \leftarrow bits + 8$

end while

$bits \leftarrow bits - b$

$baseb[out] \leftarrow (total \gg bits) \bmod 2^b$

```

end for
return baseb

```

e. *Winternitz One-Time Signature Plus Scheme (WOTS+)*

WOTS+ menggunakan dua parameter. Parameter keamanan n adalah panjang pesan dalam *byte* yang dapat ditandatangani serta panjang elemen kunci privat, elemen kunci publik, dan elemen tanda tangan. Untuk set parameter yang ditetapkan dalam NIST FIPS 205, n dapat berupa 16, 24, atau 32. Parameter kedua, lg_w menunjukkan jumlah bit yang dikodekan oleh setiap rantai *hash* yang digunakan. Dalam paper asli SPHINCS+, parameter Winternitz w digunakan sebagai parameter WOTS+ kedua, di mana w menunjukkan panjang rantai *hash* yang digunakan. Dalam FIPS 205 menggunakan parameter $lg_w = \log_2(w)$ untuk menyederhanakan notasi. lg_w adalah 4 untuk semua set parameter dalam standar ini. Parameter ini digunakan untuk menghitung empat nilai tambahan:

$$\begin{aligned}
w &= 2^{lg_w} \\
len_1 &= \left\lceil \frac{8n}{lg_w} \right\rceil \\
len_2 &= \left\lceil \frac{\log_2(len_1 \cdot (w - 1))}{lg_w} \right\rceil + 1 \\
len &= len_1 + len_2
\end{aligned}$$

Kunci privat WOTS+ terdiri dari nilai rahasia len dengan panjang n . Dalam SLH-DSA, semua ini dihasilkan dari *seed SK*. *seed* n -byte menggunakan PRF. Rantai dengan panjang w kemudian dibuat dari nilai rahasia menggunakan *chaining function*, dan nilai akhir dari setiap rantai adalah nilai publik. Kunci publik WOTS+ dihitung sebagai *hash* dari nilai publik ini. Untuk membuat tanda tangan, pesan $8n$ -bit pertama-tama diubah menjadi *array* bilangan bulat basis- w len_1 . *Checksum* kemudian dihitung untuk *string* ini, dan *checksum* diubah

menjadi *array* bilangan bulat basis- w len_2 . Tanda tangan terdiri dari entri yang sesuai dari rantai untuk setiap bilangan bulat dalam *array* pesan dan *checksum*.

Fungsi WOTS+ menggunakan dua fungsi pembantu: *base_2^b* dan *chain*. Fungsi *base_2^b* digunakan untuk memecah pesan yang akan ditandatangani dan nilai *checksum* menjadi *array* bilangan bulat basis- w . Fungsi *chain* digunakan untuk menghitung rantai *hash*. Fungsi *chain* mengambil *string* n -byte X dan bilangan bulat s dan i (di mana $i + s < w$) sebagai input dan mengembalikan hasil iterasi fungsi *hash* F pada input s kali, dimulai dari indeks i . Indeks awal 0 menunjukkan awal rantai. Fungsi *chain* juga memerlukan **PK**.seed sebagai input, yang merupakan bagian dari kunci publik SLH-DSA, dan alamat **ADRS**. Tipe dalam **ADRS** harus ditetapkan ke WOTS_HASH, dan *layer address*, *tree address*, *key pair address*, dan *chain address* harus ditetapkan ke alamat rantai yang sedang dihitung. Fungsi *chain* memperbarui alamat *hash* dalam **ADRS** dengan setiap iterasi untuk menentukan posisi saat ini dalam rantai sebelum penggunaan **ADRS** dalam **F**.

Algorithm 5 chain($X, i, s, \mathbf{PK}.seed, \mathbf{ADRS}$)

Chaining function used in WOTS⁺.

Input: Input *string* X , start index i , number of steps s , public seed **PK**.seed, address **ADRS**.

Output: Value of **F** iterated s times on X .

$tmp \leftarrow X$

for j **from** i **to** $i + s - 1$ **do**

ADRS.setHashAddress(j)

$tmp \leftarrow \mathbf{F}(\mathbf{PK}.seed, \mathbf{ADRS}, tmp)$

end for

return tmp

1) Pembangkitan Kunci Publik WOTS+

Algorithm 6 wots_pkGen(SK.seed, PK.seed, ADRS)

Generates a WOTS⁺ public key.

Input: Secret seed **SK.seed**, public seed **PK.seed**, address **ADRS**.

Output: WOTS⁺ public key *pk*.

```

    skADRS ← ADRS                                ▷ copy address to
    create key generation key address
    skADRS.setTypeAndClear(WOTS_PRF)
    skADRS.setKeyPairAddress(ADRS.getKeyPairAddress())
    for i from 0 to len - 1 do
        skADRS.setChainAddress(i)
        sk ← PRF(PK.seed, SK.seed, skADRS) ▷compute
        secret value for chain i
        ADRS.setChainAddress(i)
        tmp[i] ← chain(sk, 0, w - 1, PK.seed,
        ADRS) ▷compute public value for chain i
    end for
    wotspkADRS ← ADRS                               ▷ copy address to
    create WOTS+ public key address
    wotspkADRS.setTypeAndClear(WOTS_PK)
    wotspkADRS.setKeyPairAddress(ADRS.getKeyPairAddress())
    pk ← Tlen(PK.seed, wotspkADRS, tmp)
    ▷ compress public key
    return pk

```

Fungsi **wots_pkGen** (Algoritma 6) menghasilkan kunci publik WOTS+. Fungsi ini mengambil SK.seed dan PK.seed dari kunci privat SLH-DSA dan alamat sebagai input. Tipe dalam alamat ADRS harus ditetapkan ke WOTS_HASH, dan alamat lapisan, alamat pohon, dan

alamat pasangan kunci harus mengodekan alamat kunci publik WOTS+ yang akan dihasilkan.

Baris 4 hingga 9 dalam Algoritma 6 menghasilkan nilai publik. Untuk setiap nilai publik len , nilai rahasia yang sesuai dihasilkan dalam baris 5 dan 6, dan fungsi `chain` dipanggil untuk menghitung nilai akhir rantai dengan panjang w . Setelah nilai publik len dihitung, nilai tersebut dikompresi menjadi satu nilai n -byte dalam baris 10 hingga 13.

2) Pembangkitan Tanda Tangan WOTS+

Tanda tangan WOTS+ adalah *array string byte* len dengan panjang n , seperti yang ditunjukkan pada Gambar 3.17. Fungsi `wots_sign` (Algoritma 7) menghasilkan tanda tangan dengan mengubah pesan n -byte M menjadi *array* integer basis- w $len1$ (baris 2). *Checksum* dihitung atas M (baris 3 hingga 5). *Checksum* diubah menjadi *string byte*, yang kemudian diubah menjadi *array* integer basis- w $len2$ (baris 6 dan 7). Integer $len2$ yang mewakili *checksum* ditambahkan ke integer $len1$ yang mewakili pesan (baris 7). *Checksum* dikodekan sebagai dua *byte* dengan empat bit paling tidak signifikan menjadi nol, dan 12 bit paling signifikan ditambahkan ke pesan sebagai *array* dari tiga bilangan bulat basis-16.

Untuk setiap bilangan bulat basis- len , tanda tangan terdiri dari *node* yang sesuai dalam salah satu rantai *hash*. Untuk setiap bilangan bulat ini, baris 12 dan 13 menghitung nilai rahasia untuk rantai *hash*, dan baris 14 dan 15 menghitung *node* dalam rantai *hash* yang sesuai dengan bilangan bulat tersebut. *Node* yang dipilih digabungkan untuk membentuk tanda tangan WOTS+.

sig _{ots} [0]	n bytes
...	
sig _{ots} [$len - 1$]	n bytes

Gambar 3.17 Format data tanda tangan WOTS+

Selain pesan n -byte yang akan ditandatangani, wots_sign mengambil SK.seed dan PK.seed dari kunci privat SLH-DSA dan sebuah alamat sebagai input. Jenis pada alamat ADRS harus ditetapkan ke WOTS_HASH, dan alamat lapisan, alamat pohon, dan alamat pasangan kunci harus mengodekan alamat kunci WOTS+ yang digunakan untuk menandatangani pesan.

Algorithm 7 wots_sign(M , SK.seed, PK.seed, ADRS))

Generates a WOTS⁺ signature on an n -byte message.

Input: Message M , secret seed SK.seed, public seed PK.seed, address ADRS.

Output: WOTS⁺ signature sig .

$csum \leftarrow 0$

$msg \leftarrow \text{base_2}^b(M, lg_w, len_1) \triangleright$ convert message to base w

for i **from** 0 **to** $len_1 - 1$ **do**

$csum \leftarrow csum + w - 1 - msg[i]$

$\triangleright$ compute checksum

end for

$csum \leftarrow csum \ll ((8 - ((len_2 \cdot lg_w) \bmod 8)) \bmod 8) \triangleright$

for $lg_w = 4$, left shift by 4

$msg \leftarrow msg || \text{base_2}^b(\text{toByte}(csum, \lceil \frac{len_2 \cdot lg_w}{8} \rceil), lg_w, len_2) \triangleright$ convert to base w

$skADRS \leftarrow \text{ADRS} \triangleright$ copy address to create key generation key address

$skADRS.setTypeAndClear(WOTS_PRF)$

$skADRS.setKeyPairAddress(ADRS.getKeyPairAddress())$

```

for  $i$  from 0 to  $len - 1$  do
    skADRS.setChainAddress( $i$ )
     $sk \leftarrow \text{PRF}(\mathbf{PK}.\text{seed}, \mathbf{SK}.\text{seed}, \text{skADRS})$ 
    compute chain  $i$  secret value
    ADRS.setChainAddress( $i$ )
     $\text{sig}[i] \leftarrow \text{chain}(sk, 0, \text{msg}[i], \mathbf{PK}.\text{seed}, \mathbf{ADRS})$ 
    compute chain  $i$  signature value
end for
return  $\text{sig}$ 

```

3) Menghitung Kunci Publik WOTS+ dari Sebuah Tanda Tangan

Verifikasi tanda tangan WOTS+ melibatkan penghitungan nilai kunci publik dari pesan dan nilai tanda tangan. Verifikasi berhasil jika nilai kunci publik yang benar dihitung, yang ditentukan dengan menggunakan nilai kunci publik yang dihitung bersama dengan informasi lain untuk menghitung nilai kandidat $\mathbf{PK}.\text{root}$ dan kemudian membandingkan nilai tersebut dengan nilai $\mathbf{PK}.\text{root}$ yang diketahui dari kunci publik SLH-DSA. Bagian ini menjelaskan `wots_pkFromSig` (Algoritma 8), sebuah fungsi yang menghitung kunci publik WOTS+ kandidat dari tanda tangan WOTS+ dan pesan terkait.

Selain pesan n -byte M dan tanda tangan $len \cdot n$ -byte sig , yang ditafsirkan sebagai array *string* len n -byte, fungsi `wots_pkFromSig` mengambil $\mathbf{PK}.\text{seed}$ dari kunci publik SLH-DSA dan sebuah alamat sebagai input. Tipe alamat **ADRS** harus ditetapkan ke `WOTS_HASH`, dan alamat lapisan, alamat pohon, dan alamat pasangan kunci harus mengodekan alamat kunci WOTS+ yang digunakan untuk menandatangani pesan.

Baris 1 hingga 7 dari `wots_pkFromSig` sama dengan baris 1 hingga 7 dari `wots_sign` (Algoritma 7). Baris 8 hingga 11 dari `wots_pkFromSig` menghitung simpul akhir untuk setiap rantai menggunakan nilai tanda tangan sebagai titik awal dan nilai pesan untuk menentukan jumlah iterasi yang perlu dilakukan untuk mencapai simpul akhir. Terakhir, seperti pada baris 10 hingga 13 dari Algoritma 6, nilai kunci publik yang dihitung dikompresi dalam baris 12 hingga 15.

Algorithm 8 `wots_pkFromSig(sig, M, PK.seed, ADRS)`

Computes a WOTS⁺ public key from a message and its signature.

Input: WOTS⁺ signature `sig`, message `M`, public seed `PK.seed`, address `ADRS`.

Output: WOTS⁺ public key `pksig` derived from `sig`.

```

csum ← 0
msg ← base_2b(M, lgw, len1)                                ▷ convert
    message to base w
for i from 0 to len1 - 1 do
    csum ← csum + w - 1 - msg[i]
    ▷ compute checksum
end for
csum ← csum << ((8 - ((len2 · lgw) mod 8)) mod 8) ▷ for
    lgw = 4, left shift by 4
msg ← msg || base_2b(toByte(csum, ⌈ $\frac{len_2 \cdot lg_w}{8}$ ⌉), lgw, len2)

▷ convert to base w
for i from 0 to len - 1 do
    ADRS.setChainAddress(i)
    tmp[i] ← chain(sig[i], msg[i], w - 1 - msg[i], PK.seed,
ADRS)

```

end for

```
wotspkADRS ← ADRS                                ▷ copy address to
create WOTS+ public key address
wotspkADRS.setTypeAndClear(WOTS_PK)
wotspkADRS.setKeyPairAddress(ADRS.getKeyPairAddress())
pksig ← Tlen(PK.seed, wotspkADRS, tmp)
return pksig
```

f. *eXtended Merkle Signature Scheme (XMSS)*

XMSS memperluas skema tanda tangan WOTS+ menjadi skema yang dapat menandatangani beberapa pesan. Pohon Merkle dengan tinggi h' digunakan untuk memungkinkan kunci publik WOTS+ $2h'$ diautentikasi menggunakan satu kunci publik XMSS n -byte, yang merupakan akar pohon Merkle. Karena setiap kunci WOTS+ dapat digunakan untuk menandatangani satu pesan, kunci XMSS dapat digunakan untuk menandatangani pesan $2h'$.

Tanda tangan XMSS berukuran $(h' + len) \cdot n$ byte dan terdiri dari tanda tangan WOTS+ dan jalur autentikasi. Jalur autentikasi adalah serangkaian simpul dari pohon Merkle — satu dari setiap tingkat pohon, kecuali akar — yang memungkinkan pemeriksa untuk menghitung akar pohon saat digunakan bersama dengan kunci publik WOTS+ yang dapat dihitung dari tanda tangan WOTS+.

SIG _{WOTS+}	$len \cdot n$ bytes
AUTH[0]	n bytes
...	
AUTH[$h' - 1$]	n bytes

Gambar 3.18 Format data tanda tangan XMSS

1) Membangkitkan *Merkle Hash Tree*

Fungsi `xmss_node` (Algoritma 9) menghitung simpul-simpul pohon XMSS. Fungsi `xmss_node` mengambil input **SK**.seed dan **PK**.seed dari kunci privat SLH-DSA; indeks simpul target i , yang merupakan indeks simpul yang sedang dihitung; tinggi simpul target z , yang merupakan tinggi di dalam pohon Merkle dari simpul yang sedang dihitung; dan alamat. Alamat **ADRS** harus

memiliki alamat lapisan dan alamat pohon yang ditetapkan ke pohon XMSS di mana simpul sedang dihitung. Tinggi dan indeks simpul target harus memenuhi $z \leq h'$ dan $i < 2^{(h'-z)}$.

Setiap simpul di pohon XMSS adalah akar dari subpohon, dan Algoritma 9 menghitung akar subpohon secara rekursif. Jika subpohon terdiri dari satu simpul daun, maka fungsi tersebut hanya mengembalikan nilai kunci publik WOTS+ simpul tersebut (baris 2 hingga 4). Jika tidak, fungsi tersebut menghitung akar subpohon kiri (baris 6) dan subpohon kanan (baris 7) dan melakukan *hashing* bersama-sama (baris 8 hingga 11).

Algorithm 9 `xmss_node`(**SK**.seed, i , z , **PK**.seed, **ADRS**)

Computes the root of a Merkle subtree of WOTS+ public keys.

Input: Secret seed **SK**.seed, target node index i , target node height z , public seed **PK**.seed, address **ADRS**.

Output: n -byte root node.

```

if  $z = 0$  then
    ADRS.setTypeAndClear(WOTS_HASH)
    ADRS.setKeyPairAddress( $i$ )
     $node \leftarrow wots\_pkGen$ (SK.seed, PK.seed, ADRS)
else
```

```

    lnode ← xmss_node(SK.seed, 2i, z - 1,
PK.seed, ADRS)

    rnode ← xmss_node(SK.seed, 2i + 1, z - 1,
PK.seed, ADRS)

    ADRS.setTypeAndClear(TREE)

    ADRS.setTreeHeight(z)

    ADRS.setTreeIndex(i)

    node ← H(PK.seed, ADRS, lnode || rnode)

end if

return node

```

2) Membangkitkan Tanda Tangan XMSS

Fungsi *xmss_sign* (Algoritma 10) membuat tanda tangan XMSS pada pesan n -byte M dengan membuat jalur autentikasi (baris 1 hingga 4) dan menandatangani M dengan kunci WOTS+ yang sesuai (baris 5 hingga 7). Selain M , *xmss_sign* mengambil **SK**.seed dan **PK**.seed dari kunci privat SLH-DSA, alamat, dan indeks sebagai input. Alamat **ADRS** harus memiliki alamat lapisan dan alamat pohon yang ditetapkan ke kunci XMSS yang digunakan untuk menandatangani pesan, dan indeks idx harus menjadi indeks kunci WOTS+ dalam pohon XMSS yang akan digunakan untuk menandatangani pesan.

Jalur autentikasi terdiri dari simpul saudara dari setiap simpul yang berada di jalur dari kunci WOTS+ yang digunakan ke akar. Misalnya, jika pesan ditandatangani dengan K_2 , maka K_2 , $n_{1,1}$, dan $n_{2,0}$ adalah simpul jalur, dan jalur autentikasi terdiri dari K_3 , $n_{1,0}$, dan $n_{2,1}$. Pada baris 2 Algoritma 10, $\lfloor idx/2^j \rfloor$ adalah simpul jalur, dan $\lfloor idx/2^j \rfloor \oplus 1$ adalah simpul jalur autentikasi. Baris 3 menghitung nilai simpul jalur autentikasi.

Algorithm 10 `xmss_sign( $M$ ,  $\mathbf{SK}.\text{seed}$ ,  $\text{idx}$ ,  $\mathbf{PK}.\text{seed}$ ,  $\mathbf{ADRS}$ )`

Generates an XMSS signature.

Input: n -byte message M , secret seed $\mathbf{SK}.\text{seed}$, index idx , public seed $\mathbf{PK}.\text{seed}$, address $\mathbf{ADRS}$.

Output: XMSS signature $\text{SIG}_{\text{XMSS}} = (\text{sig} \parallel \text{AUTH})$.

for j **from** 0 **to** $h' - 1$ **do**

▷ build authentication path

$k \leftarrow \lfloor \text{idx}/2^j \rfloor \oplus 1$

$\text{AUTH}[j] \leftarrow \text{xmss_node}(\mathbf{SK}.\text{seed}, k, j, \mathbf{PK}.\text{seed}, \mathbf{ADRS})$

end for

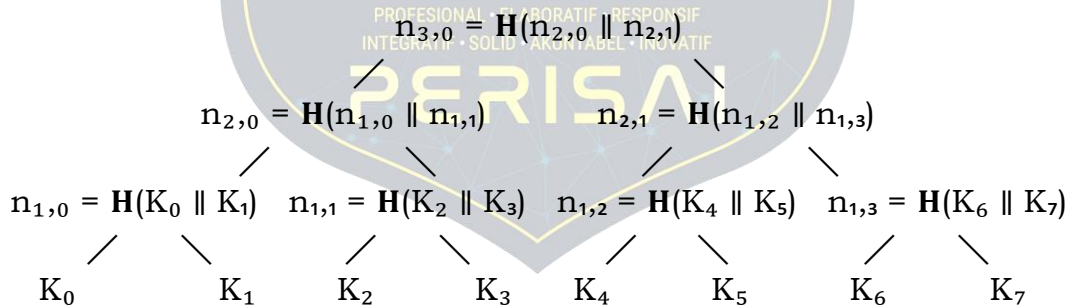
$\mathbf{ADRS}.\text{setTypeAndClear}(\text{WOTS_HASH})$

$\mathbf{ADRS}.\text{setKeyPairAddress}(\text{idx})$

$\text{sig} \leftarrow \text{wots_sign}(M, \mathbf{SK}.\text{seed}, \mathbf{PK}.\text{seed}, \mathbf{ADRS})$

$\text{SIG}_{\text{XMSS}} \leftarrow \text{sig} \parallel \text{AUTH}$

return SIG_{XMSS}



Gambar 3.19 Merkle Hash Tree

3) Menghitung Kunci Publik XMSS dari Tanda Tangan

Memverifikasi tanda tangan XMSS melibatkan penghitungan nilai kunci publik dari pesan dan nilai tanda tangan. Verifikasi berhasil jika nilai kunci publik yang benar dihitung, yang ditentukan dengan menggunakan nilai kunci publik yang dihitung bersama

dengan informasi lain untuk menghitung nilai kandidat **PK.root**, lalu membandingkan nilai tersebut dengan nilai **PK.root** yang diketahui dari kunci publik SLH-DSA. Bagian ini menjelaskan `xmss_pkFromSig` (Algoritma 11), sebuah fungsi yang menghitung kunci publik XMSS kandidat dari tanda tangan XMSS dan pesan terkait.

Selain pesan n -byte M dan tanda tangan $(len+h') \cdot n$ -byte SIG_{XMSS} , `xmss_pkFromSig` mengambil **PK.seed** dari kunci publik SLH-DSA, alamat, dan indeks sebagai input. Alamat **ADRS** harus ditetapkan ke alamat lapisan dan alamat pohon kunci XMSS yang digunakan untuk menandatangani pesan, dan indeks idx harus menjadi indeks kunci WOTS+ dalam pohon XMSS yang digunakan untuk menandatangani pesan.

Algoritma 11 dimulai dengan menghitung kunci publik WOTS+ pada baris 1 hingga 5. Akar kemudian dihitung pada baris 6 hingga 18. Dimulai dengan simpul daun (yaitu, kunci publik WOTS+), simpul pada setiap tingkat pohon dihitung dengan menggabungkan simpul yang dihitung pada iterasi sebelumnya dengan simpul jalur autentikasi yang sesuai. Pada baris 12 dan 15, **AUTH** ditafsirkan sebagai larik *string* h' n -byte.

Algorithm 11 `xmss_sign( $M$ , SK.seed,  $idx$ , PK.seed, ADRS)`

Computes an XMSS public key from an XMSS signature.

Input: Index idx , XMSS signature $SIG_{XMSS} = sig || AUTH$, n -byte message M , public seed **PK.seed**, address **ADRS**.

Output: n -byte root value `node[0]`.

`ADRS.setTypeAndClear(WOTS_HASH)`

compute WOTS⁺ pk from WOTS⁺ *sig*



```

ADRS.setKeyPairAddress(idx)
sig ← SIGXMSS.getWOTSSig()
▷ SIGXMSS[0 : len · n]
  AUTH ← SIGXMSS.getXMSSAUTH()
▷ SIGXMSS[len · n : (len + h') · n]
  node[0] ← wots_pkFromSig(sig, M, PK.seed,
ADRS)

  ADRS.setTypeAndClear(TREE) ▷
compute root from WOTS+ pk and AUTH
  ADRS.setTreeIndex(idx)
  for k from 0 to h' - 1 do
    ADRS.setTreeHeight(k + 1)
    if [idx/2k] is even then
ADRS.setTreeIndex(ADRS.getTreeIndex() / 2)
      node[1] ← H(PK.seed, ADRS, node[0] ||
AUTH[k])
    else
ADRS.setTreeIndex((ADRS.getTreeIndex() - 1) / 2)
      node[1] ← H(PK.seed, ADRS, AUTH[k] ||
node[0])
    end if
    node[0] ← node[1]
  end for
  return node[0]

```

g. SLH-DSA *Hypertree*

SLH-DSA memerlukan sejumlah besar kunci WOTS+ untuk menandatangani kunci publik FORS. Karena tidak mungkin bagi set parameter dalam standar ini untuk memiliki satu kunci XMSS dengan begitu banyak kunci WOTS+, SLH-DSA menggunakan *hypertree* untuk menandatangani kunci

FORS. Seperti yang digambarkan dalam, hypertree adalah pohon dari pohon XMSS. Kunci XMSS pada lapisan terendah digunakan untuk menandatangani kunci publik FORS, dan kunci XMSS pada setiap lapisan lainnya digunakan untuk menandatangani kunci publik XMSS pada lapisan di bawahnya.

Hypertree memiliki d lapisan pohon XMSS dengan setiap pohon XMSS menjadi pohon Merkle dengan tinggi h' , sehingga tinggi total hypertree adalah $h = d \cdot h'$. Lapisan teratas (lapisan $d - 1$) adalah pohon XMSS tunggal, dan kunci publik dari pasangan kunci XMSS ini (yaitu, akar pohon Merkle) adalah kunci publik dari hypertree (**PK.root**). Lapisan berikutnya memiliki pohon XMSS $2h'$, dan kunci publik dari masing-masing kunci XMSS ini ditandatangani oleh salah satu dari kunci WOTS+ $2h'$ yang merupakan bagian dari kunci XMSS lapisan teratas. Lapisan terendah memiliki pohon XMSS $2h-h'$, yang menyediakan kunci WOTS+ $2h$ untuk menandatangani kunci FORS.

1) Pembangkitan Tanda Tangan *Hypertree*

Tanda tangan *hypertree* memiliki panjang $(h + d \cdot \text{len}) \cdot n$ byte dan terdiri dari serangkaian tanda tangan XMSS d , dimulai dengan tanda tangan yang dihasilkan menggunakan kunci XMSS di lapisan terendah dan diakhiri dengan tanda tangan yang dihasilkan menggunakan kunci XMSS di lapisan atas.

XMSS signature SIG _{XMSS} (layer 0)	$(h' + \text{len}) \cdot n$ bytes
XMSS signature SIG _{XMSS} (layer 1)	$(h' + \text{len}) \cdot n$ bytes
...	
XMSS signature SIG _{XMSS} (layer $d - 1$)	$(h' + \text{len}) \cdot n$ bytes

Gambar 3.20 Format data Tanda tangan *Hypertree*

Selain pesan n -byte M , fungsi `ht_sign` (Algoritma 12) mengambil input **SK**.seed dan **PK**.seed dari kunci privat SLH-DSA, indeks pohon XMSS pada lapisan terendah

yang akan menandatangani pesan idx_{tree} , dan indeks kunci WOTS+ dalam pohon XMSS yang akan menandatangani pesan idx_{leaf} . Algoritma 12 dimulai pada baris 1 hingga 3 dengan menandatangani M dengan kunci XMSS yang ditentukan menggunakan kunci WOTS+ dalam kunci XMSS yang ditentukan oleh idx_{leaf} . Kunci publik XMSS diperoleh (baris 5 atau 14) untuk setiap lapisan berikutnya dan ditandatangani oleh kunci yang sesuai pada tingkat yang lebih tinggi (baris 7 hingga 11).

Algorithm 12 $ht_sign(M, SK.seed, PK.seed, idx_{tree}, idx_{leaf})$

Generates a hypertree signature.

Input: Message M , private seed $SK.seed$, public seed $PK.seed$, tree index idx_{tree} , leaf index idx_{leaf} .

Output: HT signature SIG_{HT} .

$ADRS \leftarrow toByte(0, 32)$

$ADRS.setTreeAddress(idx_{tree})$

$SIG_{tmp} \leftarrow xmss_sign(M, SK.seed, idx_{leaf}, PK.seed, ADRS)$

$SIG_{HT} \leftarrow SIG_{tmp}$

$root \leftarrow xmss_pkFromSig(idx_{leaf}, SIG_{tmp}, M, PK.seed, ADRS)$

for j **to** $d - 1$ **do**

$idx_{leaf} \leftarrow idx_{tree} \bmod 2^{h'}$ $\triangleright h'$ least significant bits of idx_{tree}

$idx_{tree} \leftarrow idx_{tree} \gg h'$ $\triangleright$ remove least significant h' bits from idx_{tree}

$ADRS.setLayerAddress(j)$

$ADRS.setTreeAddress(idx_{tree})$

```

    SIGtmp ← xmss_sign(root, SK.seed,
idxleaf, PK.seed, ADRS)
    SIGHT ← SIGHT || SIGtmp
    If j < d-1 then
        root ← xmss_pkFromSig (idxleaf, SIGtmp,
root, PK.seed, ADRS)
    end if
end for
return SIGHT

```

2) Verifikasi tanda Tangan *Hypertree*

Verifikasi tanda tangan hypertree bekerja dengan melakukan panggilan d ke `xmss_pkFromSig` (Algoritma 11) dan membandingkan hasilnya dengan kunci publik hypertree. Selain pesan n -byte M dan tanda tangan $(h + d \cdot \text{len}) \cdot n$ -byte SIG_{HT}, `ht_verify` (Algoritma 13) mengambil input **PK**.seed dan **PK**.root dari kunci publik SLH-DSA, indeks pohon XMSS pada lapisan terendah yang menandatangani pesan idx_{tree} , dan indeks kunci WOTS+ di dalam pohon XMSS yang menandatangani pesan idx_{leaf} .

Pada setiap lapisan, baik pesan M atau kunci publik terkomputasi dari kunci XMSS di lapisan bawah diberikan bersama dengan tanda tangan XMSS yang sesuai ke `xmss_pkFromSig` untuk memperoleh kunci publik XMSS terkomputasi lapisan tersebut. Jika kunci publik XMSS terkomputasi dari pohon lapisan atas sama dengan kunci publik hypertree yang diketahui, **PK**.root, maka verifikasi berhasil.

Algorithm 13 `ht_verify`(M , SIG_{HT}, PK.seed, idx_{tree} , idx_{leaf} , PK.root)
Verifies a hypertree signature.

```

Input: Message  $M$ , signature  $SIG_{HT}$ , public seed
        PK.seed, tree index  $idx_{tree}$ , leaf index
         $idx_{leaf}$ , HT public key PK.root.
Output: Boolean

ADRS  $\leftarrow$  toByte
ADRS.setTreeAddress ( $idx_{tree}$ )
SIGtmp  $\leftarrow$  SIGHT.getXMSSSignature(0)  $\triangleright$  SIGHT[0:( $h' +$ 
 $len$ ). $n$ ]
node  $\leftarrow$  xmss_pkFromSig ( $idx_{leaf}$ ,
SIGtmp,  $M$ , PK.seed, ADRS)
for  $j$  to  $d - 1$  do
     $idx_{leaf} \leftarrow idx_{tree} \bmod 2^{h'}$   $\triangleright$   $h'$  least significant
    bits of  $idx_{tree}$ 
     $idx_{tree} \leftarrow idx_{tree} \gg h'$   $\triangleright$  remove least
    significant  $h'$  bits from  $idx_{tree}$ 
    ADRS.setLayerAddress ( $j$ )
    ADRS.setTreeAddress ( $idx_{tree}$ )
    SIGtmp  $\leftarrow$  SIGHT.getXMSSSignature( $j$ )  $\triangleright$  SIGHT
    [ $j \cdot (h' + len) n : (j + 1) \cdot (h' + len) n$ ]
    node  $\leftarrow$ 
    xmss_pkFromSig ( $idx_{leaf}$ , SIGtmp, node, PK.seed, ADRS)
end for
if node = PK.root then
    return true
else
    return false
end if

```

h. Forest of Random Subsets (FORS)

FORS adalah skema tanda tangan beberapa kali yang digunakan untuk menandatangani *digest* pesan yang sebenarnya. Tidak seperti WOTS+, yang memungkinkan

pemalsuan jika kunci digunakan dua kali, keamanan kunci FORS menurun secara bertahap seiring dengan bertambahnya jumlah tanda tangan.

FORS menggunakan dua parameter, yaitu k dan $t = 2a$. Kunci privat FORS terdiri dari k set *string* t n -byte, yang semuanya dibuat secara acak dari *seed* SK.seed. Setiap set k dibentuk menjadi pohon Merkle, dan akar pohon tersebut di-hash bersama untuk membentuk kunci publik FORS. Tanda tangan pada *digest* pesan $k \cdot a$ -bit terdiri dari k elemen dari kunci privat, satu dari setiap set yang dipilih menggunakan a bit dari *digest* pesan, bersama dengan jalur autentikasi untuk setiap elemen ini.

Private key value (tree 0)	n bytes
AUTH (tree 0)	$a \cdot n$ bytes
...	
Private key value (tree $k-1$)	n bytes
AUTH (tree $k-1$)	$a \cdot n$ bytes

Gambar 3.21 Format data Tanda tangan FORS

1) Pembangkitan FORS Secret Values

Fungsi `fors_skGen` (Algoritma 14) menghasilkan *string* n -byte dari kunci privat FORS. Fungsi tersebut mengambil **SK.seed** dan **PK.seed** dari kunci privat SLH-DSA, alamat, dan indeks sebagai input. *Type* dalam alamat ADRS harus ditetapkan ke `FORS_TREE`, dan alamat pohon serta alamat pasangan kunci harus ditetapkan ke indeks kunci WOTS+ dalam pohon XMSS yang menandatangani kunci FORS. Alamat lapisan harus ditetapkan ke nol. Indeks *idx* adalah indeks nilai rahasia FORS dalam set pohon FORS.

Algorithm 14 <code>fors_skGen(SK.seed, PK.seed, ADRS, idx)</code>
<i>Generates a FORS private-key value.</i>
Input: Secret seed SK.seed, public seed PK.seed, address ADRS, secret key index <i>idx</i> .


```

Output:  $n$ -byte FORS private-key value.
skADRS  $\leftarrow$  ADRS           ▷copy address to create key
generation address
skADRS.setTypeAndClear(FORS_PRF)
skADRS.setKeyPairAddress(ADRS.getKeyPairAddress())
skADRS.setTreeIndex( $idx$ )
return PRF(PK.seed, SK.seed, skADRS)

```

2) Pembangkitan *Merkle Hash Tree*

Fungsi `fors_node` (Algoritma 15) menghitung simpul pohon Merkle. Fungsi ini sama dengan `xmss_node`, kecuali bahwa simpul daun adalah *hash* dari nilai rahasia FORS, bukan kunci publik WOTS+. Fungsi `fors_node` mengambil input **SK**.seed dan **PK**.seed dari kunci privat SLH-DSA; indeks simpul target i , yang merupakan indeks simpul yang sedang dihitung; tinggi simpul target z , yang merupakan tinggi dalam pohon Merkle dari simpul yang sedang dihitung; dan alamat. Alamat **ADRS** harus memiliki alamat lapisan yang ditetapkan ke nol (karena pohon XMSS yang menandatangani kunci FORS selalu berada di lapisan 0), alamat pohon ditetapkan ke pohon XMSS yang menandatangani kunci FORS, *type* ditetapkan ke `FORS_TREE`, dan alamat pasangan kunci ditetapkan ke indeks kunci WOTS+ dalam pohon XMSS yang menandatangani kunci FORS. Tinggi dan indeks simpul target harus memenuhi $z \leq a$ dan $i < k \cdot 2^{(a-z)}$.

Setiap simpul di pohon Merkle adalah akar dari subpohon, dan Algoritma 15 menghitung akar subpohon secara rekursif. Jika subpohon terdiri dari satu simpul daun, maka fungsi tersebut cukup mengembalikan *hash* dari *string* privat n -byte simpul tersebut (baris 2 hingga 5). Jika tidak, fungsi tersebut menghitung akar sub pohon

kiri (baris 7) dan sub pohon kanan (baris 8) dan melakukan *hash* bersama-sama (baris 9 hingga 11).

Algorithm 15

`fors_node(SK.seed, i, z, PK.seed, ADRS)`

Computes the root of a Merkle subtree of FORS public values.

Input: Secret seed `SK.seed`, target node index `i`, target node height `z`, public seed `PK.seed`, address `ADRS`.

Output: n-byte root *node*.

```

if  $z = 0$  then
     $sk \leftarrow \text{fors\_skGen}(SK.seed, PK.seed, ADRS, i)$ 
    ADRS.setTreeHeight(0)
    ADRS.setTreeIndex(i)
     $node \leftarrow F(PK.seed, ADRS, sk)$ 
else
     $lnode \leftarrow \text{fors\_node}(SK.seed, 2i, z-1, PK.seed, ADRS)$ 
     $rnode \leftarrow \text{fors\_node}(SK.seed, 2i+1, z-1, PK.seed, ADRS)$ 
    ADRS.setTreeHeight(z)
    ADRS.setTreeIndex(i)
     $node \leftarrow H(PK.seed, ADRS, lnode || rnode)$ 
end if
return node

```

3) Pembangkitan Tanda Tangan FORS

Fungsi `fors_sign` (Algoritma 16) menandatangani *digest* pesan $k \cdot a$ -bit *md*. Selain *digest* pesan, `fors_sign` mengambil **SK**.seed dan **PK**.seed dari kunci privat SLH-DSA dan sebuah alamat sebagai input. Alamat **ADRS** harus memiliki alamat lapisan yang ditetapkan ke nol (karena pohon XMSS yang menandatangani kunci FORS

selalu berada di lapisan 0), alamat pohon ditetapkan ke pohon XMSS yang menandatangani kunci FORS, *type* ditetapkan ke FORS_TREE, dan alamat pasangan kunci ditetapkan ke indeks kunci WOTS+ dalam pohon XMSS yang menandatangani kunci FORS.

Fungsi `fors_sign` membagi bit $k \cdot a$ dari md menjadi *string* k a -bit (baris 2), yang masing-masing ditafsirkan sebagai bilangan bulat antara 0 dan $t - 1$. Masing-masing bilangan bulat ini digunakan untuk memilih nilai rahasia dari salah satu set k (baris 4). Untuk setiap nilai rahasia yang dipilih, jalur autentikasi dihitung dan ditambahkan ke tanda tangan (baris 5 hingga 9).

Algorithm 16

`fors_node(SK.seed, i, z, PK.seed, ADRS)`

Generates a FORS signature.

Input: Message digest md , secret seed
SK.seed, address ADRS, public seed
PK.seed.

Output: FORS signature SIG_{FORS} .

$SIG_{FORS} = \text{NULL}$ ▷ initialize SIG_{FORS} as
a zero-length byte string

$indices \leftarrow \text{base_2}^b(md, a, k)$

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

▷compute signature elements

$SIG_{FORS} \leftarrow SIG_{FORS} \parallel \text{fors_skGen}(SK.seed,$
 $PK.seed, ADRS, i \cdot 2^a + indices[i])$

for j **from** 0 **to** $a - 1$ **do**

▷compute auth path

$s \leftarrow [indices[i] / 2^j] \oplus 1$

$AUTH[j] \leftarrow \text{fors_node}(SK.seed, i \cdot 2^{a-j} +$
 $s, j, PK.seed, ADRS)$

end for

```


$$\text{SIG}_{\text{FORS}} \leftarrow \text{SIG}_{\text{FORS}} \parallel \text{AUTH}$$

end for
return  $\text{SIG}_{\text{FORS}}$ 

```

4) Menghitung Kunci Publik FORS dari Tanda Tangan

Verifikasi tanda tangan FORS melibatkan penghitungan nilai kunci publik dari *digest* pesan dan nilai tanda tangan. Verifikasi berhasil jika nilai kunci publik yang benar dihitung, yang ditentukan dengan memverifikasi tanda tangan *hypertree* pada nilai kunci publik yang dihitung menggunakan kunci publik SLH-DSA. Bagian ini menjelaskan `fors_pkFromSig` (Algoritma 17), sebuah fungsi yang menghitung kandidat kunci publik FORS dari tanda tangan FORS dan *digest* pesan yang sesuai.

Selain *digest* pesan md dan tanda tangan $k \cdot (a + 1) \cdot n$ -byte SIG_{FORS} , `fors_pkFromSig` mengambil **PK**.seed dari kunci publik SLH-DSA dan sebuah alamat sebagai input. Alamat **ADRS** harus memiliki alamat lapisan yang ditetapkan ke nol (karena pohon XMSS yang menandatangani kunci FORS selalu berada di lapisan 0), alamat pohon ditetapkan ke pohon XMSS yang menandatangani kunci FORS, *type* ditetapkan ke FORS_TREE, dan alamat pasangan kunci ditetapkan ke indeks kunci WOTS+ dalam pohon XMSS yang menandatangani kunci FORS.

Algorithm 17 `fors_pkFromSig` ($\text{SIG}_{\text{FORS}}, md, \text{PK.seed}, \text{ADRS}$)

Computes a FORS public key from a FORS signature.

Input: FORS signature SIG_{FORS} , message digest md ,
public seed PK.seed , address ADRS .

Output: FORS public key.

```

indices  $\leftarrow$  base_2b(md, a, k)
for i from 0 to k-1 do
    sk  $\leftarrow$  SIGFORS.getSK(i)  $\triangleright$  SIGFORS[i · (a + 1) ·
    n : (i · (a + 1) + 1) · n]
    ADRS.setTreeHeight(0)  $\triangleright$  compute leaf
    ADRS.setTreeIndex(i · 2a + indices[i])
    node[0]  $\leftarrow$  F(PK.seed, ADRS, sk)
    auth  $\leftarrow$  SIGFORS.getAUTH(i)  $\triangleright$  SIGFORS[(i · (a + 1) + 1) · n :
    (i + 1) · (a + 1) · n]
    for j from 0 to a - 1 do  $\triangleright$  compute root from
    leaf and AUTH
    ADRS.setTreeHeight(j + 1)
    if [indices[i] / 2j] is even then
        ADRS.setTreeIndex(ADRS.getTreeIndex() / 2)
        node[1]  $\leftarrow$  H(PK.seed, ADRS, node[0] || auth[j])
    else
        ADRS.setTreeIndex((ADRS.getTreeIndex() - 1) / 2)
        node[1]  $\leftarrow$  H(PK.seed, ADRS, auth[j] || node[0])
    end if
    node[0]F  $\leftarrow$  node[1]L
end for
    root[i]  $\leftarrow$  node[0]
end for
forspkADRS  $\leftarrow$  ADRS  $\triangleright$  copy address to create a FORS
public-key address
forspkADRS.setTypeAndClear(FORS_ROOTS)
forspkADRS.setKeyPairAddress(ADRS.getKeyPairAddress())
pk  $\leftarrow$  Tk(PK.seed, forspkADRS, root)  $\triangleright$  compute the FORS
public key
return pk

```

Fungsi `fors_pkFromSig` dimulai dengan menghitung akar dari setiap pohon Merkle *k* (baris 2 hingga 20).

Seperti dalam *fors_sign*, bit $k \cdot a$ dari *digest* pesan dibagi menjadi *string* k a -bit (baris 1), yang masing-masing ditafsirkan sebagai bilangan bulat antara 0 dan $t - 1$. Bilangan bulat digunakan untuk menentukan lokasi di pohon Merkle dari nilai rahasia dari tanda tangan (baris 3 hingga 5). *Hash* dari nilai rahasia dihitung (baris 6), dan nilai *hash* digunakan bersama dengan jalur autentikasi yang sesuai dari tanda tangan untuk menghitung akar pohon Merkle (baris 7 hingga 19). Setelah semua akar pohon Merkle dihitung, mereka di-*hash* bersama untuk menghitung kunci publik FORS (baris 21 hingga 24).

i. SLH-DSA *Internal Functions*

SLH-DSA menggunakan *hypertree* dan kunci FORS untuk membuat skema tanda tangan berbasis *hash* tanpa status. Kunci privat SLH-DSA berisi nilai *seed* rahasia dan kunci PRF rahasia. Kunci publik terdiri dari pengenalan kunci **PK**.seed dan akar *hypertree*. Tanda tangan dibuat dengan melakukan *hashing* pada pesan, menggunakan bagian dari *digest* pesan untuk memilih kunci FORS, menandatangani bit lain dari *digest* pesan dengan kunci FORS, dan membuat tanda tangan *hypertree* untuk kunci FORS.

SLH-DSA menggunakan satu parameter tambahan m , yang merupakan panjang dalam *byte* dari *digest* pesan, dihitung sebagai:

$$m = \left\lceil \frac{h - h'}{8} \right\rceil + \left\lceil \frac{h'}{8} \right\rceil + \left\lceil \frac{k \cdot a}{8} \right\rceil$$

SLH-DSA juga menggunakan h bit dari *digest* pesan untuk memilih kunci FORS: $h - h'$ bit untuk memilih pohon XMSS di lapisan terendah dan h' bit untuk memilih kunci WOTS+ dan kunci FORS yang sesuai dari pohon tersebut. $k \cdot a$ bit dari *digest* ditandatangani oleh kunci FORS yang dipilih.

Sementara hanya $h + k \cdot a$ bit dari *digest* pesan yang digunakan, implementasi disederhanakan dengan mengekstrak bit yang diperlukan dari *digest* yang sedikit lebih besar.

Bagian ini menjelaskan fungsi untuk pembuatan kunci SLH-DSA, pembuatan tanda tangan, dan verifikasi tanda tangan. Dalam fungsi di bagian ini, jika keacakan diperlukan, nilai acak disediakan sebagai input ke fungsi. Antarmuka yang ditetapkan dalam bagian ini akan digunakan saat pengujian implementasi SLH-DSA dilakukan melalui *Cryptographic Algorithm Validation Program* (CAVP).

Selain untuk tujuan pengujian, antarmuka untuk pembuatan kunci dan pembuatan tanda tangan yang ditetapkan dalam bagian ini tidak boleh disediakan untuk aplikasi, karena nilai acak yang diperlukan untuk pembuatan kunci dan pembuatan tanda tangan harus dibuat oleh modul kriptografi.

1) Pembangkitan Kunci SLH-DSA

Kunci publik SLH-DSA berisi dua elemen. Yang pertama adalah *seed* publik n -byte PK.seed, yang digunakan dalam banyak panggilan fungsi *hash* untuk menyediakan pemisahan domain antara pasangan kunci SLH-DSA yang berbeda. Nilai kedua adalah kunci publik *hypertree* (yaitu akar dari pohon XMSS lapisan atas). PK.seed akan dibuat menggunakan generator bit acak yang disetujui, di mana instansiasi generator bit acak mendukung setidaknya $8n$ bit kekuatan keamanan.

SK.seed	N bytes
SK.prf	SK.prf
PK.seed	PK.seed
PK.root	PK.root

Gambar 3.22 SLH-DSA Kunci Privat

PK.seed	n bytes
PK.root	n bytes

Gambar 3.23 SLH-DSA Kunci Publik

Kunci privat SLH-DSA berisi dua nilai n -byte rahasia dan acak. SK.seed digunakan untuk membuat semua elemen kunci privat WOTS+ dan FORS. SK.prf digunakan untuk membuat nilai pengacakan untuk *hashing* acak pesan dalam SLH-DSA. Kunci privat juga menyertakan salinan kunci publik. Baik SK.seed maupun SK.prf akan dibuat menggunakan generator bit acak yang disetujui, di mana instansiasi generator bit acak mendukung setidaknya $8n$ bit kekuatan keamanan. Algoritma 18 membuat pasangan kunci SLH-DSA. Baris 1 hingga 3 menghitung akar pohon XMSS lapisan atas. Baris 4 menggabungkan tiga input dan PK.seed yang dihitung ke dalam kunci privat dan publik. Penandatanganan SLH-DSA memiliki dua varian — “dilindungi” dan deterministik — yang kuncinya hanya boleh digunakan untuk pembuatan dan verifikasi tanda tangan digital SLH-DSA yang dilindungi dan deterministik.

Algorithm 18

```
slh_keygen_internal(SK.seed, SK.prf, PK.seed)
```

Generates an SLH-DSA key pair.

Input: Secret seed SK.seed, PRF key SK.prf,
public seed PK.seed.

Output: SLH-DSA key pair (SK, PK).

```
ADRS ← toByte(0, 32)    ▷generate the public
key for the top-level XMSS tree
```

```
ADRS.setLayerAddress( $d - 1$ )
```

```
PK.root ← xmss_node(SK.seed, 0,  $h'$ , PK.seed,
```

```

ADRS)
return ((SK.seed, SK.prf, PK.seed, PK.root),
(PK.seed, PK.root))

```

2) Pembangkitan Tanda Tangan SLH-DSA

Tanda tangan SLH-DSA terdiri dari *string* pengacakan, tanda tangan FORS, dan tanda tangan hypertree, seperti yang ditunjukkan pada Gambar 3.24. Pembuatan tanda tangan SLH-DSA (Algoritma 19) dimulai dengan membuat *digest* pesan m -byte (baris 2 hingga 5). PRF digunakan untuk membuat pengacak pesan (baris 3), dan di-hash bersama dengan pesan untuk membuat *digest* (baris 5). Bit kemudian diekstraksi dari *digest* pesan untuk ditandatangani oleh kunci FORS (baris 6), untuk memilih pohon XMSS (baris 7 dan 9), dan untuk memilih kunci WOTS+ dan kunci FORS yang sesuai dalam pohon XMSS tersebut (baris 8 dan 10). Selanjutnya, tanda tangan FORS dihitung (baris 11 hingga 14), dan kunci publik FORS yang sesuai diperoleh (baris 16). Akhirnya, kunci publik FORS ditandatangani (baris 17).

Randomness R	n bytes
FORS signature SIG_{FORS}	$k(1+a) \cdot n$ bytes
HT signature SIG_{HT}	$(h+d \cdot len) \cdot n$ bytes

Gambar 3.24 Format data Tanda tangan SLH-DSA

Algorithm 19 `slh_sign_internal( $M$ , SK, addrnd)`

Generates an SLH-DSA signature.

Input: Message M , private key $SK = (SK.seed, SK.prf, PK.seed, PK.root)$, (optional) additional randomness *addrnd*.

Output: SLH-DSA signature SIG .

```

ADRS  $\leftarrow$  toByte(0, 32)
opt_rand  $\leftarrow$  addrnd  $\triangleright$  substitute opt_rand  $\leftarrow$  PK.seed
for the deterministic variant
R  $\leftarrow$  PRFmsg(SK.prf, opt_rand, M)  $\triangleright$ 
generate randomizer
SIG  $\leftarrow$  R
digest  $\leftarrow$  Hmsg(R, PK.seed, PK.root, M)  $\triangleright$ 
compute message digest
md  $\leftarrow$  digest[0 :  $\lceil \frac{k \cdot a}{8} \rceil$ ]  $\triangleright$ 
first  $\lceil \frac{k \cdot a}{8} \rceil$  bytes
tmp_idxtree  $\leftarrow$  digest[ $\lceil \frac{k \cdot a}{8} \rceil$  :  $\lceil \frac{k \cdot a}{8} \rceil + \lceil \frac{h-h/d}{8} \rceil$ ]
 $\triangleright$  next  $\lceil \frac{h-h/d}{8} \rceil$  bytes
tmp_idxleaf  $\leftarrow$  digest[ $\lceil \frac{k \cdot a}{8} \rceil + \lceil \frac{h-h/d}{8} \rceil$  :  $\lceil \frac{k \cdot a}{8} \rceil + \lceil \frac{h-h/d}{8} \rceil +$ 
 $\lceil \frac{h}{8d} \rceil$ ]  $\triangleright$  next  $\lceil \frac{h}{8d} \rceil$  bytes
idxtree  $\leftarrow$  toInt(tmp_idxtree,  $\lceil \frac{h-h/d}{8} \rceil$ ) mod  $2^{(h-h/d)}$ 
idxleaf  $\leftarrow$  toInt(tmp_idxleaf,  $\lceil \frac{h}{8d} \rceil$ ) mod  $2^{(h/d)}$ 
ADRS.setTreeAddress(idxtree)
ADRS.setTypeAndClear(FORS_TREE)
ADRS.setKeyPairAddress(idxleaf)
SIGFORS  $\leftarrow$  fors_sign(md, SK.seed, PK.seed, ADRS)
SIG  $\leftarrow$  SIG || SIGFORS
PKFORS  $\leftarrow$  fors_pkFromSig(SIGFORS, md, PK.seed,
ADRS)  $\triangleright$  get FORS key
SIGHT  $\leftarrow$  ht_sign(PKFORS, SK.seed, PK.seed, idxtree,
idxleaf)
SIG  $\leftarrow$  SIG || SIGHT
return SIG

```

Pengacak pesan dapat diatur dengan cara non-deterministik atau deterministik, tergantung pada

apakah *addrnd* diberikan sebagai input. Untuk varian "*hedged*", *addrnd* diberikan sebagai input, dan *opt_rand* diatur ke *addrnd*. Varian *hedged* adalah default dan harus digunakan pada platform di mana serangan side-channel menjadi perhatian. Saat menggunakan versi *hedged*, *addrnd* akan menjadi nilai acak n -byte. Meskipun *addrnd* idealnya dihasilkan oleh generator bit acak yang disetujui, metode lain untuk menghasilkan nilai acak baru dapat digunakan. Untuk varian deterministik, *addrnd* tidak diberikan sebagai input, dan *opt_rand* ditetapkan ke **PK.seed**, yang mengakibatkan penandatanganan bersifat deterministik (yaitu, menandatangani pesan yang sama dua kali akan menghasilkan tanda tangan yang sama). Varian deterministik tersedia untuk platform yang tidak memiliki generator bit acak.

3) Verifikasi Tanda Tangan SLH-DSA

Seperti halnya pembuatan tanda tangan, verifikasi tanda tangan SLH-DSA (Algoritma 20) dimulai dengan menghitung *digest* pesan (baris 8) lalu mengekstraksi *md* (baris 9), *idxtree* (baris 10 dan 12), dan *idxleaf* (baris 11 dan 13) dari *digest*. Kandidat kunci publik FORS kemudian dihitung (baris 17), dan tanda tangan pada kunci FORS diverifikasi (baris 18). Jika verifikasi tanda tangan ini berhasil, maka kunci publik FORS yang benar telah dihitung, dan SIG tanda tangan pada pesan M valid.

Algorithm 20 `slh_verify_internal( $M$ , SIG, PK)`

Verifies an SLH-DSA signature.

Input: Message M , signature SIG, public key PK
 $= (\text{PK.seed}, \text{PK.root})$.

Output: Boolean

if $|\text{SIG}| \neq (1 + k(1 + a) + h + d \cdot \text{len}) \cdot n$ **then**
return false

```

end if
ADRS  $\leftarrow$  toByte(0, 32)
R  $\leftarrow$  SIG.getR()
 $\triangleright$  SIG[0 : n]
SIGFORS  $\leftarrow$  SIG.getSIG_FORS()
 $\triangleright$  SIG[n : (1 + k(1 + a)) · n]
SIGHT  $\leftarrow$  SIG.getSIG_HT()  $\triangleright$  SIG[n : (1 + k(1 + a)) · n : (1 + k(1 + a)) + h + d · len]
digest  $\leftarrow$  Hmsg(R, PK.seed, PK.root, M)
 $\triangleright$  compute message digest
md  $\leftarrow$  digest[0 :  $\lceil \frac{k \cdot a}{8} \rceil$ ]
 $\triangleright$  first  $\lceil \frac{k \cdot a}{8} \rceil$  bytes
tmp_idxtree  $\leftarrow$  digest[ $\lceil \frac{k \cdot a}{8} \rceil$  :  $\lceil \frac{k \cdot a}{8} \rceil + \lceil \frac{h-h/d}{8} \rceil$ ]
 $\triangleright$  next  $\lceil \frac{h-h/d}{8} \rceil$  bytes
tmp_idxleaf  $\leftarrow$  digest[ $\lceil \frac{k \cdot a}{8} \rceil + \lceil \frac{h-h/d}{8} \rceil$  :  $\lceil \frac{k \cdot a}{8} \rceil + \lceil \frac{h-h/d}{8} \rceil + \lceil \frac{h}{8d} \rceil$ ]
 $\triangleright$  next  $\lceil \frac{h}{8d} \rceil$  bytes
idxtree  $\leftarrow$  toInt(tmp_idxtree,  $\lceil \frac{h-h/d}{8} \rceil$ ) mod 2(h-h/d)
idxleaf  $\leftarrow$  toInt(tmp_idxleaf,  $\lceil \frac{h}{8d} \rceil$ ) mod 2(h/d)
ADRS.setTreeAddress(idxtree)
ADRS.setTypeAndClear(FORS_TREE)
ADRS.setKeyPairAddress(idxleaf)
PKFORS  $\leftarrow$  fors_pkFromSig(SIGFORS, md, PK.seed, ADRS)
 $\triangleright$  compute
FORS
    public key
return ht_verify(PKFORS, SIGHT, PK.seed, idxtree, idxleaf, PK.root)

```

j. SLH-DSA *External Functions*

Bagian ini memberikan panduan tentang fungsi pembuatan kunci, pembuatan tanda tangan, dan verifikasi tanda tangan yang harus disediakan untuk digunakan oleh aplikasi. Fungsi-fungsi dalam bagian ini menggunakan fungsi-fungsi dalam Bagian 9 pada [20] untuk mengimplementasikan skema SLH-DSA yang mendasarinya.

1) Pembangkitan Kunci SLH-DSA

Algoritma 21 menghasilkan pasangan kunci SLH-DSA. Baris 1 hingga 3 menghasilkan nilai acak untuk kunci privat dan publik, dan baris 7 memanggil `slh_keygen_internal` untuk menghitung **PK.root** dan mengembalikan kunci privat dan publik. **PK.seed**, **SK.seed**, dan **SK.prf** akan dihasilkan menggunakan generator bit acak yang disetujui, di mana instansiasi generator bit acak mendukung setidaknya $8n$ bit kekuatan keamanan.

Algorithm 21 `slh_keygen()`

Generates an SLH-DSA key pair.

Input: (none)

Output: SLH-DSA key pair (SK, PK).

$SK.seed \xleftarrow{s} \mathbb{B}^n$ $\triangleright$ set SK.seed, SK.prf, and PK.seed to random n -byte

$SK.prf \xleftarrow{s} \mathbb{B}^n$ $\triangleright$ strings using an approved random bit generator

$PK.seed \xleftarrow{s} \mathbb{B}^n$

if SK.seed = NULL or SK.prf = NULL or PK.seed = NULL **then**

return $\perp$ $\triangleright$ return an error indication if random bit generation failed

end if

```
return slh_keygen_internal(SK.seed, SK.prf,  
PK.seed)
```

2) Pembangkitan Tanda Tangan SLH-DSA

Bagian ini menyajikan dua versi pembuatan tanda tangan SLH-DSA: versi "*pure*" (*slh_sign*) dan versi "*pra-hash*" (*hash_slh_sign*). Kedua versi menggunakan *slh_sign_internal*, tetapi keduanya berbeda dalam cara input pesan ke *slh_sign_internal* dibuat dari konten yang akan ditandatangani. Dalam versi "*pure*", konten ditandatangani oleh *slh_sign_internal* beserta beberapa informasi pemisahan domain. Dalam versi *pra-hash*, *hash* konten ditandatangani oleh *slh_sign_internal* beserta beberapa informasi pemisahan domain.

Kedua versi mengambil konten yang akan ditandatangani, kunci privat, dan konteks sebagai input. Versi *pra-hash* juga mengambil fungsi *hash* atau XOF sebagai input yang akan digunakan untuk melakukan *pra-hash* konten yang akan ditandatangani. *String* konteks memiliki panjang maksimum 255 *byte*. Secara default, konteksnya adalah *string* kosong. Namun, aplikasi dapat menentukan penggunaan *string* konteks yang tidak kosong. Pengidentifikasi untuk tanda tangan (misalnya, *object identifier* [OID]) harus menunjukkan apakah tanda tangan tersebut merupakan tanda tangan murni atau tanda tangan *pra-hash*. Dalam kasus tanda tangan *pra-hash*, pengidentifikasi juga harus menunjukkan fungsi *hash* atau XOF yang digunakan untuk menghitung *pra-hash*. Meskipun pasangan kunci tunggal dapat digunakan untuk tanda tangan murni dan *pra-hash*, disarankan agar setiap pasangan kunci hanya digunakan untuk satu versi atau yang lain. Jika *string* konteks yang tidak kosong akan digunakan, ini harus

ditunjukkan oleh pengidentifikasi tanda tangan atau aplikasi yang menggunakan tanda tangan tersebut.

Jika varian “*hedged*” nilai default dari `slh_sign_internal` digunakan, nilai acak n -byte `addrnd` akan dihasilkan oleh modul kriptografi yang menjalankan `slh_sign_internal`. Namun, M' dalam Algoritma 22 dan 23 dapat dibangun di luar modul kriptografi. Dalam kasus `hash_slh_sign`, `hash` atau XOF dari konten yang akan ditandatangani harus dihitung dalam modul kriptografi yang divalidasi FIPS 140, tetapi mungkin modul kriptografi yang berbeda dari yang menjalankan `slh_sign_internal`.

Secara umum, versi “*pure*” lebih disukai. Namun, untuk beberapa modul kriptografi yang menghasilkan tanda tangan SLH-DSA, menjalankan baris 3 dan 5 dari Algoritma 19 mungkin tidak layak jika pesan M berukuran besar. Ini mungkin, misalnya, merupakan hasil dari modul yang memiliki memori terbatas untuk menyimpan pesan yang akan ditandatangani. Demikian pula, untuk beberapa modul kriptografi yang memverifikasi tanda tangan SLH-DSA, melakukan baris 8 dari Algoritma 20 mungkin tidak layak jika pesan M besar. Untuk beberapa kasus penggunaan, masalah ini dapat diatasi dengan menandatangani *digest* konten daripada menandatangani konten secara langsung.

Dalam banyak kasus di mana konten yang akan ditandatangani besar, *hashing* konten dilakukan di tingkat aplikasi. Misalnya, dalam *Cryptographic Message Syntax*, *digest* konten dapat dihitung, dan *digest* tersebut ditandatangani bersama dengan atribut lainnya. Dalam kasus di mana konten tidak di-*hash* di tingkat aplikasi, versi *pra-hash* dari penandatanganan SLH-DSA dapat digunakan.

Untuk mempertahankan tingkat kekuatan keamanan yang sama saat konten di-*hash* pada level aplikasi atau saat menggunakan versi pra-*hash* SLH-DSA, *digest* yang ditandatangani perlu dibuat menggunakan fungsi *hash* atau XOF yang disetujui (misalnya, fungsi *hash* yang ada dalam FIPS 180-4 atau FIPS 202) yang menyediakan setidaknya $8n$ bit kekuatan keamanan klasik terhadap serangan kolisi dan *second preimage attacks*. Verifikasi tanda tangan yang dibuat dengan cara ini akan mengharuskan fungsi verifikasi untuk membuat *digest* dari pesan dengan cara yang sama untuk input ke fungsi verifikasi. Bahkan jika memungkinkan untuk menghitung kolisi pada fungsi *hash* atau XOF yang digunakan untuk membuat **H***msg*, **PRF**, **PRF***msg*, **F**, **H**, dan **T**_l, diyakini tidak ada efek buruk pada keamanan SLH-DSA. Namun, jika input ke fungsi penandatanganan adalah *digest* konten, maka kolisi pada fungsi yang digunakan untuk menghitung *digest* dapat mengakibatkan pesan palsu.

3) Pembangkitan Tanda Tangan Pure SLH-DSA

Dalam versi “*pure*”, konten yang akan ditandatangani diawali dengan pemisah domain satu *byte*, satu *byte* yang menunjukkan panjang *string* konteks, dan *string* konteks. Pemisah domain, yang memiliki nilai nol untuk penandatanganan murni, disertakan untuk mencegah tanda tangan pra-*hash* terverifikasi sebagai tanda tangan murni dan sebaliknya. Dalam kasus *default* di mana *string* konteks kosong, penandatanganan murni hanya melibatkan penambahan dua *byte* nol ke konten yang akan ditandatangani dan meneruskan hasilnya ke `slh_sign_internal` bersama

dengan kunci privat dan, dalam kasus penandatanganan terlindung, nilai acak n -byte.

Algorithm 22 $\text{slh_sign}(M, \text{ctx}, \text{SK})$
Generates a pure SLH-DSA signature.
Input: Message M , context <i>string</i> ctx , private key SK .
Output: SLH-DSA signature SIG .
if $ \text{ctx} > 255$ then
return $\perp$ $\triangleright$ return an error indication if the context <i>string</i> is too long
end if
$\text{addrnd} \xleftarrow{\$} \mathbb{B}^n$ $\triangleright$ skip lines 4 through 7 for the deterministic variant
if $\text{addrnd} = \text{NULL}$ then
return $\perp$ $\triangleright$ return an error indication if random bit generation failed
end if
$M' \leftarrow \text{toByte}(0,1) \parallel \text{toByte}(\text{ctx} ,1) \parallel \text{ctx} \parallel M$
$\text{SIG} \leftarrow \text{slh_sign_internal}(M', \text{SK}, \text{addrnd})$ $\triangleright$ omit addrnd for the deterministic variant
return SIG

4) Pembangkitan Tanda Tangan Hash SLH-DSA

Dalam versi *pre-hash*, input pesan ke slh_sign_internal adalah hasil penerapan fungsi *hash* atau XOF pada konten yang akan ditandatangani. Output fungsi *hash* atau XOF diawali oleh pemisah domain satu *byte*, satu *byte* yang menunjukkan panjang *string* konteks, *string* konteks, dan pengodean *distinguished encoding rules* (DER) dari fungsi *hash* atau OID XOF. Pemisah domain memiliki nilai satu untuk penandatanganan *pra-hash*. Pengodean DER dari OID mencakup tag dan panjang. Algoritma 23 menunjukkan

pengodean DER dari OID untuk SHA-256, SHA-512, SHAKE128, dan SHAKE256. Namun, *hash_slh_sign* dapat digunakan dengan fungsi *hash* atau XOF lainnya. SHA-256 dan SHAKE128 hanya sesuai untuk digunakan dengan set parameter SLH-DSA yang diklaim berada dalam kategori keamanan 1.

Algorithm 23 *hash_slh_sign*(*M*, *ctx*, *PH*, *SK*)

Generates a pre-hash SLH-DSA signature.

Input: Message *M*, context string *ctx*, private key *SK*.

Output: SLH-DSA signature *SIG*.

```

if |ctx| > 255 then
    return  $\perp$   $\triangleright$  return an error indication if
    the context string is too long
end if

 $\text{addrnd} \xleftarrow{s} \mathbb{B}^n$   $\triangleright$  skip lines 4 through 7 for the
deterministic variant
if addrnd = NULL then
    return  $\perp$   $\triangleright$  return an error indication if
    random bit generation failed
end if

switch PH do
    case SHA-256:
        OID  $\leftarrow$  toByte(0x0609608648016503040201, 11)
         $\triangleright$  2.16.840.1.101.3.4.2.1
        PHM  $\leftarrow$  SHA-256(M)

    case SHA-512:
        OID  $\leftarrow$  toByte(0x0609608648016503040203, 11)
         $\triangleright$  2.16.840.1.101.3.4.2.3
        PHM  $\leftarrow$  SHA-512(M)

    case SHAKE128:

```

```

    OID  $\leftarrow$  toByte(0x060960864801650304020B, 11)
 $\triangleright$  2.16.840.1.101.3.4.2.11
    PHM  $\leftarrow$  SHAKE128(M, 256)
    case SHAKE256:
        OID  $\leftarrow$  toByte(0x060960864801650304020C, 11)
 $\triangleright$  2.16.840.1.101.3.4.2.12
        PHM  $\leftarrow$  SHAKE256(M, 512)
    case ...  $\triangleright$  other
approved hash functions or XOFs
    ...
end switch
M'  $\leftarrow$  toByte(1,1) || toByte(|ctx|, 1) || ctx || OID || PHM  $\triangleright$ 
omit addrnd for the
deterministic variant
return SIG

```

4) Verifikasi Tanda Tangan SLH-DSA

Algoritma 24 dan 25 menyajikan versi “pure” dan *pra-hash* dari verifikasi tanda tangan SLH-DSA yang sesuai dengan versi murni dan *pra-hash* dari pembuatan tanda tangan SLH-DSA di Bagian 10.2. Fungsi-fungsi ini membangun M' dengan cara yang sama seperti Algoritma 22 dan 23, masing-masing, dan meneruskan M' yang dihasilkan ke [slh_verify_internal](#) untuk verifikasi. Seperti halnya pembuatan tanda tangan, M' dapat dibangun di luar modul kriptografi yang menjalankan [slh_verify_internal](#). Namun, dalam kasus [hash_slh_verify](#), *hash* atau XOF konten harus dihitung dalam FIPS 140 *validated cryptographic module*, yang mungkin merupakan modul kriptografi yang berbeda dari yang menjalankan [slh_verify_internal](#).

Pengidentifikasi yang dikaitkan dengan tanda tangan harus menunjukkan apakah versi murni atau *pra-hash*

dari verifikasi tanda tangan harus digunakan, dan dalam kasus *pra-hash*, fungsi *hash* atau XOF yang digunakan untuk menghitung *pra-hash*. *String* konteks yang tidak kosong harus digunakan dalam verifikasi jika *string* tersebut ditentukan dalam pengidentifikasi tanda tangan atau oleh aplikasi yang menggunakan tanda tangan tersebut.

Algorithm 24 `slh_verify(M, SIG, ctx, PK)`

Verifies a pure SLH-DSA signature.

Input: Message *M*, signature *SIG*, context *string* *ctx*, public key *PK*.

Output: Boolean.

if `|ctx| > 255` **then**

return false

end if

$M' \leftarrow \text{toByte}(0,1) \parallel \text{toByte}(|ctx|,1) \parallel ctx \parallel M$

return `slh_verify_internal(M', SIG, PK)`

Algorithm 25 `hash_slh_verify(M, SIG, ctx, PH, PK)`

Verifies a pre-hash SLH-DSA signature.

Input: Message *M*, signature *SIG*, context *string* *ctx*, pre-hash function *PH*, public key *PK*.

Output: Boolean.

if `|ctx| > 255` **then**

return false

end if

switch *PH* **do**

case SHA-256:

$\text{OID} \leftarrow \text{toByte}(0x0609608648016503040201, 11)$ ▷

2.16.840.1.101.3.4.2.1

$\text{PH}_M \leftarrow \text{SHA-256}(M)$

case SHA-512:


```

    OID ← toByte(0x0609608648016503040203, 11) ▷
2.16.840.1.101.3.4.2.3
    PHM ← SHA-512(M)
    case SHAKE128:
        OID ← toByte(0x060960864801650304020B, 11) ▷
2.16.840.1.101.3.4.2.11
        PHM ← SHAKE128(M, 256)
    case SHAKE256:
        OID ← toByte(0x060960864801650304020C, 11) ▷
2.16.840.1.101.3.4.2.12
        PHM ← SHAKE256(M, 512)
    case ...▷ other approved hash functions or XOFs
    ...
end switch
M' ← toByte(1, 1) || toByte(|ctx|, 1) || ctx || OID || PHM
return slh_verify_internal(M', SIG, PK)

```

k. Parameter Set

Dalam FIPS 205 ini menyetujui 12 set parameter untuk digunakan dengan SLH-DSA. Satu set parameter terdiri dari parameter untuk WOTS+ (n dan lg_w), XMSS dan *hypertree* SLH-DSA (h dan d), dan FORS (k dan a), serta instansiasi untuk fungsi H_{msg} , **PRF**, PRF_{msg} , **F**, **H**, dan T_ℓ . Tabel 3.7 mencantumkan set parameter yang disetujui untuk digunakan. Setiap nama set parameter menunjukkan keluarga fungsi *hash* (SHA2 atau SHAKE) yang digunakan untuk membuat fungsi *hash*, panjang parameter keamanan n dalam bit, dan apakah set parameter dirancang untuk membuat tanda tangan yang relatif kecil ('s') atau memiliki pembuatan tanda tangan yang relatif cepat ('f'). Ada enam set nilai untuk n , lg_w , h , d , k , dan a yang disetujui untuk digunakan. Untuk masing-masing dari enam set nilai, fungsi H_{msg} , **PRF**, PRF_{msg} ,

$\mathbf{F}$, $\mathbf{H}$, dan $\mathbf{T}_\ell$ dapat diwujudkan menggunakan SHAKE atau SHA-2.

Ke-12 set parameter yang disertakan dalam Tabel 3.7 dirancang untuk memenuhi kategori kekuatan keamanan tertentu yang ditetapkan oleh NIST dalam *Call for Proposals* berkenaan dengan kemampuan *existential unforgeability under chosen message attack* (EUF-CMA) ketika setiap pasangan kunci digunakan untuk menandatangani paling banyak 2^{64} pesan. Kategori kekuatan keamanan ini dijelaskan lebih lanjut dalam SP 800-57, Bagian 1. Dengan menggunakan pendekatan ini, kekuatan keamanan tidak dijelaskan oleh satu angka, seperti "128 bit keamanan." Sebaliknya, setiap set parameter diklaim setidaknya seaman *block cipher* generik dengan ukuran kunci yang ditentukan. Lebih tepatnya, diklaim bahwa sumber daya komputasi yang diperlukan untuk memecahkan SLH-DSA lebih besar dari atau sama dengan sumber daya komputasi yang diperlukan untuk memecahkan *block cipher* ketika sumber daya komputasi ini diperkirakan menggunakan model komputasi yang realistis. Model komputasi yang berbeda dapat lebih atau kurang realistis dan, karenanya, mengarah pada perkiraan kekuatan keamanan yang lebih atau kurang akurat. Secara konkret, set parameter dengan $n = 16$ diklaim berada dalam kategori keamanan 1, set parameter dengan $n = 24$ diklaim berada dalam kategori keamanan 3, dan set parameter dengan $n = 32$ diklaim berada dalam kategori keamanan 5 [10].

Tabel 3.7 SLH-DSA *parameter sets*

	n	h	d	h'	a	k	lg_w	m	security category	pk bytes	sig bytes
SLH-DSA-SHA2-128s	16	63	7	9	12	14	4	30	1	32	7 856
SLH-DSA-SHAKE-128s											

	n	h	d	h'	a	k	lg_w	m	security category	pk bytes	sig bytes
SLH-DSA-SHA2-128f	16	66	22	3	6	33	4	34	1	32	17 088
SLH-DSA-SHAKE-128f											
SLH-DSA-SHA2-192s	24	63	7	9	14	17	4	39	3	48	16 224
SLH-DSA-SHAKE-192s											
SLH-DSA-SHA2-192f	24	66	22	3	8	33	4	42	3	48	35 664
SLH-DSA-SHAKE-192f											
SLH-DSA-SHA2-256s	32	64	8	8	14	22	4	47	5	64	29 792
SLH-DSA-SHAKE-256s											
SLH-DSA-SHA2-256f	32	68	17	4	9	35	4	49	5	64	49 856
SLH-DSA-SHAKE-256f											

Beberapa aplikasi memerlukan properti yang dikenal sebagai *message-bound signatures*, yang secara intuitif mengharuskan tidak mungkin bagi siapa pun untuk membuat kunci publik dan tanda tangan yang valid untuk dua pesan yang berbeda. Skema tanda tangan tidak diharuskan memiliki properti ini berdasarkan definisi keamanan EUF-CMA yang digunakan dalam menetapkan kategori keamanan. Dalam kasus SLH-DSA, pemilik pasangan kunci dapat membuat dua pesan dengan tanda tangan yang sama dengan menemukan kolisi pada $Hmsg$. Karena panjang *output* $Hmsg$, menemukan kolisi tersebut diperkirakan memerlukan sumber daya komputasi yang lebih sedikit daripada yang ditentukan untuk kategori keamanan yang diklaim oleh set parameter dalam semua kasus kecuali SLH-DSA-SHA2-128f dan SLH-DSA-

SHAKE-128f. Oleh karena itu, aplikasi yang memerlukan *message-bound signatures* harus memperhitungkan biaya yang diharapkan untuk menemukan kolisi pada $Hmsg$ saat memilih set parameter yang sesuai atau menerapkan teknik (misalnya, transformasi BUFF) untuk memperoleh properti *message-bound signatures*.

1. Keunggulan dan Implementasi SLH-DSA

Keunggulan dari SLH-DSA, yaitu:

- 1) Tahan terhadap serangan dari komputer kuantum karena didasarkan pada fungsi *hash* yang dipercaya aman terhadap serangan tersebut.
- 2) Tidak memerlukan penyimpanan status (*stateless*) sehingga mengurangi risiko serangan terkait status.
- 3) Menawarkan kinerja yang relatif efisien dalam hal kecepatan dan ukuran tanda tangan.

SLH-DSA dapat digunakan dalam berbagai aplikasi yang membutuhkan tanda tangan digital, seperti:

- 1) Untuk penerbitan sertifikat digital pada Infrastruktur kunci publik (PKI).
- 2) Protokol keamanan seperti TLS/SSL untuk mengamankan komunikasi internet.
- 3) Aplikasi *blockchain* untuk mengamankan transaksi dan data.

E. Kriptografi berbasis isogeni supersingular (*Supersingular Isogeny-based cryptography*)

Isogeni antara kurva eliptis merupakan pemetaan nonkonstan yang dapat ditulis sebagai pecahan polinomial dan kompatibel dengan penjumlahan pada kedua kurva, sehingga *image* jumlah dua titik pada kurva pertama sama dengan jumlah *image*, bila dihitung pada kurva kedua. Kriptografi berbasis isogeni menggunakan isogeni antara kurva eliptis pada *finite field*. Masalah isogeni (*isogeny problem*) adalah mencari isogeni antara dua kurva eliptis yang diketahui isogen [21].

Masalah ini diperkenalkan pada tahun 2005 oleh Denis Xavier Charles, Kristin E. Lauter, dan Eyal Z. Goren dalam makalahnya yang berjudul *Cryptographic hash functions from expander graphs* yang dipublikasikan dalam *Journal of Cryptology* pada bulan Januari 2009 dan menjadi dasar bagi salah satu kandidat algoritma *post-quantum* yang diajukan ke kompetisi NIST yaitu *Supersingular Isogeny Key Encapsulation* (SIKE).

Pada tanggal 5 Agustus 2022 Castryck dan Decru mengeluarkan pracetak [22] yang menguraikan algoritma pemulihan kunci klasik yang efisien terhadap *Supersingular Isogeny Diffie-Hellman* (SIDH) beserta kode yang mengimplementasikan algoritma tersebut dan menunjukkan kepraktisannya. Serangan ini bergantung pada pemilihan *public starting curve* (kurva awal publik) yaitu kurva Montgomery dengan parameter $A = 6$ dan $B = 1$ yang digunakan dalam SIKE.

Penelitian lanjutan oleh Maino dan Martindale [23] serta Robert [24] menunjukkan bahwa modifikasi SIDH apa pun yang sejalan dengan perubahan kurva awal tersebut tetap tidak aman. Moriya [25] dan Fouotsa [26] secara independen telah mengusulkan pendekatan yang berbeda untuk memodifikasi SIDH untuk menghindari serangan Castryck dan Decru, tetapi kedua usulan ini secara signifikan menurunkan kinerja dan ukuran kunci SIDH setidaknya satu orde besaran (*magnitude*). Konsensus umum saat ini adalah bahwa tidak ada varian SIDH yang diketahui yang dapat mempertahankan karakteristik kinerja yang hampir sama dengan yang awalnya diusulkan oleh Tim SIKE namun tetap aman.

Berdasarkan hal di atas, Tim SIKE mempertimbangkan kemungkinan untuk menarik diri dari proses standardisasi NIST PQC. Tetapi Tim SIKE memutuskan untuk tetap mengajukan proposal ke NIST PQC putaran ke-4 dengan catatan tambahan yang menyatakan bahwa SIKE dan SIDH tidak aman dan sebaiknya tidak digunakan. Sehingga versi akhir dari proposal SIKE secara akurat mencerminkan status kriptosistem saat ini. Tim SIKE juga meminta NIST untuk menginformasikan hal tersebut di tempat yang paling mungkin dilihat

oleh para pengguna potensial mengingat tidak semua orang yang menelusuri kandidat NIST PQC mengikuti perkembangan forum diskusi

Berikut adalah deskripsi singkat SIKE sesuai dengan proposal yang diajukan pada NIST putaran ke-4 (dapat dilihat pada dokumen *Supersingular Isogeny Key Encapsulation* yang diterbitkan pada tanggal 15 September 2022 dan dapat diunduh di <https://sike.org/>).

1. Dasar Matematika

a. *Finite Field*

Finite field terdiri atas himpunan berhingga elemen yang tertutup di bawah operasi penjumlahan dan perkalian yang didefinisikan di atas himpunan tersebut. Ada elemen identitas aditif (0) dan elemen identitas perkalian (1). Setiap elemen memiliki invers aditif yang unik, dan setiap elemen bukan nol memiliki invers perkalian yang unik.

Untuk bilangan bulat positif q , ada *finite field* dari elemen q jika dan hanya jika q adalah pangkat dari bilangan prima p . Jika $\mathbb{F}_q$ adalah *finite field* dengan $q = p^t$ untuk bilangan prima p , karakteristik dari $\mathbb{F}_q$ didefinisikan dengan $\text{char}(\mathbb{F}_q)$ sebagai p . *Finite field* yang digunakan dalam kriptografi *supersingular isogeny* adalah *field* ekstensi kuadratik dari *field* prima $\mathbb{F}_q$, dengan $p = 2^{e_2} 3^{e_3} - 1$, dengan e_2 dan e_3 adalah parameter publik tetap, dan *field* ekstensi dibentuk $\mathbb{F}_{p^3} = \mathbb{F}_q(i)$ dengan $i^2 + 1 = 0$. Ketika abstraksi akan merujuk ke $l, m \in \{2, 3\}$, sehingga $l \neq m$.

b. *Finite Field* $\mathbb{F}_p$

Elemen pada $\mathbb{F}_p$ dipresentasikan dengan bilangan bulat

$$\{0, 1, \dots, p-1\}$$

dengan operasi *field* didefinisikan sebagai berikut:

- 1) Penjumlahan: Jika $a, b \in \mathbb{F}_p$ maka $a + b = r$ di $\mathbb{F}_p$ dengan $r \in [0, p-1]$ adalah sisa $a + b$ dibagi dengan p , juga dikenal sebagai penjumlahan modulus p .

- 2) Perkalian: Jika $a, b \in \mathbb{F}_p$ maka $ab = s$ di $\mathbb{F}_p$ dengan $s \in [0, p-1]$ adalah sisa ab dibagi dengan p , juga dikenal sebagai perkalian modulus p .
- 3) Invers penjumlahan: Jika $a \in \mathbb{F}_p$ solusi unik di $[0, p-1]$ pada persamaan $a + x \equiv 0 \pmod{p}$ adalah invers penjumlahan $(-a)$.
- 4) Invers perkalian: Jika $a \in \mathbb{F}_p, a \neq 0$ solusi unik di $[0, p-1]$ pada persamaan $ax \equiv 1 \pmod{p}$ adalah invers perkalian a^{-1} .

Konvensi yang digunakan dalam SIKE terkait *finite field* adalah $a - b = a + (-b)$ dan $a/b = a \cdot b^{-1}$ dalam $\mathbb{F}_p$.

c. *Finite Field* $\mathbb{F}_{p^2}$

Elemen $\mathbb{F}_{p^2}$ direpresentasikan sebagai $s = s_0 + s_1 \cdot i$ dengan $s_0, s_1 \in \mathbb{F}_p$ dengan operasi *field* didefinisikan sebagai berikut:

- 1) Penjumlahan: Jika $a, b \in \mathbb{F}_{p^2}$ maka $(a_0 + a_1 \cdot i) + (b_0 + b_1 \cdot i) = (a_0 + b_0) + (a_1 + b_1) \cdot i$ di $\mathbb{F}_{p^2}$ dengan penjumlahan $(a_i + b_i)$ dilakukan di $\mathbb{F}_p$.
- 2) Perkalian: Jika $a, b \in \mathbb{F}_{p^2}$ maka $(a_0 + a_1 \cdot i)(b_0 + b_1 \cdot i) = (a_0b_0 - a_1b_1) + (a_0b_1 + a_1b_0) \cdot i$ di $\mathbb{F}_{p^2}$ dengan operasi penjumlahan, invers penjumlahan dan perkalian dilakukan di $\mathbb{F}_p$.
- 3) Invers penjumlahan: Jika $a \in \mathbb{F}_p, a \neq 0$ maka $(-a_0) + (-a_1) \cdot i \in \mathbb{F}_{p^2}$ adalah invers penjumlahan $(-a)$ dengan nilai $(-a_i)$ dihitung dalam field $\mathbb{F}_p$.
- 4) Invers perkalian: Jika $a \in \mathbb{F}_p, a \neq 0$ maka $(a_0(a_0^2 + a_1^2)^{-1} + (-a_1)(a_0^2 + a_1^2)^{-1}) \cdot i \in \mathbb{F}_{p^2}$ adalah invers perkalian a^{-1} dengan operasi dilakukan di $\mathbb{F}_p$.
- 5) Akar kuadrat: Jika terdapat $r = \alpha + \beta \in \mathbb{F}_p$ dengan $\alpha, \beta \in \mathbb{F}_p$ sedemikian sehingga $r^2 = s$, maka didefinisikan $\sqrt{s} = r$ jika $\alpha \neq 0$ merupakan bilangan bulat genap atau $\alpha = 0$

dan β merupakan bilangan bulat genap, jika tidak $\sqrt{s} = -r$.

d. Kurva Montgomery

Kurva Montgomery adalah bentuk spesial dari kurva eliptis. Misalkan $A, B \in \mathbb{F}_q$ adalah elemen *field* yang memenuhi $B(A^2 - 4) \neq 0$ di $\mathbb{F}_q$ (dengan $\text{char}(\mathbb{F}_q) \neq 2$). Kurva Montgomery $E_{A,B}$ yang didefinisikan di atas $\mathbb{F}_q$, dinotasikan $E_{A,B}/\mathbb{F}_q$, didefinisikan sebagai himpunan titik $P = (x, y)$ dari solusi dalam $\mathbb{F}_q$ untuk persamaan

$$By^2 = x^3 + Ax^2 + x$$

bersama dengan titik tambahan $\mathcal{O}$, yang disebut titik di takhingga. Untuk mempermudah, berikut notasi terkait dengan kurva:

- $E_{A,B}$ ketika *field* yang menjadi dasar $\mathbb{F}_q$ ditetapkan oleh konteks, atau tidak ditentukan,
- $E(\mathbb{F}_q)$ ketika parameter kurva ditetapkan oleh konteks, atau tidak ditentukan,
- E ketika *field* dan parameter kurva A, B ditetapkan oleh konteks, atau tidak ditentukan.
- E_A ketika *field* yang menjadi dasar $\mathbb{F}_q$ ditetapkan oleh konteks, atau tidak ditentukan, dan ketika B (yang menentukan *quadratic twist*) dianggap $B = 1$ atau tidak relevan.

Akan lebih mudah untuk merujuk ke koordinat- x dari titik P . Selanjutnya akan digunakan notasi x_P untuk merujuk ke koordinat- x dari P , dan secara analog y_P untuk merujuk ke koordinat- y .

Himpunan titik-titik E bersama dengan titik pada takhingga membentuk grup finite abelian di bawah aturan penjumlahan titik. Orde kurva eliptis E pada finite field $\mathbb{F}_q$, dinotasikan $\#E(\mathbb{F}_q)$, adalah jumlah titik di E termasuk $\mathcal{O}$.

Sering kali, kurva Montgomery ditunjukkan oleh $M_{A,B}$ tetapi dalam dokumen ini akan digunakan notasi $E_{A,B}$.

e. Penjumlahan titik

Diberikan 2 titik $P = (x_P, y_P)$ dan $Q = (x_Q, y_Q)$ dengan $P \neq \pm Q$ pada kurva Montgomery $E_{A,B}$. Pada finite field $\mathbb{F}_q$, dapat dihitung $R = P + Q$ sebagai

$$x_R = B\lambda^2 - (x_P + x_Q) - A$$

dan

$$y_R = \lambda(x_P - x_R) - y_P$$

dengan $R = (x_R, y_R)$ dan $\lambda = (y_P - y_Q)/(x_P - x_Q)$.

Untuk menambahkan sebuah titik ke dirinya sendiri beberapa kali, misalnya k kali, dinyatakan sebagai $P + P + \dots + P = [k]P$. Orde dari sebuah titik P $\text{ord}(P)$ adalah bilangan bulat positif terkecil n sehingga $[n]P = \mathcal{O}$ (titik di takhingga).

f. Doubling titik

Misalkan $(x_P, y_P) \in E_{A,B}$ adalah titik yang ordenya tidak membagi 2. Maka $[2]P = (x_{[2]P}, y_{[2]P}) \in E_{A,B}$ dapat dihitung sebagai

$$(x_{[2]P}, y_{[2]P}) = \left(\frac{(x_P^2 - 1)^2}{4x_P(x_P^2 + Ax_P + 1)}; y_P \cdot \frac{(x_P^2 - 1)(x_P^4 + 2Ax_P^3 + 6x_P^2 + 2Ax_P + 1)}{8x_P^2(x_P^2 + Ax_P + 1)^2} \right)$$

Perhatikan bahwa $x_{[2]P}$ hanya bergantung pada x_P dan A .

g. Tripling titik

Misalkan $(x_P, y_P) \in E_{A,B}$ adalah titik yang ordenya tidak membagi 3. Maka $[3]P = (x_{[3]P}, y_{[3]P}) \in E_{A,B}$ dapat dihitung sebagai

$$x_{[3]P} = \frac{(x_P^4 - 4Ax_P - 6x_P^2 - 3)^2 x_P}{(4Ax_P^3 + 3x_P^4 + 6x_P^2 - 1)^2}$$

dan

$$y_{[3]P} = y_P \cdot \frac{(x_P^4 - 4Ax_P - 6x_P^2 - 3)(x_P^8 + 4Ax_P^7 + 28Ax_P^5 + (16A^2 + 6)x_P^4 + 28Ax_P^3 + 28Ax_P^2 + 4Ax_P + 1)}{(4Ax_P^3 + 3x_P^4 + 6x_P^2 - 1)^3}$$

Perhatikan bahwa $x_{[3]P}$ hanya bergantung pada x_P dan A .

h. Properti tambahan kurva eliptis

Untuk sebarang grup G dan himpunan elemen $\{P_1, P_2, \dots, P_t\} \subseteq G$, dapat didefinisikan subgrup $\langle P_1, P_2, \dots, P_t \rangle$ yang dibangkitkan oleh himpunan tersebut sebagai subgrup terkecil dari G yang memuat elemen-elemen $\{P_1, P_2, \dots, P_t\}$. Untuk grup abelian G , dikatakan himpunan elemen $\{P_1, P_2, \dots, P_t\} \subseteq G$ membentuk basis G jika setiap elemen P dari G memiliki ekspresi unik dalam bentuk:

$$P = [k_1]P_1 + [k_2]P_2 + \dots + [k_t]P_t$$

dengan $0 \leq k_i < \text{ord}(P_i)$ untuk semua i . Secara analog, dikatakan himpunan $\{P_1, P_2, \dots, P_t\} \subseteq H$ membentuk basis dari subgrup $H \subseteq G$ ketika semua elemen dari subgrup H memiliki ekspresi unik seperti di atas. *Weil pairing* [6] dapat membantu dalam menentukan apakah suatu himpunan membentuk basis atau tidak, karena untuk $n = \text{ord}(P) = \text{ord}(Q)$, *Weil pairing* orde- n (e_n) memiliki sifat bahwa $\text{ord}(e_n(P, Q)) = n$ jika dan hanya jika $\langle P \rangle \cap \langle Q \rangle = \{O\}$.

Untuk bilangan integer positif m , didefinisikan himpunan $E[m]$ elemen torsi m dari kurva eliptis $E(\mathbb{F}_q)$ sebagai himpunan titik-titik di $E(\mathbb{F}_q)$ sehingga $[m]P = O$.

Kurva eliptis $E(\mathbb{F}_q)$ pada field karakteristik p disebut supersingular jika $p \mid (q + 1 - \#E(\mathbb{F}_q))$ dan biasa jika tidak.

Invarian j dari kurva eliptik $E_{A,B}$ dihitung sebagai

$$j(E_{A,B}) = \frac{256(A^2 - 3)^3}{A^2 - 4}$$

Invarian j dari kurva eliptis pada field $\mathbb{F}_q$ bersifat unik hingga isomorfisma kurva eliptis. Protokol SIKE mendefinisikan *shared secret* sebagai invarian j dari kurva eliptis.

i. Isogeni

Misalkan E_1 dan E_2 adalah kurva eliptis pada finite field $\mathbb{F}_q$. Isogeni $\phi = E_1 \rightarrow E_2$ adalah pemetaan rasional non-konstan

yang didefinisikan pada $\mathbb{F}_q$ yang juga merupakan homomorfisma grup dari $E_1(\mathbb{F}_q)$ ke $E_2(\mathbb{F}_q)$. Jika pemetaan tersebut ada, dikatakan E_1 isogen terhadap E_2 , dan dua kurva E_1 dan E_2 pada $\mathbb{F}_q$ isogen jika dan hanya jika $\#E_1(\mathbb{F}_q) = \#E_2(\mathbb{F}_q)$.

Isogeni ϕ dapat dinyatakan dalam bentuk dua pemetaan rasional f dan g pada $\mathbb{F}_q$ sehingga $\phi((x,y)) = (f(x), y \cdot g(x))$. Dapat ditulis $f(x) = p(x)/q(x)$ dengan polinomial $p(x)$ dan $q(x)$ pada $\mathbb{F}_q$ yang tidak memiliki faktor persekutuan, dan hal yang sama berlaku untuk $g(x)$. Derajat $\deg(\phi)$ dari isogeni didefinisikan sebagai $\max\{\deg(p(x)), \deg(q(x))\}$, dengan $p(x)$ dan $q(x)$ seperti di atas. Sering kali lebih mudah untuk melakukan perhitungan isogeni hanya menggunakan $f(x)$ komponen dari isogeni.

Diberikan isogeni $\phi = E_1 \rightarrow E_2$, kernel ϕ didefinisikan sebagai berikut:

$$\ker(\phi) = \{P \in E_1 : \phi(P) = \mathcal{O}\}$$

Untuk setiap subgrup berhingga H dari $E(\mathbb{F}_q)$ ada isogeni unik (hingga isomorfisme) $\phi = E \rightarrow E'$ sehingga $\ker(\phi) = H$ dan $\deg(\phi) = |H|$, dimana $|H|$ menunjukkan kardinalitas H . Dalam kasus ini, kurva E' dinyatakan dengan E/H . Diberikan subgrup $H \subseteq E(\mathbb{F}_q)$, rumus Vélu [7] dapat digunakan untuk menemukan isogeni ϕ dan kurva isogen E/H . Rumus Vélu tidak praktis secara komputasi untuk subgrup sembarang. SIKE menggunakan isogeni atas subgrup yang merupakan pangkat 2, 3, dan 4.

1) 2-isogeni

Misalkan $(x_2, y_2) \in E_{A,B}$ adalah titik orde 2 dengan $x_2 \neq \pm 0$ dan misalkan $\phi_2 : E_{A,B} \rightarrow E_{A',B'}$ adalah isogeni 2 unik (hingga isomorfisme) dengan kernel $\langle (x_2, y_2) \rangle$. Maka $E_{A',B'}$ dapat dihitung sebagai:

$$(A', B') = (2 \cdot (1 - 2x_2^2), Bx_2)$$

Perhatikan bahwa A' hanya bergantung pada x_2 .

Jika $P = (x_p, y_p)$ adalah sebarang titik pada $E_{A,B}$ yang tidak berada di $\ker(\phi_2)$, maka $\phi_2: (x_p, y_p) \mapsto (x_{\phi_2(P)}, y_{\phi_2(P)})$, dan dapat dihitung sebagai

$$x_{\phi_2(P)} = \frac{x_p^2 x_2 - x_p}{x_p - x_2}$$

dan

$$y_{\phi_2(P)} = \frac{x_p^2 x_2 - 2x_p x_2^2 + x_2}{(x_p - x_2)^2}$$

$x_{\phi_2(P)}$ hanya bergantung pada x_p dan x_2 .

2) 4-isogeni

Misalkan $(x_4, y_4) \in E_{A,B}$ adalah titik orde 4 dengan $x_4 \neq \pm 1$ dan misalkan $\phi_4: E_{A,B} \rightarrow E_{A',B'}$ adalah isogeni 4 unik (hingga isomorfisme) dengan kernel $\langle (x_4, y_4) \rangle$. Maka $E_{A',B'}$ dapat dihitung sebagai:

$$(A', B') = (4x_4^4 - 2, -x_4(x_4^2 + 1) \cdot B/2)$$

Perhatikan bahwa A' hanya bergantung pada x_4 .

Jika $P = (x_p, y_p)$ adalah sebarang titik pada $E_{A,B}$ yang tidak berada di $\ker(\phi_4)$, maka $\phi_4: (x_p, y_p) \mapsto (x_{\phi_4(P)}, y_{\phi_4(P)})$, dan dapat dihitung sebagai:

$$x_{\phi_4(P)} = \frac{-(x_p x_4^2 + x_p - 2x_4)x_p(x_p x_4 - 1)^2}{(x_p - x_4)^2(2x_p x_4 - x_4^2 - 1)}$$

dan

$$y_{\phi_4(P)} = y_p \cdot \frac{-2x_4^2(x_p x_4 - 1)(x_p^4(x_4^2 + 1) - 4x_p^3(x_4^3 + x_4) + 2x_p^2(x_4^4 + 5x_4^2) - 4x_p(x_4^3 + x_4) + x_4^2 + 1)}{(x_p - x_4)^3(2x_p x_4 - x_4^2 - 1)^2}$$

Perhatikan bahwa $x_{\phi_4(P)}$ hanya bergantung pada x_p dan x_4 .

3) 3-isogeni

Misalkan $(x_3, y_3) \in E_{A,B}$ adalah titik orde 3 dan misalkan $\phi_3: E_{A,B} \rightarrow E_{A',B'}$ adalah isogeni 3 unik (hingga isomorfisme) dengan kernel $\langle (x_3, y_3) \rangle$. Maka $E_{A',B'}$ dapat dihitung sebagai:

$$(A', B') = ((Ax_3 - 6x_3^3 + 6)x_3, Bx_3^2)$$

Koefisien baru A' hanya bergantung pada x_p dan x_3 .

Jika $P = (x_P, y_P)$ adalah sebarang titik pada $E_{A,B}$ yang tidak berada di $\ker(\phi_3)$, maka $\phi_3: (x_P, y_P) \mapsto (x_{\phi_3(P)}, y_{\phi_3(P)})$, dan dapat dihitung sebagai:

$$(x_{\phi_3(P)}, y_{\phi_3(P)}) = \left(\frac{x_P(x_P x_3 - 1)^2}{(x_P - x_3)^2}, y_P \cdot \frac{(x_P x_3 - 1)(x_P^2 x_3 - 3x_P x_3^2 + x_P + x_3)}{(x_P - x_3)^3} \right)$$

Perhatikan bahwa $x_{\phi_3(P)}$ hanya bergantung pada x_P dan x_3 .

Protokol SIKE mendefinisikan kunci rahasia dari dua ruang kunci terpisah, $\mathcal{K}_1$ dan $\mathcal{K}_2$. Kunci rahasia sk mendefinisikan subgrup H dari $E(\mathbb{F}_q)$, yang selanjutnya mendefinisikan isogeni $\phi_{sk}: E \rightarrow E/H$. Kunci publik ditentukan oleh isogeni ϕ_{sk} dan titik $P, Q \in E(\mathbb{F}_q)$ yang ditetapkan secara global sebagai parameter publik dan tidak bergantung pada sk . Lebih khusus lagi, kunci publik yang sesuai dengan sk ditentukan oleh $\{E/H, \phi_{sk}(P), \phi_{sk}(Q)\}$. Titik P dan Q dipilih sehingga $\{P, Q\}$ membentuk basis untuk $E[\ell^{e_\ell}]$. Dalam implementasinya, untuk alasan efisiensi, kunci publik direpresentasikan sebagai triplet elemen *field*, yaitu tiga koordinat $\{x_{\phi_{sk}(P)}, x_{\phi_{sk}(Q)}, x_{\phi_{sk}(P-Q)}\}$ dari tiga titik di bawah isogeni. Dimungkinkan untuk mengonversi antar representasi menggunakan metode yang diberikan dalam [27]. Misalnya, koefisien kurva Montgomery A dari E/H dapat dipulihkan oleh tiga koordinat x dari kunci publik $\{x_{\phi_{sk}(P)}, x_{\phi_{sk}(Q)}, x_{\phi_{sk}(P-Q)}\}$ menggunakan persamaan.

$$A = \frac{(1 - x_{\phi_{sk}(P)} x_{\phi_{sk}(Q)} - x_{\phi_{sk}(P)} x_{\phi_{sk}(P-Q)} - x_{\phi_{sk}(Q)} x_{\phi_{sk}(P-Q)})^2}{4x_{\phi_{sk}(P)} x_{\phi_{sk}(Q)} x_{\phi_{sk}(P-Q)}} - x_{\phi_{sk}(P)} - x_{\phi_{sk}(Q)} - x_{\phi_{sk}(P-Q)}$$

Demikian pula, titik $\phi_{sk}(P)$ dan $\phi_{sk}(Q)$ dapat dipulihkan (hingga tanda simultan) dari $x_{\phi_{sk}(P)}$ dan $x_{\phi_{sk}(Q)}$ menggunakan rumus:

$$y_{\phi_{sk}(P)} = \sqrt{x_{\phi_{sk}(P)}^3 + Ax_{\phi_{sk}(P)}^2 + x_{\phi_{sk}(P)}}$$

dan

$$y_{\phi_{sk}(Q)} = \sqrt{x_{\phi_{sk}(Q)}^3 + Ax_{\phi_{sk}(Q)}^2 + x_{\phi_{sk}(Q)}}$$

dan jika

$$x_{\phi_{sk}(P-Q)} + x_{\phi_{sk}(Q)} + x_{\phi_{sk}(P)} + A \neq \left(\frac{y_{\phi_{sk}(Q)} - y_{\phi_{sk}(P)}}{x_{\phi_{sk}(Q)} - x_{\phi_{sk}(P)}} \right)^2$$

Maka himpunan $y_{\phi_{sk}(Q)} = -y_{\phi_{sk}(Q)}$.

2. Spesifikasi Protokol SIKE

a. Parameter-parameter Publik

Parameter-parameter publik pada SIKE adalah:

- 1) Dua integer positif e_2 dan e_3
- 2) Kurva eliptis supersingular awal $E_0/\mathbb{F}_{p^2}$
- 3) Himpunan tiga koordinat x yang sesuai dengan titik-titik pada $E_0[2^{e_2}]$
- 4) Himpunan tiga koordinat x yang sesuai dengan titik-titik pada $E_0[3^{e_3}]$

b. Kurva Awal

Kurva awal publik adalah kurva eliptis supersingular:

$$E_0/\mathbb{F}_{p^2} : y^2 = x^3 + 6x^2 + x$$

dengan $\#(E_0/\mathbb{F}_{p^2}) = (2^{e_2} 3^{e_3})^2$ dan invarian- j sama dengan $j(E_0) = 287496$.

Berikut adalah contoh khusus kurva Montgomery $By^2 = x^3 + Ax^2 + x$, dengan $A = 6$ dan $B = 1$. Kurva ini telah diperbarui sejak proposal awal SIKE, karena alasan yang dijelaskan lebih lanjut dalam [28]. Kurva asli memiliki $j = 1728$, dengan E_0 adalah satu-satunya kurva 2-isogen non-isomorfis, yang berarti penyerang akan mengetahui dengan pasti langkah pertama yang diambil dari titik awal ini dalam grafik 2-isogeni, terlepas dari rahasianya.

Hanya ada dua (berlawanan dengan pada umumnya yaitu empat) kelas isomorfisme yang 3-isogen terhadap $j = 1728$, sehingga kernel yang berbeda dapat mengarah ke isogeni isomorfis. Memulai pada E_0 untuk menghindari kedua masalah ini. Selain itu, kombinasi titik dasar pada E_0 dan

perhitungan subgrup kernel rahasia memastikan bahwa 2-isogeni pertama yang diambil dari E_0 tidak searah kurva dengan $j = 1728$, melainkan ke salah satu dari dua kurva 2-isogen lainnya.

c. Titik-titik pembangkit parameter Publik

Tiga koordinat x dalam parameter publik yang sesuai dengan titik-titik di $E_0[2^{e_2}]$ ditentukan sebagai berikut.

Pertama-tama tentukan dua titik:

$$P_2 \in E_0(\mathbb{F}_{p^2}) \text{ dan } Q_2 \in E_0(\mathbb{F}_{p^2})$$

sedemikian sehingga kedua titik tersebut memiliki orde 2^{e_2} yang tepat, dan $\{P_2, Q_2\}$ membentuk basis untuk $E_0(\mathbb{F}_{p^2})[2^{e_2}]$, yaitu, *Weil pairing* orde-2 $e_2(P_2, Q_2) \in \mathbb{F}_{p^2}^\times$ memiliki orde penuh, atau ekuivalennya, $e_2([2^{e_2-1}]P_2, [2^{e_2-1}]Q_2) \in \mathbb{F}_{p^2}^\times$ tidak sama dengan 1.

Ditentukan juga dua titik

$$P_3 \in E_0(\mathbb{F}_p) \text{ dan } Q_3 \in E_0(\mathbb{F}_{p^2}) \setminus E_0(\mathbb{F}_p)$$

Sedemikian sehingga kedua titik tersebut memiliki orde eksak 3^{e_3} , dan $\{P_3, Q_3\}$ membentuk basis untuk $E_0(\mathbb{F}_{p^2})[3^{e_3}]$.

Misalkan $f := X^3 + 6X^2 + X$ dan ingat $\mathbb{F}_{p^2} = \mathbb{F}_p(i)$ dengan $i^2 + 1 = 0$. Titik-titik P_2, Q_2, P_3, Q_3 ditentukan menurut prosedur berikut:

- 1) $P_2 = [3^{e_3}](i + c, \sqrt{f(i+c)})$, dengan c adalah integer nonnegatif terkecil sedemikian sehingga $P_2 \in E_0(\mathbb{F}_{p^2})$ dan $[2^{e_2-1}]P_2 = (-3 \pm 2\sqrt{2}, 0)$
- 2) $Q_2 = [3^{e_3}](i + c, \sqrt{f(i+c)})$, dengan c adalah integer nonnegatif terkecil sedemikian sehingga $Q_2 \in E_0(\mathbb{F}_{p^2})$ dan $[2^{e_2-1}]Q_2 = (0, 0)$
- 3) $P_3 = [2^{e_2-1}](c, \sqrt{f(c)})$, dengan c adalah integer nonnegatif terkecil sedemikian sehingga $f(c)$ adalah kuadrat di $\mathbb{F}_p$ dan P_3 mempunyai orde 3^{e_3} .

- 4) $Q_3 = [2^{e_2-1}](c, \sqrt{f(c)})$, dengan c adalah integer nonnegatif terkecil sedemikian sehingga $f(c)$ adalah bukan kuadrat di $\mathbb{F}_p$ dan Q_3 mempunyai orde 3^{e_3} .

Titik P_2, Q_2, P_3, Q_3 , dapat berfungsi sebagai parameter publik untuk SIKE, tetapi sebagai gantinya, demi alasan efisiensi (seperti dijelaskan dalam [28], titik P_2 dan Q_2 menggunakan tiga koordinat- x x_{P_2}, x_{Q_2} dan x_{R_2} , dengan $R_2 = P_2 - Q_2$. Demikian pula, P_3, Q_3 dikodekan dengan menggunakan tiga koordinat- x x_{P_3}, x_{Q_3} dan x_{R_3} , dengan $R_3 = P_3 - Q_3$.

- d. Titik-titik pembangkit parameter Publik (SIKE terkompresi)

Berikut ini dijelaskan metodologi untuk membangun generator untuk varian terkompresi SIKE. Metodologi ini diperkenalkan di [29] dimotivasi oleh optimasi kompresi kunci.

Tentukan dua titik:

$$P_3 \in E_0(\mathbb{F}_p) \text{ dan } Q_3 \in E_0(\mathbb{F}_{p^2}) \setminus E_0(\mathbb{F}_p)$$

sedemikian sehingga kedua titik tersebut memiliki orde 3^{e_3} yang tepat, dan $\{P_3, Q_3\}$ membentuk basis untuk $E_0(\mathbb{F}_{p^2})[3^{e_3}]$.

Misalkan $f := X^3 + 6x^2 + x$ dan ingat $\mathbb{F}_{p^2} = \mathbb{F}_p(i)$ dengan $i^2 + 1 = 0$. Titik-titik P_2, Q_2, P_3, Q_3 ditentukan menurut prosedur berikut:

- 1) $P_2 = \begin{bmatrix} 1 \\ 2 \end{bmatrix} P'_2$, sedemikian sehingga $P'_2 = [3^{e_3}](c, \sqrt{f(c)})$ adalah titik orde 2^{e_2-1} , dengan c adalah bilangan bulat nonnegatif terkecil sedemikian sehingga $f(c)$ bukan kuadrat di $\mathbb{F}_p$.
- 2) $Q_2 = \begin{bmatrix} 1 \\ 2 \end{bmatrix} Q'_2$, sedemikian sehingga $Q'_2 = [3^{e_3}](c, \sqrt{f(c)})$ adalah titik orde 2^{e_2-1} dan independen linear dengan P'_2 , dengan c adalah bilangan bulat nonnegatif terkecil sedemikian sehingga $f(c)$ kuadrat di $\mathbb{F}_p$.
- 3) $P_3 = [2^{e_2}](c, \sqrt{f(c)})$, dengan c adalah bilangan bulat nonnegatif terkecil sedemikian sehingga $f(c)$ kuadrat di $\mathbb{F}_p$ dan P_3 mempunyai orde 3^{e_3} .

4) $Q_3 = (-x(P_3), i \cdot y(P_3))$ adalah peta distorsi dari P_3 .

Titik P_2, Q_2, P_3, Q_3 , dapat berfungsi sebagai parameter publik untuk SIKE terkompresi, tetapi sebaliknya demi alasan efisiensi, titik titik P_2 dan Q_2 dikodekan dengan menggunakan tiga koordinat- x x_{P_2}, x_{Q_2} dan x_{R_2} , dengan $R_2 = P_2 - Q_2$. Demikian pula, P_3, Q_3 dikodekan dengan menggunakan tiga koordinat- x x_{P_3}, x_{Q_3} dan x_{R_3} , dengan $R_3 = P_3 - Q_3$.

e. Perhitungan Isogeni

Pada bagian ditetapkan $\ell, m \in \{2, 3\}$ sedemikian sehingga $\ell \neq m$. Dua algoritma isogeni fundamental yang dijelaskan adalah isogen_ℓ dan isoex_ℓ . Pada input parameter publik dan kunci rahasia, isogen_ℓ mengeluarkan kunci publik yang sesuai dengan input kunci rahasia. Pada input kunci rahasia dan kunci publik, isoex_ℓ mengeluarkan kunci bersama yang sesuai. Kedua algoritma ini akan digunakan sebagai blok penyusun untuk skema PKE dan KEM yang didefinisikan pada bagian selanjutnya.

Kedua algoritma tersebut menghitung isogeni derajat ℓ^e melalui komposisi e_ℓ isogeni derajat ℓ ; isogeni derajat ℓ dievaluasi pada setidaknya satu titik yang terletak pada kurva domain. Mengikuti [30] dan [31], daripada mengevaluasi *image* isogeni pada titik $R = (x_R, y_R)$, lebih efisien untuk mengevaluasi *image* di bawah proyeksi x -saja $(x_R, y_R) \mapsto x_R$. Karena pemetaan koordinat untuk isogeni $\psi: E \rightarrow E', R \mapsto \psi(R)$ selalu dapat ditulis sedemikian sehingga $x_{\psi(R)} = f(x_R)$ untuk beberapa fungsi f [8], algoritma isogen_ℓ dan isoex_ℓ akan mengasumsikan isogeni derajat ℓ ke- i ϕ_i mematuhi kerangka kerja ini dengan menulis $\phi_i: (x, -) \mapsto (f_i(x), -)$.

Perhatikan bahwa definisi parameter publik dan kunci publik memungkinkan kemungkinan implementasi generik yang kembali ke komputasi isogeni penuh yang menghitung

koordinat x dan y dari titik *image* baik dalam kerangka Montgomery maupun kerangka Weierstrass pendek. Secara khusus, kurva awal E_0 yang didefinisikan dalam kurva awal di huruf b adalah contoh khusus dari kurva Montgomery dan kurva Weierstrass pendek. Titik generator publik dalam titik-titik pembangkit parameter publik di huruf c secara unik mendefinisikan koordinat y dari P_2, Q_2, P_3 dan Q_3 .

f. Perhitungan kunci publik isogen $_{\ell}$

Sepasang kunci isogeni supersingular terdiri atas kunci rahasia sk_{ℓ} , yang merupakan bilangan integer, dan himpunan tiga koordinat- x $pk_{\ell} = (x_P, x_Q, x_R)$. Perhitungan kunci publik isogen $_{\ell}$ sebagai berikut:

Parameter publik

- Prima $p = 2^{e_2} 3^{e_3} - 1$
- kurva awal $E_0 \setminus \mathbb{F}_{p^2}$ dan
- pembangkit publik $\{x_{P_2}, x_{Q_2}, x_{R_2}\}$ dan $\{x_{P_3}, x_{Q_3}, x_{R_3}\}$

Input

- kunci rahasia sk_{ℓ}

Output

- Kunci publik pk_{ℓ}

Aksi

Hitung kunci publik pk_{ℓ} , sebagai berikut:

1. Set $x_s \leftarrow x_{P_{\ell} + [sk_{\ell}]Q_{\ell}}$;
2. Set $(x_1, x_2, x_3) \leftarrow (x_{P_m}, x_{Q_m}, x_{R_m})$;
3. For i from 0 to $e_{\ell} - 1$ do
 - (a) Compute the x portion for an ℓ -isogeny

$$\phi_i: E_i \rightarrow E'$$

$$(x, -) \mapsto (f_i(x), -)$$

such that $\ker \phi_i = \langle [\ell^{e_{\ell}-i-1}]S \rangle$ where S is a point on E_i with x -coordinate x_s .

- (b) Set $E_{i+1} \rightarrow E'$;

- (c) Set $x_s \leftarrow f_i(x_s)$;

- (d) $Set(x_1, x_2, x_3) \leftarrow (f_i(x_1), f_i(x_2), f_i(x_3));$
4. Output $pk_\ell = (x_1, x_2, x_3)$
- g. Pembuatan kunci bersama $isoex_\ell$

Parameter publik

- Prima $p = 2^{e_2} 3^{e_3} - 1$

Input

- Kunci publik $pk_m = (x_{P_m}, x_{Q_m}, x_{R_m})$ dan
- kunci rahasia sk_ℓ

Output

- Kunci bersama j ,
- *string* oktet dengan panjang $2N_p$

Aksi

Hitung kunci bersama j , sebagai berikut:

1. Compute E'_0 from pk_m using cfpk (lihat pada dokumen *Supersingular Isogeny Key Encapsulation* bab 1.2.1)
2. Set $x_s \leftarrow x_{m+[sk_\ell]Q_m}$
3. For i from 0 to $e_\ell - 1$ do
 - (a) Compute the x portion for an ℓ -isogeny

$$\phi_i: E_i \rightarrow E'$$

$$(x, -) \mapsto (f_i(x), -)$$
 such that $\ker \phi_i = \langle [\ell^{e_\ell-i-1}]S \rangle$ where S is a point on E_i with x -coordinate x_s .
 - (b) Set $E_{i+1} \rightarrow E'$;
 - (c) Set $x_s \leftarrow f_i(x_s)$;
4. Encode $j(E'_{e_\ell})$ into j using fp2toos (lihat pada dokumen *Supersingular Isogeny Key Encapsulation* bab 1.2.8)

- h. Optimasi isogen $_\ell$ dan isoex $_\ell$

Algoritma isogen $_\ell$ dan isoex $_\ell$ yang dijelaskan di atas, meskipun waktu polinomial, relatif tidak efisien dalam praktiknya. Dalam kedua kasus, bagian yang paling mahal adalah perhitungan titik $[\ell^{e_\ell-i-1}]S$ karena salah satu perhitungan tersebut memerlukan paling banyak perkalian ℓ^{e_ℓ}

dengan skalar ℓ , dan diulang e_ℓ kali, dengan total operasi dasar $O(e_\ell)$.

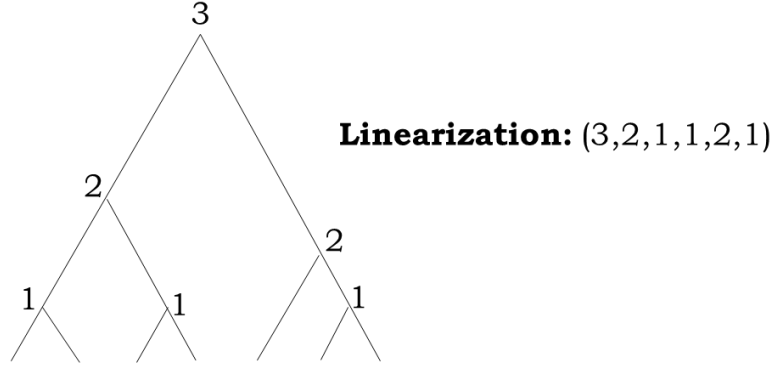
Untuk implementasi yang optimal direkomendasikan [31] untuk mengganti *loop for* dengan dekomposisi rekursif dari perhitungan isogeni menjadi operasi dasar, yang hanya memerlukan perkalian $O(e_\ell \log e_\ell)$ dengan skalar ℓ , dan jumlah evaluasi isogeni ℓ yang serupa. Dekomposisi semacam ini disebut sebagai strategi komputasional yang digambarkan dengan pohon biner penuh pada simpul $e_\ell - 1$. Jika pohon tersebut digambar sehingga semua simpul terletak di dalam wilayah segitiga latis heksagonal, dengan semua daun pada satu batas, maka panjang lintasan pohon tersebut proporsional dengan upaya komputasional yang diperlukan oleh strategi tersebut (sebagai contoh lihat Gambar 3.25).



Gambar 3.25 Tiga strategi komputasi berukuran $e_\ell - 1 = 6$.

Pendekatan sederhana yang digunakan di huruf f dan huruf g sesuai dengan strategi paling kiri

Dalam praktiknya, sebarang pohon biner lengkap pada simpul $e_\ell - 1$ direpresentasikan dengan cara mengaitkan jumlah daun di sebelah kanannya ke simpul internal apa pun, lalu menelusuri pohon dalam urutan kedalaman-pertama kiri-pertama dan mengeluarkan label saat label tersebut ditemukan (sebagai contoh lihat Gambar 3.26).



Gambar 3.26 Representasi linear strategi pada simpul 6.

Dengan pohon biner lengkap apa pun yang direpresentasikan dengan cara ini, perhitungan pada langkah 3 isogen_ℓ dapat digantikan dengan prosedur rekursif berikut:

Input

- Kurva awal E , koordinat- x x_s pada titik S pada E ,
- daftar koordinat- x $(x_1, x_2, \dots)$ pada E .
- Strategi $(s_1, \dots, s_{t-1})$ dengan ukuran $t - 1$

Output

- *Image* kurva $E' = E/\langle S \rangle$ pada isogeni $\psi: E \rightarrow E/\langle S \rangle$ dengan kernel $\langle S \rangle$
- daftar koordinat *image* $(\psi(x_1), \psi(x_2), \dots)$ pada E'

Aksi

1. If $t = 1$ (i.e., the strategy is empty) then
 - (a) Compute an ℓ -isogeny

$$\phi: E \rightarrow E'$$

$$(x, -) \mapsto (f(x), -)$$

such that $\ker \phi = \langle S \rangle$;
 - (b) Return $(E', f(x_1), f(x_2), \dots)$;
2. Let $n = s_1$;
3. Let $L = (s_2, \dots, s_{t-n})$ dan $R = (s_{t-n+1}, \dots, s_{t-1})$
4. Set $x_T \leftarrow x_{[\ell^n]S}$;
5. Set $(E, (x_U, x_1, x_2, \dots)) \leftarrow \text{Recurse on } (E, x_T(x_s, x_1, x_2, \dots))$ with strategi L ;

6. Set $(E, (x_1, x_2, \dots)) \leftarrow \text{Recurse on } (E, (x_U, x_1, x_2, \dots))$ with strategi R ;
7. Return $(E, (x_1, x_2, \dots))$.

Penjelasan lebih lanjut dapat dilihat pada dokumen *Supersingular Isogeny Key Encapsulation* bab 1.3.8.

i. Kunci rahasia

Skema PKE dan KEM memerlukan dua kunci rahasia, sk_2 dan sk_3 , yang masing-masing digunakan untuk menghitung isogeni 2^{e_2} dan isogeni 3^{e_3} .

Misalkan $N_{sk_2} = \lfloor e_2/8 \rfloor$. Kunci rahasia sk_2 sesuai dengan integer dalam rentang $\{0, 1, \dots, 2^{e_2} - 1\}$ yang dikodekan sebagai *string* oktet dengan panjang N_{sk_2} menggunakan itoos (dapat dilihat pada dokumen *Supersingular Isogeny Key Encapsulation* bab 1.2.6). Ruang kunci yang sesuai dilambangkan dengan $\mathcal{K}_2$.

Misalkan $s = \lfloor \log_2 3^{e_3} \rfloor$ dan $N_{sk_3} = \lfloor e_2/8 \rfloor$. Kunci rahasia sk_3 berkorespondensi dengan integer dalam rentang $\{0, 1, \dots, 2^s - 1\}$, yang dikodekan sebagai *string* oktet dengan panjang N_{sk_3} menggunakan itoos (dapat dilihat pada dokumen *Supersingular Isogeny Key Encapsulation* bab 1.2.6). Ruang kunci yang berkorespondensi dilambangkan dengan $\mathcal{K}_2$.

j. Enkripsi Kunci Publik

Algoritma 1 mendefinisikan skema enkripsi kunci publik PKE = (Gen, Enc, Dec). Dua ruang kunci $\mathcal{K}_2$ dan $\mathcal{K}_2$ didefinisikan dalam huruf i. Ukuran ruang pesan $\mathcal{M} = \{0, 1\}^n$, serta fungsi F yang memetakan rahasia bersama j ke *bitstring* penjelasan fungsi F dapat dilihat pada dokumen *Supersingular Isogeny Key Encapsulation* bab 1.4.

Fungsi Enc membangkitkan sk_2 secara acak. Dalam kasus mekanisme enkapsulasi kunci, keacakan tersebut

diimplementasikan dalam input yang dinyatakan dengan $(c_0, c_1) \leftarrow \text{Enc}(\text{pk}_3, m; \text{sk}_2)$ (lihat Baris 7 dari Algoritma 2).

k. Mekanisme enkapsulasi kunci

Algoritma 2 mendefinisikan mekanisme enkapsulasi kunci $\text{KEM} = (\text{KeyGen}, \text{Encaps}, \text{Decaps})$, dengan menerapkan transformasi Hofheinz, Hövelmanns, dan Kiltz [32] ke PKE yang didefinisikan dalam huruf j . Transformasi ini dimodifikasi dengan memasukkan pk_3 dalam input ke G (seperti dalam [33]) dan dengan menyederhanakan "enkripsi ulang (*re-encryption*)".

Dua ruang kunci $\mathcal{K}_2$ dan $\mathcal{K}_3$ didefinisikan dalam huruf i . Ukuran $\mathcal{M} = \{0,1\}^n$ serta fungsi G dan H . Penjelasan fungsi G dan H dapat dilihat pada dokumen *Supersingular Isogeny Key Encapsulation* bab 1.4.

Algorithm 1: $\text{PKE} = (\text{Gen}, \text{Enc}, \text{Dec})$		
function Gen Input: () Output: $(\text{pk}_3, \text{sk}_3)$ $\text{sk}_3 \leftarrow_R \mathcal{K}_3$ $\text{pk}_3 \leftarrow \text{isogen}_3(\text{sk}_3)$ $s \leftarrow_R [0,1]^n$ return $(s, \text{sk}_3, \text{pk}_3)$	function Enc Input: $\text{pk}_3, m \in \mathcal{M}, r \in \mathcal{K}_2$ Output: (c_0, c_1) $\text{sk}_2 \leftarrow r$ $c_0 \leftarrow \text{isogen}_2(\text{sk}_2)$ $j \leftarrow \text{isoex}_2(\text{pk}_3, \text{sk}_2)$ $h \leftarrow F(j)$ $c_1 \leftarrow h \oplus m$ return (c_0, c_1)	function Dec Input: $\text{sk}_3(c_0, c_1)$ Output: m $j \leftarrow \text{isoex}_2(c_0, \text{sk}_3)$ $h \leftarrow F(j)$ $m \leftarrow h \oplus c_1$ return m

Algorithm 2: $\text{KEM} = (\text{KeyGen}, \text{Encaps}, \text{Decaps})$		
function KeyGen Input: () Output: $(s, \text{sk}_3, \text{pk}_3)$ $\text{sk}_3 \leftarrow_R \mathcal{K}_3$ $\text{pk}_3 \leftarrow \text{isogen}_3(\text{sk}_3)$ $s \leftarrow_R [0,1]^n$ return $(s, \text{sk}_3, \text{pk}_3)$	function Encaps Input: pk_3 Output: (c, K) $m \leftarrow_R [0,1]^n$ $r \leftarrow G(m \parallel \text{pk}_3)$ $(c_0, c_1) \leftarrow \text{Enc}(\text{pk}_3, m; r)$ $K \leftarrow H(m \parallel (c_0, c_1))$ return $((c_0, c_1), K)$	function Decaps Input: $(s, \text{sk}_3, \text{pk}_3), (c_0, c_1)$ Output: K $m' \leftarrow \text{Dec}(\text{sk}_3, (c_0, c_1))$ $r' \leftarrow G(m' \parallel \text{pk}_3)$ $c'_0 \leftarrow \text{isogen}_2(r')$ if $c'_0 = c_0$ then $K \leftarrow H(m' \parallel (c_0, c_1))$ else

		$K \leftarrow H(s\ (c_0, c_1))$ return K
--	--	--

3. Primitif Simetris

Ketiga fungsi *hash* F, G , dan H yang digunakan dalam mekanisme enkapsulasi kunci KEM semuanya diwujudkan dengan fungsi turunan SHA-3 SHAKE256 sebagaimana ditetapkan oleh NIST dalam [34].

Secara spesifik, fungsi G melakukan *hash* pada *string* bit acak $m \in \mathcal{M} = \{0,1\}^n$ yang dirangkai dengan kunci publik pk_3 . Fungsi ini diwujudkan dengan SHAKE256, dengan mengambil $m\|pk_3$ sebagai input, yang meminta bit output e_2 yang dinotasikan [9] dengan $G(m\|pk_3) = \text{SHAKE256}(m\|pk_3, e_2)$. Nilai n merupakan $n \in \{128, 192, 256\}$.

Fungsi F digunakan sebagai fungsi derivasi kunci pada invarian- j selama enkripsi kunci publik dan dihitung sebagai $F(j) = \text{SHAKE256}(j, n)$ menggunakan notasi [9], dengan *output* yang diminta terdiri dari n bit dengan n merupakan $n \in \{128, 192, 256\}$.

Fungsi ketiga H digunakan untuk mendapatkan kunci bersama k -bit K dari *string* bit acak m dan ciphertext c yang dihasilkan oleh Enc. Fungsi ini dihitung sebagai $\text{SHAKE256}(m\|c, k)$ dengan $m\|cc$ sebagai input. Nilai k sesuai dengan jumlah bit keamanan klasik, yaitu, $k \in \{128, 192, 256\}$.

4. Kompresi Kunci Publik

Kunci publik SIKE yang tidak dikompresi berbentuk $(x_P, x_Q, x_R) \in (\mathbb{F}_{p^2})^3$, yang sesuai dengan tiga titik $P, Q, R \in E_A(\mathbb{F}_{p^2})$ dengan orde eksak ℓ^e , untuk $\ell \in \{2, 3\}$. Mengikuti [35] dan perbaikan lebih lanjut yang dijelaskan dalam [27] [36], ide kompresi kunci publik adalah untuk merepresentasikan titik-titik ini sebagai elemen $\mathbb{Z}_{\ell^e} \times \mathbb{Z}_{\ell^e}$ berkenaan dengan basis torsi ℓ^e yang dipilih secara deterministik sebagai fungsi E_A . Pengodean elemen $\mathbb{Z}_{\ell^e} \times \mathbb{Z}_{\ell^e}$ memerlukan sekitar setengah bit dari jumlah elemen pengodean $\mathbb{F}_{p^2}$. Secara khusus,

kunci yang dikompresi memerlukan $3.5 \log p$ bit, bukan $6 \log p$ untuk merepresentasikan tiga (x_P, x_Q, x_R) . Ini berarti penghematan $\approx 41\%$ dalam ukuran kunci.

Perlu diperhatikan bahwa kompresi *ciphertext* juga dimungkinkan meskipun *ciphertext* terkompresi menempati sekitar $4 \log p$ bit (penghematan $\approx 33\%$) daripada $3.5 \log p$ bit yang dicapai dalam kompresi kunci karena adanya *tradeoff* yang diperkenalkan dalam [37] yang memungkinkan dekapsulasi yang jauh lebih cepat. Algoritma kompresi dan dekompresi dijelaskan dalam dokumen *Supersingular Isogeny Key Encapsulation* Lampiran C.

5. Parameter

Bagian ini berisi informasi mengenai delapan set parameter SIKE.

Field prima yang mendasari delapan set parameter SIKE adalah $p = 2^{e_2} 3^{e_3} - 1$ dengan $2^{e_2} = 3^{e_3}$. Delapan set parameter SIKE adalah SIKEp434, SIKEp434_compressed, SIKEp503, SIKEp503_compressed, SIKEp610, SIKEp610_compressed, SIKEp751 dan SIKEp751_compressed. Penamaan delapan set parameter tersebut disesuaikan dengan panjang bit karakteristik field prima.

Berikut adalah rincian dari masing parameter yang terdapat dalam masing-masing parameter SIKE:

- a. bilangan prima p dan nilai e_2 dan e_3 ,
- b. nilai $x_{Q_2,0}$ dan $x_{Q_2,1}$ sedemikian sehingga $x_{Q_2} = x_{Q_2,0} + x_{Q_2,1} \cdot i$;
- c. nilai $x_{P_2,0}$ dan $x_{P_2,1}$ sedemikian sehingga $x_{P_2} = x_{P_2,0} + x_{P_2,1} \cdot i$;
- d. nilai $x_{R_2,0}$ dan $x_{R_2,1}$ sedemikian sehingga $x_{R_2} = x_{R_2,0} + x_{R_2,1} \cdot i$;
- e. nilai $x_{Q_3,0}$ dan $x_{Q_3,1}$ sedemikian sehingga $x_{Q_3} = x_{Q_3,0} + x_{Q_3,1} \cdot i$;
- f. nilai $x_{P_3,0}$ dan $x_{P_3,1}$ sedemikian sehingga $x_{P_3} = x_{P_3,0} + x_{P_3,1} \cdot i$;
- g. nilai $x_{R_3,0}$ dan $x_{R_3,1}$ sedemikian sehingga $x_{R_3} = x_{R_3,0} + x_{R_3,1} \cdot i$

Berikut contoh salah satu set parameter SIKE yaitu SIKEp434.

P = 0002341F 27177344 6CFC5FD6 81C52056 7BC65C78 3158AEA3
FDC1767A E2FFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFF FFFFFFFF

e2 = 000000D8

e3 = 00000089

xQ20 = 0000C746 1738340E FCF09CE3 88F666EB 38F7F3AF D42DC0B6
64D9F461 F31AA2ED C6B4AB71 BD42F4D7 C058E13F 64B237EF
7DDD2ABC 0DEB0C6C

xQ21 = 000025DE 37157F50 D75D320D D0682AB4 A67E4715 86FBC2D3
1AA32E69 57FA2B26 14C4CD40 A1E27283 EAAF4272 AE517847
197432E2 D61C85F5

yQ20 = 0001D407 B70B01E4 AEE172ED F491F4EF 32144F03 F5E054CE
F9FDE5A3 5EFA3642 A1181790 5ED0D4F1 93F31124 264924A5
F64EFE14 B6EC97E5

yQ21 = 0000E7DE C8C32F50 A4E735A8 39DCDB89 FE0763A1 84C525F7
B7D0EBC0 E84E9D83 E9AC53A5 72A25D19 E1464B50 9D97272A
E761657B 4765B3D6

xP20 = 00003CCF C5E1F050 030363E6 920A0F7A 4C6C71E6 3DE63A0E
6475AF62 1995705F 7C84500C B2BB61E9 50E19EAB 8661D25C
4A50ED27 9646CB48

xP21 = 0001AD1C 1CAE7840 EDDA6D8A 924520F6 0E573D3B 9DFAC6D1
89941CB2 2326D284 A8816CC4 249410FE 80D68047 D823C97D
705246F8 69E3EA50

yP20 = 0001AB06 6B849495 82E3F666 88452B92 55E72A01 7C45B148
D719D9A6 3CDB7BE6 F48C812E 33B68161 D5AB3A0A 36906F04
A6A6957E 6F4FB2E0

yP21 = 0000FD87 F67EA576 CE97FF65 BF9F4F76 88C4C752 DCE9F8BD
2B36AD66 E04249AA F8337C01 E6E4E1A8 44267BA1 A1887B43
3729E1DD 90C7DD2F

xR20 = 0000F37A B34BA0CE AD94F43C DC50DE06 AD19C67C E4928346
E829CB92 580DA84D 7C36506A 2516696B BE3AEB52 3AD7172A
6D239513 C5FD2516

xR21 = 000196CA 2ED06A65 7E90A735 43F3902C 208F4108 95B49CF8
 4CD89BE9 ED6E4EE7 E8DF90B0 5F3FDB8B DFE489D1 B3558E98
 7013F980 6036C5AC
 xQ30 = 00012E84 D7652558 E694BF84 C1FBDAAF 99B83B42 66C32EC6
 5B10457B CAF94C63 EB063681 E8B1E739 8C0B241C 19B9665F
 DB9E1406 DA3D3846
 xQ31 = 00000000
 yQ30 = 00000000
 yQ31 = 0000EBAA A6C73127 1673BEEC E467FD5E D9CC29AB 564BDED7
 BDEAA86D D1E0FDDF 399EDCC9 B49C829E F53C7D7A 35C3A074
 5D73C424 FB4A5FD2
 xP30 = 00008664 865EA7D8 16F03B31 E223C26D 406A2C6C D0C3D667
 466056AA E85895EC 37368BFC 009DFAFC B3D97E63 9F65E9E4
 5F46573B 0637B7A9
 xP31 = 00000000
 yP30 = 00006AE5 15593E73 97609197 8DFBD70B DA0DD6BC AEEDFDD4
 FB1E748D DD9ED3FD CF679726 C67A3B2C C12B3980 5B32B612
 E058A428 0764443B
 yP31 = 00000000
 xR30 = 0001CD28 597256D4 FFE7E002 E8787075 2A8F8A64 A1CC78B5
 A2122074 783F51B4 FDE90E89 C48ED91A 8F4A0CCB ACBFA7F5
 1A89CE51 8A52B76C
 xR31 = 00014707 3290D78D D0CC8420 B1188187 D1A49DBF A24F26AA
 D46B2D9B B547DBB6 F63A760E CB0C2B20 BE52FB77 BD2776C3
 D14BCBC4 04736AE4

BAB IV

STANDARDISASI

POST-QUANTUM CRYPTOGRAPHY



BAB IV

STANDARDISASI POST-QUANTUM CRYPTOGRAPHY

Upaya untuk melakukan standarisasi algoritma PQC telah menjadi agenda strategis di berbagai negara dan lembaga internasional guna mengantisipasi ancaman dari komputer kuantum terhadap algoritma kriptografi konvensional. Di tingkat global, *National Institute of Standards and Technology* (NIST) dari Amerika Serikat memimpin proses seleksi dan standarisasi algoritma PQC sejak tahun 2016, yang kini telah menghasilkan algoritma finalis seperti Kyber, Dilithium, dan SPHINCS+ melalui tahapan kompetisi terbuka dan tinjauan komunitas kriptografi internasional. Di Eropa, *European Telecommunications Standards Institute* (ETSI) melalui *Quantum-Safe Cryptography* (QSC) *Working Group* telah menerbitkan berbagai *Technical Report* dan *Technical Specification* yang memandu implementasi dan migrasi teknologi ke PQC dalam infrastruktur digital dan jaringan telekomunikasi. *International Organization for Standardization* (ISO/IEC JTC 1/SC 27) juga telah menginisiasi penyusunan standar-standar PQC melalui jalur teknis seperti ISO/IEC 14888-4 dan ISO/IEC 18031 yang mengadopsi algoritma berbasis *lattice* dan *hash*.

Negara-negara Asia seperti Tiongkok dan Korea Selatan juga turut aktif mengembangkan algoritma PQC nasionalnya. Tiongkok melalui *Institute of Commercial Cryptography Standards* (ICCS), yang berada di bawah *Chinese Cryptography Standardization Technical Committee*, bertanggung jawab atas pengembangan dan standarisasi algoritma kriptografi komersial, termasuk PQC. ICCS meluncurkan program *Next-generation Commercial Cryptographic Algorithms* (NGCC), yang mencakup tiga jenis, yaitu *public-key* (NGCC-PK), *hash* (NGCC-CH), dan *block cipher* (NGCC-BC). Program ini bersifat global, memanggil kontribusi internasional, dan mengevaluasi kandidat berdasarkan keamanan, performa, dan kelayakan implementasi.

Korea Selatan melakukan standarisasi PQC melalui program *Korean Post-Quantum Cryptography Competition* (KpqC) yang digagas oleh *National Intelligence Service* (NIS) dan *National Security Research Institute* (NSR). Program KpqC dimulai pada November 2022 dan berakhir pada Januari

2025, dengan pengumuman 4 (empat) algoritma PQC, yaitu 2 (dua) algoritma tanda tangan digital: AIMER dan HAETAETAE serta 2 (dua) algoritma PKE/KEM : NTRU+ dan SMAUG-T.

A. NIST

Sejak Tahun 2016, *National Institute of Standards and Technology* (NIST) sedang memulai proyek standardisasi algoritma PQC dengan nama *Post-Quantum Cryptography Standardization*. Proyek PQC yang diselenggarakan oleh NIST terdiri dari beberapa *round submission*.

1. Round 1 Submission

Round 1 Submissions bertujuan untuk mengumpulkan proposal algoritma dari komunitas kriptografi global yang dapat menggantikan atau melengkapi algoritma kriptografi saat ini (seperti RSA, ECC, dan lainnya) yang rentan terhadap serangan komputer kuantum. Pada tanggal 30 November 2017, NIST menetapkan batas akhir penerimaan proposal algoritma dari para peneliti, akademisi, dan perusahaan.

Algoritma yang diajukan harus memiliki dasar keamanan yang kuat terhadap serangan klasik maupun kuantum. Selain itu, algoritma juga harus efisien dalam komputasi dan penggunaan sumber daya serta dapat diterapkan di berbagai platform dan skenario dunia nyata. Pada tanggal 21 Desember 2017, NIST mengumumkan 69 algoritma yang berhasil lulus pada round 1 submission, yaitu: BIG QUAKE, BIKE, CFPKM, Classic McEliece, Compact LWE, CRYSTALS-DILITHIUM, CRYSTALS-KYBER, DAGS, Ding Key Exchange, DME, DRS, DualModeMS, *Edon-K, EMBLEM and R.EMBLEM, FALCON, FrodoKEM, GeMSS, Giophantus, Gravity-SPHINCS, Guess Again, Gui, HILA5, HiMQ-3, *HK17, HQC, KCL (*pka* OKCN/AKCN/CNKE), KINDI, LAC, LAKE, LEDAkem, LEDApkc, Lepton, LIMA, Lizard, LOCKER, LOTUS, LUOV, McNie, Mersenne-756839, MQDSS, NewHope, NTRUEncrypt, pqNTRUSign, NTRU-HRSS-KEM, NTRU Prime, NTS-KEM, Odd Manhattan, Ouroboros-R, Picnic, Post-quantum RSA-Encryption, Post-quantum RSA-Signature, pqsigRM, QC-MDPC KEM, qTESLA,

RaCoSS, Rainbow, Ramstake, *RankSign, RLCE-KEM, Round2, RQC, *RVB, SABER, SIKE, SPHINCS+, *SRTPI, Three Bears, Titanium, WalnutDSA.

2. *Round 2 Submission*

Pada *round 2 submission* tim pengembang algoritma diperbolehkan melakukan perbaikan dan optimalisasi terhadap proposal mereka berdasarkan masukan dari komunitas kriptografi. NIST menilai algoritma berdasarkan beberapa aspek:

- a. Keamanan, yaitu seberapa tahan algoritma terhadap berbagai jenis serangan, baik klasik maupun kuantum.
- b. Kinerja, yaitu hal mengenai kecepatan enkripsi, dekripsi, tanda tangan digital, dan verifikasi dalam berbagai platform (CPU, FPGA, mikroprosesor, dll.).
- c. Ukuran kunci dan *ciphertext* yang berpengaruh terhadap efisiensi dalam penggunaan *bandwidth* dan penyimpanan.
- d. Kompleksitas Implementasi yaitu terkait kemudahan penerapan dalam berbagai sistem.

Dari hasil penyaringan yang dilakukan pada *round 2 submission*, pada 30 Januari 2019 NIST mengumumkan 26 algoritma yang berhasil lolos dari 69 algoritma Round 1, yaitu BIKE, Classic McEliece, CRYSTALS-KYBER, FrodoKEM, HQC, LAC, LEDAcrypt (*merger of LEDAkem and LEDApkc*), NewHope, NTRU (*merger of NTRUEncrypt and NTRU-HRSS-KEM*), NTRU Prime, NTS-KEM, ROLLO (*merger of LAKE, LOCKER and Ouroboros-R*), Round5 (*merger of HILA5 and Round2*), RQC, SABER, SIKE, Three Bears, CRYSTALS-DILITHIUM, FALCON, GeMSS, LUOV, MQDSS, Picnic, qTESLA, Rainbow, SPHINCS+.

3. *Round 3 Submission*

Pada tahap ini NIST menentukan beberapa algoritma terbaik yang nantinya akan menjadi inti dari standar PQC pertama dimana diharapkan mampu melindungi informasi sensitif untuk masa depan, termasuk setelah hadirnya komputer kuantum yang kuat. Pada tanggal 22 Juli 2020, NIST mengumumkan bahwa 15

algoritma dari *Round 2* berhasil melanjutkan ke *Round 3* (7 Finalis dan 8 Alternatif).

Evaluasi yang dilakukan pada *round 3* adalah analisis terhadap keamanan berdasarkan penelitian terbaru, pengujian efisiensi implementasi dalam berbagai lingkungan (perangkat keras dan perangkat lunak), studi lebih lanjut mengenai ukuran kunci, *ciphertext*, dan performa komputasi. Finalis pada *round 3* diantaranya adalah algoritma Classic McEliece, CRYSTALS-KYBER, NTRU, SABER, CRYSTALS-DILITHIUM, FALCON, Rainbow, BIKE, FrodoKEM, HQC, NTRU Prime, SIKE, GeMSS, Picnic, dan SPHINCS+.

4. *Round 4 Submission*

Pada 5 Juli 2022, NIST mengumumkan bahwa 4 algoritma alternatif dari *Round 3* akan dievaluasi lebih lanjut dalam *Round 4*. NIST memilih empat kandidat untuk dievaluasi lebih lanjut dalam putaran ini:

- a. BIKE (*Code-based*)
- b. Classic McEliece (*Code-based*)
- c. HQC (*Code-based*)
- d. SIKE (*Isogeny-based*)

Bit-based Interactive Key Exchange (BIKE), Classic McEliece, dan *Hamming Quasi-Cyclic* (HQC) adalah algoritma kriptografi berbasis kode yang sedang dipertimbangkan dalam upaya standar kriptografi pasca-kuantum oleh NIST. BIKE menggunakan kode biner untuk pertukaran kunci yang aman, sedangkan McEliece memanfaatkan kode Goppa untuk sistem enkripsi yang tahan terhadap serangan kuantum. Di sisi lain, HQC juga berbasis kode, dengan pendekatan pada kode quasi-siklik Hamming untuk menawarkan keamanan terhadap ancaman komputasi kuantum. SIKE adalah algoritma kriptografi berbasis isogeni yang digunakan untuk pertukaran kunci. SIKE mengandalkan struktur matematika dari isogeni pada eliptik kurva supersingular. Namun, pada Agustus 2022, ditemukan bahwa SIKE memiliki kerentanan

keamanan yang signifikan dan dinyatakan tidak aman untuk digunakan.

5. Algoritma Terpilih oleh NIST

Pada tanggal 5 Juli 2022, NIST memilih tiga algoritma yang terbagi dalam dua kategori:

a. KEM – Mekanisme enkapsulasi kunci

CRYSTALS-Kyber (*Lattice-based*)

- 1) Dipilih sebagai standar utama untuk pertukaran kunci *post-quantum* karena efisiensinya dalam kecepatan dan ukuran kunci yang relatif kecil.
- 2) Menggunakan pendekatan *lattice-based cryptography*, yang dianggap aman terhadap serangan komputer kuantum.

b. *Digital Signatures* - Tanda Tangan Digital

1) CRYSTALS-Dilithium (*Lattice-based*)

- Dipilih sebagai standar utama tanda tangan digital *post-quantum*.
- Menawarkan keseimbangan yang baik antara keamanan, efisiensi, dan ukuran tanda tangan.

2) SPHINCS+ (*Hash-based*)

- Dipilih sebagai solusi tanda tangan digital berbasis *hash*, berbeda dari pendekatan *lattice-based*.
- Tidak seefisien Dilithium atau Falcon dalam ukuran tanda tangan, tetapi menawarkan fondasi keamanan yang kuat berdasarkan struktur *hash*.

6. Pengumuman tiga FIPS untuk PQC

Pada bulan Agustus 2024, NIST mempublikasikan 3 dokumen standar untuk algoritma PQC, yaitu:

a. *FIPS 203, Module-Lattice-Based Key-Encapsulation Mechanism Standard*

Standar ini berbasis algoritma CRYSTALS-Kyber, yang kini diganti namanya menjadi *Module-Lattice-Based Key-Encapsulation Mechanism (ML-KEM)*.

- b. *FIPS 204, Module-Lattice-Based Digital Signature Standard*
Standar ini menggunakan algoritma CRYSTALS-Dilithium, yang telah diganti namanya menjadi *Module-Lattice-Based Digital Signature Algorithm* (ML-DSA).
- c. *FIPS 205 Stateless Hash-Based Digital Signature Standard*
Standar ini menggunakan algoritma Sphincs+, yang kini dinamai *Stateless Hash-Based Digital Signature Algorithm* (SLH-DSA). Standar ini berbasis pendekatan matematika yang berbeda dari ML-DSA, dan ditujukan sebagai metode cadangan jika di kemudian hari ML-DSA terbukti memiliki kerentanan.

Ketiga dokumen ini telah dibahas pada BAB III. Algoritma PQC.

Selain ketiga dokumen FIPS yang telah dirilis, terdapat pula draf standar FIPS 206 yang sedang dalam proses perumusan berdasarkan algoritma FALCON. Algoritma tersebut akan dinamakan FN-DSA, yang merupakan singkatan dari *FFT (fast-Fourier transform) over NTRU-Lattice-Based Digital Signature Algorithm*.

NIST sedang melanjutkan proses standardisasi PQC ke putaran keempat. Dalam tahap ini, masih terdapat beberapa algoritma KEM yang dipertimbangkan, yaitu: BIKE, Classic McEliece, HQC, dan SIKE. Namun, tidak ada lagi kandidat skema tanda tangan digital yang tersisa dalam proses seleksi saat ini. Oleh karena itu, pada September 2022, NIST mengajukan permintaan terbuka untuk usulan skema tanda tangan digital baru untuk dipertimbangkan dalam proses standardisasi PQC.

NIST secara khusus tertarik pada skema tanda tangan digital umum yang berbasis selain *lattice*. Namun, untuk kebutuhan aplikasi tertentu seperti *certificate transparency*, NIST juga membuka kemungkinan untuk mempertimbangkan skema tanda tangan yang memiliki ukuran tanda tangan yang pendek dan proses verifikasi yang cepat.

Meski NIST tetap membuka peluang bagi usulan tanda tangan yang berbasis *lattice*, namun menekankan pentingnya diversifikasi dalam standar tanda tangan digital. Artinya, jika usulan tersebut masih menggunakan *structured lattices*, maka usulan tersebut harus menunjukkan kinerja yang jauh lebih unggul dibandingkan dengan CRYSTALS-Dilithium dan FALCON, baik dari segi aplikasi praktis maupun jaminan keamanan yang kuat, agar dapat dipertimbangkan untuk standardisasi. Batas waktu pengiriman paket usulan ke NIST adalah 1 Juni 2023. Berikut merupakan linimasa usulan algoritma tanda tangan digital.

1. *Round 1 Additional Signatures*

Berikut hasil seleksi *round 1* untuk usulan skema tanda tangan digital tambahan yang diumumkan 17 Juli 2023:

- a. *Code-based Signatures*: CROSS, Enhanced pqsigRM, FuLeeca, LESS, MEDS, Wave
- b. *Isogeny-based Signatures*: SQIsign
- c. *Lattice-based Signatures*: EagleSign, EHTv3/EHTv4, HAETAE, HAWK, HuFu, Raccoon, SQUIRRELS
- d. *MPC-in-the-Head Signatures* (Secure multiparty compute): Biscuit, MIRA, MiRiTH, MQOM, PERK, RYDE, SDitH
- e. *Multivariate Signatures*: 3WISE, Biscuit, DME-Sign, HPPC, MAYO, PROV, QR-UOV, SNOVA, TUOV, UOV, VOX
- f. *Symmetric-based Signatures*: AImeR, Ascon-Sign, FAEST, SPHINCS-alpha
- g. Skema lainnya: ALTEQ, eMLE-Sig 2.0, KAZ-SIGN, Preon, Xifrat1-Sign.I

2. *Round 2 Additional Signatures*

Berikut hasil seleksi *round 1* untuk usulan skema tanda tangan digital tambahan yang diumumkan 24 Oktober 2024:

- a. *Code-based Signatures*: CROSS, LESS
- b. *Isogeny-based Signatures*: SQIsign
- c. *Lattice-based Signatures*: HAWK

- d. *MPC-in-the-Head Signatures (Secure multiparty compute)*: Mirath (*merger MIRA/MiRitH*), MQOM, PERK, RYDE, SDitH
- e. *Multivariate Signatures*: MAYO, QR-UOV, SNOVA, UOV
- f. *Symmetric-based Signatures*: FAEST

Pada tanggal 11 Maret 2025, NIST secara resmi memilih algoritma HQC sebagai algoritma kelima dalam portofolio enkripsi tahan kuantum. HQC diusulkan sebagai *backup* untuk algoritma utama ML-KEM, yang telah ditetapkan pada 2024. HQC adalah algoritma KEM keempat yang dipilih sejak putaran keempat PQC (setelah BIKE, Classic McEliece, SIKE).

HQC berbasis pada masalah matematika *error-correcting codes*, berbeda dengan ML-KEM yang menggunakan *structured lattices*. Diversifikasi ini penting agar jika suatu saat ditemukan kelemahan pada basis matematika ML-KEM, HQC dapat menjadi cadangan algoritma yang aman.

B. ISO

Pada pertengahan tahun 2020, *International Organization for Standardization* (ISO) mulai menyusun standar untuk tanda tangan digital yang tahan terhadap serangan dari komputer kuantum. Standar terkait tanda tangan digital sebelumnya, seperti yang diatur dalam ISO/IEC 14888-2:2008 *Information technology — Security techniques — Digital signatures with appendix Part 2: Integer factorization based mechanisms*, menyatakan keamanan tanda tangan digital bergantung pada kesulitan faktorisasi bilangan bulat. Sementara itu, standar lain terkait tanda tangan digital yaitu ISO/IEC 14888-3:2018 *IT Security techniques — Digital signatures with appendix Part 3: Discrete logarithm based mechanisms*, menyatakan keamanan tanda tangan digital bergantung pada perhitungan logaritma diskrit. Namun, ketika komputer kuantum telah tersedia dan digunakan secara masif, semua teknik tersebut tidak aman untuk ukuran kunci yang praktis.

Pada Tahun 2024, ISO mempublikasikan standar baru terkait tanda tangan digital yaitu ISO/IEC 14888-4:2024 *Information security —*

Digital signatures with appendix Part 4: Stateful hash-based mechanisms, yang mengatur tanda tangan digital dengan mekanisme berbasis *hash* (*Hash-Based Signature* - HBS). Keamanan pada tanda tangan digital yang terdapat pada dokumen standar tersebut bergantung pada keamanan fungsi *hash* yang telah distandardisasi dan diyakini aman bahkan terhadap serangan yang menggunakan komputer kuantum skala besar. Standar ini mencakup empat algoritma tanda tangan digital yang telah distandardisasi sebelumnya sebagai RFC 8391 oleh IETF pada 2018 dan sebagai bagian dari standar NIST SP 800-208 tahun 2020 untuk skema tanda tangan berbasis *hash*. Di dalam standar ini memuat penjelasan terkait proses pembangkitan kunci (*key generation*), proses *signing*, proses verifikasi (*verification*), dan parameter – parameter yang ada pada setiap algoritma. Berikut penjelasan singkat mengenai keempat algoritma tersebut:

1. *eXtended Merkle Signature Scheme* (XMSS)

Skema XMSS adalah skema tanda tangan berbasis *hash* yang *stateful*. Dalam skema ini, satu kunci privat hanya bisa digunakan untuk membuat tanda tangan dalam jumlah terbatas. Keamanan XMSS didasarkan pada kesulitan masalah *Target Collision Resistance* (TCR), dan skema ini telah terbukti aman dalam model standar.

2. *XMSS Multi-Tree* (XMSS-MT)

Skema XMSS-MT, yang merupakan varian dari XMSS, adalah skema tanda tangan berbasis *hash stateful* yang mendukung jumlah tanda tangan lebih banyak dibandingkan XMSS. XMSS-MT mewarisi semua properti keamanan dari XMSS.

3. *Leighton-Micali Signature* (LMS)

Skema LMS (*Leighton-Micali Signature*) dapat digunakan untuk menandatangani sejumlah pesan dalam jumlah yang sudah ditentukan. Skema ini menggunakan tanda tangan LM-OTS (*Leighton-Micali One-Time Signature*) dan membangun struktur pohon biner berbasis *hash*. Dalam pohon ini, setiap daun berisi kunci publik dari pasangan kunci LM-OTS.

LM-OTS adalah skema tanda tangan sekali pakai yang digunakan sebagai komponen dasar dalam skema LMS dan HSS. Skema ini tidak boleh digunakan secara terpisah di luar LMS atau HSS.

4. *Hierarchical Signature System (HSS)*

Skema HSS menggunakan jumlah simpul yang sangat besar dalam sebuah pohon memerlukan biaya yang tinggi, karena semakin besar ukuran pohon, semakin besar pula upaya yang dibutuhkan untuk menghasilkan pasangan kunci. Jika dibutuhkan tanda tangan dalam jumlah yang sangat banyak, skema HSS dapat digunakan sebagai solusi.

Sejak tahun 2023, ISO juga mulai melakukan amandemen terhadap ISO/IEC 18033-2:2006 *Information technology — Security techniques — Encryption algorithms Part 2: Asymmetric ciphers*, yaitu standar internasional yang membahas algoritma kriptografi asimetris. Pada bulan Februari 2025, proses amandemen dokumen standar tersebut telah mencapai tahap konsultasi dan pemungutan suara oleh anggota ISO. Pada draf dokumen amandemen ini ditambahkan tiga algoritma KEM yang ketiganya merupakan algoritma yang didapat dari proses standarisasi PQC oleh NIST, yaitu:

1. Classic McEliece KEM

Classic McEliece KEM berasal dari algoritma McEliece (1978) yang menggunakan *code-based cryptography*. Classic McEliece KEM tidak dipilih dalam finalisasi utama NIST PQC 2022, namun tetap dipertahankan dalam daftar kandidat alternatif. Classic McEliece KEM ini terdiri dari tiga fungsi matematika, yaitu CM.KeyGen, CM.Encap (CM.Encrypt), dan CM.Decap (CM.Decrypt), untuk setiap set parameter yang ditentukan (Klausul 13.16 pada standar).

2. FrodoKEM

FrodoKEM menggunakan *lattice-based cryptography* tanpa penggunaan struktur aljabar. FrodoKEM dipilih sebagai kandidat alternatif Putaran 3 dalam proyek Standardisasi PQC NIST, tetapi tidak dipilih untuk standarisasi.

FrodoKEM menggunakan parameter berbasis *pseudorandom generator* (PRG) yang digunakan untuk menghasilkan matriks yang dilambangkan sebagai A . Berdasarkan klausul 14 standar ini memungkinkan dua varian utama, yang ditentukan berdasarkan penggunaan salah satu dari:

- a. AES128, sebagaimana didefinisikan dalam Klausul 5.2 ISO/IEC 18033-3, atau
- b. SHAKE128, sebagaimana didefinisikan dalam Lampiran C ISO/IEC 10118-3:2018, untuk menghasilkan A .

Algoritma FrodoKEM terdiri dari dua varian utama, yaitu

- a. Varian "standar", yang tidak membatasi penggunaan ulang pasangan kunci. Penggunaan FrodoKEM standar direkomendasikan untuk aplikasi di mana jumlah *ciphertext* yang dihasilkan untuk satu kunci publik diperkirakan sama dengan atau lebih besar dari 2^8 .
- b. Varian "ephemeral", yang ditujukan untuk aplikasi di mana jumlah *ciphertext* yang dihasilkan untuk setiap kunci publik relatif kecil. FrodoKEM ephemeral harus digunakan untuk aplikasi di mana jumlah *ciphertext* yang dihasilkan untuk satu kunci publik diperkirakan lebih kecil dari 2^8 .

3. ML-KEM

ML-KEM merupakan mekanisme enkapsulasi kunci berbasis modul-lattice yang dikembangkan dari CRYSTALS-Kyber. ML-KEM merupakan salah satu algoritma yang terpilih dalam *Round 3* dan resmi ditetapkan sebagai standar dalam NIST PQC *Final Selections* (2022) untuk PKE dan KEM.

C. ETSI

European Telecommunications Standards Institute (ETSI) adalah organisasi standar Eropa yang didirikan untuk mengembangkan spesifikasi dan standar dalam bidang teknologi informasi dan komunikasi (TIK). ETSI berperan penting dalam memastikan interoperabilitas, efisiensi, dan keamanan sistem serta layanan digital

yang digunakan secara luas di berbagai sektor industri dan pemerintahan. Saat ini, ETSI mulai berfokus pada isu keamanan informasi yang ditimbulkan oleh kemampuan komputer kuantum dalam memecahkan algoritma kriptografi konvensional, seperti RSA dan *Elliptic Curve Cryptography* (ECC), yang selama ini telah populer digunakan dalam sistem keamanan digital. Ancaman tersebut menimbulkan kebutuhan mendesak akan kriptografi yang tahan terhadap serangan komputer kuantum, yang dikenal dengan istilah *Quantum Safe Cryptography* (QSC) atau PQC.

Sebagai respons terhadap hal tersebut, ETSI membentuk sebuah kelompok kerja khusus yang dinamakan *Quantum Safe Cryptography (QSC) Working Group*. Kelompok kerja tersebut dibentuk dengan tujuan untuk mengevaluasi dan merekomendasikan algoritma, protokol, serta strategi implementasi yang dianggap aman terhadap ancaman komputasi kuantum. ETSI tidak menciptakan algoritma kriptografi baru terkait QSC, namun berperan untuk melakukan penilaian terhadap algoritma yang telah dikembangkan oleh komunitas akademik dan industri, dengan mempertimbangkan faktor keamanan, performa, efisiensi, dan kesiapan implementasi di lingkungan dunia nyata.

Hingga saat ini, ETSI telah menerbitkan berbagai dokumen standar terkait QSC, salah satunya adalah dokumen TR 103 619 yang memberikan panduan strategis mengenai langkah-langkah migrasi dari sistem kriptografi konvensional menuju sistem yang tahan terhadap serangan komputer kuantum. Dokumen tersebut memuat rekomendasi teknis serta pertimbangan praktis bagi organisasi yang hendak memulai transisi menuju lingkungan digital yang lebih aman terhadap risiko dari komputer kuantum di masa mendatang. Standar terkait QSC yang telah diterbitkan oleh ETSI selain dokumen TR 103 619, disajikan dalam bentuk daftar dokumen beserta ruang lingkupnya sesuai Tabel 4.1 berikut.

Tabel 4.1 Daftar Dokumen terkait QSC dari ETSI

No.	Dokumen	Ruang Lingkup
1.	ETSI EG 203 310 V1.1.1 (2016-06) : CYBER; <i>Quantum Computing Impact on security of ICT Systems; Recommendations on Business Continuity and Algorithm Selection.</i> [ETSI Guide]	Dokumen ini membahas mengenai bagaimana bisnis dapat terus berjalan di tengah ancaman dari Komputasi Kuantum yang dapat melemahkan sistem keamanan saat ini yang masih menggunakan kriptografi asimetris seperti RSA dan ECC. Dokumen ini juga menjelaskan bagaimana cara beralih ke era pasca-kuantum, termasuk memastikan kembali keandalan Otoritas Sertifikat (CA) pada Infrastruktur Kunci Publik (PKI), pendistribusian algoritma baru dan pendistribusian kunci baru. Dokumen ini juga menyarankan agar perencanaan keberlanjutan bisnis memperhitungkan dampak Komputasi Kuantum terhadap TIK. Dokumen ini terdiri dari beberapa bab, sebagai berikut: 1. Ruang lingkup 2. Referensi 3. Definisi dan Singkatan 4. Gambaran Umum Masalah 5. Pertimbangan Kelangsungan Bisnis 6. Lampiran A: Tinjauan Umum Komputasi Kuantum 7. Lampiran B: Algoritma Shor 8. Lampiran C: Algoritma Grover
2.	ETSI GR QSC 001 V1.1.1 (2016-07) : <i>Quantum-Safe Cryptography (QSC); Quantum-safe algorithmic framework.</i>	Dokumen ini memberikan gambaran tentang pemahaman terkini dan praktik baik di dunia akademik dan industri mengenai <i>Quantum-Safe Cryptography (QSC)</i>. Dokumen ini berfokus pada

No.	Dokumen	Ruang Lingkup
	[Group Report]	identifikasi dan penilaian primitif kriptografi yang telah diusulkan untuk proses pertukaran kunci dan autentikasi yang efisien, serta yang mungkin cocok untuk standardisasi ETSI, dan penggunaan selanjutnya oleh industri untuk mengembangkan solusi <i>quantum-safe</i> untuk aplikasi dunia nyata. Dokumen ini terdiri dari beberapa bab, sebagai berikut: 1. Ruang lingkup 2. Referensi 3. Singkatan 4. Primitif yang dipertimbangkan 5. Kerangka Penilaian 6. Primitif berbasis Lattice 7. Skema Multivariat 8. Primitif berbasis Kode 9. Primitif berbasis Hash 10. Primitif berbasis Isogeni 11. Ringkasan Panjang Kunci 12. Kesimpulan 13. Lampiran A: Perbandingan Ukuran Kunci Klasik 14. Lampiran B: Perbandingan Ukuran Kunci Kuantum
3.	ETSI GR QSC 003 V1.1.1 (2017-02) : <i>Quantum Safe Cryptography; Case Studies and Deployment Scenarios</i>. [Group Report]	Dokumen ini meninjau berbagai kasus penggunaan nyata dalam penerapan <i>Quantum-Safe Cryptography</i> (QSC). Secara khusus, dokumen ini membahas beberapa aplikasi umum yang saat ini menggunakan primitif kriptografi dan menguraikan beberapa hal yang perlu dipertimbangkan oleh pengembang, termasuk fitur yang mungkin perlu diubah agar mengakomodasi

No.	Dokumen	Ruang Lingkup
		<i>Quantum-Safe Cryptography</i>. Fokus utama dari dokumen ini adalah pada opsi peningkatan primitif kriptografi kunci publik untuk pertukaran kunci dan autentikasi, meskipun beberapa alternatif non kunci publik juga dibahas. Dokumen ini terdiri dari beberapa bab, sebagai berikut: 1. Ruang Lingkup 2. Referensi 3. Singkatan 4. Skenario Penerapan QSC 5. Protokol Keamanan Jaringan 6. Layanan Luring 7. <i>Internet of Things</i> 8. Komunikasi Satelit 9. Pusat Distribusi Kunci 10. Autentikasi 11. Fungsionalitas Eksotis 12. Kesimpulan 13. Lampiran A : Tabel Ringkasan
4.	ETSI GR QSC 006 V1.1.1 (2017-02) : <i>Quantum-Safe Cryptography</i> (QSC); <i>Limits to Quantum Computing applied to symmetric key sizes</i> [Group Report]	Dokumen ini menganalisis dampak keberadaan komputer kuantum terhadap primitif kriptografi simetris. Perkiraan skenario terburuk dibuat untuk memperkirakan kekuatan maksimum komputasi kuantum yang tersedia pada tahun 2050. Dari analisis tersebut disimpulkan bahwa kriptografi simetris dan fungsi <i>hash</i> dengan panjang 256-bit diperkirakan akan masih tidak dapat dipecahkan pada tahun 2050. Dokumen ini terdiri dari beberapa bab, sebagai berikut:

No.	Dokumen	Ruang Lingkup
		1. Ruang Lingkup 2. Referensi 3. Simbol dan Singkatan 4. Latar Belakang 5. Komputer Kuantum di 2050 6. Ukuran Parameter dan Kunci 7. Kesimpulan
5.	ETSI GR QSC 004 V1.1.1 (2017-03) : <i>Quantum-Safe Cryptography; Quantum-Safe threat assessment</i> [Group Report]	Dokumen ini menjelaskan pertimbangan sederhana mengenai kemungkinan serangan yang dapat terjadi saat Komputer Kuantum yang layak digunakan telah tersedia, dan mencocokkan risiko tersebut dengan kebutuhan tiap sektor bisnis, dalam rangka untuk memahami bagaimana sebaiknya teknologi kriptografi digunakan di masing-masing sektor. Informasi tersebut digunakan untuk membantu industri menentukan berapa lama waktu yang mereka miliki untuk merespons kehadiran Komputer Kuantum, agar tetap menjaga kepercayaan dan keamanan dalam operasional bisnis mereka. Dokumen ini terdiri dari beberapa bab, sebagai berikut: 1. Ruang Lingkup 2. Referensi 3. Singkatan 4. Tinjauan Umum Pendekatan untuk Asesmen Ancaman 5. Jadwal Asesmen Komputasi Kuantum 6. Penilaian Ancaman terhadap Aspek Penerapan QC 7. Masalah Khusus Industri

No.	Dokumen	Ruang Lingkup
		8. Ringkasan, Kesimpulan dan Rekomendasi
6.	ETSI TR 103 570 V1.1.1 (2017-10) : <i>CYBER; Quantum-Safe Key Exchanges</i> [<i>Technical Report</i>]	Dokumen ini membandingkan beberapa usulan <i>quantum-safe key exchange</i> dari literatur akademik. Secara khusus, dokumen ini mencakup pertukaran kunci yang didasarkan pada masalah <i>Learning with Errors</i> (LWE), Ring-LWE, dan <i>Supersingular Isogeny Diffie-Hellman</i> (SIDH), serta pertukaran kunci yang dibangun dari skema <i>key transport</i> NTRU dan Niederreiter. Dokumen ini memberikan gambaran umum tentang setiap metode pertukaran kunci, mencantumkan parameter yang diusulkan, dan memberikan perkiraan kinerja perangkat lunak pada berbagai jenis prosesor. Dokumen ini juga membahas berbagai pertimbangan terkait keamanan dan implementasi, seperti serangan aktif dan kerentanan terhadap serangan <i>side-channel</i>. Dokumen ini terdiri dari beberapa bab, sebagai berikut: 1. Ruang Lingkup 2. Referensi 3. Singkatan 4. Pertukaran kunci <i>quantum-safe</i> 5. Pertimbangan Implementasi 6. <i>Learning with Errors</i> 7. Isogeni Supersingular 8. Pertukaran kunci dari Mekanisme <i>key transport</i>

No.	Dokumen	Ruang Lingkup
		9. Kesimpulan 10. Lampiran A : Desain LWE dan pertimbangan keamanan 11. Lampiran B : Latar Belakang SIDH dan pertimbangan keamanan 12. Lampiran C : <i>Open Quantum-Safe benchmarks</i>
7.	ETSI TR 103 617 V1.1.1 (2018-09) : <i>Quantum-Safe Virtual Private Networks</i> [Technical Report]	Dokumen ini membahas mengenai persyaratan protokol yang diperlukan untuk menambahkan ketahanan terhadap serangan kuantum pada teknologi VPN. Pembahasan tersebut juga mencakup pertimbangan dari sisi klien, server dan arsitektur. Secara khusus, dokumen ini mempertimbangkan kebutuhan terkait protokol dan penetapan kunci, berdasarkan banyaknya sistem yang berisiko dan perlu diperbarui keamanannya, sebelum komputer kuantum yang mampu menyerang kriptografi komersial dikembangkan. Dokumen ini terdiri dari beberapa bab, sebagai berikut: 1. Ruang Lingkup 2. Referensi 3. Singkatan 4. Persyaratan <i>Virtual Private Network</i> (VPN) Umum 5. Persyaratan berbasis <i>Internet Key Exchange</i> (IKE) 6. Persyaratan berbasis <i>Transport Layer Security</i> (TLS) 7. Persyaratan berbasis <i>Media Access Control Security</i> (MACsec)

No.	Dokumen	Ruang Lingkup
		8. Persyaratan berbasis <i>Secure Shell</i> (SSH) 9. Kesimpulan 10. Lampiran A : Hasil Eksperimen Terkait Fragmentasi Pesan 11. Lampiran B : Dampak Protokol Tambahan
8.	ETSI TR 103 618 V1.1.1 (2019-12) : CYBER <i>Quantum-Safe Identity-Based Encryption</i>; [<i>Technical Report</i>]	Dokumen ini menjelaskan usulan skema enkripsi berbasis identitas hierarkis yang aman terhadap serangan kuantum. Dokumen ini memberikan gambaran umum tentang fungsi yang ditawarkan oleh enkripsi berbasis identitas hierarkis, menyajikan beberapa contoh kasus penggunaannya, dan menyediakan penjelasan tingkat tinggi tentang solusi potensial berdasarkan <i>structured lattice</i>. Dokumen ini terdiri dari beberapa bab, sebagai berikut: 1. Ruang Lingkup 2. Referensi 3. Definisi istilah, simbol, dan singkatan 4. <i>Identity-Based Encryption</i> (IBE) 5. <i>Lattice-based</i> HIBE 6. Pemilihan Parameter 7. Estimasi performa 8. Kesimpulan 9. Lampiran A : Latar Belakang Matematika 10. Lampiran B : Pertimbangan Implementasi 11. Lampiran C : Pertimbangan Keamanan
9.	ETSI TR 103 619 V1.1.1 (2020-07) : CYBER;	Dokumen ini membahas seputar migrasi dari lingkungan yang belum

No.	Dokumen	Ruang Lingkup
	<i>Migration strategies and recommendations to Quantum Safe schemes</i> [Technical Report]	menggunakan kriptografi <i>quantum-safe</i> menuju lingkungan yang sepenuhnya berada dalam Kondisi Kriptografi Aman-Kuantum (<i>Fully Quantum Safe Cryptographic State/FQSCS</i>). Dokumen ini juga memberikan rekomendasi dan panduan untuk memastikan proses transisi antara kedua kondisi tersebut berlangsung dengan aman. Dokumen ini terdiri dari beberapa bab, sebagai berikut: 1. Ruang Lingkup 2. Referensi 3. Definisi istilah, simbol, dan singkatan 4. Pendekatan bertahap untuk Migrasi QSC 5. Tahap 1 - Penyusunan Inventaris 6. Tahap 2 - Persiapan Rencana Migrasi 7. Tahap 3 - Eksekusi Migrasi 8. Lampiran A : Daftar Periksa Migrasi 9. Lampiran B : Pertanyaan yang sering Diajukan
10.	ETSI TS 103 744 V1.1.1 (2020-12) : <i>CYBER; Quantum-safe Hybrid Key Exchanges</i> [Technical Specification]	Dokumen ini menentukan beberapa metode untuk melakukan derivasi kunci kriptografi dari beberapa <i>shared secret</i>. <i>Shared secret</i> tersebut diperoleh melalui skema <i>key agreemet</i> klasik yang sudah ada, seperti Diffie-Hellman (ECDH) pada dokumen NIST SP800-56Ar3, dan <i>key encapsulation mechanisms</i> (KEM) <i>quantum safe</i> yang baru.

No.	Dokumen	Ruang Lingkup
		Dokumen ini terdiri dari beberapa bab sebagai berikut: 1. Ruang lingkup 2. Referensi 3. Istilah definisi, simbol dan singkatan 4. Tujuan dari <i>quantum-safe hybrid key exchange</i> 5. Arsitektur untuk <i>quantum-safe hybrid key exchange</i> 6. Informasi pendahuluan 7. Primitif kriptografi 8. Skema <i>hybrid key agreement</i> 9. Lampiran A (informatif): Latar Belakang 10. Lampiran B (informatif) : Pertimbangan keamanan 11. Lampiran C (informatif): <i>Test vector</i>
11.	ETSI TR 103 616 V1.1.1 (2021-09) : <i>CYBER; Quantum-Safe Signatures</i> [<i>Technical Report</i>]	Dokumen ini memberikan deskripsi teknis mengenai skema tanda tangan digital yang telah diajukan ke <i>National Institute of Standards and Technology</i> (NIST) untuk putaran ketiga dalam proses standardisasi <i>post-quantum cryptography</i>. Dokumen ini terdiri dari beberapa bab sebagai berikut: 1. Ruang lingkup 2. Referensi 3. Istilah definisi, simbol dan singkatan 4. Pendahuluan 5. Latar belakang 6. Finalis 7. Kandidat alternatif 8. Lampiran A: Properti keamanan

No.	Dokumen	Ruang Lingkup
		9. Lampiran B: Kerangka kerja untuk mengontruksi tanda tangan digital 10. Lampiran C: Hasil kriptanalisis terbaru 11. Lampiran D: Parameter tambahan untuk GeMSS 12. Lampiran E: Parameter untuk SPHINCS+
12.	ETSI TR 103 823 V1.1.2 (2021-10) : <i>CYBER; Quantum-Safe Public-Key Encryption and Key Encapsulation</i> [Technical Report]	Dokumen ini memberikan deskripsi teknis mengenai <i>Public-Key Encryption</i> (PKE) dan <i>Key Encapsulation Mechanisms</i> (KEM) yang telah diajukan ke <i>National Institute for Standards and Technology</i> (NIST) untuk putaran ketiga dalam proses standarisasi <i>Post-Quantum Cryptography</i> (PQC). Dokumen ini terdiri dari beberapa bab sebagai berikut: 1. Ruang lingkup 2. Referensi 3. Istilah definisi, simbol dan singkatan 4. Pendahuluan 5. Latar belakang 6. Finalis 7. Kandidat alternatif 8. Lampiran A: Pembuktian keamanan 9. Lampiran B: Properti keamanan 10. lampiran C: <i>Code-based costing methodology</i> 11. Lampiran D: <i>Lattice coasting methodology</i>
13.	ETSI TR 103 949 V1.1.1 (2023-05) : <i>Quantum-Safe Cryptography (QSC)</i>	Dokumen ini mereviu status penerapan mekanisme keamanan kriptografi dalam <i>Intelligent</i>

No.	Dokumen	Ruang Lingkup
	<i>Migration; ITS and C-ITS migration study</i> <i>[Technical Report]</i>	<i>Transport System (ITS) dan Cooperative Intelligent Transport Systems (C-ITS) serta kerentanannya terhadap serangan oleh komputer kuantum. Dokumen ini juga memberikan sejumlah rekomendasi terkait adopsi Quantum Safe Cryptography (QSC) untuk memimalkan risiko ITS dan C-ITS terhadap serangan kuantum.</i> Dokumen ini terdiri dari beberapa bab sebagai berikut: 1. Ruang lingkup 2. Referensi 3. Istilah definisi, simbol, dan singkatan 4. Reviu arsitektur dan model keamanan C-ITS 5. Penerapan ETSI TR 103 619 pada C-ITS 6. Lampiran A: Panduan migrasi untuk ketentuan QSC dalam standar ETSI ITS 7. Lampiran B: Panduan migrasi untuk ketentuan QSC dalam IEEE 1609.2 dan standar terkait 8. Lampiran C: Panduan migrasi khusus untuk model CCMS Uni Eropa 9. Lampiran D: Panduan migrasi untuk model SVI 10. Lampiran E: Panduan migrasi khusus untuk model ExVe 11. Lampiran F: Reviu sederhana tentang ITS dan C-ITS 12. Lampiran G: Daftar pustaka
14.	ETSI TR 103 966 V1.1.1 (2024-10) : <i>CYBER</i>	Dokumen ini mengeksplorasi berbagai permasalahan terkait

No.	Dokumen	Ruang Lingkup
	<i>Security (CYBER); Quantum-Safe Cryptography (QSC); Deployment Considerations for Hybrid Schemes</i> [<i>Technical Report</i>]	penggabungan algoritma tradisional dan <i>post-quantum</i> untuk membangun skema kriptografi <i>hybrid</i>. Secara spesifik, dokumen ini menelaah beberapa alasan di balik usulan dan adopsi skema <i>hybrid</i> sebagai berikut: 1. untuk pembentukan kunci maupun tanda tangan digital; 2. menjelaskan beberapa terminologi yang digunakan dalam skema <i>hybrid</i>; 3. membahas berbagai pertimbangan terkait keamanan, efisiensi, dan fleksibilitas; 4. menyoroti aspek penting dalam pemilihan kombinasi algoritma dan parameter; 5. mengeksplorasi potensi tantangan dalam penerapan dan migrasi ; serta 6. mengidentifikasi kondisi dengan skema <i>hybrid</i> perlu dihentikan untuk penggunaan algoritma <i>post-quantum</i> secara penuh Dokumen ini tidak memberikan panduan mengenai apakah skema <i>hybrid</i> sebaiknya digunakan atau tidak. Dokumen ini terdiri dari beberapa bab sebagai berikut: 1. Ruang lingkup 2. Referensi 3. Istilah definisi, simbol dan singkatan 4. Pendahuluan

No.	Dokumen	Ruang Lingkup
		5. Motivasi 6. Pertimbangan desain 7. Pertimbangan penerapan 8. Kesimpulan 9. Lampiran A: Mekanisme <i>key encapsulation</i> 10. Lampiran B: Algoritma tanda tangan digital 11. Lampiran C: Histori perubahan
15.	ETSI TR 103 965 V1.1.1 (2024-10) : <i>CYBER; Quantum-Safe Cryptography (QSC); Impact of Quantum Computing on Cryptographic Security Proofs</i> [Technical Report]	Dokumen ini bertujuan untuk memberikan gambaran tentang dampak komputasi kuantum terhadap pembuktian keamanan beberapa protokol kriptografi. Dokumen ini berfokus pada protokol kriptografi yang dapat dijalankan pada perangkat keras klasik. Dokumen ini membahas pembuktian keamanan mana yang menjadi tidak valid atau terpengaruh jika terdapat penyerang dengan akses ke <i>Cryptographically Relevant Quantum Computer (CRQC)</i>, serta membahas untuk setiap sistem yang terpengaruh apakah: 1. telah ditemukan pembuktian alternatif yang memberikan keamanan terhadap serangan kuantum, namun mungkin dengan tingkat keamanan yang lebih rendah; 2. belum ditemukan pembuktian alternatif, tetapi keamanan masih diharapkan tetap bertahan; 3. sistem kriptografi diperkirakan akan diretas oleh serangan kuantum dengan cara yang

No.	Dokumen	Ruang Lingkup
		tidak tercakup dalam pembuktian keamanan klasik, meskipun belum ada serangan kuantum konkret yang diketahui; atau 4. terdapat serangan kuantum konkret yang dapat merusak keamanan dengan cara yang tidak tercakup dalam pembuktian keamanan klasik. Dalam hal pembuktian keamanan dan permasalahan yang dipertimbangkan, dokumen ini mencakup hal-hal berikut: 1. Model kuantum <i>random oracle</i>, khususnya penggunaannya dalam: a) Transformasi Fiat-Shamir. b) Transformasi Fujisaki-Okamoto. 2. Teknik <i>rewinding</i> untuk sistem pembuktian <i>zero-knowledge</i>. 3. Properti <i>binding</i> dalam skema komitmen. 4. Kerangka kerja <i>universal-composability</i>. 5. Kerangka kerja <i>indifferentiability</i>. 6. Pembuktian keamanan fungsi semu-acak (<i>pseudo-random functions</i>). Selain menyajikan perkembangan teoritis, dokumen ini juga menguraikan konsekuensi praktisnya. Dalam beberapa kasus, keamanan skema yang aman secara klasik menjadi tidak pasti ketika menghadapi lawan kuantum.

No.	Dokumen	Ruang Lingkup
		Sementara itu, dalam kasus lain, keamanan skema tetap bertahan, tetapi parameter perlu disesuaikan untuk mempertahankan tingkat keamanan yang sama. Dokumen ini terdiri dari beberapa bab sebagai berikut: 1. Ruang lingkup 2. Referensi 3. Istilah definisi, simbol dan singkatan 4. Pembuktian keamanan kriptografi dan penyerang kuantum 5. Pendahuluan matematis 6. Teknik <i>rewinding</i> untuk pembuktian <i>zero-knowledge</i> 7. Pembuktian keamanan dalam model kuantum <i>random oracle</i> 8. Skema komitmen 9. Keamanan dalam komposisi paralel 10. Fungsi acak semu (<i>pseudo-random</i>) 11. Lampiran A: Histori perubahan
16.	ETSI TR 104 016 V1.1.1 (2024-10) : <i>CYBER; Quantum-Safe Cryptography (QSC); A Repeatable Framework for Quantum-Safe Migrations</i> [<i>Technical Report</i>]	Dokumen ini menjelaskan kerangka kerja <i>divide and conquer</i> yang dapat diterapkan secara berulang untuk memigrasikan aset keamanan informasi suatu perusahaan dari kondisi rentan terhadap kuantum ke kondisi aman terhadap kuantum berdasarkan skala prioritas. Pendekatan ini memberikan rekomendasi untuk membagi perusahaan ke dalam elemen yang terpisah. Selanjutnya, melalui berbagai analisis dalam dan antar elemen tersebut, dokumen ini

No.	Dokumen	Ruang Lingkup
		menguraikan metodologi untuk menetapkan rencana migrasi menuju keamanan kuantum bagi setiap elemen yang telah dipartisi. Dokumen ini terdiri dari beberapa bab, sebagai berikut: 1. Ruang lingkup 2. Referensi 3. Istilah definisi, simbol dan singkatan 4. Ringkasan kerangka kerja 5. Pendahuluan 6. Deskripsi kerangka kerja
17.	ETSI TR 103 967 V1.1.1 (2025-01) : <i>Cyber Security (CYBER); Quantum-Safe Cryptography (QSC); Impact of Quantum Computing on Symmetric Cryptography</i> [Technical Report]	Dokumen ini memberikan gambaran umum tentang dampak komputasi kuantum pada algoritma simetris seperti <i>block cipher</i> dan <i>hash function</i>. Dokumen ini membahas kepraktisan paralelisasi algoritma Grover, pengaruh pembatasan kedalaman sirkuit kuantum, serta beban tambahan yang ditimbulkan oleh <i>quantum error correction</i>. Dokumen ini melengkapi ETSI GR QSC 006 dengan merangkum estimasi sumber daya kuantum yang diperlukan untuk menyerang algoritma simetris yang banyak digunakan, dengan asumsi kedalaman sirkuit yang wajar. Dokumen ini juga memberikan panduan mengenai kebutuhan untuk meningkatkan panjang kunci simetris untuk berbagai skenario penggunaan yang berbeda.

No.	Dokumen	Ruang Lingkup
		Dokumen ini terdiri dari beberapa bab, sebagai berikut: 1. Ruang lingkup 2. Referensi 3. Istilah definisi, simbol dan singkatan 4. Pendahuluan 5. Algoritma Grover 6. Dampak Grover pada algoritma simetris 7. Algoritma kuantum lainnya 8. Rekomendasi 9. Lampiran A: Latar belakang koreksi eror kuantum 10. Lampiran B: Histori perubahan
18.	ETSI TS 104 015 V1.1.1 (2025-02) : <i>Cyber Security (CYBER); Quantum-Safe Cryptography (QSC); Efficient Quantum-Safe Hybrid Key Exchanges with Hidden Access Policies</i> [Technical Specification]	Dokumen ini menetapkan metode untuk membangun dan menerapkan <i>Key Encapsulation Mechanisms</i> (KEM) secara efisien dengan kebijakan akses tersembunyi, dengan memastikan bahwa privasi kunci yang dienkapsulasi dengan mengandalkan keamanan terbaik dari 2 (dua) skema hibridisasi, yaitu dengan instansiasi dimana privasi bergantung pada asumsi klasik <i>Computational Diffie-Hellman</i> (CDH) dan asumsi <i>post-quantum Learning With Errors</i> (LWE). Kedua masalah tersebut harus dipecahkan untuk dapat membahayakan privasi dari kunci yang dienkapsulasi. Dokumen ini terdiri dari beberapa bab, sebagai berikut: 1. Pendahuluan 2. Ruang lingkup 3. Referensi

No.	Dokumen	Ruang Lingkup
		4. Istilah, definisi, simbol dan singkatan 5. <i>Cryptographic primitives</i> 6. <i>Hybrid Traceable</i> KEMAC (HTKEMAC) 7. <i>Access structure</i> 8. Spesifikasi 9. Kesimpulan 10. Lampiran A Definisi keamanan untuk KEM dan KEMAC

D. Amandemen *Executive Order* 14144 oleh Presiden USA

Sebagai respons atas eskalasi ancaman siber terhadap pemerintah Amerika Serikat (AS), sektor privat, dan infrastruktur informasi kritikal, pada 6 Juni 2025 Presiden Trump menetapkan amandemen terhadap *Executive Order* (EO) 14144 (terbit pada 16 Januari 2025). EO 14144 berfokus pada kebijakan, inovasi dan standar teknis untuk memperkuat keamanan siber nasional AS. Amandemen tersebut masih menyatakan *Cryptographically Relevant Quantum Computer* (CRQC) sebagai ancaman terhadap kriptografi kunci publik. Amandemen terhadap EO 14144 menekankan pada:

1. Daftar kategori produk PQC

Daftar kategori produk PQC wajib dikeluarkan oleh *Cybersecurity and Infrastructure Security Agency* (CISA)/ *Department of Homeland Security* (DHS) dengan koordinasi *National Security Agency* (NSA). Pada amandemen EO 14114, kewajiban tersebut tetap namun terdapat tambahan tenggat waktu secara eksplisit pada 1 Desember 2025.

2. Pemicu pengadaan (*procurement trigger*) berbasis PQC

Semula, EO 14144 menetapkan “pemicu pengadaan” yang mewajibkan setiap proses pengadaan pemerintah yang terjadi 90 hari setelah suatu kategori produk tercantum dalam daftar resmi untuk mensyaratkan dukungan PQC. Melalui amandemen terbaru,

ketentuan ini dicabut sepenuhnya, sehingga tidak ada lagi kewajiban pengadaan berbasis PQC-readiness. Dengan demikian, tidak ada lagi keterkaitan antara daftar kategori produk PQC yang diterbitkan CISA dan kewajiban pembelian pemerintah.

3. Penerapan algoritma PQC atau KEM secara hibrida

EO 14144 mewajibkan instansi federal mengadopsi algoritma PQC atau KEM hibrida segera setelah teknologinya tersedia. Pada dokumen amandemen, ketentuan kewajiban tersebut dihapus seluruhnya. Akibatnya, meskipun solusi PQC atau KEM hibrida sudah siap secara teknis, instansi tidak diwajibkan untuk menerapkannya. Sehingga, adopsi dilakukan berdasarkan kebijakan internal, prioritas risiko, atau kesiapan operasional instansi masing-masing.

4. Mandat untuk adopsi TLS 1.3

EO 14144 mewajibkan NSA untuk *National Security Systems* (NSS) dan *Office of Management and Budget* (OMB) untuk sistem non-NSS, menerbitkan pedoman agar *Transport Layer Security* (TLS) 1.3 atau penerusnya diadopsi selambat-lambatnya 2 Januari 2030. Melalui amandemennya, substansi kewajiban tetap namun tenggat penerbitan pedoman diperjelas menjadi 1 Desember 2025. Dengan demikian, yang berubah adalah penegasan batas waktu administratif, sedangkan tujuan akhir adopsi TLS 1.3 (atau versi penerus) paling lambat 2030 tetap sama.

5. Keterlibatan internasional untuk PQC

EO 14144 mewajibkan pemerintah Amerika Serikat untuk dalam 90 hari mempromosikan adopsi global algoritma PQC yang distandardisasi oleh NIST. Melalui amandemen terbaru, ketentuan ini **dihapus**, sehingga koordinasi diplomatik maupun pengaturan ekspor terkait PQC tidak lagi menjadi mandat. Dengan demikian, peran advokasi internasional AS dalam mendorong migrasi PQC ditiadakan, dan percepatan adopsi di tingkat global akan lebih bergantung pada inisiatif tiap negara serta dinamika pasar dan industri.

6. Pedoman *Secure Software Development Framework* (SSDF)

Ketentuan EO 14144 yang memerintahkan pembaruan berkala SSDF (NIST SP 800-218) dan kontrol keamanan NIST SP 800-53 dengan fokus pada praktik pengembangan, keamanan, dan operasi perangkat lunak yang menunjang kesiapan PQC tetap dipertahankan sepenuhnya melalui amandemen (Pasal 2(c)). Amandemen hanya menegaskan tenggat pembaruan pada Desember 2025, tanpa mengubah arah kebijakan atau ruang lingkup teknis. Dengan demikian, jalur panduan teknis tidak mengalami perubahan: SSDF dan SP 800-53 tetap menjadi rujukan utama untuk meningkatkan *software assurance* dan tata kelola patch/pembaruan di lingkungan federal.

Secara keseluruhan, sinyal kebijakan Amerika Serikat terhadap migrasi PQC memang melemah. Penghapusan “*procurement trigger*” dan kewajiban adopsi oleh instansi federal mengurangi daya tarik dari sisi belanja pemerintah, namun sinyal teknis tetap kuat. Pengakuan akan adanya ancaman CRQC, penerbitan daftar kategori produk siap-PQC oleh CISA/NSA, dan target dukungan TLS 1.3 tetap dipertahankan.

Dampaknya di pasar, vendor global masih akan mengintegrasikan KEM/DSA berbasis PQC karena tekanan rantai pasok (*cloud, chipset, browser, sistem operasi*), tetapi tanpa paksaan pengadaan federal, adopsi di sektor yang bergantung kontrak pemerintah cenderung bergeser dari “wajib segera” menjadi “sesuai kesiapan”.

Pada level internasional, dihapusnya mandat advokasi global AS meningkatkan risiko fragmentasi. Korea dengan KpqC, Tiongkok dengan program nasionalnya, dan blok lain berpotensi mendorong profil algoritma/implementasi yang tidak identik dengan jalur NIST. Konsekuensinya adalah periode “multi-standar” yang lebih panjang dan kebutuhan *crypto-agility* yang lebih tinggi. Sejalan dengan itu, tanpa kewajiban instansi AS untuk segera menerapkan PQC atau skema KEM hibrida saat tersedia, banyak organisasi global akan mempertahankan *dual-stack* (klasik + hibrida PQC) lebih lama demi kompatibilitas.

Akibatnya, beban manajemen kunci akan bertambah, *overhead* kinerja, dan validasi keamanan. Risiko “*harvest-now, decrypt-later*” pun bertahan, sehingga organisasi yang menyimpan data bernilai panjang tetap perlu memajukan PQC pada arsip dengan retensi panjang, terlepas dari perubahan kebijakan pengadaan AS.

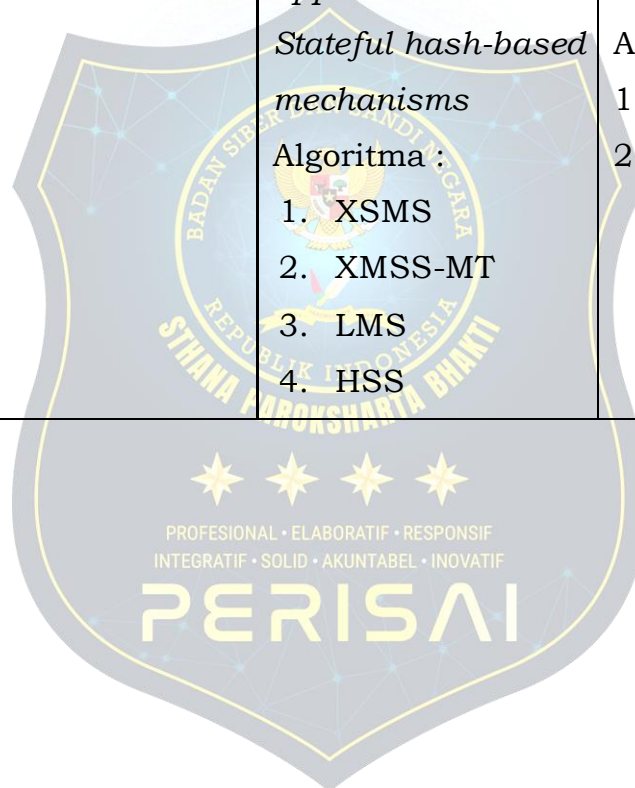
Meskipun demikian, standar NIST masih menjadi acuan utama. Pembaruan SSDF (NIST SP 800-218) dan kontrol SP 800-53 untuk *patch*/pembaruan memberi rute teknis yang jelas, dan industri besar kemungkinan tetap menjadikan ML-KEM/ML-DSA serta praktik SSDF sebagai *baseline* global. Kekosongan dorongan dari pengadaan federal kemungkinan akan diisi oleh regulator sektoral (keuangan, kesehatan, energi, telekomunikasi) dan skema label atau sertifikasi (misalnya label keamanan IoT dan profil kepatuhan industri) akan mendorong adopsi bertahap yang berbasis risiko dan kebutuhan kasus penggunaan.

Dari ketiga institusi standardisasi global yang telah melakukan proses standardisasi algoritma PQC, berikut ini merupakan resume standardisasi algoritma PQC.

Tabel 4.2 Resume Standardisasi Algoritma PQC

	NIST	ISO	ETSI
Key Encapsulation Mechanism (KEM)	FIPS 203 (ML-KEM)	Amandemen ISO/IEC 18033- 2:2006 <i>Information technology — Security techniques — Encryption algorithms Part 2: Asymmetric ciphers</i> Algoritma : 1. Classic McEliece KEM 2. FRODO KEM	ETSI TS 104 015 V1.1.1 (2025-02) : <i>Cyber Security (CYBER); Quantum-Safe Cryptography (QSC); Efficient Quantum-Safe Hybrid Key Exchanges with Hidden Access Policies</i>

	NIST	ISO	ETSI
		3. ML-KEM	Algoritma : <i>Hybrid Traceable</i> KEMAC (HTKEMAC)
<i>Digital Signature</i>	FIPS 204 (ML-DSA) FIPS 205 (SLH-DSA)	ISO/IEC 14888- 4:2024 <i>Information security — Digital signatures with appendix Part 4: Stateful hash-based mechanisms</i> Algoritma : 1. XSMS 2. XMSS-MT 3. LMS 4. HSS	ETSI TR 103 616 V1.1.1 (2021-09) : CYBER; <i>Quantum-Safe Signatures</i> Algoritma : 1. SPHINCS+ 2. GeMSS



BAB V

MIGRASI KE POST- QUANTUM CRYPTOGRAPHY



BAB V

MIGRASI KE POST-QUANTUM CRYPTOGRAPHY

Migrasi ke PQC bukan hanya tantangan teknis, tetapi juga strategis dan operasional. Proses ini memerlukan penilaian menyeluruh terhadap aset kriptografi yang ada, kesiapan sistem untuk beradaptasi, serta pemahaman mendalam terhadap algoritma baru yang direkomendasikan oleh badan standardisasi global. Pada bagian ini akan dideskripsikan mengenai perkembangan terkini terkait panduan migrasi dari berbagai lembaga internasional.

A. NIST SP 1800-38A, NIST SP 1800-38B dan NIST SP 1800C

Cybersecurity and Infrastructure Security Agency (CISA), National Security Agency (NSA) dan NIST membuat lembar fakta Quantum-Readiness: Migration to Post-Quantum Cryptography pada Agustus 2023 untuk memberi informasi kepada organisasi, terutama yang mendukung infrastruktur kritis, tentang dampak kemampuan kuantum, serta mendorong perencanaan awal untuk migrasi ke standar PQC melalui pengembangan Peta Jalan Kesiapan Kuantum. NIST juga menerbitkan set pertama standar PQC untuk melindungi dari kemampuan komputer-kuantum-yang-relevan-secara-kriptografi (cryptographically-relevant quantum computer (CRQC)) di masa depan dan berpotensi digunakan oleh pihak lawan. CRQC akan memiliki potensi untuk memecahkan sistem kriptografi kunci-publik (yang juga disebut sebagai kriptografi asimetris) yang saat ini digunakan untuk melindungi sistem informasi. Migrasi PQC yang sukses membutuhkan waktu untuk direncanakan dan dilaksanakan. CISA, NSA, dan NIST mendesak organisasi untuk mulai bersiap sekarang dengan membuat peta jalan kesiapan kuantum, melakukan inventarisasi, menerapkan penilaian dan analisis risiko, serta berinteraksi dengan vendor. Perencanaan awal sangat diperlukan karena pelaku ancaman siber bisa saja menargetkan data saat ini yang masih akan memerlukan perlindungan di masa depan (disebut juga data dengan masa kerahasiaan panjang), melalui operasi “harvest now, decrypt later”.

Banyak produk, protokol, dan layanan kriptografi yang digunakan saat ini dan bergantung pada algoritma kunci publik (misalnya, RSA, ECDH, dan ECDSA) perlu diperbarui, diganti, atau diubah secara signifikan agar menggunakan algoritma PQC yang tahan terhadap serangan kuantum. Organisasi dianjurkan untuk mempersiapkan diri secara proaktif terhadap migrasi masa depan ke produk-produk yang menerapkan standar PQC, termasuk berdiskusi dengan vendor tentang kesiapan kuantum mereka dan menerapkan langkah-langkah yang bijaksana dan terencana dalam organisasi mereka untuk mengurangi risiko dari CRQC.

Penjelasan dari setiap tahapan migrasi adalah sebagai berikut:

1. Menyusun Peta Jalan Kesiapan terhadap Kuantum

Sebagai langkah awal menuju migrasi ke PQC, organisasi dianjurkan untuk membentuk tim manajemen proyek guna merancang dan menentukan ruang lingkup transisi. Tim ini bertanggung jawab melakukan penemuan kriptografi secara proaktif untuk mengidentifikasi sistem dan aset yang masih menggunakan algoritma kriptografi rentan terhadap ancaman komputer kuantum, termasuk sistem tanda tangan digital serta pembaruan perangkat lunak dan *firmware*. Inventaris tersebut menjadi dasar bagi penilaian risiko dan prioritas migrasi. Kegiatan ini perlu melibatkan pakar pengadaan Teknologi Informasi (TI) dan Teknologi Operasional (TO) serta vendor dalam rantai pasokan, mengingat kriptografi kunci publik tersebar luas namun seringkali tidak terlihat dalam infrastruktur organisasi. Pelibatan manajer risiko keamanan siber dan privasi juga penting untuk memastikan bahwa aset paling kritis mendapatkan perhatian utama dalam upaya mitigasi risiko terhadap CRQC.

2. Menyusun Inventarisasi Kriptografi

Memiliki inventaris teknologi yang rentan terhadap kuantum dan tingkat kritikalitas data yang terkait memungkinkan organisasi untuk mulai merencanakan proses penilaian risiko dalam rangka memprioritaskan migrasi ke PQC. Inventaris kriptografi ini akan:

- a. Membantu organisasi mencapai kesiapan kuantum, kondisi di mana CRQC tidak lagi menjadi ancaman,
- b. Membantu organisasi mempersiapkan transisi ke arsitektur *zero trust*,
- c. Membantu mengidentifikasi atau mengaitkan akses luar terhadap dataset, karena data tersebut lebih terbuka dan lebih berisiko, dan
- d. Memberikan informasi untuk analisis mendatang tentang data apa yang bisa menjadi target sekarang dan didekripsi nanti saat CRQC tersedia.

Organisasi harus membuat inventaris kriptografi yang memberi visibilitas tentang bagaimana organisasi menggunakan kriptografi di sistem TI dan TO-nya. Alat penemuan kriptografi harus digunakan untuk mengidentifikasi algoritma yang rentan terhadap kuantum pada:

- a. Protokol jaringan, guna menelusuri algoritma yang rentan di protokol tersebut,
- b. Aset di sistem pengguna akhir dan server, termasuk aplikasi dan pustaka terkait, baik dalam fungsi aplikasi maupun untuk pembaruan firmware dan perangkat lunak, dan
- c. Kode kriptografi atau dependensi dalam *pipeline* pengembangan *Continues Integration* (CI)/ *Continues Delivery*(CD).

Catatan: Alat penemuan mungkin tidak dapat mengidentifikasi kriptografi tertanam yang digunakan secara internal dalam produk, sehingga menghambat proses penemuan atau dokumentasi. Organisasi sebaiknya meminta daftar kriptografi tertanam dari vendor dalam produk mereka.

Organisasi harus mencatat kapan dan di mana kriptografi yang rentan terhadap kuantum digunakan untuk melindungi dataset paling sensitif dan kritis, termasuk estimasi berapa lama dataset ini perlu dilindungi. Organisasi juga harus:

- a. Menghubungkan inventaris kriptografi dengan program yang sudah ada seperti Inventaris Aset, Manajemen Identitas dan Akses (ICAM, IdAM), Deteksi dan Respon *Endpoint* (EDR), serta Diagnostik dan Mitigasi Berkelanjutan (CDM),
- b. Memahami sistem dan protokol mana yang digunakan untuk memindahkan atau mengakses dataset paling kritis, dan
- c. Mengidentifikasi kriptografi yang rentan terhadap kuantum yang melindungi proses penting, terutama dalam Infrastruktur Kritis.

Organisasi harus mengintegrasikan inventaris ini ke dalam proses penilaian risikonya, sehingga pejabat risiko dapat memprioritaskan area mana yang harus segera diubah ke PQC saat tersedia.

3. Mendiskusikan Peta Jalan Kesiapan terhadap Kuantum dengan Penyedia Teknologi

Organisasi didorong untuk secara proaktif menjalin komunikasi dengan vendor teknologi guna memahami peta jalan kesiapan kuantum yang mereka miliki, termasuk rencana migrasi ke PQC. Peta jalan tersebut idealnya memuat rencana pengujian algoritma PQC dan tahapan integrasinya ke dalam produk, baik untuk produk *commercial-off-the-shelf* (COTS) maupun layanan berbasis *cloud*. Penerbitan peta jalan oleh vendor menunjukkan komitmen mereka terhadap implementasi PQC. Selain itu, organisasi disarankan untuk mulai menyesuaikan ketentuan kontrak, baik yang sedang berlaku maupun yang akan datang, guna memastikan produk baru dilengkapi dengan dukungan PQC dan produk lama dapat ditingkatkan sesuai dengan jadwal transisi yang telah direncanakan.

4. Menentukan Kesiapan Rantai Pasokan terhadap Kuantum

Organisasi harus memahami ketergantungan mereka terhadap kriptografi yang rentan terhadap kuantum di sistem dan aset, serta bagaimana vendor dalam rantai pasokan mereka akan melakukan migrasi ke PQC. Seperti disebutkan sebelumnya,

memahami ketergantungan ini mencakup penemuan algoritma rentan dalam sistem dan perangkat TI/TO saat ini (baik yang dibuat khusus maupun COTS), serta layanan *cloud* yang digunakan, guna memastikan bahwa strategi transisi organisasi mengurangi risiko kuantum sebisa mungkin. Organisasi juga harus mulai menanyakan kepada vendor bagaimana mereka mengatasi kesiapan kuantum dan mendukung migrasi ke PQC. Pertimbangan tambahan:

- a. Prioritaskan sistem berdampak tinggi, sistem kontrol industri, dan sistem dengan kebutuhan kerahasiaan jangka panjang.
- b. Jika ditemukan kriptografi yang rentan dalam teknologi buatan sendiri, organisasi harus mengidentifikasi risikonya terhadap data atau fungsi yang mengandalkannya. Organisasi bisa memilih untuk bermigrasi ke PQC dalam teknologi tersebut atau membangun peningkatan keamanan sistem sebagai mitigasi. Produk buatan sendiri, terutama pada sistem lama, kemungkinan besar membutuhkan upaya terbesar untuk menjadi tahan terhadap kuantum.
- c. Untuk produk COTS, keterlibatan dengan vendor mengenai peta jalan PQC mereka sangat penting. Migrasi ke PQC harus dilihat sebagai upaya modernisasi TI/TO. Peta jalan organisasi harus mencakup kapan dan bagaimana tiap vendor akan menyediakan pembaruan atau peningkatan untuk memungkinkan penggunaan PQC, serta perkiraan biaya migrasi.
- d. Untuk produk *cloud*, organisasi harus berdialog dengan penyedia layanan *cloud* mereka untuk memahami kesiapan kuantum dari pihak penyedia. Setelah standar PQC tersedia, dialog tersebut harus berfokus pada bagaimana mengaktifkan penggunaan PQC, misalnya melalui perubahan konfigurasi atau pembaruan aplikasi.

Seiring berjalannya proyek migrasi, NIST mengembangkan contoh implementasi, panduan, dan praktik yang direkomendasikan. Proyek ini juga menunjukkan contoh-contoh tersebut dalam mendukung berbagai skenario penggunaan. Temuan dari demonstrasi ini dipublikasikan dalam panduan praktik ini sebagai bagian dari publikasi khusus seri NIST 1800 yang terdiri dari beberapa volume yang ditujukan untuk topik dan audiens yang berbeda berdasarkan *workstream* yang ditentukan. Saat ini telah diterbitkan draf awal NIST *Special Publication* 1800-38 volume A, B dan C. Deskripsi singkat dari ketiga volume tersebut adalah sebagai berikut:

1. NIST SP 1800-38A

Standar NIST SP 1800-38A berjudul *Migration to Post-Quantum Cryptography: Preparation for Considering the Implementation and Adoption of Quantum Safe Cryptography Volume A: Executive Summary* [38] atau dalam Bahasa Indonesia “Migrasi ke *Post-Quantum Cryptography*: Persiapan untuk Mempertimbangkan Implementasi dan Adopsi Kriptografi Aman terhadap Ancaman Kuantum Volume A: Ringkasan Eksekutif”. Dokumen ini dirilis dalam bentuk draf awal (*preliminary draft*) pada bulan April 2023 dan telah mengalami update pada tanggal 2 Mei 2023.

Standar NIST SP 1800-38A berisi ringkasan eksekutif yang bertujuan memberikan gambaran awal tentang upaya migrasi dari algoritma kriptografi kunci publik yang rentan terhadap serangan kuantum menuju algoritma pasca-kuantum yang lebih aman. Dokumen ini memperkenalkan pendekatan berbasis risiko melalui dua jalur kerja utama: identifikasi penggunaan kriptografi dan pengujian interoperabilitas algoritma baru dalam protokol komunikasi dan perangkat keras.

2. NIST SP 1800-38B

Standar NIST SP 1800-38B berjudul *Migration to Post-Quantum Cryptography: Quantum Readiness – Cryptographic Discovery, Volume B: Approach, Architecture, and Security Characteristics of Public Key Application Discovery Tools* [39] atau dalam Bahasa

Indonesia “Migrasi ke Post-Quantum Cryptography: Kesiapan Menghadapi Kuantum – Penemuan Kriptografi, Volume B: Pendekatan, Arsitektur, dan Karakteristik Keamanan Alat Penemuan Aplikasi Kriptografi Kunci Publik”. Dokumen ini dirilis dalam bentuk draf awal (*preliminary draft*) pada bulan Desember 2023.

Tujuan dari standar ini adalah untuk mendukung proses transisi menuju PQC dengan menyediakan pendekatan sistematis dalam melakukan *cryptographic discovery*, yakni proses penemuan dan pengkajian terhadap elemen-elemen kriptografi yang sedang digunakan, khususnya dalam konteks aplikasi berbasis kunci publik. Volume B dari standar ini secara khusus membahas pendekatan metodologis, rancangan arsitektur, serta karakteristik keamanan dari alat bantu (*tools*) yang digunakan dalam proses penemuan aplikasi kriptografi, yang menjadi langkah awal dalam mempersiapkan sistem terhadap ancaman yang ditimbulkan oleh kemajuan teknologi komputasi kuantum.

Pendekatan yang diambil melibatkan dua jenis demonstrasi utama: pertama, penggunaan alat penemuan otomatis untuk mengidentifikasi keberadaan algoritma kunci publik dalam sistem; dan kedua, pengujian kinerja algoritma PQC dalam protokol yang ada. Studi kasus yang digunakan adalah ZetaMSB Inc., sebuah perusahaan menengah fiktif yang mulai merancang strategi migrasi berdasarkan audit sistem TI, pemantauan lalu lintas jaringan, dan analisis kode DevOps. Strategi ini menekankan otomatisasi proses penemuan dan penyatuan laporan dalam format standar agar dapat diintegrasikan ke dalam sistem keamanan yang ada.

Dokumen ini juga membahas algoritma-asimetris rentan terhadap serangan kuantum, termasuk RSA, ECDSA, DH, dan MQV. Sebagai gantinya, NIST sedang memfinalisasi algoritma baru seperti CRYSTALS-KYBER untuk pembentukan kunci dan CRYSTALS-Dilithium serta SPHINCS+ untuk tanda tangan digital. Untuk memudahkan transisi, diperkenalkan konsep *Cryptographic*

Bill of Materials (CBOM), yang mencatat informasi terkait aset kriptografi dalam perangkat lunak dan sistem organisasi.

Risiko yang ditimbulkan oleh komputer kuantum sangat nyata, termasuk serangan “*store now, decrypt later*” yang memungkinkan data yang dienkripsi hari ini didekripsi di masa depan oleh pelaku ancaman. Untuk itu, dokumen ini mengusulkan penggunaan kerangka kerja penilaian risiko seperti Teorema Mosca dan *Crypto Agility Risk Assessment Framework* (CARAF), serta model khusus dari sektor keuangan (CAI) dan sektor lain seperti telekomunikasi dan sistem siber-fisik.

Arsitektur yang diusulkan mencakup tiga area utama: pengembangan kode (CI/CD *pipeline*), sistem dan aplikasi operasional, serta protokol jaringan. Penekanan utama adalah pada deteksi algoritma rentan dan integrasi hasil temuan dalam sistem pengelolaan risiko organisasi. Panduan ini merupakan draf awal yang masih akan diperbarui berdasarkan masukan dari komunitas dan hasil demonstrasi lanjutan, dan ditujukan bagi organisasi menengah hingga besar serta lembaga pemerintah yang ingin bersiap menghadapi era PQC.

3. NIST SP 1800-38C

Standar NIST SP 1800-38C berjudul *Migration to Post-Quantum Cryptography - Quantum Readiness: Testing Draft Standards Volume C: Quantum-Resistant Cryptography Technology Interoperability and Performance Report* [40] atau dalam Bahasa Indonesia “Migrasi ke Post-Quantum Cryptography – Kesiapan Kuantum: Pengujian Standar Draft, Volume C: Laporan Interoperabilitas dan Kinerja Teknologi Kriptografi Tahan Kuantum”. Dokumen ini dirilis dalam bentuk draf awal (*preliminary draft*) pada bulan Desember 2023.

Tujuan dari standar ini adalah mengidentifikasi tantangan interoperabilitas dan performa yang mungkin dihadapi kriptografer saat menerapkan algoritma tahan kuantum pertama yang akan distandarkan NIST pada tahun 2024. Pengujian awal melibatkan QUIC, TLS, SSH, profil hibrida sertifikat X.509 untuk mendukung

algoritma tradisional dan *post-quantum*, serta operasi *post-quantum* dari HSM generasi berikutnya.

Pada NIST SP 1800-38C terdapat dua alur kerja yaitu Alur Kerja Penemuan Kriptografi Rentan Kuantum dan Alur Kerja Interoperabilitas dan Kinerja Algoritma PQC. Alur pertama berfokus pada identifikasi penggunaan algoritma kriptografi klasik yang rentan terhadap serangan komputer kuantum, sementara alur kedua mendemonstrasikan interoperabilitas dan performa algoritma PQC melalui implementasi langsung dalam berbagai protokol dan perangkat umum.

a. Alur Kerja Penemuan Kriptografi Rentan Kuantum

Alur kerja ini bertujuan membantu organisasi menemukan dan mencatat penggunaan algoritma kriptografi lama (seperti RSA dan ECC) yang dianggap tidak lagi aman jika komputer kuantum berkembang di masa depan. Dengan pendekatan yang terstruktur, organisasi dapat mengidentifikasi sistem atau protokol mana yang masih menggunakan algoritma tersebut. Proses ini melibatkan pemindaian jaringan, pemeriksaan kode program, dan pengaturan perangkat lunak. Hasil dari kegiatan ini menjadi dasar untuk merencanakan transisi ke PQC sekaligus menilai kesiapan infrastruktur dan sistem manajemen risiko untuk menjaga kelangsungan dan keamanan sistem informasi.

b. Alur Kerja Interoperabilitas dan Kinerja Algoritma PQC

Alur ini menjadi bagian utama dalam Volume C. NIST bersama mitra industri menguji penerapan algoritma PQC dalam berbagai protokol komunikasi dan sistem keamanan yang umum digunakan. Tujuannya adalah melihat apakah algoritma ini bisa bekerja dengan baik dan saling terhubung antar sistem (interoperabel) serta tetap efisien. Pengujian dilakukan di laboratorium, bukan di sistem produksi, dan meliputi hal-hal seperti kecepatan proses jabat tangan (*handshake*), kecocokan antar perangkat, dan masalah teknis

lain yang mungkin timbul. Hasil pengujian ini menjadi dasar penting untuk merancang standar dan produk yang siap menggunakan algoritma PQC secara luas.

Beberapa skenario pada NIST SP 1800-38C di antaranya yaitu protokol *Transport Layer Security* (TLS), protokol *Secure Shell* (SSH), modul keamanan perangkat keras (HSM), QUIC, dan X.509. Pada Testing Scope Pengujian interoperabilitas dalam konteks ini memungkinkan untuk identifikasi masalah kompatibilitas antara algoritma yang siap terhadap kuantum; penyelesaian masalah kompatibilitas dalam lingkungan yang terkendali dan nonproduksi; dan pengurangan waktu yang dihabiskan oleh masing-masing organisasi untuk melakukan pengujian interoperabilitas serupa untuk upaya migrasi mereka sendiri. Pada bagian *Testing Scope* dijelaskan juga tentang Algoritma *Post-Quantum* yang telah terpilih oleh NIST, Protokol, Standar dan *Use-Cases*, dan ada protokol, standar, dan kasus penggunaan utama yang tidak dibahas dalam pengujian awal yang diuraikan dalam dokumen NIST SP 1800-38C.

NIST telah menetapkan standar terkait PQC, yaitu FIPS 203, FIPS 204, dan FIPS 205. Standar-standar ini dirancang untuk mendukung interoperabilitas, menjamin keamanan, dan mempercepat adopsi PQC di berbagai protokol dan sistem komunikasi. Dalam pengembangannya, berbagai protokol, standar, dan skenario penggunaan telah disiapkan untuk memfasilitasi transisi menuju ekosistem yang tahan terhadap serangan kuantum. Namun demikian, terdapat beberapa protokol komunikasi yang dikecualikan dari ruang lingkup proyek ini. Protokol-protokol tersebut mencakup TLS 1.2, VPN dengan IKEv2/IPsec, *Datagram Transport Layer Security* (DTLS), dan *Message Queuing Telemetry Transport* (MQTT). Meskipun protokol ini masih banyak digunakan, pengujian

dan penerapan PQC dalam konteks tersebut tidak termasuk dalam fokus proyek ini.

Beberapa organisasi yang berkolaborasi dengan NIST dalam kegiatan penelitian dan pengembangan implementasi algoritma PQC pra-standar mencakup berbagai pelaku utama di bidang teknologi dan keamanan siber global. Di antaranya adalah Amazon Web Services (AWS), Crypto4A Technologies, CryptoNext Security, Entrust, IBM, Information Security Corporation (ISC), Keyfactor, Kudelski IoT, Microsoft, PQShield, Samsung SDS, dan wolfSSL. Selain itu, Thales turut berkontribusi melalui berbagai produknya seperti Thales Luna 7 Hardware Security Module (HSM), Thales TCT Luna T-Series HSM (untuk Pemerintah AS), Thales CipherTrust Manager untuk manajemen kunci, serta Thales High Speed Encryptors (HSEs) untuk enkripsi jaringan. Utimaco juga menjadi mitra penting dalam proyek ini dengan solusi HSM dan perlindungan data berbasis perangkat keras. Kolaborasi lintas sektor ini mencerminkan komitmen bersama untuk memastikan kesiapan kriptografi di era pascakuantum melalui pengujian interoperabilitas dan performa dalam berbagai protokol dan lingkungan sistem yang nyata.

B. ENISA PQC INTEGRATION STUDY

European Union Agency for Cybersecurity (ENISA) hingga saat ini, belum menerbitkan standar resmi khusus untuk kriptografi pasca-kuantum. ENISA lebih fokus pada memberikan panduan dan analisis mengenai perkembangan dan integrasi PQC [41].

1. Post-Quantum Cryptography: Current State and Quantum Mitigation

Laporan ini memberikan wawasan mendalam mengenai status terkini PQC dan langkah mitigasi terhadap ancaman yang ditimbulkan oleh komputer kuantum. Laporan ini dapat digunakan sebagai panduan strategis dalam mempersiapkan transisi global menuju sistem kriptografi yang aman terhadap ancaman kuantum.

Dengan menyediakan pandangan menyeluruh tentang algoritma PQC, tantangan, serta langkah mitigasi, ENISA membantu meningkatkan kesadaran dan kesiapan menghadapi era post-kuantum.

Seperti yang telah dibahas pada bab sebelumnya, algoritma PQC dapat dikategorikan menjadi lima, yaitu:

- a. Berbasis Kode (*Code-Based*): Mengandalkan kesulitan dalam decoding kode linier.
- b. Berbasis Latis (*Lattice-Based*): Fokus pada masalah matematika seperti LWE
- c. Berbasis Hash (*Hash-Based*): Cocok untuk tanda tangan digital yang sangat tahan terhadap serangan kuantum.
- d. Isogeny (*Supersingular Isogeny*): Meskipun efisien untuk kunci kecil, membutuhkan sumber daya besar.
- e. Multivariat (*Multivariate Polynomial*): Menggunakan kesulitan dalam memecahkan persamaan polinomial multivariat.

Laporan ini memberikan pembaruan tentang kompetisi NIST untuk standardisasi PQC, dengan fokus pada algoritma finalis putaran ketiga dan kandidat alternatif.

Ancaman yang ditimbulkan oleh Komputer Kuantum juga dijelaskan. Bagaimana komputer kuantum, dengan algoritma seperti Shor's dan Grover, dapat mendekripsi sistem kriptografi klasik seperti RSA, ECC, dan AES (dalam kasus ukuran kunci kecil). Sorotan pada urgensi mitigasi untuk infrastruktur kritis yang menggunakan sistem kriptografi klasik.

Langkah-Langkah Mitigasi Kuantum dan Persiapan Transisi ke PQC dapat dilakukan melalui:

- a. Mengidentifikasi aset kriptografi dalam sistem yang ada.
- b. Melakukan analisis dampak terhadap ancaman kuantum.
- c. Merancang rencana transisi menuju algoritma PQC.

Pendekatan hibrida, yaitu kombinasi algoritma klasik dan pasca-kuantum, untuk memastikan keamanan selama transisi. Strategi Multi-Fase, yang dapat diterapkan yaitu:

- a. Fase 1: Penilaian risiko dan kesiapan.
- b. Fase 2: Pengujian dan eksperimen PQC.
- c. Fase 3: Implementasi penuh algoritma PQC.

Beberapa tantangan dalam Adopsi PQC, yaitu:

- a. Kompleksitas Implementasi
Kesulitan dalam memperbarui protokol dan perangkat keras lama untuk mendukung algoritma baru.
- b. Kinerja
Sebagian besar algoritma PQC membutuhkan lebih banyak daya komputasi dan ruang memori dibandingkan algoritma klasik.
- c. Interoperabilitas
Tantangan dalam memastikan kompatibilitas antara sistem lama dan baru selama transisi.

2. *Post-Quantum Cryptography - Integration Study*

Laporan ini mengeksplorasi kebutuhan untuk merancang protokol kriptografi baru dan mengintegrasikan sistem pasca-kuantum ke dalam protokol yang ada. Laporan ini dirancang untuk memberikan wawasan teknis dan strategis bagi organisasi, penyusun kebijakan, serta pengembang teknologi dalam mendukung transisi ke era post-kuantum.

Laporan ini bertujuan untuk memberikan panduan tentang cara mengintegrasikan PQC ke dalam sistem keamanan yang ada, mengidentifikasi tantangan teknis, operasional, dan strategis dalam proses integrasi dan mengusulkan langkah mitigasi untuk memastikan transisi yang aman dan efisien.

Persiapan Integrasi PQC dilakukan dengan identifikasi Aset Kriptografi dan Analisis Risiko. Pada saat identifikasi aset, Organisasi perlu melakukan inventarisasi terhadap semua aset yang menggunakan kriptografi, termasuk protokol, perangkat keras, dan perangkat lunak. Pada analisis risiko, dilakukan penilaian potensi dampak jika algoritma klasik dikompromikan dan

pengidentifikasian prioritas dalam transisi berdasarkan risiko terhadap infrastruktur kritis.

Laporan ini menyertakan contoh bagaimana organisasi dapat memulai pengujian algoritma PQC pada skala kecil, misalnya Penggunaan PQC dalam protokol *Transport Layer Security* (TLS). Penerapan tanda tangan digital berbasis PQC dalam sistem elektronik.

C. BSI JERMAN

Federal Office for Information Security (*Bundesamt für Sicherheit in der Informationstechnik*, BSI) merupakan lembaga pemerintah Jerman yang berada di bawah Kementerian Dalam Negeri dan bertanggung jawab atas keamanan informasi nasional. Sejak didirikan pada tahun 1991, BSI berperan dalam pengembangan standar keamanan siber, menyediakan panduan terkait perlindungan data dan kriptografi, menangani ancaman siber yang dapat berdampak pada infrastruktur kritis Jerman, termasuk juga dalam menghadapi ancaman dari perkembangan komputer kuantum. Berkaitan dengan hal tersebut, pada Mei 2021 BSI telah menerbitkan dokumen berjudul “*Migration to Post-Quantum Cryptography*” [42] yang berisi rekomendasi langkah-langkah untuk beralih ke kriptografi yang tahan terhadap ancaman komputer kuantum.

BSI memandang bahwa PQC akan menjadi standar di masa mendatang dan oleh karena itu, peralihan ke algoritma yang tahan terhadap komputer kuantum harus sudah dipertimbangkan sejak saat ini. Proses migrasi harus dilaksanakan berdasarkan manajemen risiko dan dievaluasi berkelanjutan seiring perkembangan teknologi. Berkaitan dengan hal tersebut, BSI menekankan dua risiko yang perlu menjadi perhatian utama organisasi, yaitu skenario ancaman penyimpanan data untuk dibobol di masa depan (*store now – decrypt later*) dan periode migrasi yang panjang. Pada skenario *store now – decrypt later*, data yang hari ini dienkripsi dengan algoritma konvensional bisa saja dicuri dan disimpan oleh pihak penyerang untuk didekripsi di masa mendatang,

yakni ketika komputer kuantum sudah benar-benar hadir dan mampu memecahkan algoritma konvensional tersebut. Hal ini sangat berbahaya untuk data yang membutuhkan perlindungan jangka panjang seperti informasi pribadi yang sensitif atau rahasia bisnis perusahaan. Sementara itu, proses migrasi ke PQC membutuhkan waktu yang panjang bahkan bisa bertahun-tahun, terutama untuk sistem yang kompleks seperti *Public Key Infrastructure* (PKI) yang tidak bisa diubah secara instan atau untuk perangkat dengan masa pakai yang panjang. Kesiapan organisasi sangat bervariasi dan adopsi teknologi baru juga membutuhkan waktu. Jika migrasi tidak dimulai sejak dini, maka akan sulit menyelesaikannya saat komputer kuantum benar-benar hadir. Dua hal tersebut menjadi fokus perhatian BSI dalam mendorong kesiapan organisasi untuk bermigrasi ke PQC karena menyangkut perlindungan jangka panjang terhadap data dan kesiapan jangka panjang atas sistem.

Adapun langkah-langkah yang direkomendasikan oleh BSI untuk beralih ke PQC berdasarkan dokumen "*Migration to Post Quantum Cryptography*" adalah sebagai berikut:

1. Menerapkan prinsip *Cryptoagility*

BSI menekankan pentingnya *cryptoagility*, yaitu kemampuan sistem untuk mengganti mekanisme kriptografi secara fleksibel seiring berkembangnya ancaman dan teknologi. Dengan menerapkan prinsip *cryptoagility*, maka dimungkinkan untuk menerapkan algoritma baru tanpa melakukan perubahan besar pada infrastruktur.

2. Penggunaan *Hash-Based Signatures* untuk pembaruan *Firmware*

BSI merekomendasikan penggunaan *Stateful Hash-Based Signatures* untuk pembaruan firmware. Meski *Stateful Hash-Based Signatures* memiliki keterbatasan dalam hal dependensi *state* yang berdampak pada terbatasnya jumlah tanda tangan yang dapat dihasilkan, namun masih dapat memenuhi kebutuhan tanda tangan untuk pembaruan firmware. Terlebih, *Stateful Hash-Based*

Signatures sudah dianggap sebagai algoritma tanda tangan yang tahan terhadap komputer kuantum.

3. Mempertimbangkan Ukuran Panjang Kunci untuk Algoritma Simetris

Meski algoritma simetris lebih tahan terhadap serangan kuantum, algoritma Grover tetap dapat mempercepat *brute force*. Oleh karena itu, disarankan penggunaan panjang kunci 256-bit untuk data yang membutuhkan perlindungan jangka panjang.

4. Menerapkan Langkah Proteksi Jangka Pendek

Untuk mitigasi awal terhadap potensi ancaman komputer kuantum sebelum algoritma PQC diadopsi secara luas, terdapat beberapa mekanisme perlindungan sementara yang direkomendasikan oleh BSI, meliputi:

- a. menambahkan *pre-distributed symmetric long-term key* dalam proses pertukaran kunci untuk meningkatkan ketahanan terhadap serangan komputer kuantum. *Pre-distributed symmetric long-term key* tersebut berperan sebagai elemen tambahan dalam proses derivasi *session key*;
- b. melindungi pertukaran kunci asimetris dengan enkripsi simetris menggunakan *pre-distributed secret*; dan/atau
- c. menerapkan parameter kurva yang dirahasiakan ke dalam kriptografi kurva eliptik agar lebih tahan terhadap serangan komputer kuantum.

5. Menggunakan Solusi Hibrida (*Hybrid Solutions*)

BSI sangat merekomendasikan penggunaan solusi hibrida yakni penggabungan algoritma klasik dengan PQC dengan tujuan untuk menjaga keamanan jika salah satu algoritma terbukti rentan. Penerapan solusi hibrida direkomendasikan hingga algoritma PQC sudah lebih matang baik dalam implementasi maupun analisisnya.

6. Melakukan Adaptasi pada Protokol Kriptografi

Migrasi ke PQC menuntut penyesuaian protokol kriptografi yang ada, seperti TLS dan IKEv2 yang saat ini sudah tersedia rancangan

awal penyesuaiannya. BSI menekankan bahwa adaptasi protokol dapat dimulai tanpa menunggu finalisasi semua algoritma PQC.

7. Implementasi Algoritma PQC untuk *Key Agreement*

Dalam konteks menghadapi ancaman komputer kuantum, perlindungan terhadap pertukaran kunci kriptografi (*key agreement*) memiliki prioritas yang lebih tinggi jika dibandingkan dengan tanda tangan digital. Hal ini disebabkan karena algoritma *key agreement* memiliki peran besar dalam menjaga kerahasiaan data jangka panjang yang rentan terhadap skenario serangan *store now, decrypt later*. BSI merekomendasikan dua algoritma untuk *key agreement* dalam solusi hibrida yakni FrodoKEM dan Classic McElice. Pada akhir 2019, BSI menyatakan bahwa kedua algoritma tersebut layak digunakan dalam solusi hibrida tanpa menunggu hasil akhir proses standardisasi oleh NIST.

Sebagai bentuk komitmen internasional, pada November 2024, BSI bersama dengan mitra dari 17 negara Uni Eropa lainnya menerbitkan pernyataan bersama (*joint statement*) dalam publikasi berjudul "*Securing Tomorrow, Today: Transitioning to Post-Quantum Cryptography*"[43]. Pernyataan bersama tersebut mendorong industri, penyedia infrastruktur kritis dan administrasi publik untuk memulai transisi ke PQC, mengingat terdapat dua risiko yang perlu menjadi fokus perhatian yakni *store now decrypt later* dan lamanya waktu yang dibutuhkan untuk migrasi pada sistem kompleks seperti *Public Key Infrastructure* (PKI). Dalam pernyataan bersama tersebut, BSI dan mitranya menekankan langkah-langkah yang diperlukan untuk migrasi ke PQC meliputi:

- a. melakukan analisis ancaman kuantum melalui inventarisasi aset dan sistem kriptografi yang digunakan;
- b. menyusun roadmap transisi berbasis risiko;
- c. merencanakan proses migrasi yang mencakup alokasi anggaran dan pelibatan seluruh proses bisnis; dan
- d. mendorong riset dan standardisasi PQC lebih lanjut.

BSI dan 17 negara anggota Uni Eropa juga berkomitmen untuk memastikan kesiapan yang memadai hingga akhir 2030, mengembangkan perencanaan transisi yang komprehensif untuk infrastruktur kunci publik hingga akhir 2030, serta berkolaborasi dalam melindungi sistem TI dari ancaman komputer kuantum. Sebagai bagian dari komitmen tersebut dan tindak lanjut dari rekomendasi Komisi Eropa, dibentuk sebuah *Work Stream* dalam kerangka kerja NIS *Cooperation Group* yang diketuai oleh Perancis, Jerman, dan Belanda, untuk mendorong partisipasi aktif seluruh anggota Uni Eropa dalam mengembangkan roadmap transisi menuju PQC di tingkat Uni Eropa.

D. AIVD – TNO – CWI Belanda

Panduan ini [44] memberikan panduan praktis dan rekomendasi bagi organisasi untuk mengurangi risiko yang ditimbulkan oleh komputer kuantum terhadap sistem kriptografi saat ini. Edisi ini merupakan pengembangan dari publikasi sebelumnya yang dirilis pada Maret 2023, mencakup pembaruan berdasarkan perkembangan terbaru serta tambahan materi baru. Waktu pasti kapan komputer kuantum akan mampu melemahkan kriptografi yang ada sulit untuk diprediksi. Namun, potensi dampaknya cukup signifikan sehingga beberapa organisasi perlu segera memulai langkah mitigasi. Hal ini terutama berlaku bagi organisasi yang mengelola data yang harus tetap rahasia selama beberapa dekade atau yang membangun sistem jangka panjang yang masih akan digunakan dalam waktu lama.

Pendekatan yang paling komprehensif untuk menghadapi ancaman kuantum dikenal sebagai PQC. PQC dapat diterapkan pada sistem dan komputer yang ada saat ini, sekaligus memberikan perlindungan terhadap serangan dari komputer kuantum. Karena keterbatasan bawaan serta tantangan praktis dan keamanan tertentu, banyak lembaga keamanan saat ini secara resmi merekomendasikan hanya PQC sebagai langkah mitigasi terhadap ancaman kuantum.

Migrasi dari kriptografi yang rentan terhadap kuantum ke PQC akan menjadi tugas yang memakan waktu dan sumber daya yang besar.

Berdasarkan migrasi sebelumnya, proses ini dapat memakan waktu lebih dari lima tahun. Baru-baru ini, pada bulan Agustus 2024, standar PQC pertama diterbitkan oleh Institut Standar dan Teknologi Nasional AS (NIST), yang menandai fase berikutnya dalam migrasi PQC. Berbagai organisasi telah memulai migrasi PQC dan badan-badan regulasi di seluruh dunia sedang mempersiapkan dan menerapkan undang-undang terkait PQC. Ketersediaan standar PQC akan memungkinkan lebih banyak organisasi untuk memulai migrasi PQC mereka. Buku panduan ini mengikuti pendekatan tiga langkah untuk mengurangi ancaman kuantum: (1) Diagnosis Kerentanan Kuantum, (2) Perencanaan, dan (3) Eksekusi.

1. Diagnosis Kerentanan Kuantum

Langkah pertama, Diagnosis Kerentanan Kuantum, mencakup serangkaian tindakan “tanpa penyesalan” yang akan meningkatkan ketahanan siber organisasi, terlepas dari ancaman kuantum. Langkah-langkah ini membantu organisasi dalam mengelola kriptografi dengan lebih efektif dan mempermudah transisi sistem kriptografi. Pengelolaan kriptografi yang baik memungkinkan identifikasi dan mitigasi risiko dengan lebih cepat serta mengurangi waktu respons saat terjadi insiden, bahkan jika insiden tersebut tidak terkait dengan komputer kuantum.

Secara lebih spesifik, setiap organisasi disarankan untuk memulai dengan pendataan aset kriptografi guna membuat inventarisasi dari seluruh sistem kriptografi yang digunakan. Selain itu, organisasi perlu segera melakukan penilaian risiko kuantum dan mengintegrasikannya ke dalam prosedur manajemen risiko yang sudah ada.

a. PQC Persona

Sebelum memulai migrasi ke PQC, organisasi harus menentukan kapan waktu yang tepat untuk memulai proses tersebut. Untuk membantu pengambilan keputusan ini dan memenuhi beragam kebutuhan organisasi dalam migrasi PQC,

panduan ini mengelompokkan organisasi ke dalam beberapa kategori yang disebut PQC persona.

Pendekatan ini memiliki dua manfaat utama. Pertama, membantu mengidentifikasi organisasi yang perlu segera memulai migrasi dan organisasi yang masih dapat menundanya untuk sementara waktu. Kedua, memungkinkan pemberian rekomendasi yang lebih spesifik dan sesuai bagi organisasi dengan struktur yang serupa.

Panduan ini merancang berbagai langkah tindakan konkret untuk setiap PQC persona, yang berbeda dalam hal urgensi, timeline, analisis risiko, tingkat perhatian yang diperlukan, dan faktor lainnya. Untuk membedakan setiap kategori, digunakan karakteristik berikut:

- 1) *Attack surface*
- 2) Tipe Sistem
- 3) Tipe Data
- 4) *Time pressure*
- 5) Dependensi dengan organisasi lain
- 6) Level Ancaman

PQC persona dapat dikelompokkan menjadi 3 kategori, yaitu:

- 1) *Urgent Adopters*

Urgent Adopters harus segera memulai migrasi karena mereka menangani data yang harus tetap rahasia selama bertahun-tahun atau sistem yang berumur panjang (misalnya, sektor pemerintah, kesehatan, keuangan, dan infrastruktur kritis).

- 2) *Regular Adopters*

Regular Adopters adalah Organisasi yang dapat menunggu perkembangan lebih lanjut tetapi harus mulai mempersiapkan rencana migrasi.

- 3) *Cryptographic Expert*

Cryptographic Expert merupakan penyedia layanan atau produk kriptografi yang harus memastikan kompatibilitas dengan standar PQC terbaru.

b. Manajemen Aset Kriptografi

- 1) Membuat daftar semua algoritma kriptografi yang digunakan, termasuk protokol komunikasi (TLS, SSH, IPsec, dll.).
- 2) Mengidentifikasi sistem yang menggunakan kriptografi dan menentukan umur operasionalnya.
- 3) Menganalisis dependensi vendor terkait dengan penyedia layanan atau perangkat keras yang digunakan.

c. Asesmen Risiko Kuantum

"*Quantum Risk Assessment*" bertujuan untuk menilai apakah suatu aset kriptografi berisiko terhadap serangan komputer kuantum serta menilai dampak jika aset kriptografi dikompromikan, baik dari segi keuangan, hukum, maupun operasional.

Akhirnya, setiap organisasi harus meninjau dan memperbarui kebijakan kriptografi mereka sesuai dengan perubahan regulasi yang berlaku. Informasi ini akan memberikan wawasan tentang pendekatan yang harus diambil dalam migrasi ke PQC. Dengan memiliki informasi ini sejak awal, organisasi dapat menghindari migrasi yang tergesa-gesa dan rentan kesalahan, yang berpotensi menimbulkan biaya serta risiko yang tidak perlu di masa mendatang.

2. Perencanaan

Langkah kedua, fase Perencanaan, penting untuk membentuk tim khusus yang bertanggung jawab dalam mengawasi proses migrasi serta memastikan bahwa seluruh proses bisnis telah siap untuk mendukung transisi yang lancar. Panduan ini memberikan perhatian khusus kepada "pengadopsi mendesak", yaitu organisasi yang harus segera memulai migrasi ke PQC karena dampak kebocoran data akibat serangan kriptografi dalam beberapa dekade

mendatang dianggap tidak dapat diterima. Selain itu, panduan ini juga menyediakan *tools* untuk menilai kesiapan dan tingkat kematangan suatu organisasi dalam melakukan migrasi ke PQC.

Dari sisi teknis, ada berbagai pilihan dalam penerapan PQC, dan tidak semua solusi PQC cocok untuk setiap skenario penggunaan. Panduan ini menawarkan panduan konkret dalam merancang strategi implementasi PQC dengan mempertimbangkan berbagai kebutuhan aplikasi. Penerapan PQC juga dapat memerlukan perangkat keras baru atau pergantian ke penyedia layanan yang mendukung solusi PQC yang sesuai.

Setelah mengidentifikasi risiko, organisasi perlu membuat rencana strategis untuk migrasi PQC.

a. Membentuk Tim Migrasi

Tim ini harus mencakup berbagai disiplin ilmu, termasuk keamanan siber, arsitektur TI, tim hukum, dan manajemen risiko. Koordinasi dengan vendor diperlukan untuk memastikan kompatibilitas PQC dengan sistem yang digunakan.

b. Penyusunan Rencana Implementasi

Penyusunan rencana bertujuan untuk menentukan prioritas migrasi berdasarkan tingkat risiko dan urgensi serta mengidentifikasi kebutuhan perangkat keras dan perangkat lunak yang mendukung PQC dan juga aplikasi dan infrastruktur yang perlu diperbarui atau diganti.

c. Estimasi Biaya dan Sumber Daya

Tahapan ini bertujuan untuk menghitung biaya pengembangan, implementasi, dan pemeliharaan PQC serta Memastikan anggaran yang cukup tersedia untuk mendukung proses transisi.

d. Menyiapkan Mekanisme Kecepatan Adaptasi

Memilih solusi yang memungkinkan pergantian algoritma dengan mudah tanpa harus melakukan perubahan besar pada sistem.

e. Menentukan Waktu Migrasi

Memilih waktu yang optimal untuk migrasi, misalnya saat pembaruan infrastruktur atau perangkat lunak secara berkala. Migrasi mendadak sangat dihindari karena dapat meningkatkan risiko kesalahan dan gangguan operasional.

3. Eksekusi

Fase terakhir dalam migrasi PQC adalah tahap Eksekusi. Pada tahap ini, organisasi harus berhati-hati agar tidak menimbulkan kerentanan baru dalam sistem mereka. Panduan ini memberikan panduan tentang cara melaksanakan migrasi untuk berbagai jenis kriptografi serta strategi yang telah dirancang selama fase perencanaan.

a. Penerapan Strategi Migrasi

Terdapat beberapa strategi yang bisa digunakan:

- "Big Bang" Approach: Migrasi secara menyeluruh dalam waktu singkat (berisiko tinggi, tetapi lebih cepat).
- "Phased Approach": Migrasi secara bertahap, dimulai dari sistem yang paling kritis (lebih aman, tetapi memakan waktu lebih lama).
- "Hybrid Approach": Menggunakan kombinasi kriptografi klasik dan PQC selama masa transisi.

b. Pengujian dan Validasi

- Melakukan uji coba PQC pada sistem sebelum implementasi penuh.
- Menggunakan "shadow deployment" untuk menjalankan PQC secara paralel dengan sistem lama untuk mendeteksi potensi masalah.
- Mengidentifikasi dampak terhadap performa sistem, termasuk latency dan kebutuhan komputasi tambahan.

c. Implementasi PQC dalam Infrastruktur yang Ada

- Mengganti algoritma yang rentan terhadap kuantum dengan standar PQC yang telah disertifikasi oleh NIST.

- Memperbarui perangkat lunak dan perangkat keras yang tidak mendukung algoritma PQC.
 - Menyesuaikan kebijakan enkripsi dalam organisasi agar selaras dengan standar PQC.
- d. Manajemen Risiko dan Respons Insiden
- Memonitor implementasi PQC untuk memastikan tidak ada kerentanan baru yang muncul.
 - Menyiapkan strategi rollback, jika terjadi kesalahan dalam proses migrasi.
 - Melakukan audit berkala untuk menyesuaikan dengan perkembangan teknologi PQC terbaru.
- e. Edukasi dan Pelatihan Tim
- Memberikan pelatihan kepada tim keamanan TI tentang penerapan dan pengelolaan PQC.
 - Mensosialisasikan perubahan kepada seluruh *stakeholder* yang terlibat dalam proses migrasi.

Kriptografi akan terus berkembang di masa depan. Misalnya, algoritma baru dapat ditemukan, kerentanan baru mungkin terungkap, atau kemajuan dalam kriptanalisis bisa saja menuntut penggunaan kunci kriptografi yang lebih besar. Oleh karena itu, penting bagi organisasi untuk membangun atau mempertahankan kelincahan kriptografi (*cryptographic agility*).

Kelincahan kriptografi memungkinkan organisasi untuk dengan cepat menyesuaikan atau mengganti algoritma kriptografi yang digunakan tanpa menyebabkan gangguan besar pada proses operasional. Hal ini menjadi sangat krusial ketika terjadi peningkatan dalam protokol kriptografi atau ditemukan kerentanan baru. Panduan ini memberikan panduan tentang cara mengintegrasikan kelincahan kriptografi ke dalam proses manajemen perubahan yang sudah ada.

BAB VI

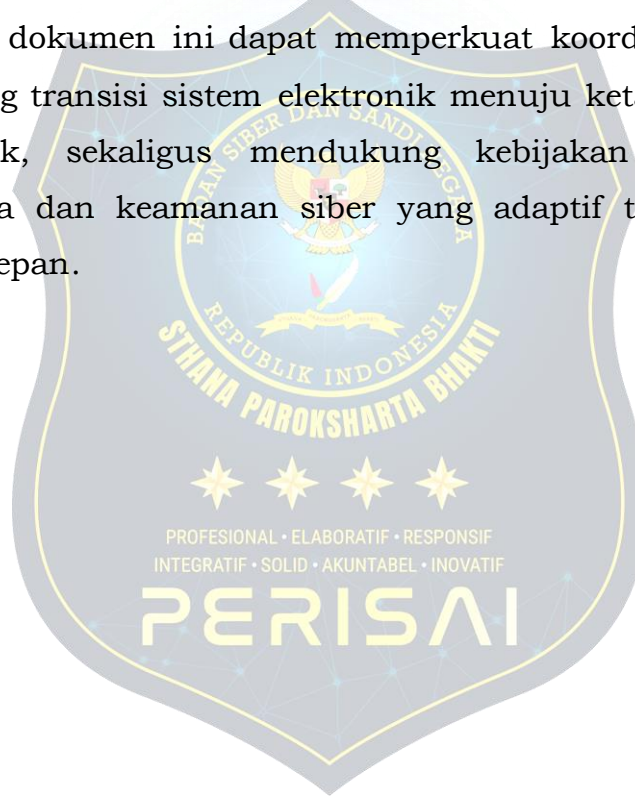
PENUTUP



BAB VI PENUTUP

Dalam menghadapi tantangan keamanan informasi era komputasi kuantum, adopsi dan integrasi PQC menjadi suatu keniscayaan. Dokumen ini disusun sebagai referensi strategis bagi para pemangku kepentingan, termasuk pengambil kebijakan, perumus kebijakan, praktisi keamanan siber dan sandi, serta penyedia layanan dan produk keamanan teknologi informasi, dalam memahami dan mengimplementasikan mekanisme kriptografi yang tahan terhadap ancaman kuantum.

Diharapkan dokumen ini dapat memperkuat koordinasi lintas sektor dalam mendorong transisi sistem elektronik menuju ketahanan kriptografi yang lebih baik, sekaligus mendukung kebijakan nasional dalam perlindungan data dan keamanan siber yang adaptif terhadap dinamika teknologi masa depan.



REFERENSI

- [1] C. Paar, J. Pelzl, and T. Guneysu, *Understanding Cryptography: From Established Symmetric and Asymmetric Ciphers to Post-Quantum Algorithms*, 2nd ed. Cham, Switzerland: Springer, 2024.
- [2] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. Boca Raton, FL, USA: CRC Press, 1996.
- [3] D. J. Bernstein and T. Lange, “Post-quantum cryptography,” *Nature*, vol. 549, pp. 188–194, 2017.
- [4] National Institute of Standards and Technology (NIST), “What is Post-Quantum Cryptography?,” [Online]. Tersedia: <https://www.nist.gov/cybersecurity/what-post-quantum-cryptography> [diakses: 25 Juni 2025 pukul 09.00 WIB].
- [5] U. S. Reddy, *Quantum Computing and Post-Quantum Cryptography: Secure and Efficient Algorithms*. 2021.
- [6] D. Micciancio and C. Peikert, *A Decade of Lattice Cryptography, Foundations and Trends in Theoretical Computer Science*, vol. 10, no. 4, pp. 283–424, Now Publishers Inc., 2016.
- [7] V. Lyubashevsky, C. Peikert, and O. Regev, “On Ideal Lattices and Learning with Errors Over Rings,” *Cryptography ePrint Archive*, Paper 2012/230, 2012. [Online]. Available: <https://eprint.iacr.org/2012/230> [diakses: 25 Juni 2025 pukul 09.30 WIB].
- [8] National Institute of Standards and Technology, “NIST releases first 3 finalized Post-Quantum Encryption Standards,” Aug. 2024. [Online]. Tersedia: <https://www.nist.gov/news-events/news/2024/08/nist-releases-first-3-finalized-post-quantum-encryption-standards> [diakses: 25 Juni 2025 pukul 10.00 WIB].
- [9] National Institute of Standards and Technology (NIST), *FIPS 203: Module-Lattice-Based Key-Encapsulation Mechanism Standard*. Gaithersburg, MD, USA: NIST, 2024. [Online]. Tersedia: <https://csrc.nist.gov/pubs/fips/203/final> [diakses: 25 Juni 2025 pukul 10.30 WIB].

- [10] PQCrystals Project, “CRYSTALS–Dilithium,” [Online]. Tersedia : <https://pq-crystals.org/dilithium/> [diakses: 24 Juni 2025 pukul 10.00 WIB].
- [11] National Institute of Standards and Technology (NIST), *FIPS 204: Module Lattice-Based Digital Signature Algorithm (ML-DSA)*. U.S. Department of Commerce, 2024. [Online]. Tersedia: <https://doi.org/10.6028/NIST.FIPS.204> [diakses: 24 Juni 2025 pukul 10.30 WIB].
- [12] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA, USA: W. H. Freeman, 1979.
- [13] J. Patarin, “Cryptanalysis of the Matsumoto and Imai public key scheme of Eurocrypt’88,” in *Advances in Cryptology — EUROCRYPT ’95*, Lecture Notes in Computer Science, vol. 921, Springer, Berlin, Heidelberg, 1995, pp. 248–261.
- [14] J. Patarin, “The oil and vinegar signature scheme,” in *Workshop on Cryptography — Dagstuhl Seminar Proceedings*, Dagstuhl, Germany, 1996.
- [15] A. Kipnis and A. Shamir, “Cryptanalysis of the Oil and Vinegar signature scheme,” in *Advances in Cryptology — CRYPTO ’98*, Lecture Notes in Computer Science, vol. 1462, Springer, Berlin, Heidelberg, 1998, pp. 257–266.
- [16] J. Patarin, L. Goubin, and N. Courtois, “Unbalanced Oil and Vinegar signature schemes,” in *Advances in Cryptology — EUROCRYPT ’99*, Lecture Notes in Computer Science, vol. 1592, Springer, Berlin, Heidelberg, 1999, pp. 206–222.
- [17] J. Ding and D. Schmidt, “Rainbow, a new multivariable polynomial signature scheme,” in *Applied Cryptography and Network Security (ACNS 2005)*, Lecture Notes in Computer Science, vol. 3531, Springer, Berlin, Heidelberg, 2005, pp. 164–175.
- [18] J. Ding et al., “Rainbow – Algorithm Specification and Documentation: The 3rd Round Proposal, Type: Signature Scheme, Family: Multivariate

Cryptography, Single-Field Schemes,”.2020.

<https://www.pqc rainbow.org>

- [19] W. Beullens, “*Breaking Rainbow takes a weekend on a laptop*”, *Cryptology ePrint Archive*, Paper 2022/214, 2022. [Online]. Tersedia: <https://eprint.iacr.org/2022/214> [diakses: 24 Juni 2025 pukul 11.00 WIB].
- [20] National Institute of Standards and Technology, *FIPS 205: Stateless Hash-Based Digital Signature Standard (SLH-DSA)*, Gaithersburg, MD, USA, Aug. 13, 2024. [Online]. Tersedia: <https://doi.org/10.6028/NIST.FIPS.205> [diakses: 24 Juni 2025 pukul 11.30 WIB]
- [21] European Union Agency for Cybersecurity (ENISA), *Post-Quantum Cryptography: Current State and Quantum Mitigation*. Heraklion, Greece: ENISA, 2021. [Online]. Tersedia: <https://www.enisa.europa.eu/publications/post-quantum-cryptography-current-state-and-quantum-mitigation> [diakses: 24 Juni 2025 pukul 11.40 WIB].
- [22] W. Castryck and T. Decru, “*An efficient key recovery attack on SIDH*,” *Cryptology ePrint Archive*, Paper 2022/975, 2022. [Online]. Tersedia: <https://eprint.iacr.org/2022/975>
- [23] L. Maino and C. Martindale, “*An attack on SIDH with arbitrary starting curve*,” *Cryptology ePrint Archive*, Paper 2022/1026, 2022. [Online]. Tersedia: <https://eprint.iacr.org/2022/1026> [diakses: 24 Juni 2025 pukul 11.55 WIB].
- [24] D. Robert, “*Breaking SIDH in polynomial time*,” *Cryptology ePrint Archive*, Paper 2022/1038, 2022. [Online]. Available: <https://eprint.iacr.org/2022/1038> [diakses: 24 Juni 2025 pukul 13.00 WIB]
- [25] T. Moriya, “*Masked-degree SIDH*,” *Cryptology ePrint Archive*, Paper 2022/1019, 2022. [Online]. Tersedia: <https://eprint.iacr.org/2022/1019> [diakses: 24 Juni 2025 pukul 13.05 WIB].

- [26] T. B. Fouotsa, “*SIDH with masked torsion point images*,” *Cryptology ePrint Archive*, Paper 2022/1054, 2022. [Online]. Tersedia: <https://eprint.iacr.org/2022/1054> [diakses: 24 Juni 2025 pukul 13.15 WIB].
- [27] C. Costello, P. Longa, M. Naehrig, J. Renes, and D. Urbanik, “Efficient compression of SIDH public keys,” in *Advances in Cryptology – EUROCRYPT 2017: 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Paris, France, 2017, pp. 679–706.
- [28] C. Costello, P. Longa, M. Naehrig, J. Renes, and F. Virdia, “Improved classical cryptanalysis of the computational supersingular isogeny problem,” *Cryptology ePrint Archive*, Paper 2019/298, 2019. [Online]. Tersedia: <https://eprint.iacr.org/2019/298> [diakses: 24 Juni 2025 pukul 13.30 WIB].
- [29] M. Naehrig and J. Renes, “Dual isogenies and their application to public-key compression for isogeny-based cryptography,” in *Advances in Cryptology – ASIACRYPT 2019: 25th International Conference on the Theory and Application of Cryptology and Information Security*, Kobe, Japan, 2019, pp. 243–272.
- [30] C. Costello, P. Longa, and M. Naehrig, “Efficient algorithms for supersingular isogeny Diffie–Hellman,” in *Advances in Cryptology – CRYPTO 2016: 36th Annual International Cryptology Conference*, Santa Barbara, CA, USA, 2016, pp. 572–601.
- [31] D. Jao and L. De Feo, “Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies,” *Journal of Mathematical Cryptology*, vol. 8, no. 3, pp. 209–247, 2014.
- [32] D. Hofheinz, K. Hövelmanns, and E. Kiltz, “A modular analysis of the Fujisaki–Okamoto transformation,” *Cryptology ePrint Archive*, Paper 2017/604, 2017. [Online]. Tersedia: <https://eprint.iacr.org/2017/604> [diakses: 24 Juni 2025 pukul 14.00 WIB].
- [33] J. W. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, “CRYSTALS–Kyber: A CCA-secure module-lattice-based KEM,” *Cryptology ePrint Archive*, Paper 2017/634,

2017. [Online]. Available: <https://eprint.iacr.org/2017/634> [diakses: 24 Juni 2025 pukul 14.00 WIB].
- [34] M. J. Dworkin, *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions (FIPS PUB 202)*, Federal Information Processing Standards Publication, National Institute of Standards and Technology (NIST), Gaithersburg, MD, USA, 2015. [Online]. Tersedia: <https://doi.org/10.6028/NIST.FIPS.202> [diakses: 24 Juni 2025 pukul 14.30 WIB].
- [35] R. Azarderakhsh, D. Jao, K. Kalach, M. Koziel, and C. Leonardi, “Key compression for isogeny-based cryptosystems,” in *Proceedings of the 3rd ACM International Workshop on ASIA Public-Key Cryptography (AsiaPKC 2016)*, Xi’an, China, 2016, pp. 1–10.
- [36] G. Zanon, M. Armand, S. S. J. Galbraith, C. Costello, C. F. P. Jao, D. Page, S. L. M. Benger, and P. Schwabe, “Faster isogeny-based compressed key agreement,” in *Post-Quantum Cryptography – 9th International Conference, PQCrypto 2018*, Fort Lauderdale, FL, USA, 2018, pp. 407–423.
- [37] G. Pereira and D. Jao, “ x -only point addition formula and faster torsion basis generation in compressed SIKE,” *Cryptology ePrint Archive*, Paper 2020/431, 2020. [Online]. Tersedia: <https://eprint.iacr.org/2020/431> [diakses: 24 Juni 2025 pukul 14.30 WIB].
- [38] National Institute of Standards and Technology (NIST), *NIST Special Publication 1800-38A: Migration to Post-Quantum Cryptography — Preparation for the Coming Transition*, Gaithersburg, MD, USA, 2024. [Online]. Tersedia: <https://doi.org/10.6028/NIST.SP.1800-38> [diakses: 24 Juni 2025 pukul 14.45 WIB].
- [39] National Institute of Standards and Technology (NIST), *NIST Special Publication 1800-38B: Migration to Post-Quantum Cryptography — Quantum Readiness: Cryptographic Discovery, Volume B – Approach, Architecture, and Security Characteristics of Public Key Application Discovery Tools*, Gaithersburg, MD, USA, 2024. [Online]. Tersedia: <https://doi.org/10.6028/NIST.SP.1800-38B> [diakses: 24 Juni 2025 pukul 14.50 WIB].

- [40] National Institute of Standards and Technology (NIST), *NIST Special Publication 1800-38C: Migration to Post-Quantum Cryptography — Quantum Readiness: Testing Draft Standards, Volume C – Quantum-Resistant Cryptography Technology Interoperability and Performance Report*, Gaithersburg, MD, USA, 2024. [Online]. Available: <https://doi.org/10.6028/NIST.SP.1800-38C> [diakses: 24 Juni 2025 pukul 14.50 WIB].
- [41] ENISA, *Post-Quantum Cryptography — Integration Study*, ENISA, Oct. 18, 2022. [Online]. Tersedia: <https://www.enisa.europa.eu/publications/post-quantum-cryptography-integration-study> [diakses: 24 Juni 2025 pukul 15.00 WIB].
- [42] Federal Office for Information Security (BSI), *Migration to Post-Quantum Cryptography*, Bonn, Germany: Bundesamt für Sicherheit in der Informationstechnik (BSI), 2023. [Online]. Tersedia: https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Crypto/Migration_to_Post_Quantum_Cryptography.pdf [diakses: 24 Juni 2025 pukul 15.10 WIB].
- [43] European Union cybersecurity authorities, *Securing Tomorrow, Today: Transitioning to Post-Quantum Cryptography* (Joint Statement by 18 EU Member States), 2024. [Online]. Tersedia: <https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Crypto/PQC-joint-statement.pdf> [diakses: 24 Juni 2025 pukul 15.20 WIB].
- [44] A. Amadori, T. Attema, M. Bombar, J. D. Duarte, V. Dunning, S. Etinski, D. van Gent, M. Lequesne, W. van der Schoot, M. Stevens, and AIVD Cryptologists and Advisors, *The PQC Migration Handbook: Guidelines for Migrating to Post-Quantum Cryptography*, Revised and Extended 2nd ed., The Hague, Netherlands: AIVD – Unit Weerbaarheid; CWI – Cryptology Group; and TNO – Applied Cryptography and Quantum Algorithms, Dec. 2024. [Online]. Tersedia: <https://publications.tno.nl/publication/34643386/fXcPVHsX/TNO-2024-pqc-en.pdf> [diakses: 24 Juni 2025 pukul 15.30 WIB].



**BADAN SIBER
DAN SANDI
NEGARA**



**Deputi Bidang Strategi dan Kebijakan Keamanan Siber dan Sandi
Badan Siber dan Sandi Negara**

Jl. Cijayanti No. 70, Kec. Babakan Madang, Kabupaten Bogor

2025

PUBLIK