HAWK Signature Algorithm

https://youtu.be/dX0X1oMu7YE

한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB

1. HAWK 소개

- HAWK
  - NIST additional Digital signature schemes에 제출된 격자 기반 알고리즘
  - FALCON과 같은 해시 및 서명 설계를 공유하는 것 외에는 공통점이 없으나, 비슷한 점이 있기 때문에 오마주로 네이밍을 HAWK로 함

Hawk Falcon

- HAWK-512 / HAWK-1024 두개의 파라미터를 제공
  - NIST 보안 레벨 1과 레벨 5를 만족함

Speed

The procedures for generating and verifying signatures are fast on all devices, including low end devices.

HAWK-512 signature generation and verification take <0.1ms on an average desktop PC.

Compactness

The keys and signature sizes are all rather small.

A HAWK-512 public key is 1024 bytes, a signature 555 bytes.

Well-suited for various hardware

HAWK is free of floating-point arithmetic, i.e. CPUs do not need to support `double`.

Moreover, HAWK-512 only needs 14 kiB of RAM to work.

2. HAWK 알고리즘

Table 2: Key and signature sizes for HAWK in bytes.

	HAWK-512	HAWK-1024
Private key size	184	360
Public key size	1024	2440
Signature size	555	1221

Speed on ARM Cortex M4 (clock cycles)

Key pair generation	5.23×10^7	2.26×10^8
Signature generation	2.80×10^6 (1.16×10^6)	1.42×10^6 (1.23×10^6)
Signature verification	1.42×10^6 (1.23×10^6)	3.01×10^6 (2.61×10^6)
	53,200,000	226,000,000
	2,800,000	1,420,000
	1,420,000	3,010,000

Scheme	Type	Key Generation	Sign	Verify
dilithium2	m4	1 400 412	6 157 001	1 461 284
dilithium3	m4	2 282 485	9 289 499	2 228 898
dilithium4	m4	3 097 421	8 468 805	3 173 500
falcon1024	m4-ct	480 910 965	83 482 883	977 140
falcon512	m4-ct	197 793 925	38 090 446	474 052
falcon512-tree	m4-ct	201 459 670	17 181 744	475 278

NIST Security Level	2	3	5
Output Size			
public key size (bytes)	1312	1952	2592
signature size (bytes)	2420	3293	4595

Parameter set	Parameter set alias	Security model	Claimed NIST Level	Public key size (bytes)	Secret key size (bytes)
Dilithium2	NA	EUF-CMA	2	1312	2528
Dilithium3	NA	EUF-CMA	3	1952	4000
Dilithium5	NA	EUF-CMA	5	2592	4864

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

Parameter set	Parameter set alias	Security model	Claimed NIST Level	Public key size (bytes)	Secret key size (bytes)
Falcon-512	NA	EUF-CMA	1	897	1281
Falcon-1024	NA	EUF-CMA	5	1793	2305
Falcon-padded-512	NA	EUF-CMA	1	897	1281
Falcon-padded-1024	NA	EUF-CMA	5	1793	2305

2. HAWK 알고리즘

- 키 생성 / 서명 생성 / 서명 검증
  - 키 생성 과정이 가장 오랜 시간이 걸림.

degree	kg(ms)	sd(us)	sf(us)	vv(us)	vf(us)
256:	0.62	30.96	17.95	26.10	21.36
512:	2.66	56.66	35.46	53.86	43.86
1024:	14.39	119.72	72.26	112.98	91.25

Algorithm Sketch 1 High level HawkKeyGen

Ensure: $\mathbf{B} \in \mathsf{GL}_2(R_n)$ and $\mathbf{Q} = \mathbf{B}^{\star}\mathbf{B}$

Sample coefficients of $f, g \in R_n$ i.i.d. from $\mathrm{Bin}(\eta)$
if $\mathsf{f\text{-}g\text{-}conditions}(f, g)$ is false **then restart**
$r \leftarrow \mathsf{NTRUSolve}(f, g)$
if r is $\perp$ **then restart**
$(F, G) \leftarrow r$
$\mathbf{B} \leftarrow \begin{pmatrix} f & F \\ g & G \end{pmatrix}$, $\mathbf{Q} \leftarrow \mathbf{B}^{\star}\mathbf{B}$
if $\mathsf{KGen\text{-}encoding}(\mathbf{Q}, \mathbf{B})$ is false **then restart**
$\mathsf{hpub} \leftarrow \mathsf{H}(\mathbf{Q})$
return $(\mathsf{pk}, \mathsf{sk}) \leftarrow (\mathbf{Q}, (\mathbf{B}, \mathsf{hpub}))$

Algorithm Sketch 2 High level HawkSign

Require: A message m and secret key $\mathsf{sk} = (\mathbf{B}, \mathsf{hpub})$
Ensure: A signature sig formed of a uniform salt $\mathsf{salt} \in \{0,1\}^{\mathsf{saltlen_{bits}}}$ and $s_1 \in R_n$

$M \leftarrow \mathsf{H}(\mathsf{m} \parallel \mathsf{hpub})$
$\mathsf{salt} \leftarrow \mathsf{Rnd}(\mathsf{saltlen_{bits}})$
$\mathbf{h} \leftarrow \mathsf{H}(M \parallel \mathsf{salt})$
$\mathbf{t} \leftarrow \mathbf{B} \cdot \mathbf{h} \bmod 2$
$\mathbf{x} \leftarrow D_{2\mathbb{Z}^{2n}+\mathbf{t}, 2\sigma_{\mathsf{sign}}}$
if $\|\mathbf{x}\|^2 > 4 \cdot \sigma_{\mathsf{verify}}^2 \cdot 2n$ **then restart**
$\mathbf{w} \leftarrow \mathbf{B}^{-1}\mathbf{x}$
if $\mathsf{sym\text{-}break}(\mathbf{w})$ is false **then** $\mathbf{w} = -\mathbf{w}$
$\mathbf{s} \leftarrow \frac{1}{2}(\mathbf{h} - \mathbf{w})$
$s_1 \leftarrow \mathsf{Compress}(\mathbf{s})$
if $\mathsf{sig\text{-}encoding}(\mathsf{salt}, s_1)$ is false **then restart**
return $\mathsf{sig} \leftarrow (\mathsf{salt}, s_1)$

Algorithm Sketch 3 High level HawkVerify

Require: A message m, a public key $\mathsf{pk} = \mathbf{Q}$, and a signature $\mathsf{sig} = (\mathsf{salt}, s_1)$
Ensure: A bit determining whether sig is a valid signature on m

$\mathsf{hpub} \leftarrow \mathsf{H}(\mathbf{Q})$
$M \leftarrow \mathsf{H}(\mathsf{m} \parallel \mathsf{hpub})$
$\mathbf{h} \leftarrow \mathsf{H}(M \parallel \mathsf{salt})$
$\mathbf{s} \leftarrow \mathsf{Decompress}(s_1, \mathbf{h}, \mathbf{Q})$
$\mathbf{w} \leftarrow \mathbf{h} - 2\mathbf{s}$
if $\mathsf{len_{bits}}(\mathsf{salt}) = \mathsf{saltlen_{bits}}$ and $\mathbf{s} \in R_n^2$ and $\mathsf{sym\text{-}break}(\mathbf{w})$ and $\|\mathbf{w}\|_{\mathbf{Q}}^2 \leq 4 \cdot \sigma_{\mathsf{verify}}^2 \cdot 2n$ **then**
    return 1
else
    return 0

2. HAWK 알고리즘 – 키 생성

Algorithm Sketch 1 High level HawkKeyGen

Ensure: $\mathbf{B} \in \mathsf{GL}_2(R_n)$ and $\mathbf{Q} = \mathbf{B}^\star\mathbf{B}$

Sample coefficients of $f, g \in R_n$ i.i.d. from $\mathrm{Bin}(\eta)$

if f-g-conditions(f, g) is false **then restart**

$r \leftarrow \mathsf{NTRUSolve}(f, g)$

if r is $\perp$ **then restart**

$(F, G) \leftarrow r$

$\mathbf{B} \leftarrow \begin{pmatrix} f & F \\ g & G \end{pmatrix}$, $\mathbf{Q} \leftarrow \mathbf{B}^\star\mathbf{B}$

if KGen-encoding$(\mathbf{Q}, \mathbf{B})$ is false **then restart**

hpub $\leftarrow \mathsf{H}(\mathbf{Q})$

return $(\mathsf{pk}, \mathsf{sk}) \leftarrow (\mathbf{Q}, (\mathbf{B}, \mathsf{hpub}))$

Coefficients i.i.d
independent and identically distributed coefficients

Algorithm 12 Regeneratefg: Regenerate (f, g)

Require: Key generation seed kgseed

Ensure: Polynomials (f, g)

$b \leftarrow n/64$ ▷ $b = 4$, 8 or 16, depending on n.

$y \leftarrow \mathsf{SHAKE256x4}(\mathsf{kgseed})[0 : 2bn]$ ▷ b bits for each coefficient of f and g.

for $i = 0$ to $n - 1$ **do**

  $f[i] \leftarrow \left(\sum_{j=0}^{b-1} y[ib + j]\right) - b/2$ ▷ centred binomial with $\eta = b/2$.

for $i = 0$ to $n - 1$ **do**

  $g[i] \leftarrow \left(\sum_{j=0}^{b-1} y[(i + n)b + j]\right) - b/2$

return (f, g)

다항식 f와 g는 난수씨드에서 초기화된 SHAKE256x4 인스턴스를 사용하여 얻은 의사 난수 비트로부터 생성되는 중앙 집중 이항 분포를 사용하여 샘플링됨.

2. HAWK 알고리즘 – 키 생성

Algorithm Sketch 1 High level HawkKeyGen

Ensure: $\mathbf{B} \in \mathsf{GL}_2(R_n)$ and $\mathbf{Q} = \mathbf{B}^\star\mathbf{B}$

1: Sample coefficients of $f, g \in R_n$ i.i.d. from $\mathrm{Bin}(\eta)$
2: **if** f-g-conditions(f, g) is false **then restart**
3: $r \leftarrow \mathsf{NTRUSolve}(f, g)$
4: **if** r is $\perp$ **then restart**
5: $(F, G) \leftarrow r$
6: $\mathbf{B} \leftarrow \begin{pmatrix} f & F \\ g & G \end{pmatrix}$, $\mathbf{Q} \leftarrow \mathbf{B}^\star\mathbf{B}$
7: **if** KGen-encoding$(\mathbf{Q}, \mathbf{B})$ is false **then restart**
8: hpub $\leftarrow \mathsf{H}(\mathbf{Q})$
9: **return** $(\mathsf{pk}, \mathsf{sk}) \leftarrow (\mathbf{Q}, (\mathbf{B}, \mathsf{hpub}))$

Algorithm 12 Regeneratefg: Regenerate (f, g)

Require: Key generation seed kgseed
Ensure: Polynomials (f, g)

1: $b \leftarrow n/64$ ▷ $b = 4$, 8 or 16, depending on n.
2: $y \leftarrow \mathsf{SHAKE256x4}(\mathsf{kgseed})[0 : 2bn]$ ▷ b bits for each coefficient of f and g.
3: **for** $i = 0$ to $n - 1$ **do**
4: $\quad f[i] \leftarrow \left(\sum_{j=0}^{b-1} y[ib + j]\right) - b/2$ ▷ centred binomial with $\eta = b/2$.
5: **for** $i = 0$ to $n - 1$ **do**
6: $\quad g[i] \leftarrow \left(\sum_{j=0}^{b-1} y[(i + n)b + j]\right) - b/2$
7: **return** (f, g)

- 중앙 집중 이항 분포
  - 평균이나 기대값이 0이 되는 이항 분포

2. HAWK 알고리즘 – 키 생성

Algorithm Sketch 1 High level $\mathsf{HawkKeyGen}$

Ensure: $\mathbf{B} \in \mathsf{GL}_2(R_n)$ and $\mathbf{Q} = \mathbf{B}^\star\mathbf{B}$

1: Sample coefficients of $f, g \in R_n$ i.i.d. from $\mathrm{Bin}(\eta)$
2: **if** f-g-conditions(f, g) is false **then restart**
3: $\boxed{r \leftarrow \mathsf{NTRUSolve}(f, g)}$
4: **if** r is $\perp$ **then restart**
5: $(F, G) \leftarrow r$
6: $\mathbf{B} \leftarrow \begin{pmatrix} f & F \\ g & G \end{pmatrix}$, $\mathbf{Q} \leftarrow \mathbf{B}^\star\mathbf{B}$
7: **if** KGen-encoding$(\mathbf{Q}, \mathbf{B})$ is false **then restart**
8: $\mathsf{hpub} \leftarrow \mathsf{H}(\mathbf{Q})$
9: **return** $(\mathsf{pk}, \mathsf{sk}) \leftarrow (\mathbf{Q}, (\mathbf{B}, \mathsf{hpub}))$

Then, NTRUSolve comes up with (F, G) such that

$$\boxed{\|(F, G)\|_\infty \leq 127,} \text{ and } fG - gF = 1. \tag{31}$$

$$\|\cdot\|\colon K_n \to \mathbb{Q},\ f \mapsto \sqrt{\langle f, f\rangle}, \tag{13}$$

$$\|\cdot\|_\infty\colon K_n \to \mathbb{Q},\ f \mapsto \max_{0\leq i<n}(|f[i]|). \tag{14}$$

2. HAWK 알고리즘 – 키 생성

Algorithm Sketch 1 High level HawkKeyGen

Ensure: $\mathbf{B} \in \mathsf{GL}_2(R_n)$ and $\mathbf{Q} = \mathbf{B}^\star\mathbf{B}$

Sample coefficients of $f, g \in R_n$ i.i.d. from $\mathrm{Bin}(\eta)$
if f-g-conditions(f, g) is false **then restart**
$r \leftarrow \mathsf{NTRUSolve}(f, g)$
if r is $\perp$ **then restart**
$(F, G) \leftarrow r$
$\mathbf{B} \leftarrow \begin{pmatrix} f & F \\ g & G \end{pmatrix}$, $\mathbf{Q} \leftarrow \mathbf{B}^\star\mathbf{B}$
if KGen-encoding$(\mathbf{Q}, \mathbf{B})$ is false **then restart**
$\mathsf{hpub} \leftarrow \mathsf{H}(\mathbf{Q})$
return $(\mathsf{pk}, \mathsf{sk}) \leftarrow (\mathbf{Q}, (\mathbf{B}, \mathsf{hpub}))$

Algorithm 13 HawkKeyGen: HAWK key pair generation

Require: Cryptographically secure source of random bits
Ensure: New key pair $(\mathsf{priv}, \mathsf{pub})$

$\mathsf{kgseed} \leftarrow \mathsf{Rnd}(\mathsf{kgseedlen_{bits}})$ ▷ $\mathsf{kgseedlen_{bits}}$ is defined in Table 4.
$(f, g) \leftarrow \mathsf{Regeneratefg}(\mathsf{kgseed})$
if $\mathsf{IsInvertible}(f, 2) \neq \mathsf{true}$ or $\mathsf{IsInvertible}(g, 2) \neq \mathsf{true}$ **then**
    restart
if $\|(f, g)\|^2 \leq 2n\sigma^2_{\mathsf{krsec}}$ **then**
    restart
$q_{00} \leftarrow ff^\star + gg^\star$
$(p_1, p_2) \leftarrow (2147473409, 2147389441)$
if $\mathsf{IsInvertible}(q_{00}, p_1) \neq \mathsf{true}$ or $\mathsf{IsInvertible}(q_{00}, p_2) \neq \mathsf{true}$ **then**
    restart
if $(1/q_{00})[0] \geq \beta_0$ **then** ▷ Inverse over $\mathbb{Q}[X]/(X^n + 1)$.
    restart
$r \leftarrow \mathsf{NTRUSolve}(f, g, 1)$
if $r = \perp$ **then**
    restart
$(F, G) \leftarrow r$
if $\|(F, G)\|_\infty > 127$ **then**
    restart
$q_{01} \leftarrow Ff^\star + Gg^\star$
$q_{11} \leftarrow FF^\star + GG^\star$
if $|q_{11}[i]| \geq 2^{\mathsf{high}_{11}}$ for any $i > 0$ **then**
    restart
$\mathsf{pub} \leftarrow \mathsf{EncodePublic}(q_{00}, q_{01})$
if $\mathsf{pub} = \perp$ **then**
    restart
$\mathsf{hpub} \leftarrow \mathsf{SHAKE256}(\mathsf{pub})$ ▷ $\mathsf{hpublen_{bits}}$ is defined in Table 4.
$\mathsf{priv} \leftarrow \mathsf{EncodePrivate}(\mathsf{kgseed}, F \bmod 2, G \bmod 2, \mathsf{hpub})$
return $(\mathsf{priv}, \mathsf{pub})$

f, g 다항식 생성

NTRUSolve 만족하는
F, G 다항식 생성

-> 공개키 생성

생성된 공개키로 비밀키 생성

2. HAWK 키 생성 코드 확인

유클리드 노름
벡터의 제곱합의 제곱근으로 계산
유클리드 공간에서 거리나 길이

Algorithm 13 HawkKeyGen: HAWK key pair generation

Require: Cryptographically secure source of random bits
Ensure: New key pair (priv, pub)

kgseed $\leftarrow$ Rnd(kgseedlen$_{\text{bits}}$) ▷ kgseedlen$_{\text{bits}}$ is defined in Table 4.
$(f, g) \leftarrow$ Regeneratefg(kgseed)
if IsInvertible$(f, 2) \neq$ true or IsInvertible$(g, 2) \neq$ true **then**
  restart
if $\|(f, g)\|^2 \leq 2n\sigma^2_{\text{krsec}}$ **then**
  restart
$q_{00} \leftarrow ff^\star + gg^\star$
$(p_1, p_2) \leftarrow (2147473409, 2147389441)$
if IsInvertible$(q_{00}, p_1) \neq$ true or IsInvertible$(q_{00}, p_2) \neq$ true **then**
  restart
if $(1/q_{00})[0] \geq \beta_0$ **then** ▷ Inverse over $\mathbb{Q}[X]/(X^n + 1)$.
  restart
$r \leftarrow$ NTRUSolve$(f, g, 1)$
if $r = \perp$ **then**
  restart
$(F, G) \leftarrow r$
if $\|(F, G)\|_\infty > 127$ **then**
  restart
$q_{01} \leftarrow Ff^\star + Gg^\star$
$q_{11} \leftarrow FF^\star + GG^\star$
if $|q_{11}[i]| \geq 2^{\text{high}_{11}}$ for any $i > 0$ **then**
  restart
pub $\leftarrow$ EncodePublic(q_{00}, q_{01})
if pub $= \perp$ **then**
  restart
hpub $\leftarrow$ SHAKE256(pub) ▷ hpublen$_{\text{bits}}$ is defined in Table 4.
priv $\leftarrow$ EncodePrivate(kgseed, F mod 2, G mod 2, hpub)
return (priv, pub)

```c
/*
 * Generate f and g.
 */
rng(rng_context, seed_buf, seed_len);
Hawk_regen_fg(logn, f, g, seed_buf);

/*
 * Start again if f and g are not both odd.
 */
if (parity(logn, f) != 1 || parity(logn, g) != 1) {
    continue;
}

/*
 * Check that (f,g) has an acceptable norm; this is a
 * _minimum_ bound (2*n*sigma_sec^2).
 */
uint32_t norm2_fg = poly_sqnorm(logn, f) + poly_sqnorm(logn, g);
if (norm2_fg < l2low) {
    continue;
}

/*
 * Check that f*adj(f) + g*adj(g) is invertible modulo
 * X^n+1 mod p1 (with p1 = 2147473409 = PRIMES[0].p).
 * We also output f*adj(f) + g*adj(g) into t1.
 */
```

2. HAWK 키 생성 코드 확인

Algorithm 13 HawkKeyGen: HAWK key pair generation

Require: Cryptographically secure source of random bits
Ensure: New key pair (priv, pub)

kgseed $\leftarrow$ Rnd(kgseedlen$_{\mathsf{bits}}$) $\triangleright$ kgseedlen$_{\mathsf{bits}}$ is defined in Table 4.
$(f, g) \leftarrow$ Regeneratefg(kgseed)
if IsInvertible$(f, 2) \neq$ true or IsInvertible$(g, 2) \neq$ true **then**
    restart
if $\|(f, g)\|^2 \leq 2n\sigma^2_{\mathsf{krsec}}$ **then**
    restart
$q_{00} \leftarrow ff^\star + gg^\star$
$(p_1, p_2) \leftarrow (2147473409, 2147389441)$
if IsInvertible$(q_{00}, p_1) \neq$ true or IsInvertible$(q_{00}, p_2) \neq$ true **then**
    restart
if $(1/q_{00})[0] \geq \beta_0$ **then** $\triangleright$ Inverse over $\mathbb{Q}[X]/(X^n + 1)$.
    restart
$r \leftarrow$ NTRUSolve$(f, g, 1)$
if $r = \perp$ **then**
    restart
$(F, G) \leftarrow r$
if $\|(F, G)\|_\infty > 127$ **then**
    restart
$q_{01} \leftarrow Ff^\star + Gg^\star$
$q_{11} \leftarrow FF^\star + GG^\star$
if $|q_{11}[i]| \geq 2^{\mathsf{high}_{11}}$ for any $i > 0$ **then**
    restart
pub $\leftarrow$ EncodePublic(q_{00}, q_{01})
if pub $= \perp$ **then**
    restart
hpub $\leftarrow$ SHAKE256(pub) $\triangleright$ hpublen$_{\mathsf{bits}}$ is defined in Table 4.
priv $\leftarrow$ EncodePrivate(kgseed, F mod 2, G mod 2, hpub)
return (priv, pub)

```
	/*
	 * Solve the NTRU equation.
	 */
#if NTRUGEN_STATS
	stats_solve_attempt ++;
#endif
	int err = solve_NTRU(prof, logn, f, g, tt32);
	switch (err) {
	case SOLVE_OK:
#if NTRUGEN_STATS
		stats_solve_success ++;
#endif
		break;
#if NTRUGEN_STATS
	case SOLVE_ERR_GCD:
		stats_solve_err_gcd ++;
		continue;
	case SOLVE_ERR_REDUCE:
		stats_solve_err_reduce ++;
		continue;
	case SOLVE_ERR_LIMIT:
		stats_solve_err_limit ++;
		continue;
#endif
	default:
		continue;
	}
```

2. HAWK 키 생성 코드 확인

Algorithm 13 HawkKeyGen: HAWK key pair generation

Require: Cryptographically secure source of random bits
Ensure: New key pair (priv, pub)

kgseed ← Rnd($\mathsf{kgseedlen}_{\mathsf{bits}}$) ▷ $\mathsf{kgseedlen}_{\mathsf{bits}}$ is defined in Table 4.
$(f, g) \leftarrow$ Regeneratefg(kgseed)
if IsInvertible$(f, 2) \neq$ true or IsInvertible$(g, 2) \neq$ true **then**
  restart
if $\|(f, g)\|^2 \leq 2n\sigma^2_{\mathsf{krsec}}$ **then**
  restart
$q_{00} \leftarrow ff^\star + gg^\star$
$(p_1, p_2) \leftarrow (2147473409, 2147389441)$
if IsInvertible$(q_{00}, p_1) \neq$ true or IsInvertible$(q_{00}, p_2) \neq$ true **then**
  restart
if $(1/q_{00})[0] \geq \beta_0$ **then** ▷ Inverse over $\mathbb{Q}[X]/(X^n + 1)$.
  restart
$r \leftarrow$ NTRUSolve$(f, g, 1)$
if $r = \perp$ **then**
  restart
$(F, G) \leftarrow r$
if $\|(F, G)\|_\infty > 127$ **then**
  restart
$q_{01} \leftarrow Ff^\star + Gg^\star$
$q_{11} \leftarrow FF^\star + GG^\star$
if $|q_{11}[i]| \geq 2^{\mathsf{high}_{11}}$ for any $i > 0$ **then**
  restart
pub ← EncodePublic(q_{00}, q_{01})
if pub $= \perp$ **then**
  restart
hpub ← SHAKE256(pub) ▷ $\mathsf{hpublen}_{\mathsf{bits}}$ is defined in Table 4.
priv ← EncodePrivate(kgseed, $F \bmod 2$, $G \bmod 2$, hpub)
return (priv, pub)

```c
/*
 * Compute q00, q01 and q1, and check that they are in the
 * expected range.
 *
 * F and G use the first 2*n bytes = hn words.
 */
if (!make_q001(logn, lim00, lim01, lim11,
    f, g, tF, tG, (uint32_t *)(tG + n)))
{
#if NTRUGEN_STATS
    stats_solve_err_limit ++;
#endif
    continue;
}

int16_t *tq00 = (int16_t *)(tG + n);
int16_t *tq01 = tq00 + n;
int32_t *tq11 = (int32_t *)(tq01 + n);
uint8_t *tseed = (uint8_t *)(tq11 + n);
memmove(tseed, seed_buf, seed_len);

/*
 * Return the computed F, G, q00, q01, q11 and seed.
 */
if (F != NULL) {
    memmove(F, tF, n);
}
if (G != NULL) {
    memmove(G, tG, n);
}
if (q00 != NULL) {
    memmove(q00, tq00, n * sizeof *tq00);
}
if (q01 != NULL) {
    memmove(q01, tq01, n * sizeof *tq01);
}
if (q11 != NULL) {
    memmove(q11, tq11, n * sizeof *tq11);
}
if (seed != NULL) {
    memmove(seed, tseed, seed_len);
}
if (tt32 != tmp) {
    memmove(tmp, tt32, 10 * n + seed_len);
}
```

3. HAWK 키 생성 코드 확인

Algorithm 13 HawkKeyGen: HAWK key pair generation

Require: Cryptographically secure source of random bits
Ensure: New key pair (priv, pub)

$\mathsf{kgseed} \leftarrow \mathsf{Rnd}(\mathsf{kgseedlen}_{\mathsf{bits}})$ ▷ $\mathsf{kgseedlen}_{\mathsf{bits}}$ is defined in Table 4.
$(f, g) \leftarrow \mathsf{Regeneratefg}(\mathsf{kgseed})$
if $\mathsf{IsInvertible}(f, 2) \neq \mathsf{true}$ or $\mathsf{IsInvertible}(g, 2) \neq \mathsf{true}$ **then**
    restart
if $\|(f, g)\|^2 \leq 2n\sigma^2_{\mathsf{krsec}}$ **then**
    restart
$q_{00} \leftarrow ff^\star + gg^\star$
$(p_1, p_2) \leftarrow (2147473409, 2147389441)$
if $\mathsf{IsInvertible}(q_{00}, p_1) \neq \mathsf{true}$ or $\mathsf{IsInvertible}(q_{00}, p_2) \neq \mathsf{true}$ **then**
    restart
if $(1/q_{00})[0] \geq \beta_0$ **then** ▷ Inverse over $\mathbb{Q}[X]/(X^n + 1)$.
    restart
$r \leftarrow \mathsf{NTRUSolve}(f, g, 1)$
if $r = \bot$ **then**
    restart
$(F, G) \leftarrow r$
if $\|(F, G)\|_\infty > 127$ **then**
    restart
$q_{01} \leftarrow Ff^\star + Gg^\star$
$q_{11} \leftarrow FF^\star + GG^\star$
if $|q_{11}[i]| \geq 2^{\mathsf{high}_{11}}$ for any $i > 0$ **then**
    restart
$\mathsf{pub} \leftarrow \mathsf{EncodePublic}(q_{00}, q_{01})$
if $\mathsf{pub} = \bot$ **then**
    restart
$\mathsf{hpub} \leftarrow \mathsf{SHAKE256}(\mathsf{pub})$ ▷ $\mathsf{hpublen}_{\mathsf{bits}}$ is defined in Table 4.
$\mathsf{priv} \leftarrow \mathsf{EncodePrivate}(\mathsf{kgseed}, F \bmod 2, G \bmod 2, \mathsf{hpub})$
return (priv, pub)

```c
    for (;;) {
        if (Hawk_keygen(logn, f, g, 0, 0, 0, 0, 0, 0, rng, rng_context,
            tt8, (size_t)(((int8_t *)tmp + tmp_len) - tt8)) != 0)
        {
            return 0;
        }

        if (encode_public(logn, tpub, pub_len, q00, q01)) {
            (void)encode_private(logn, tpriv,
                seed, F, G, tpub, pub_len);
#if HAWK_DEBUG
            printf("#### Keygen (n=%u):\n", 1u << logn);
            print_blob("kgseed", seed, seed_len);
            print_i8(logn, "f", f);
            print_i8(logn, "g", g);
            print_i8(logn, "F", F);
            print_i8(logn, "G", G);
            print_i16(logn, "q00", q00);
            print_i16(logn, "q01", q01);
            print_i32(logn, "q11", q11);
            print_blob("priv", tpriv, priv_len);
            print_blob("pub", tpub, pub_len);
#endif
            if (priv != NULL) {
                memcpy(priv, tpriv, priv_len);
            }
            if (pub != NULL) {
                memcpy(pub, tpub, pub_len);
            }
            memmove(tmp, tpriv, priv_len + pub_len);
            return 1;
        }
    }
```

3. HAWK 최적화 구현

```
ng_hawk Hawk keygen ftimer print
ftimer_ntt = 0.000273
fs->rng = 0.000021
fs->Hawk_regen_fg_t = 0.000808
fs->mp_mkgmigm_t = 0.000015
fs->mp_add_t = 0.000300
fs->mp_montymul = 0.000340
fs->mp_norm = 0.000300
fs->mp_mkgm_t = 0.000013
fs->vect_FFT_t = 0.000307
fs->solve_NTRU_t = 0.057962
```

```
ng_ntru Hawk keygen FTIMER_ntru print
solve_NTRU_deepest = 0.007041
solve_NTRU_intermediate = 0.049215
solve_NTRU_depth0 = 0.001656
poly_mp_set_small_t = 0.000000
mp_mkgm_t = 0.000000
mp_NTT_t = 0.000000
mp_mkigm_t = 0.000000
zint_rebuild_CRT_t = 0.000000
```

```c
/*
 * Memory layout: we keep Ft, Gt, ft and gt; we append_ntru:
 *   gm    NTT support (n)
 *   igm   iNTT support (n)
 *   fx    temporary f mod p (NTT) (n)
 *   gx    temporary g mod p (NTT) (n)
 */
uint32_t *gm = t1;
uint32_t *igm = gm + n;
uint32_t *fx = igm + n;
uint32_t *gx = fx + n;
mp_mkgmigm(logn, gm, igm, PRIMES[u].g, PRIMES[u].ig, p, p0i);
if (u < slen) {
    memcpy(fx, ft + u * n, n * sizeof *fx);
    memcpy(gx, gt + u * n, n * sizeof *gx);
    mp_iNTT(logn, ft + u * n, igm, p, p0i);
    mp_iNTT(logn, gt + u * n, igm, p, p0i);
} else {
    uint32_t Rx = mp_Rx31((unsigned)slen, p, p0i, R2);
    for (size_t v = 0; v < n; v ++) {
        fx[v] = zint_mod_small_signed(ft + v, slen, n,
            p, p0i, R2, Rx);
        gx[v] = zint_mod_small_signed(gt + v, slen, n,
            p, p0i, R2, Rx);
    }
    mp_NTT(logn, fx, gm, p, p0i);
    mp_NTT(logn, gx, gm, p, p0i);
}

/*
 * We have (F,G) from deeper level in Ft and Gt, in
 * RNS. We apply the NTT modulo p.
 */
uint32_t *Fe = Ft + u * n;
uint32_t *Ge = Gt + u * n;
mp_NTT(logn - 1, Fe + hn, gm, p, p0i);
mp_NTT(logn - 1, Ge + hn, gm, p, p0i);
```

```c
uint32_t *Fe = Ft + u * n;
uint32_t *Ge = Gt + u * n;
mp_NTT(logn - 1, Fe + hn, gm, p, p0i);
mp_NTT(logn - 1, Ge + hn, gm, p, p0i);

/*
 * Compute F and G (unreduced) modulo p.
 */
for (size_t v = 0; v < hn; v ++) {
    uint32_t fa = fx[(v << 1) + 0];
    uint32_t fb = fx[(v << 1) + 1];
    uint32_t ga = gx[(v << 1) + 0];
    uint32_t gb = gx[(v << 1) + 1];
    uint32_t mFp = mp_montymul(Fe[v + hn], R2, p, p0i);
    uint32_t mGp = mp_montymul(Ge[v + hn], R2, p, p0i);
    Fe[(v << 1) + 0] = mp_montymul(gb, mFp, p, p0i);
    Fe[(v << 1) + 1] = mp_montymul(ga, mFp, p, p0i);
    Ge[(v << 1) + 0] = mp_montymul(fb, mGp, p, p0i);
    Ge[(v << 1) + 1] = mp_montymul(fa, mGp, p, p0i);
}

/*
 * We want the new (F,G) in RNS only (no NTT).
 */
mp_iNTT(logn, Fe, igm, p, p0i);
mp_iNTT(logn, Ge, igm, p, p0i);
```

감 사 합 니 다