Lattice Cryptosystems (III)

Kyber, Saber, NTRU Prime, Dilithium, Falcon

楊柏因 Bo-Yin Yang

Sinica/CCoE/IACR PQCrypto Mini-School,
July 13, 2022

LWE Story

Stories of Historical (Hysterical?) Interest

- 1996, Ajtai article describing theoretical construction of lattice crypto
- 1996, Hoffstein-Pipher-Silverman article circulated (after provisional patent)
  A section specifically mentions distinguishing $(a, as + e)$ from random
- 2005, Regev article re-introduces Learning with Errors problem/systems
- 2010, Gaborit/Aguilar-Melchor patent filed
- 2010, Lybashevsky-Peikert-Regev article, just after the G/AM patent.
- 2012, Ding article on Ring-LWE key exchange and patent
- 2014, Peikert article on Ring-LWE cryptosystems.
- 2016, NewHope key exchange by Alkim, Ducas, Pöppelman, Schwabe.
- 2017, Crystals-Kyber, Saber, NewHope KEMs as NIST-PQC submissions
- 2018, Kyber is modified to use incomplete NTT.
- 2020, Kyber is modified again, NewHope eliminated.
- 2022, Kyber selected as future standard

What is usually called “R-LWE based Encryption”

Source of the Random Looking Distribution (everything is a polynomial)

NTRU	“Ring-LWE-based” Lattice Encryption
$h = g/f, g, f$ ternary	$b = as + e$, s, e discrete Gaussian, a random
$h \sim U$, U means uniform	$(a, b) \sim U \times U$, U means uniform

1. $c = (a, b + m)$ obviously does not secure m.
2. $c = (ra, rb + m)$ still insecure, because we can compute r.
3. $c = (ra + e', rb + m)$ look secure but not semantically secure
   Semantically secure: ciphertexts leak no information, in particular, we shouldn’t be able to tell that ciphertexts point to the same plaintext. Which is because $b^{-1}((r'b + m) - (rb + m)) = (r' - r)$ is small.
4. $c = (ra + e', rb + e'' + m)$ is really provably secure if Ring-LWE is.

Procedures of R-LWE over $\mathbb{Z}_q$

KeyGen
Generate random uniform a, (small) discrete Gaussian s, e, s is the secret key, and the pubkey is (a, b), $b = as + e$.

Encrypt m **using** $pk = (a, b)$**,** $m \in \{0,1\}^n$
Generate discrete Gaussian r, e', e'', $c = (ra + e', rb + e'' + \lfloor \frac{q}{2} \rfloor m)$

Decrypt from $c = (u, v)$ **using** $sk = s$
compute $v - us = rb + e'' + \lfloor \frac{q}{2} \rfloor m - ras - e's = \lfloor \frac{q}{2} \rfloor m + \underbrace{re + e'' - e's}_{\text{small}}$

Each component (bit) is 1 if the number is closer to $q/2$ than 0, else 0.

When Implemented
$c = (u, v)$ rounded down to powers of 2 before transmission. Use not Discrete Gaussian but centered Binomial. Sample $NTT(a)$ for speed. Maybe error-correct.

Gaborit/Aguilar-Melchior/Lyubashevsky/Peikert/Regev (2010, Ring-LWE case)

$\mathbf{s}, \mathbf{e} \leftarrow \chi^*$(discrete Gaussian) $\quad \mathbf{A}$ uniform random

$SK = \mathbf{s}, PK = \mathbf{t} = \mathbf{As} + \mathbf{e}$ $\quad$ globally fixed

$\mathbf{t} \longrightarrow$

$\mathbf{r}, \mathbf{e}', \mathbf{e}'' \leftarrow \chi^*$

$\mathbf{u} = \mathbf{Ar} + \mathbf{e}'$

$\mathbf{v} = \mathbf{tr} + \mathbf{e}'' + \lfloor \frac{q}{2} \rceil \mathbf{M}$

$\longleftarrow \mathbf{u}, \mathbf{v}$

$\mathbf{M}' = \left\lfloor \frac{2}{q} (\mathbf{v} - \mathbf{su}) \right\rceil$

Note $\mathbf{v} - \mathbf{su} = \mathbf{er} - \mathbf{se}' + \mathbf{e}'' + \left\lfloor \frac{q}{2} \right\rceil \mathbf{M}$. This is *not* sufficient to ensure that $\mathbf{M}' = \mathbf{M}$. This is in the G/AM patent (Feb. 2010) and May 2010 talk @PQC, not in LPR Eurocrypt 2010 paper (deadline Feb. 2010), but in LPR Apr. and May 2010 talks.

“Games” for the Gaborit–Aguilar-Melchor/LPR construction

1. Distinguishing $(a, \overbrace{b}^{as+e})$ and (u, v) from Random.
   R-LWE says we can replace $b = as + e$ with a (uniform) random Y.
2. Distinguishing (a, Y) and $(\overbrace{u}^{ra+e'}, \overbrace{v}^{r\mathrm{Y}+e''+m})$ from random
   R-LWE says we can replace $u = ra + e'$ and $rb + e''$ with random Ψ, Φ.
3. Distinguishing (a, Y), $(\Psi, \Phi + m)$ from random
   but the sum of a uniform random and anything is uniform random
4. Distinguishing (a, Y), (Ψ, Ξ) from random
   where Y, Ψ, Ξ are random, and we see that (GAM)LPR is secure.

We don’t talk “IND-CCA2 conversions” here, but Cf. Prof. Kai-Min Chung’s lectures.

Ding's Key (Diffie-Hellman-like) Exchange (2012)

Note: at nearly 1/2 the transmission size

Party i	Party j	
Public Key: $p_i = as_i + 2e_i \in R_q$	Public Key: $p_j = as_j + 2e_j \in R_q$	a is uniform random, public
Secret Key: $s_i \in R_q$	Secret Key: $s_j \in R_q$	Keys are ephemeral!
where $s_i, e_i \leftarrow_r \chi_\alpha$	where $s_j, e_j \leftarrow_r \chi_\alpha$	
	$\xrightarrow{p_i}$	
	$k_j = s_j p_i \in R_q$	$\mathrm{Cha}(x) = 1$ if $-\frac{q}{2} < x \bmod^{\pm} q < \frac{q}{2}$
	$w_j = \mathrm{Cha}(k_j) \in \{0,1\}^n$	$\mathrm{Cha}(x) = 0$ otherwise
	$\xleftarrow{w_j, p_j}$	
$k_i = s_i p_j \in R_q$		$k_i - k_j = 2(s_i e_j - s_j e_i)$
$\sigma_i = \mathrm{Mod}_2(k_i, w_j) \in \{0,1\}^n$	$\sigma_j = \mathrm{Mod}_2(k_j, w_j) \in \{0,1\}^n$	$\mathrm{Mod}_2(x, y) =$
$sk_i = \mathrm{H}_2(i, j, w_j, \sigma_i)$	$sk_j = \mathrm{H}_2(i, j, w_j, \sigma_j)$	$\left((x + y \cdot \frac{q-1}{2}) \bmod q\right) \bmod 2$

- $\mathbb{Z}$:= integers, $R := \mathbb{Z}[x]/(x^n + 1)$, $R_q : \mathbb{Z}_q[x]/(x^n + 1)$
- $\leftarrow_r \chi_\alpha$ denotes a random choice from χ_α (discrete Gaussian distribution centered at 0, std.= α).

Ding's Authenticated Key Exchange

Party i

Public Key: $p_i = as_i + 2e_i \in R_q$

Secret Key: $s_i \in R_q$

where $s_i, e_i \leftarrow_r \chi_\alpha$

$x_i = ar_i + 2f_i \in R_q$

where $r_i, f_i \leftarrow_r \chi_\beta$

Party j

Public Key: $p_j = as_j + 2e_j \in R_q$

Secret Key: $s_j \in R_q$

where $s_j, e_j \leftarrow_r \chi_\alpha$

$y_j = ar_j + 2f_j \in R_q$

where $r_j, f_j \leftarrow_r \chi_\beta$

$\xrightarrow{x_i, p_i}$

$k_j = (p_i c + x_i)(s_j d + r_j) + 2g_j \in R_q$

where $g_j \leftarrow_r \chi_\beta$

$w_j = \mathrm{Cha}(k_j) \in \{0,1\}^n$

$\xleftarrow{y_j, w_j, p_j}$

$k_i = (p_j d + y_j)(s_i c + r_i) + 2g_i$

where $g_i \leftarrow_r \chi_\beta$

$\sigma_i = \mathrm{Mod}_2(k_i, w_j) \in \{0,1\}^n$

$sk_i = \mathrm{H}_2(i, j, x_i, y_j, w_j, \sigma_i)$

$\sigma_j = \mathrm{Mod}_2(k_j, w_j) \in \{0,1\}^n$

$sk_j = \mathrm{H}_2(i, j, x_i, y_j, w_j, \sigma_j)$

$c = \mathrm{H}_1(i, j, x_i) \in R, d = \mathrm{H}_1(j, i, y_j, x_i) \in R$

“Peikert’s Version of Ding’s Key Exchange (2014)”

Achieves nearly the same thing as Ding, and is what is usually known as “R-LWE” today

Parameters: q, n, χ

KEM.Setup() :

$\mathbf{a} \xleftarrow{\$} \mathcal{R}_q$

Alice (server)		Bob (client)
KEM.Gen($\mathbf{a}$) :		KEM.Encaps($\mathbf{a}, \mathbf{b}$) :
$\mathbf{s}, \mathbf{e} \xleftarrow{\$} \chi$		$\mathbf{s}', \mathbf{e}', \mathbf{e}'' \xleftarrow{\$} \chi$
$\mathbf{b} \leftarrow \mathbf{as} + \mathbf{e}$	$\xrightarrow{\mathbf{b}}$	$\mathbf{u} \leftarrow \mathbf{as}' + \mathbf{e}'$
		$\mathbf{v} \leftarrow \mathbf{bs}' + \mathbf{e}''$
		$\bar{\mathbf{v}} \xleftarrow{\$} \mathrm{dbl}(\mathbf{v})$
KEM.Decaps($\mathbf{s}, (\mathbf{u}, \mathbf{v}')$) :	$\xleftarrow{\mathbf{u}, \mathbf{v}'}$	$\mathbf{v}' = \langle \bar{\mathbf{v}} \rangle_2$
$\mu \leftarrow \mathrm{rec}(2\mathbf{us}, \mathbf{v}')$		$\mu \leftarrow \lfloor \bar{\mathbf{v}} \rceil_2$

$$\Pr(e = 0) = \frac{1}{2}, \Pr(e = 1) = \Pr(e = -1) = \frac{1}{4},$$

$$\mathrm{dbl}(v, e) := 2v - e, \langle v \rangle_2 := \lfloor \frac{4}{q} \cdot v \rceil \bmod 2,$$

$$\lfloor v \rceil_2 := \lfloor \frac{2}{q} \cdot v \rceil \bmod 2,$$

$$\mathrm{rec}(w, b) := \begin{cases} 0, & \text{if } w \in I_b + E \pmod q \\ 1, & \text{else} \end{cases}$$

$$I_0 := \{0, 1, \ldots, \lfloor \frac{q}{2} \rceil - 1\},$$

$$I_1 := \{-\lfloor \frac{q}{2} \rfloor, \ldots, -1\},$$

$$E := [-\frac{q}{4}, \frac{q}{4})$$

Bos, Costello, Naehrig, Stebila (2015)

Public parameters		
Decision R-LWE parameters q, n, χ		
$a \xleftarrow{\$} \mathcal{U}(R_q)$		
Alice		**Bob**
$s, e \xleftarrow{\$} \chi$		$s', e' \xleftarrow{\$} \chi$
$b \leftarrow as + e \in R_q$	$\xrightarrow{b}$	$b' \leftarrow as' + e' \in R_q$
		$e'' \xleftarrow{\$} \chi$
		$v \leftarrow bs' + e'' \in R_q$
		$\overline{v} \xleftarrow{\$} \mathrm{dbl}(v) \in R_{2q}$
	$\xleftarrow{b',c}$	$c \leftarrow \langle \overline{v} \rangle_{2q,2} \in \{0,1\}^n$
$k_A \leftarrow \mathrm{rec}(2b's, c) \in \{0,1\}^n$		$k_B \leftarrow \lfloor \overline{v} \rceil_{2q,2} \in \{0,1\}^n$

$$\mathrm{dbl}(v, e) := 2v - e,$$

$$\Pr(e = 0) = \frac{1}{2}, \Pr(e = 1) = \Pr(e = -1) = \frac{1}{4}$$

$$\langle v \rangle_{q,2} := \lfloor \frac{4}{q} \cdot v \rceil \bmod 2,$$

$$\lfloor v \rceil_{q,2} := \lfloor \frac{2}{q} \cdot v \rceil \bmod 2,$$

$$\chi := \psi^n, \psi(x) = D_{\mathbb{Z},8/\sqrt{2\pi}}(x) = \frac{1}{8} e^{-\pi x^2/32}$$

$$\mathrm{rec}(w, b) := \begin{cases} 0, & \text{if } w \in I_b + E \pmod q, \\ 1, & \text{else;} \end{cases}$$

$$n = 1024, q = 2^{32} - 1.$$

NewHope (Alkim, Ducas, Pöppelman, Schwabe, 2015)

Parameters: $q = 12289 < 2^{14}$, $n = 1024$
Error distribution: ψ_{16}

Alice (server)		Bob (client)
$seed \xleftarrow{\$} \{0,1\}^{256}$		
$\mathbf{a} \leftarrow \text{Parse}(\text{SHAKE-128}(seed))$		
$\mathbf{s}, \mathbf{e} \xleftarrow{\$} \psi_{16}^n$		$\mathbf{s}', \mathbf{e}', \mathbf{e}'' \xleftarrow{\$} \psi_{16}^n$
$\mathbf{b} \leftarrow \mathbf{as} + \mathbf{e}$	$\xrightarrow{(\mathbf{b}, seed)}$	$\mathbf{a} \leftarrow \text{Parse}(\text{SHAKE-128}(seed))$ $\mathbf{u} \leftarrow \mathbf{as}' + \mathbf{e}'$ $\mathbf{v} \leftarrow \mathbf{bs}' + \mathbf{e}''$
$\mathbf{v}' \leftarrow \mathbf{us}$	$\xleftarrow{(\mathbf{u}, \mathbf{r})}$	$\mathbf{r} \xleftarrow{\$} \text{HelpRec}(\mathbf{v})$
$\nu \leftarrow \text{Rec}(\mathbf{v}', \mathbf{r})$		$\nu \leftarrow \text{Rec}(\mathbf{v}, \mathbf{r})$
$\mu \leftarrow \text{SHA3-256}(\nu)$		$\mu \leftarrow \text{SHA3-256}(\nu)$

$$\psi_{16} := \text{Centered Binomial, 16 fair coins}$$

$$\text{HelpRec}(\mathbf{x}; b) := \text{CVP}_{\tilde{D}_4}\left(\frac{2^r}{q}(\mathbf{x} + b\mathbf{g})\right) \bmod 2^r,$$

$$D_4 := \mathbb{Z}[\mathbf{u}_0, \mathbf{u}_1, \mathbf{u}_2, (\tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2})]$$

$$\text{Rec}(\mathbf{x}, \mathbf{r}) := \text{Decode}\left(\frac{1}{q}\mathbf{x} - \frac{1}{2^r}\mathbf{Br}\right),$$

$$\text{Decode}(\mathbf{x}) := \begin{cases} 0, & \mathbf{x} - \lfloor \mathbf{x} \rceil \text{ closer to } (0,0,0,0) \\ 1, & \mathbf{x} - \lfloor \mathbf{x} \rceil \text{ closer to } (\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}) \end{cases}$$

A similar version without rings (Frodo) was proposed; it was much larger and not very competitive.

Procedures of M-LWE (Kyber) over $\mathbb{Z}_q$

KeyGen
Generate random uniform matrix A, (small) centered binomial vector s, e, s is the secret key, and the pubkey is (A, b), $b = As + e$.

Encrypt m **using** $pk = (A, b)$**,** m
Generate error (note: not discrete Gaussian) r, e', e'', $c = (Ar + e', r \cdot b + e'' + \lfloor \frac{q}{2} \rfloor m)$

Decrypt from $c = (u, v)$ **using** $sk = s$
compute $v - u \cdot s = rb + e'' + \lfloor \frac{q}{2} \rfloor m - r^T As - e's = \lfloor \frac{q}{2} \rfloor m + \underbrace{re + e'' - e's}_{\text{small}}$

Each component (bit) is 1 if the number is closer to $q/2$ than 0, else 0.

In Actual Implementation
Neither u nor v in $c = (u, v)$ is transmitted in its entirety. They are rounded to a power of two. We sample $NTT(A)$ and not A to faciliate fast multiplication.

Crystals-Kyber, a Module LWE Cryptosystem (NIST Submission 2017)

Avanzi, Bos, Ducas, Kiltz, Lepoint, Lyubashevsky, Schanck, Schwabe, Seiler, Stehlé

$$\begin{aligned}
\mathrm{Compress}_q(x,d) &:= \left\lfloor (2^d/q)x \right\rceil \bmod 2^d;\ \mathrm{Decomp}_q(x,d) := \left\lfloor (q/2^d)x \right\rceil \bmod q \\
R_q &:= \mathbb{Z}_{3329}[X]/(X^{256}+1),\ \mathbf{A} \in R_q^{k\times k} \\
\chi^* &:= (\Pr(\chi^* = \pm 1) = 5/16,\ \Pr(\chi^* = 0) = 3/8),\ U := \text{uniform} \in R_q^{k\times k}
\end{aligned}$$

$\mathbf{s},\mathbf{e} \leftarrow \chi^*$	$\mathbf{A} \leftarrow U$(public)	
$SK = \mathbf{s}, PK = \mathbf{t} = \mathbf{A}\mathbf{s} + \mathbf{e}$		
	$\mathbf{t} \longrightarrow$	
$\mathbf{s},\mathbf{t},\mathbf{u},\mathbf{e},\mathbf{e}' \in R_q^k$		$\mathbf{r},\mathbf{e}',e'' \leftarrow \chi^*$
		$\mathbf{u} = \mathrm{Compress}_q(\mathbf{A}^T\mathbf{r} + \mathbf{e}', d_u)$
		$v = \mathrm{Compress}_q(\mathbf{t}^T r + e'' + \lfloor \frac{q}{2} \rceil M, d_v)$
	$\longleftarrow \mathbf{u}, v$	
$M' = \left\lfloor \frac{2}{q} \left(\mathrm{Decomp}_q(v,d_v) - \mathbf{s}^T \mathrm{Decomp}_q(\mathbf{u},d_u) \right) \right\rceil$	check M' components for proximity to $0, q/2$	

Procedures of R-LWR over $\mathbb{Z}_q$ and $\mathbb{Z}_p$

KeyGen
Generate random uniform a, (small) discrete Gaussian s, which is the secret key, and the pubkey is (a, b), $b = \lceil as \rfloor$.

Encrypt m **using** $pk = (a, b)$**,** $m \in \{0,1\}^n$
Generate error (note: not discretee Gaussian) r, $c = (\lceil ra \rfloor, \lceil rb + \lfloor \frac{q}{2} \rfloor m \rfloor)$

Decrypt from $c = (u, v)$ **using** $sk = s$
Can write $\lceil as \rfloor = as + e, \lceil ra \rfloor = ra + e', \lceil rb + \lfloor \frac{q}{2} \rfloor m \rfloor = rb + \lfloor \frac{q}{2} \rfloor m + e"$.

compute $v - us = rb + e" + \lfloor \frac{q}{2} \rfloor m - ras - e's = \lfloor \frac{q}{2} \rfloor m + \underbrace{re + e" - e's}_{\text{small}}$

Each component (bit) is 1 if the number is closer to $q/2$ then 0, else 0.

We compress u, v again in practice.

Procedures of M-LWR (Saber) over $\mathbb{Z}_q$ and $\mathbb{Z}_p$

KeyGen
Generate random uniform $\ell \times \ell$ matrix of ring elements A, (small) centered binomial vector s, (the secret key), and the pubkey is (A, b), $b = \lceil As \rfloor$.

Encrypt m **using** $pk = (A, b)$, $m \in \{0,1\}^n$
Generate centered binomial vector r, $c = (\lceil Ar \rfloor, \lceil r \cdot b + \lfloor \frac{q}{2} \rfloor m \rfloor)$

Decrypt from $c = (u, v)$ **using** $sk = s$
Can write $\lceil As \rfloor = As + e, \lceil Ar \rfloor = Ar + e', \lceil r \cdot b + \lfloor \frac{q}{2} \rfloor m \rfloor = r \cdot b + \lfloor \frac{q}{2} \rfloor m + e''$.

compute $v - u \cdot s = r \cdot b + e'' + \lfloor \frac{q}{2} \rfloor m - r^T As - e's = \lfloor \frac{q}{2} \rfloor m + \underbrace{r \cdot e + e'' - e' \cdot s}_{\text{small}}$

Each component (bit) is 1 if the number is closer to $q/2$ than 0, else 0.

We compress u, v again in practice.

SABER, a Module LWR (D'Anvers, Karmakar, Roy, Vercautern)

$$\chi^* := \text{CenteredBinomial}(8, 1/2)$$
$$R_q := \mathbb{Z}_{8192}[X]/(X^{256}+1),\ \mathbf{A} \in R_q^{k\times k}$$

$\mathbf{s}, \mathbf{e} \leftarrow \chi^*$	$\mathbf{A} \leftarrow U(\text{public})$	
$SK = \mathbf{s}, PK = \mathbf{b} = \left\lfloor \frac{p}{q}\mathbf{A}\mathbf{s} \right\rceil$		
	$\mathbf{A}, \mathbf{b} \longrightarrow$	
$\mathbf{s}, \mathbf{s}', \mathbf{b}, \mathbf{b}' \in R_q^k$		$\mathbf{s}' \leftarrow \chi^*$
		$\mathbf{b}' = \left\lfloor \frac{1}{8}\mathbf{A}^T\mathbf{s}' \right\rceil$: coefficients mod 1024
		$v' = \left\lfloor \frac{1}{128}\mathbf{b}^T s' + 4M \right\rceil$: coefficients mod 8
	$\longleftarrow \mathbf{b}', v'$	
$M = \left\lfloor \frac{1}{4096}\left(1024v' - 8\mathbf{b}'^T\mathbf{s}\right) \right\rceil$		

Some Misunderstandings about (M,R)-LWE i

Regev first suggested the LWE problem
Hoffstein-Pipher-Silverman noted (1996) LWE has to be hard for NTRU to exist.

SVP has been studied since Gauss
Gauss reduced dim-2 lattices; the Gaussian Heuristic is not endorsed by him.

NTRU has been studied for 1/4-century, power-of-2-cyclotomic rings much less.

There are no known attacks exploiting the ideal structure
The S-Unit attack against cyclotomic rings ideal-SVP is being ignored or denied.

M-LWE has easier scaling of security
Compared to 2-power-cyclotomic Ring-LWE, yes, but not using
2-power-cyclotomic rings has much better scaling of security.

Some Misunderstandings about (M,R)-LWE ii

M-LWE is more (or "no less") secure than R-LWE

$\mathbb{Z}_q[X]/\langle X^{2^k}+1\rangle$ is a module over $\mathbb{Z}_q[X]/\langle X^{2^\ell}+1\rangle$ if $\ell < k$, so if M-LWEs on *all* dimension $2^{k-\ell}$ modules over $\mathbb{Z}_q[X]/\langle X^{2^\ell}+1\rangle$ are insecure, naturally an R-LWE over $\mathbb{Z}_q[X]/\langle X^{2^k}+1\rangle$ is insecure. But there are no reductions from an M-LWE over $\mathbb{Z}_q[X]/\langle X^{256}+1\rangle$ to an R-LWE over $\mathbb{Z}_q[X]/\langle X^{1020}+X^{1019}+\cdots+X+1\rangle$.

NTRU doesn't have a security reduction like (R,M)-LWE

There are larger (Stehlè-Steinfeld) NTRU instances with perfectly fine and meaningful security reductions, the same way that there are larger M-LWE instances with valid security reductions.

(R,M)-LWE is protected by a worst-case-to-average-case reduction theorem

Practical instances much too small compared to the dimensions in the theorem.

Some Misunderstandings about (M,R)-LWE iii

M-LWE (Kyber) is more efficient than R-LWE (NewHope)
No, a matrix-to-vector multiplication over a module takes time that is square in the module dimension; a polynomial multiplication over an equally-sized ring takes time that is linearithmic in the ring size.

Kyber is faster than NewHope because NewHope picked its parameters (e.g., error width) more conservatively where Kyber did not.

Kyber's security analysis is thorough
No, it failed to include "hybrid attacks".

Some Misunderstandings about (M,R)-LWE iv

NewHope picked its parameters in accordance with security theorems
What NewHope did (without logically connecting the dots):

- pointing at worst-to-average-case reductions on continuous Gaussians;
- saying that those trivially also cover rounded Gaussians;
- giving a Renyi-divergence proof that the NewHope distribution is about as good as a rounded-Gaussian distribution

Kyber is more secure than NewHope
No, Kyber's much narrower error distribution makes it probably less secure than NewHope at each corresponding security level, due to hybrid attacks.

NTRU Prime

Streamlined NTRU Prime (NTRU Variation), the Public Key Encryption

The Ring is $R/q = \mathbb{F}_q[x]/(x^p - x - 1)$, a field

- p, q primes, w posint, $2p \geq 3w$, $q \geq 16w + 1$.
- **small** is a polynomial with coefficients in $\{0, \pm 1\}$.
- **short** is small and has exactly w non-zero coefficients.
- **rounded** is each coefficient normalized and divisible by 3.

KeyGen: Take small $g \in R/3 = \mathbb{F}_3[x]/(x^p - x - 1)$ until invertible and compute $1/g \in R/3$. Take short $f \in R/3$. SecKey is $(f, 1/g) \in (R/3) \times (R/3)$. Pubkey $h = g^\dagger/(3f^\dagger) \in R/q$. Here $\cdot^\dagger : R/3 \to R/q$ is a "centerlift".

Encrypt: Input short r, ciphertext $c = \text{round}(hr)$.

Decrypt: Take SecKey (f, v), compute $a = 3f^\dagger c \in R/q$, consider as in R. Then compute $r = av \in R/3$, answer is correct if it is short.

Streamlined NTRU Prime, The Key Establishment Method

Note: Most of this talk omitted KEM conversion details; Kyber, and SABER used the same transformations

KeyGen
Generate keys as before, but with an extra short ρ in the secret key.

Encapsulation
Take a random short r, compute $c = \text{round}(hr)$ Compute $C = (c, \text{hash}(r, h))$. Key is $\text{hash}'(1, r, C)$, Send C.

Decapsulation
Decrypt c using secret key to find r'. Check to see if $\text{hash}(r', h)$ matches. If so output $\text{hash}'(1, r, C)$, else output $\text{hash}'(0, \rho, C)$.

NTRU LPRime (GAMLPRime?), a R-LWR system

$(\chi^*$ = ternary, fixed-weight), $(R_q := \mathbb{Z}_{4591}[X]/(X^{761} - X - 1), A \in R_q)$

$G \leftarrow U,\ a \leftarrow \chi^*$

$SK = a, PK = \{G, A\}, A = \text{Round}_3(aG)$

$G, A \longrightarrow$

$a, b \in R_q$

$\mathbf{b} \leftarrow \chi^*$

$B = \text{Round}_3(bG)$

$T = \left\lfloor \frac{16}{q} \text{Round}_{q/16}\left(\text{Truncate}_{256}(bA) + \frac{q}{2}M\right)\right\rceil$

$\longleftarrow B, T$

$M = \text{Round}_2\left(\frac{q}{16}T - aB\right)$

Digital Signature Schemes

About Digital Signature Schemes

Succinct Non-Interactive Zero-Knowledge Proof of Secret Key

- Signature is (reasonably) short
- Signing a message needs no interaction; otherwise called an “ID Scheme”
- Signature establishes that (s)he owns private key
- Signature does not leak information about private key

The Secret Knowledge

RSA (Pre-Quantum): Knows factorization of public key $N = pq$

Discrete Logarithm (Pre-Quantum): Knows x such that $a = g^x$ (or $A = xP$)

Lattices (Post-Quantum): Knows short s such that $b = As + e \approx As$

Multivariates (Post-Quantum): Knows S, T such that $P = T \circ Q \circ S$

Summary of NISTPQC Signatures (Skylake)

scheme	security	privkey	pubkey	signature	keygen	sign	verify
SPHINCS+128s	NIST I	64 B	32 B	7856 B	85M cc	645M cc	861k cc
SPHINCS+128f	NIST I	64 B	32 B	17088 B	1.3M cc	33M cc	2150k cc
SPHINCS+192s	NIST III	96 B	48 B	16224 B	125M cc	1246M cc	1444k cc
SPHINCS+192f	NIST III	96 B	48 B	35664 B	1.9M cc	55M cc	3492k cc
SPHINCS+256s	NIST V	128 B	64 B	29792 B	192M cc	1025M cc	1987k cc
SPHINCS+256f	NIST V	128 B	64 B	49856 B	5.0M cc	109M cc	2559k cc
Falcon-512	NIST I	1281 B	897 B	666 B	≈ 20M cc	≈ 387k cc	≈ 82k cc
Falcon-1024	NIST V	2305 B	1793 B	1280 B	≈ 63M cc	≈ 790k cc	≈ 168k cc
Dilithium2	NIST II	2528 B	1312 B	2420 B	124k cc	333k cc	118k cc
Dilithium3	NIST III	4000 B	1952 B	3293 B	256k cc	529k cc	179k cc
Dilithium5	NIST V	4864 B	2592 B	4595 B	298k cc	642k cc	280k cc
UOV-16-160-64	NIST I	378404 B	412160 B	96 B	11.3 M cc	2.3M cc	861k cc
Ed25519(ref.)	≈ 0	32 B	32 B	64 B	≈ 55k cc	≈ 60k cc	≈ 200k cc

Lattice Signatures

Ideas Behind Lattice Signatures

Earliest Methods: Unknown Good Basis

Public key is a known, bad basis; private key an unknown, good basis. Signature is a point that is close to (some hash of) the message. The hard problem is the (approximate) Closest Vector Problem (CVP) in a given class of lattice.

Later came Small Integer Solutions and (Ring) Learning with Errors (Product NTRU)

Private key are small vectors s and s', the public key is a matrix A and $b = As + s'$. The hard problems are SIS and LWE variants.

Overview

Falcon (Lattices)
Reasonably fast signing and verifying on big CPUs, smaller (897B pubkey, 666B signature), very complex (hard to program correctly), slow on microcontrollers.

Dilithium (Lattices)
Reasonably fast signing and verifying, larger (1312B pubkey, 2420B signature).

Security Concerns: Recent Attacks

- "S-Unit Attacks" by Daniel J. Bernstein *et al* might affect Falcon/Dilithium security. Back in the early 1990s it was similarly unclear what impact NFS would have on RSA security.

Early Lattice Signatures

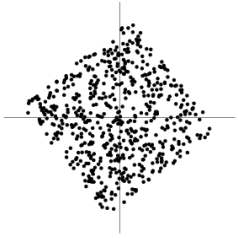
GGH (Goldreich-Goldwasser-Halevi): CVP for random lattice

$S = \{s_1, s_2, \ldots, s_n\}$ secret (good) basis, $B = \{b_1, b_2, \ldots, b_n\}$ equivalent public (bad) basis, such that $\mathbb{Z}[s_1, s_2, \ldots, s_n] = \mathbb{Z}[b_1, b_2, \ldots, b_n]$. The message M is a point in space. Then the signature σ is a lattice point very close to M, constructed by Babai's algorithm: let $M = \sum_{i=1}^{n} a_i s_i$, then $\sigma = \sum_{i=1}^{n} \lceil a_i \rfloor s_i$, transformed into a linear combination of the b_i. Verification: $M - \sigma$ is small.

Approximate CVP Signatures and Leakage

Signatures leak information

Since $M - \sigma = \sum_{i=1}^{n} \left(a_i - \lceil a_i \rfloor\right) s_i$ $M - \sigma$ for any M is in a parallelpiped $\{\sum_{i=1}^{n} \alpha_i s_i, -\frac{1}{2} < \alpha_1, \alpha_2, \ldots, \alpha_n < \frac{1}{2}\}$. We can learn this parallelpiped from sufficiently many samples by first transforming it into a hypercube.



Can do hypercube transformation as distribution's covariance matrix $\propto S^T S$.

Rejection Sampling

Bright Idea: Make the Distribution Something Known

Suppose in a lattice and CVP-based signature there is some randomness in the signing process, and we discard the results and restart on any attempts in which the error vector $m - \sigma$ doesn't fit a given (publicly known) distribution. Then the signature will no longer leak any information about the key.

Generally, this public distribution used to be a discrete Gaussian, and is (in Dilithium) uniform over a standard hypercube or a hypersphere, and presumably the number of expected retries is not too high.

This is called *rejection sampling* and is now standard in lattice-based digital signatures.

NTRUMLS (pqNTRUSign): Uses $R_q := \mathbb{Z}_q[x]/\langle x^{2^\ell}+1\rangle (\ell = 9, 10), q = 65537, p = 2$

KeyGen

f, g ternary with fixed weights, $h = g/(pf) \in R_q$

Sign: find $(s,t) \equiv (s_p, t_p) \pmod p$ **and** $t = hs$ **and not too large**

1. hash $(M\|h)$ (via XOF) uniform $\pmod p$ polynomials s_p and t_p.
2. get random polynomial r (via XOF or PRF) with $\|r\| < q/(2p)$
3. $s_0 := pr + s_p$, $t_0 := s_0 h$, $a := g^{-1}\left((t_p - t_0) \bmod p\right)$, $s := s_0 + paf$, $t := t_0 + ag$
4. $sig := s_1 = (s - s_p)/p$ if $\|s\| < q/2 - B_s$, $\|t\| < q/2 - B_t$, $\|af\| < B_s$, $\|ag\| < B_t$.

Verify

1. hash $(M\|h)$ to obtain digest and uniform $\pmod p$ polynomials s_p and t_p.
2. $s := s_p + ps_1$, $t = sh$, check $t = t_p \pmod p$, $\|s\| < q/2 - B_s$ and $\|t\| < q/2 - B_t$.

Crystals-Dilithium

"Fiat-Shamir" Transforms an ID scheme to a Signature Scheme

An Identification Scheme based on ECC

KeyGen: Public Key $A = aB$, B base point, a is Secret key.

Commit: Peter (Prover) picks random nonce r, computes, sends $R = rB$

Challenge: Vera (Verifier) picks and sends random c.

Response: Peter sends $s = r + ac$.

Verify: Vera checks that $sB = R + cA$.

Digital Signature Scheme (Ed25519, $(a, a') = H(secret), H =$**SHA-512)**

KeyGen: Public Key $A = aB$, B base point, a is Secret key.

Sign: "Nonce" $r = H(a', M)$, $c = H^{\dagger}(\underline{R}, \underline{A}, M)$, sig is $(R = rB, s = r + ac)$.

Verify: On receiving (R, s), check that $sB = R + cA$.

A Corresponding "Fiat-Shamir" *Almost* Lattice Signature Scheme

An Almost-Identification Scheme based on Ring-LWE/SIS

KeyGen: Public Key $t \approx As$, s is Secret key.

Commit: Peter (Prover) picks random y, computes, sends $x \approx Ay$.

Challenge: Vera (Verifier) picks and sends random c.

Response: Peter sends $z = y + sc$.

Verify: Vera checks that $Az \approx x + ct$.

Corresponding *almost* Digital Signature Scheme

KeyGen: Public Key $t \approx As$, s is Secret key.

Sign: Random y, $x \approx Ay$, $c = H^{\dagger}(x, M)$, sig is $(c, z = y + sc)$.

Verify: On receiving (z, c), check that $c = H^{\dagger}(\boxed{\text{Az-ct}}, M)$.

A Corresponding "Fiat-Shamir" *Almost* Lattice Signature Scheme

An Almost-Identification Scheme based on Ring-LWE/SIS

KeyGen: Public Key $t \approx As$, s is Secret key.

Commit: Peter (Prover) picks random y, computes, sends $x \approx Ay$.

Challenge: Vera (Verifier) picks and sends random c.

Response: Peter sends $z = y + sc$.

Verify: Vera checks that $Az \approx x + ct$.

- Use Rejection Sampling
- Need Exact Equality — Use Only High Part of Ay, Send Adjustment Hint

Corresponding *almost* Digital Signature Scheme

KeyGen: Public Key $t \approx As$, s is Secret key.

Sign: Random y, $x \approx Ay$, $c = H^{\dagger}(x, M)$, sig is $(c, z = y + sc)$.

Verify: On receiving (z, c), check that $c = H^{\dagger}(\boxed{Az - ct}, M)$.

Dilithium2 Algorithms

KeyGen()
$\mathbf{A} \leftarrow R^{4\times 4}$; $\mathbf{s_1} \leftarrow [-5, 5]^4$, $\mathbf{s_2} \leftarrow [-5, 5]^4$
$\mathbf{As_1+s_2 = t}$ = low(**t**)+high(**t**)
SK: ($\mathbf{s_1}$, $\mathbf{s_2}$), **PK:** ($\mathbf{A} \leftarrow R^{4\times 4}$, high(**t**))

Sign(μ)

$\mathbf{y} \leftarrow [-\gamma, \gamma]^4$
$c := H(high(\mathbf{Ay}), \mu)$
$\mathbf{z} := \mathbf{y} + c\mathbf{s_1}$
Restart if $|\mathbf{z}| > \gamma - \beta$ or $|low(\mathbf{Ay} - c\mathbf{s_2})| > \gamma - \beta$
Create a small carry bit hint vector **h**
Signature = (**z**, c, **h**)

Makes the distribution of **z** independent of $\mathbf{s_i}$

Carry bits caused by ignoring c·low(**t**)

Verify(**z**, c, **h**, μ)

Use **h** and **Az** - c·high(**t**) to reconstruct high(**Az** - c**t**)
Verify: $|\mathbf{z}| \le \gamma - \beta$ and $c = H(high(\mathbf{Az} - c\mathbf{t}), \mu)$

= high(**Ay**)

- A sampled in NTT (Number Theoretic Transform) domain
- high() and low() split a value in $\mathbb{Z}_q$ into top and bottom parts
- H hashes into short ball: w of the coefficients are ±1, and the rest are 0
- h ("hint") marks where $high(Az - c \cdot high(t)) \neq high(Az - ct)$

Falcon (Sketch)

Original Idea — NTRU Sign

(Original) NTRU Signatures, an analogue of GGH

Private Key: short f, g, Public Key: $h = g/f$. From message M, pick nonce r, compute hash $m = H(M \| r)$, Sig$= (r, \sigma)$ where $\sigma = \lceil -(1/q)mg \rfloor f + \lceil (1/q)mf \rfloor g$ (Babai). Verification: let $m = H(M \| r)$ check that $(\sigma, \sigma h - m)$ is small $\pmod q$.

This is a CVP problem because we have a private basis $\begin{bmatrix} f & g \\ F & G \end{bmatrix}$ for the public basis $\begin{bmatrix} 1 & h \\ 0 & q \end{bmatrix}$, where $fG - gF = q$, and there is a X such that $[\sigma X]\begin{bmatrix} 1 & h \\ 0 & q \end{bmatrix}$ is close to $\begin{bmatrix} 0 \\ m \end{bmatrix}$.

Changes toward Falcon

- Rejection Sampling
- Fast Fourier Transforms
- Babai's nearest-plane algorithm (not Babai's original algorithm)

Falcon, Most Complex Selection, Uses Double-Precision Floats

The GPV over NTRU Lattices framework

- Public Basis $A = [1 | h^*]$ where $\left(\sum_{i=0}^{n-1} h_i x^i\right)^* = a_0 - \sum_{i=1}^{n-1} a_i x^{n-i}$.
- Secret Basis $B = \left[\begin{array}{c|c} g & -f \\ \hline G & -F \end{array}\right]$. $B \times A^* = 0 \pmod q$, $h = g/f \pmod q$.
- Signature of message $M \longrightarrow (r, s_2)$ where we have small $s_1 + s_2 h = H(r \| M)$. Don't send s_1 (computable from r, s_2), To verify, check $\|(s_1, s_2)\|$ small.

Reason to Use DP Floats: Fast Fourier Transforms (not NTT) Representation

- Generation of F, G from f, g (in $\mathbb{Z}[x]$, big integer RNS arithmetic)
- LDL^* Decomposition of BB^* into a tree-form
- Trapdoor Sampling and Nearest-Plane Algorithm using the tree-form

Summary of NISTPQC Signatures (recap, Skylake))

scheme	security	privkey	pubkey	signature	keygen	sign	verify
SPHINCS+128s	NIST I	64 B	32 B	7856 B	85M cc	645M cc	861k cc
SPHINCS+128f	NIST I	64 B	32 B	17088 B	1.3M cc	33M cc	2150k cc
SPHINCS+192s	NIST III	96 B	48 B	16224 B	125M cc	1246M cc	1444k cc
SPHINCS+192f	NIST III	96 B	48 B	35664 B	1.9M cc	55M cc	3492k cc
SPHINCS+256s	NIST V	128 B	64 B	29792 B	192M cc	1025M cc	1987k cc
SPHINCS+256f	NIST V	128 B	64 B	49856 B	5.0M cc	109M cc	2559k cc
Falcon-512	NIST I	1281 B	897 B	666 B	≈ 20M cc	≈ 387k cc	≈ 82k cc
Falcon-1024	NIST V	2305 B	1793 B	1280 B	≈ 63M cc	≈ 790k cc	≈ 168k cc
Dilithium2	NIST II	2528 B	1312 B	2420 B	124k cc	333k cc	118k cc
Dilithium3	NIST III	4000 B	1952 B	3293 B	256k cc	529k cc	179k cc
Dilithium5	NIST V	4864 B	2592 B	4595 B	298k cc	642k cc	280k cc
UOV-16-160-64	NIST I	378404 B	412160 B	96 B	11.3 M cc	2.3M cc	861k cc
Ed25519 (ref.)	≈ 0	32 B	32 B	64 B	≈ 55k cc	≈ 60k cc	≈ 200k cc

Thank you for Listening

That’s it Folks!

Appendix

Obtaining (f, g, F, G) such that $fG - gF = q \pmod{x^{2^k} + 1}$

Fast Fourier Transform NTRU Solve in Falcon

Discrete Gaussian Samples for f, g coefficients

Sample f, g, ensure: f^{-1} exists, $\|(f, g)\|$, $\|\frac{qg}{ff^* + gg^*}, \frac{qf}{ff^* + gg^*}\|$ are small

A Recursive Process

If $k = 0$: find $af + bg = 1$ using extended GCD variant, $F = -bq$, $G = aq$

If $k > 0$: recursively find F', G' where $f'G' - g'F' = q \pmod{x^{2^{k-1}} + 1}$, with
$f'(x^2) = f(x)f(-x)$, $g'(x^2) = g(x)g(-x)$, then
$F(x) = g(-x)F'(x^2)$, $G(x) = f(-x)G'(x^2)$.

Things to Note

- Entire computation is in $\mathbb{Z}[x]$ with thousands-of-bits coefficients.
- Uses Residual Number System with multiple $(2048k + 1)$ primes $< 2^{31}$.

Computing the Falcon (LDL^*) Tree from $G = BB^*$

Algorithm 9 $\mathsf{ffLDL}^*(\mathbf{G})$

Require: A full-rank Gram matrix $\mathbf{G} \in \mathsf{FFT}\left(\mathbb{Q}[x]/(x^n+1)\right)^{2\times 2}$
Ensure: A binary tree T
Format: All polynomials are in FFT representation.

1: $(\mathbf{L}, \mathbf{D}) \leftarrow \mathsf{LDL}^*(\mathbf{G})$ ▷ $\mathbf{L} = \left[\begin{array}{c|c} 1 & 0 \\ \hline L_{10} & 1 \end{array}\right], \mathbf{D} = \left[\begin{array}{c|c} D_{00} & 0 \\ \hline 0 & D_{11} \end{array}\right]$
2: T.value $\leftarrow L_{10}$
3: **if** $(n = 2)$ **then**
4: T.leftchild $\leftarrow D_{00}$
5: T.rightchild $\leftarrow D_{11}$
6: return T
7: **else**
8: $d_{00}, d_{01} \leftarrow \mathsf{splitfft}(D_{00})$ ▷ $d_{ij} \in \mathsf{FFT}\left(\mathbb{Q}[x]/(x^{n/2}+1)\right)$
9: $d_{10}, d_{11} \leftarrow \mathsf{splitfft}(D_{11})$
10: $\mathbf{G}_0 \leftarrow \left[\begin{array}{c|c} d_{00} & d_{01} \\ \hline d_{01}^\star & d_{00} \end{array}\right], \mathbf{G}_1 \leftarrow \left[\begin{array}{c|c} d_{10} & d_{11} \\ \hline d_{11}^\star & d_{10} \end{array}\right]$ ▷ Since D_{00}, D_{11} are self-adjoint, (3.30) applies
11: T.leftchild $\leftarrow \mathsf{ffLDL}^*(\mathbf{G}_0)$ ▷ Recursive calls
12: T.rightchild $\leftarrow \mathsf{ffLDL}^*(\mathbf{G}_1)$
13: return T

Fast Fourier Transform Trapdoor Sampling

Algorithm 11 $\mathsf{ffSampling}_n(\mathbf{t}, \mathsf{T})$

Require: $\mathbf{t} = (t_0, t_1) \in \mathsf{FFT}\left(\mathbb{Q}[x]/(x^n+1)\right)^2$, a FALCON tree T
Ensure: $\mathbf{z} = (z_0, z_1) \in \mathsf{FFT}\left(\mathbb{Z}[x]/(x^n+1)\right)^2$
Format: All polynomials are in FFT representation.

1: **if** $n = 1$ **then**
2: $\quad \sigma' \leftarrow \mathsf{T.value}$ ▷ It is always the case that $\sigma' \in [\sigma_{\min}, \sigma_{\max}]$
3: $\quad z_0 \leftarrow \mathsf{SamplerZ}(t_0, \sigma')$ ▷ Since $n = 1$, $t_i = \mathsf{invFFT}(t_i) \in \mathbb{Q}$ and $z_i = \mathsf{invFFT}(z_i) \in \mathbb{Z}$
4: $\quad z_1 \leftarrow \mathsf{SamplerZ}(t_1, \sigma')$
5: $\quad$ **return** $\mathbf{z} = (z_0, z_1)$
6: $(\ell, \mathsf{T}_0, \mathsf{T}_1) \leftarrow (\mathsf{T.value}, \mathsf{T.leftchild}, \mathsf{T.rightchild})$
7: $\mathbf{t}_1 \leftarrow \mathsf{splitfft}(t_1)$ ▷ $\mathbf{t}_0, \mathbf{t}_1 \in \mathsf{FFT}\left(\mathbb{Q}[x]/(x^{n/2}+1)\right)^2$
8: $\mathbf{z}_1 \leftarrow \mathsf{ffSampling}_{n/2}(\mathbf{t}_1, \mathsf{T}_1)$ ▷ First recursive call to $\mathsf{ffSampling}_{n/2}$
9: $z_1 \leftarrow \mathsf{mergefft}(\mathbf{z}_1)$ ▷ $\mathbf{z}_0, \mathbf{z}_1 \in \mathsf{FFT}\left(\mathbb{Z}[x]/(x^{n/2}+1)\right)^2$
10: $t_0' \leftarrow t_0 + (t_1 - z_1) \odot \ell$
11: $\mathbf{t}_0 \leftarrow \mathsf{splitfft}(t_0')$
12: $\mathbf{z}_0 \leftarrow \mathsf{ffSampling}_{n/2}(\mathbf{t}_0, \mathsf{T}_0)$ ▷ Second recursive call to $\mathsf{ffSampling}_{n/2}$
13: $z_0 \leftarrow \mathsf{mergefft}(\mathbf{z}_0)$
14: **return** $\mathbf{z} = (z_0, z_1)$

Signing with Assistance from Fast Fourier Transform Trapdoor

Algorithm 10 Sign $(\mathsf{m}, \mathsf{sk}, \lfloor\beta^2\rfloor)$

Require: A message m, a secret key sk, a bound $\lfloor\beta^2\rfloor$
Ensure: A signature sig of m

$\mathsf{r} \leftarrow \{0,1\}^{320}$ uniformly
$c \leftarrow \mathsf{HashToPoint}(\mathsf{r}\|\mathsf{m}, q, n)$
$\mathbf{t} \leftarrow \left(-\frac{1}{q}\mathsf{FFT}(c) \odot \mathsf{FFT}(F), \frac{1}{q}\mathsf{FFT}(c) \odot \mathsf{FFT}(f)\right)$ ▷ $\mathbf{t} = (\mathsf{FFT}(c), \mathsf{FFT}(0)) \cdot \hat{\mathbf{B}}^{-1}$
do
  do
    $\mathbf{z} \leftarrow \mathsf{ffSampling}_n(\mathbf{t}, \mathsf{T})$
    $\mathbf{s} = (\mathbf{t} - \mathbf{z})\hat{\mathbf{B}}$ ▷ At this point, $\mathbf{s}$ follows a Gaussian distribution: $\mathbf{s} \sim D_{(c,0)+\Lambda(\mathbf{B}),\sigma,0}$
  while $\|\mathbf{s}\|^2 > \lfloor\beta^2\rfloor$ ▷ Since $\mathbf{s}$ is in FFT representation, one may use (3.8) to compute $\|\mathbf{s}\|^2$
  $(s_1, s_2) \leftarrow \mathsf{invFFT}(\mathbf{s})$ ▷ $s_1 + s_2 h = c \bmod (\phi, q)$
  $\mathbf{s} \leftarrow \mathsf{Compress}(s_2, 8 \cdot \mathsf{sbytelen} - 328)$ ▷ Remove 1 byte for the header, and 40 bytes for r
while $(\mathbf{s} = \perp)$
return $\mathsf{sig} = (\mathsf{r}, \mathbf{s})$

Babai's Nearest Plane Algorithm

Babai's original algorithm for short basis (s_i)
Let $M = \sum_{i=1}^{n} a_i s_i$, then $\sigma = \sum_{i=1}^{n} \lceil a_i \rfloor s_i$,

Nearest Plane Algorithm

1. Let $\delta \leftarrow M$, find the Gram-Schmidt orthogonalization $\left(\hat{s}_i\right)$ of lattice basis (s_i).
2. for $j = n$ downto 1 do
3. $\delta \leftarrow \delta - c_j s_j$, where $c_j = \lceil \langle \delta, \hat{s}_j \rangle / \|\hat{s}_j\|^2 \rfloor$,
   this basically finds recursively integers c_j such that the hyperplane $c_j s_j + \text{span}(s_1, s_2, \ldots, s_{j-1})$ is closest to δ.
4. end do
5. output $\sigma = M - \delta$.

Pieces of the Puzzle of NTRU Prime: Deterministic Encode/Decode

Let $M = (m_0, \ldots, m_{n-1})$ and $R = (r_0, \ldots, r_{n-1})$ be int sequences with $0 \le r_i < m_i < 2^{14}, \forall i$. Length of $S = \text{Encode}(R, M)$ depends only on M.

Encoding

- If $n = 0$ then S is the empty sequence ().
- If $n = 1$ then S is r_0 in little-endian (if $m_0 > 2^8$, 2 bytes, if $2^8 \ge m_0 > 1$, 1 byte)
- If $n \ge 2$ then S is a prefix then $\text{Encode}(R_0, M_0)$. Each pair r_i, r_{i+1} modulo m_i, m_{i+1} for even i is merged into $r = r_i + m_i r_{i+1}$ modulo $m = m_i m_{i+1}$, and then r, m are reduced to r_0, m_0 with $0 < r_0 < m_0 < 2^{14}$, producing an entry for R_0 and an entry for M_0 respectively. For odd n, include $(r_{n-1}, m_{n-1}$ in (R_0, M_0).
- Reduction: if $m \ge 2^{14}$ then $r \bmod 2^8$ is appended to the prefix while r, m are replaced by $\lfloor r/2^8 \rfloor, \lceil m/2^8 \rceil$, repeat 0–2 times to reduce m to the correct range.

Pieces of the Puzzle of NTRU Prime: Decoding

```
def Decode(S,M):
  if len(M) == 0: return []
  if len(M) == 1: return [sum(S[i]*256**i for i in range(len(S)))%M[0]]
  k = 0; bottom,M2 = [],[]
  for i in range(0,len(M)-1,2):
    m,r,t = M[i]*M[i+1],0,1
    while m >= limit:
      r,t,k,m = r+S[k]*t,t*256,k+1,(m+255)//256
    bottom += [(r,t)]; M2 += [m]
  if len(M)&1:  M2 += [M[-1]]
  R2 = Decode(S[k:],M2);   R = []
  for i in range(0,len(M)-1,2):
    r,t = bottom[i//2]; r += t*R2[i//2]
    R += [r%M[i]];  R += [(r//M[i])%M[i+1]]
  if len(M)&1: R += [R2[-1]]
  return R
```

Pieces of the Puzzle of NTRU Prime

Other Encodings and Hashing

- Encoding of polynomial: center-lift coefficients and add $(q-1)/2$, apply Encode with $M = p$ copies of q
- Encoding of rounded polynomial: center-lift coefficients, divide by 3, add $\frac{q-1}{6}$ ($q \equiv 1 \pmod 6$), Encode with $M = p$ copies of $\frac{q+2}{3}$.
- Encoding of Small polynomial: add 1, pack as 2 bits, small-endian.
- $(f, v = 1/g)$ are encoded as two small polynomials.
- Hash is SHA-512 cut to 256 bits. $\mathrm{Hash}_b(x)$ is $Hash(b\|x)$ for byte b. $\mathrm{hash}(r,h) = \mathrm{Hash}_2(\mathrm{Hash}_3(r), \mathrm{Hash}_4(h))$ ($\mathrm{Hash}_4(h)$ is cached after first use); $\mathrm{hash}'(b,y,z) = \mathrm{Hash}_b(\mathrm{Hash}_3(y), z)$.