FALCON with message recovery, a specification

Felix Günther[1], Vadim Lyubashevsky[1], and Rolfe Schmidt[2]

[1]IBM Research Europe, Zurich
[2]Signal Foundation

Abstract

FALCON (Prest et al. NIST submission) is a lattice-based digital signature scheme that is intended to be standardized by NIST under the name FN-DSA. This scheme has smaller signature and public key sizes than the "primary" NIST scheme, ML-DSA, but is more complicated to implement due to floating-point requirements. For this reason, NIST has recommended the scheme be used in special situations where smaller key and signature sizes are particularly important.

There are certain situations, where one can make small modifications to FALCON which results in even shorter outputs, which is particularly interesting for FALCON's intended use cases where smaller outputs are important. In the scenario where one would like to minimize the total *public key plus signature* length, one could use FALCON in *key recovery mode*, which is a fairly straight-forward procedure specified in the FALCON document.

In the scenario where one would like to minimize the total *signature plus message* length, one could use the *message recovery* mode (which is a somewhat less straight-forward modification) as described in (del Pino et al. SCN 2017). The purpose of this note is to fully specify the details of this latter mode for developers wishing to implement it. The savings of using FALCON in message recovery mode is up to 226 bytes as compared to FALCON-512 and up to 434 bytes compared to FALCON-1024 for the same security levels (i.e. NIST levels 1 and 5, respectively). We also sketch how the same technique can be applied to the ring signature version of FALCON from (Gajland et al. Crypto 2024).

The main algorithmic building blocks of FALCON such as key generation and trapdoor sampling, remain exactly the same. The only algorithmic changes are in the hashing and parsing of the messages, randomness, and the hash function.

1 FALCON with message recovery

This document proposes a standard way in which one would use the FALCON [4] signature scheme in message recovery mode (first proposed in [1]), called FALCON-MRM. The purpose of this document is to specify the concrete specifications of this mode.

We define $\mathcal{R}_f$ to be the polynomial ring $\mathbb{Z}_q[X]/(f(X))$. For both security levels of FALCON and FALCON-MRM, $q = 12289$. The polynomial $f(X)$ is

either $X^{512}+1$ or $X^{1024}+1$ for 128 and 256-bit security levels, respectively. As notation, we will write (a, b) to mean the concatenation of elements a and b. A polynomial $y \in \mathcal{R}_f$ is represented by either a 512 or a 1024 coefficient vector over $\mathbb{Z}_q$. If we write $y = (a, b)$ for $a \in \mathbb{Z}_q^{\lambda}, b \in \mathbb{Z}_q^{512-\lambda}$, then we mean that y is interpreted as a concatenation of two vectors a and b, where a is a vector consisting of the first λ coefficients of y and b is a vector consisting of the other $512-\lambda$.

We write $a \leftarrow b$ to mean that the variable a is set to the value b. If b is a set, then it means that a is chosen at random from b. If the above b is a distribution, then a is chosen according to the distribution b. We will also write $[N]$ to mean the set of natural numbers $\{1, \ldots, N\}$.

1.1 Hash Functions

The FALCON-MRM signature uses two hash functions, modeled as random oracles.

- $\mathcal{H}_1 : \mathbb{Z}_q^{\gamma} \times \mathbb{Z}_q^{n-\lambda-\gamma} \times \{0,1\}^* \rightarrow \mathbb{Z}_q^{\lambda}$
- $\mathcal{H}_2 : \mathbb{Z}_q^{\lambda} \rightarrow \mathbb{Z}_q^{n-\lambda}$

The function $\mathcal{H}_1$ will be used to hash the salt $\rho \in \mathbb{Z}_q^{\gamma}$ and the message (consisting of two parts $M_1 \in \mathbb{Z}_q^{n-\lambda-\gamma}$ and $M_2 \in \{0,1\}^*$) to the domain of $\mathcal{H}_2$. Outputs of $\mathcal{H}_1$ and $\mathcal{H}_2$, when concatenated together, will form a vector in $\mathbb{Z}_q^n$, whose coefficients represent a polynomial in the FALCON ring $\mathbb{Z}_q[X]/(X^n+1)$.

To convert vectors over $\mathbb{Z}_q$ to bit-strings (for creating inputs to the hash functions), we simply write each element in $\mathbb{Z}_q$ as a 14-bit string encoded as a 14-bit big-endian integer. The resulting bits are concatenated in index order and packed into bytes; if the length of the resulting bitstring is not a multiple of 8, the last byte is zero-padded in its least-significant bits. Using hash functions $\mathcal{H}_1$ and $\mathcal{H}_2$ to hash into vectors over $\mathbb{Z}_q$, we use the HashToPoint algorithm in the FALCON specification with input prefixed by an appropriate domain separation tag.

The parameters for the two security levels, corresponding to the security of FALCON-512 and FALCON-1024 are given in the below table. The values of λ and γ are chosen to be the smallest allowable by the proof of Lemma 1 (see Equation (1)).

Security Level	n	λ	γ
128	512	19 ($\lvert\mathbb{Z}_q^{\lambda}\rvert > 256$)	15 ($\lvert\mathbb{Z}_q^{\gamma}\rvert > 192$)
256	1024	38 ($\lvert\mathbb{Z}_q^{\lambda}\rvert > 512$)	24 ($\lvert\mathbb{Z}_q^{\gamma}\rvert > 320$)

The message M that is to be hashed will be split into two parts as $M = (M_1, M_2)$. The message M_1 will be recovered from the signature, while M_2 is simply sent in the clear. We will assume that $M_2 \in \{0,1\}^*$, while M_1 has already been converted to a vector in $\mathbb{Z}_q^{n-\lambda-\gamma}$. In Section 1.4, we give a method of converting a bit-string into a vector over $\mathbb{Z}_q$ in a nearly optimal way. The message M_1 will always be of the maximal allowed length $n-\lambda-\gamma$ $\mathbb{Z}_q$ elements, and we leave it to the application to "meaningfully" encode into, and decode from, M_1.

Algorithm 1 FALCON-sign

Input: $M \in \{0,1\}^*$
Output: s_2, ρ
  repeat
    $\rho \leftarrow \{0,1\}^{320}$
    $c \leftarrow \mathcal{H}(\rho, M)$
    $s_1, s_2 \leftarrow D_\sigma$ such that $s_1 + s_2 h = c$ ▷ Using the FALCON sampler.
  until $\|(s_1, s_2)\| \leq \beta$
  return ρ, s_2

Algorithm 2 FALCON-verify

Input: $M \in \{0,1\}^*, s_2, \rho \in \{0,1\}^{320}$
Output: True / False
  $c \leftarrow \mathcal{H}(\rho, M)$
  $s_1 \leftarrow c - s_2 h$
  if $\|(s_1, s_2)\| > \beta$ **then return** False
  end if
  return True

1.2 Signature Scheme

For completeness, we first present the signing and verification algorithms of the original FALCON scheme in Algorithms 1 and 2 before giving the FALCON-MRM algorithms in Algorithms 3 and 4, since we will be proving the security of FALCON-MRM via a reduction from FALCON. The key generation for both version is the same – the public key is a polynomial h and the secret key is a "trapdoor" allowing for pre-image-sampling polynomials s_1, s_2 (with $\|(s_1, s_2)\| < \beta$) according to a discrete Gaussian distribution D_σ satisfying $s_1 + hs_2 = t$ for arbitrary $t \in \mathcal{R}_f$.[1] FALCON also uses a hash function $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_q^n = \mathcal{R}_f$.

Unlike FALCON, which uses the same salt size $|\rho| = 320$ for both security levels, mostly for simplicity, we vary the size of the salt (i.e., the value γ) in addition to the output size of $\mathcal{H}_1$ (i.e., the value λ), both of which have an effect on the security, and compactness, of the schemes. The intuition for why we put ρ into the part of the message that can be recovered is in the case that if the whole message is smaller than what fits in M_1, we can use the leftover space to recover ρ. Since ρ is always part of the signature, there is no downside in always including it into the "recoverable" part.

Lemma 1. *Breaking FALCON-MRM implies breaking FALCON.*

Proof. Let $\mathcal{A}$ be the algorithm breaking FALCON-MRM and $\mathcal{S}$ be the simulator that will break FALCON. During the algorithm, $\mathcal{S}$ will be creating an injective mapping $f : \mathbb{Z}_q^\gamma \to \{0,1\}^{320}$. When $\mathcal{A}$ makes a query:

1. $\mathcal{H}_1(\rho, M = (M_1, M_2))$. If this value was already queried, $\mathcal{S}$ simply returns it. Otherwise, $\mathcal{S}$ checks whether $f(\rho)$ has been set and if it hasn't, it assigns $f(\rho)$ to the first (in e.g. lexicographic order) $\rho' \in \{0,1\}^{320}$ for

[1] $\|(s_1, s_2)\|$ will be less than β only most of the time, and so we need to check for this fact during the signing process.

Algorithm 3 FALCON-MRM-sign

Input: $M = (M_1, M_2) \in (\mathbb{Z}_q^{n-\lambda-\gamma}, \{0,1\}^*)$
Output: s_1, s_2, ρ
  repeat
    $\rho \leftarrow \mathbb{Z}_q^{\gamma}$
    $c = (c_1, c_2) \leftarrow (\mathcal{H}_1(\rho, M), \mathcal{H}_2(c_1) + (\rho, M_1) \bmod q)$
    $s_1, s_2 \leftarrow D_\sigma$ such that $s_1 + s_2 h = c$ ▷ Using the FALCON sampler.
  until $\|(s_1, s_2)\| \leq \beta$
  return s_1, s_2, M_2

Algorithm 4 FALCON-MRM-verify

Input: M_2, s_1, s_2
Output: M_1 or $\perp$ (reject)
  if $\|s_1, s_2\| > \beta$ **then return** $\perp$
  end if
  $(c_1, c_2) = c \leftarrow s_1 + s_2 h$ ▷ $(c_1, c_2) \in \left(\mathbb{Z}_q^{\lambda}, \mathbb{Z}_q^{n-\lambda}\right)$
  $(\rho, M_1) \leftarrow c_2 - \mathcal{H}_2(c_1) \bmod q$ ▷ $(\rho, M_1) \in \left(\mathbb{Z}_q^{\gamma}, \mathbb{Z}_q^{n-\lambda-\gamma}\right)$
  if $\mathcal{H}_1(\rho, (M_1, M_2)) \neq c_1$ **then return** $\perp$
  end if
  return M_1

which $f^{-1}(\rho')$ hasn't been set yet.[2] $\mathcal{S}$ computes $c = \mathcal{H}(f(\rho), M)$, sets $c = (c_1, c_2)$. If $\mathcal{H}_2(c_1)$ has already been programmed, then $\mathcal{S}$ aborts. Otherwise it programs $c_1 = \mathcal{H}_1(\rho, M)$ and $c_2 - (\rho, M_1) = \mathcal{H}_2(c_1)$.

2. To the FALCON-MRM signing oracle on $M = (M_1, M_2)$. $\mathcal{S}$ chooses a random value $\rho \in \mathbb{Z}_q^{\gamma}$. If $f(\rho)$ has already been set, it aborts. $\mathcal{S}$ makes a FALCON signing query on M, obtaining the signature ρ', s_2, M. If $f^{-1}(\rho')$ has already been set, it aborts. Otherwise, it sets $f(\rho) = \rho'$. It then computes $c = \mathcal{H}(\rho', M)$, $s_1 = c - hs_2$, and sets $c = (c_1, c_2)$. It programs $\mathcal{H}_1(\rho, M) = c_1$. Then $c_2 - (\rho, M_1)$ is programmed to $\mathcal{H}_2(c_1)$, unless $\mathcal{H}_2(c_1)$ has already been set, in which case, it aborts. The signature is M_2, s_1, s_2.

3. $\mathcal{H}_2(x)$. $\mathcal{S}$ checks if $\mathcal{H}_2(x)$ has already been programmed. If not, it programs it to a random value y and returns it.

In (1), because $\mathcal{H}$ is a random oracle, c_1 and c_2 are random values and so the outputs of $\mathcal{H}_1$ and $\mathcal{H}_2$ are programmed to random values. In (2), again because $\mathcal{H}$ is a random oracle and ρ' is fresh, c_1, c_2 are fresh random values. Moreover, ρ is fresh and chosen at random, so the signature follows the correct distribution. In (3), $\mathcal{H}_2$ is always programmed to a random value.

To see that the signature returned in (2) is valid, notice that the FALCON signer returns values such that

$$s_1 + hs_2 = \mathcal{H}(\rho', M) = (c_1, c_2),$$

[2]Note that such ρ' exists as any practical $\mathcal{A}$ will make fewer than 2^{320} queries.

and then the functions $\mathcal{H}_1$ and $\mathcal{H}_2$ are programmed such that $c_1 = \mathcal{H}_1(\rho, M)$ and $c_2 - (\rho, M_1) = \mathcal{H}_2(c_1)$. Thus

$$s_1 + hs_2 = (\mathcal{H}_1(\rho, M), \mathcal{H}_2(c_1) + (\rho, M_1))$$

as in the proper FALCON-MRM signing algorithm. So as long as no aborts occur, the simulation is perfect. We now compute the probability of an abort occurring.

Let Q_1, Q_2 be the number of times $\mathcal{H}_1, \mathcal{H}_2$ are queried, and Q_S be the number of signature queries. First, let's compute the probability of an abort happening when programming f. The number of times that f (and f^{-1}) is set is $Q_1 + Q_S$. An abort can happen only in (2), and happens with probability upper bounded by

$$\frac{Q_1 Q_S}{|\mathbb{Z}_q^\gamma|} + \frac{Q_S^2}{|\mathbb{Z}_q^\gamma|}$$

where the first term is when a collision happens when f was originally programmed in (1) and the second term is the probability that it was programmed in (2). The time it takes to obtain an abort in the first term is $Q_1 + Q_S$ and the time in the second term is Q_S. Thus it would take approximately $\frac{|\mathbb{Z}_q^\gamma|}{Q_S}$ time to obtain an abort. The calculation for obtaining an abort because f^{-1} was already set is the same except the $|\mathbb{Z}_q^\gamma|$ term is replaced with $|\{0,1\}^{320}|$, thus requiring $\frac{2^{320}}{Q_S}$ expected number of queries for an abort to occur.

Since $|Q_S| \leq 2^{64}$, per NIST requirements (and this is also important for the FALCON security proof), we have

$$|\mathbb{Z}_q^\gamma|/Q_S \geq 2^{128} \text{ for } \gamma = 15$$
$$|\mathbb{Z}_q^\gamma|/Q_S \geq 2^{256} \text{ for } \gamma = 24$$
$$2^{320}/Q_S \geq 2^{256}$$

We now compute the probability that an abort occurs because $\mathcal{H}_2(c_1)$ was already programmed. The total number of times that $\mathcal{H}_2$ is programmed is $Q_1 + Q_2 + Q_S$. Let $Q = \max(Q_1, Q_2, Q_S)$ be the largest of the three values. Then the probability of an abort is upper bounded by $\frac{3Q^2}{|\mathbb{Z}_q^\lambda|}$. Since it takes at least around Q time to perform these queries, the expected time until an abort occurs is approximately $\frac{|\mathbb{Z}_q^\lambda|}{3Q}$, which is greater than $2^{256}/Q$ for $\lambda = 19$ and greater than $2^{512}/Q$ for $\lambda = 38$. Since Q is at most 2^{128} and 2^{256}, respectively,[3] we have that the time complexity of hitting an abort is 2^{128} and 2^{256}, respectively.

We now show that when $\mathcal{A}$ produces a forgery for FALCON-MRM, $\mathcal{S}$ produces a forgery for FALCON. When $\mathcal{A}$ comes up with a forgery (s_1, s_2, M_2), $\mathcal{S}$ computes $s_1 + hs_2 = (c_1, c_2)$ and then $(\rho, M_1) = c_2 - \mathcal{H}_2(c_1)$. $\mathcal{S}$ then returns $s_2, M = (M_1, M_2), \rho' = f(\rho)$ as the forgery for FALCON. To see that this is a valid FALCON forgery, first note that $\mathcal{A}$ must have previously queried $\mathcal{H}_1(\rho, M)$ (because otherwise there is only a $1/|\mathbb{Z}_q^\lambda|$ chance that $\mathcal{H}_1(\rho, M) = c_1$), and so by construction, we have that $\mathcal{H}(f(\rho), M) = (c_1, c_2)$. Thus $s_1 + hs_2 = (c_1, c_2) = \mathcal{H}(f(\rho), M)$, and so the signature for FALCON is valid.

[3]Performing more than this number of queries queries would make the adversary run-time more than AES-128 and AES-256 security levels, which are the targeted security levels for FALCON-512 and FALCON-1024.

	s_1	s_2	M_1	ρ+header	total
FALCON-512	0	625	$\lvert\mathbb{Z}_q^{478}\rvert = 811$	41	1477
FALCON-MRM-512	625	625	0	1	1251
FALCON-1024	0	1239	$\lvert\mathbb{Z}_q^{962}\rvert = 1633$	41	2913
FALCON-MRM-1024	1239	1239	0	1	2479

Table 1: The output size, in bytes, of the inputs to the verifier of FALCON and FALCON-MRM when signing a message $M = (M_1, M_2 = \{\})$. If the message M_2 is not null, then the size of this message gets added to the total number of bytes in both schemes. The "header" is a one byte element described in the FALCON specification.

If M was never queried to the signing oracle of FALCON-MRM, then it was also never queried to the signing oracle of FALCON, and so $\mathcal{S}$ returned a forgery on a fresh M. If M were queried, then (s_1, s_2) must be different than the signature $\mathcal{A}$ received and so it broke the strong-unforgeability of FALCON-MRM. In this case s_2 must also be different than the one received in the signing query to FALCON, and so we have a strong forgery for FALCON as well. □

In terms of an advantage bound, the proof of Lemma 1 yields:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{SUF\text{-}CMA}}_{\text{FALCON-MRM}}(t, Q_1, Q_2, Q_S) &\leq \mathsf{Adv}^{\mathsf{SUF\text{-}CMA}}_{\text{FALCON}}(t', Q_H, Q_S) \\ &\quad + \frac{(Q_1 + Q_S) \cdot Q_S}{|\mathbb{Z}_q^\gamma|} + \frac{3Q^2}{|\mathbb{Z}_q^\lambda|} \\ &\leq \mathsf{Adv}^{\mathsf{SUF\text{-}CMA}}_{\text{FALCON}}(t', Q_H, Q_S) \\ &\quad + \frac{Q_1 + Q_S}{2^b} + \frac{Q^2}{2^{2b}}, \end{aligned} \tag{1}$$

where $t' \approx t$, $Q_H = Q_1 + Q_S$, $Q = \max(Q_1, Q_2, Q_S)$, and $b \in \{128, 256\}$ is the intended bit security level of FALCON-512 and FALCON-1024, respectively.

For bit security b, we want that $\mathsf{Adv} \leq T/2^b$, where T is the total run-time of the adversary, which is satisfied for each term above.

1.3 Setting Parameters

The size of s_1, s_2 for FALCON-512 and FALCON-1024 is 625 bytes and 1239 bytes, respectively. The norm bound β and Gaussian width σ take the same values as in the corresponding FALCON instance; see [4] for their exact values. In the below table, we compute, in bytes, the total transmission size of message + signature of FALCON and FALCON-MRM when signing a message of the maximum size that can be recovered in FALCON-MRM (i.e., for M_2 being empty). The result is that one can get up to a 226 byte and 434 byte saving in the transmission size, for FALCON-MRM-512 compared to FALCON-512, and respectively FALCON-MRM-1024 and FALCON-1024. If the message being signed isn't natively a $\mathbb{Z}_q$-vector (e.g., it's a bit string), then it has to be converted into a $\mathbb{Z}_q$-vector by using an encoding, such as the one in Section 1.4. The encoding described in that section is close to optimal and results in a loss of only one bit over the information-theoretic maximum when encoding

Domain	Information-Theoretic Maximum	Alg. in Sec. 1.4
$\mathbb{Z}_q^{478}$	$\lfloor 478 \cdot \log_2 12289 \rfloor = 6493$	6492
$\mathbb{Z}_q^{962}$	$\lfloor 962 \cdot \log_2 12289 \rfloor = 13068$	13067

Table 2: Efficiency of the encode / decode algorithm. The numbers in the last 2 columns represent the number of bits that can be encoded into the specified domain.

bit strings into $\mathbb{Z}_q^{478}$ and $\mathbb{Z}_q^{962}$. If the message being signed is not long enough to fit into $\mathbb{Z}_q^{n-\lambda-\gamma}$, however, then the saving will be smaller. In particular, if the message being signed is shorter than 585 bytes (resp. 1199 bytes), then using FALCON-MRM-512 (resp. FALCON-MRM-1024) is worse than using plain FALCON.

1.4 Encoding bit strings into $\mathbb{Z}_q^k$

The encoding and decoding algorithms convert bit strings to vectors over $\mathbb{Z}_q$, where each element in $\mathbb{Z}_q$ is represented by a positive integer between 0 and $q-1$. For efficient packing, we would like to find values of x, y such that q^x is only a little bit larger than 2^y, which will allow us to pack y bits into $\mathbb{Z}_q^x$ with minimal waste. We do this encoding by interpreting the y bits as an integer and then converting it into base q. The parameters $x = 12, y = 163$ produce a rather good encoding with $x = 2, y = 27$ being only a slightly worse one. Our procedure thus first goes through consecutive blocks of 163 bits, and converts them to vectors in $\mathbb{Z}_q^{12}$. Once there are less than 163 bits left, it goes through blocks of 27 bits and converts them to vectors in $\mathbb{Z}_q^2$. The algorithms in Section 1.4.1 give pseudo-code for the encoding and decoding procedures.

Maximum encoded length. Table 2 shows the maximum bit-string length that can be encoded using the above method into $\mathbb{Z}_q^{478}$ and $\mathbb{Z}_q^{962}$, respectively. Note that in both cases, it is just one bit less than the information-theoretic maximum. Letting $B(k)$ denote the number of bits encodable into $\mathbb{Z}_q^k$ using our encoding procedure (i.e. 6492 and 13067), the two values of k (478 and 962) satisfy $B(k) = 163 \cdot \lfloor k/12 \rfloor + 27 \cdot \lfloor (k \bmod 12)/2 \rfloor$. This ensures that encoding consumes exactly k elements with no partial blocks.

Padding. The encoding and decoding algorithms presented here operate on bit strings of precisely $B(k)$ bits. Padding and encoding of shorter bit-strings are left to the application developer.

1.4.1 Encoding and Decoding Algorithms

Bit and integer conversions. For a bit string $b_0 b_1 \cdots b_{\ell-1}$, we define

$$\mathsf{BitsToInt}(b_0, \ldots, b_{\ell-1}) = \sum_{i=0}^{\ell-1} b_i \cdot 2^{\ell-1-i}$$

(big-endian convention). The inverse $\mathsf{IntToBits}(x, \ell)$ returns the unique ℓ-bit big-endian representation $(b_0, \ldots, b_{\ell-1})$ of x when $0 \le x < 2^\ell$.

Algorithm 5 IntToBase(x, q, k) – Convert integer to base-q representation

Input: Integer $x \in [0, q^k)$, modulus q, output length k
Output: $(z_0, z_1, \ldots, z_{k-1}) \in \mathbb{Z}_q^k$ such that $x = \sum_{i=0}^{k-1} z_i \cdot q^i$

```
for i = 0 to k − 1 do
    z_i ← x mod q
    x ← ⌊x/q⌋
end for
return (z_0, z_1, ..., z_{k−1})
```

Algorithm 6 BaseToInt$(z_0, \ldots, z_{k-1}, q)$ – Convert base-q representation to integer

Input: $(z_0, z_1, \ldots, z_{k-1}) \in \mathbb{Z}_q^k$, modulus q
Output: Integer $x = \sum_{i=0}^{k-1} z_i \cdot q^i$

```
x ← 0
for i = k − 1 downto 0 do
    x ← x · q + z_i
end for
return x
```

Algorithm 7 Encode(μ, k) – Encode bit string into $\mathbb{Z}_q^k$

Input: Bit string $\mu \in \{0,1\}^B$ where B is the maximum encodable length for $\mathbb{Z}_q^k$
Output: Vector $z \in \mathbb{Z}_q^k$

```
i ← 0                                          ▷ Index into input bits μ
j ← 0                                          ▷ Index into output vector z
while B − i ≥ 163 do                           ▷ Encode full 163-bit blocks
    x ← BitsToInt(μ_i, μ_{i+1}, ..., μ_{i+162})
    (z_j, z_{j+1}, ..., z_{j+11}) ← IntToBase(x, q, 12)
    i ← i + 163
    j ← j + 12
end while
while B − i ≥ 27 do                            ▷ Encode remaining 27-bit blocks
    x ← BitsToInt(μ_i, μ_{i+1}, ..., μ_{i+26})
    (z_j, z_{j+1}) ← IntToBase(x, q, 2)
    i ← i + 27
    j ← j + 2
end while
if i ≠ B then                                  ▷ Partial block indicates invalid input length
    return ⊥
end if
return (z_0, z_1, ..., z_{k−1})
```

Algorithm 8 Decode(z, B) – Decode $\mathbb{Z}_q^k$ vector into bit string

Input: Vector $z \in \mathbb{Z}_q^k$, bit-length B of padded message
Output: Bit string $\mu \in \{0,1\}^B$ or $\perp$ on invalid encoding

```
i ← 0                                              ▷ Index into output bits μ
j ← 0                                              ▷ Index into input vector z
r ← B                                              ▷ Remaining bits to decode
while r ≥ 163 do                                   ▷ Decode full 163-bit blocks
    x ← BaseToInt(z_j, z_{j+1}, ..., z_{j+11}, q)
    if x ≥ 2^163 then                              ▷ Validity check: integer out of range
        return ⊥
    end if
    (μ_i, μ_{i+1}, ..., μ_{i+162}) ← IntToBits(x, 163)
    j ← j + 12
    i ← i + 163
    r ← r − 163
end while
while r ≥ 27 do                                    ▷ Decode full 27-bit blocks
    x ← BaseToInt(z_j, z_{j+1}, q)
    if x ≥ 2^27 then                               ▷ Validity check: integer out of range
        return ⊥
    end if
    (μ_i, μ_{i+1}, ..., μ_{i+26}) ← IntToBits(x, 27)
    j ← j + 2
    i ← i + 27
    r ← r − 27
end while
if r ≠ 0 then                                      ▷ Partial block indicates invalid B
    return ⊥
end if
return (μ_0, μ_1, ..., μ_{B−1})
```

1.5 A note on Ring Signatures

A ring signature scheme allows any member of an N-member group to produce a signature which can only be traced to the group, but not to the individual user.[4] One motivating application for this construction is the deniability feature in the Signal Protocol; Signal Messenger is interested in using Ring-FALCON-MRM in a future update to the Signal Protocol.

The work of [2] gave a simple transformation of a FALCON signature scheme to a ring-signature. The idea is that if every ring member $i \in [N]$ has a FALCON public key h_i and a trapdoor T_i allowing him to perform trapdoor sampling (i.e. signing messages using FALCON), then these members can create a signature on a message M with salt ρ as $s_0, s_1, \ldots, s_N$ satisfying

[4] Unlike a related concept known as "group signatures", a ring signature scheme does not have any central authority, and it's only the individuals themselves who know their secret keys.

$$\|(s_0, s_1, \ldots, s_N)\| \leq \beta \text{ and } s_0 + \sum_{j \in [N]} h_j s_j = \mathcal{H}(M, \rho).^5$$

The idea is that user i can create the s_j for $j \in [N] \setminus i$ from the discrete Gaussian distribution D_σ, and then use the FALCON sampler with his trapdoor T_i to create a "FALCON" signature s_0, s_i satisfying

$$s_0 + h_i s_i = \mathcal{H}(M, \rho) - \sum_{j \in [N], j \neq i} h_j s_j.$$

As in the FALCON signature, one does not need to send a part of the signature – s_0 – in the ring version because it can be recovered from the other s_i. And as in the message-recovery version of FALCON we can instead choose to send s_0, but recover a part of the message M, allowing for the exact same possible savings. We sketch the algorithms for the ring versions and message recovery ring versions of the FALCON signature below.

Algorithm 9 Ring-FALCON-sign

Input: $M \in \{0,1\}^*, i \in [N], T_i$
Output: $s_1, \ldots, s_N, \rho$
  repeat
    $\rho \leftarrow \{0,1\}^{320}$
    $c \leftarrow \mathcal{H}(\rho, M)$
    $\forall j \in [N] \setminus i, s_j \leftarrow D_\sigma$ ▷ Using a discrete Gaussian sampler
    $\tilde{c} \leftarrow c - \sum_{j \in [N], j \neq i} h_j s_j$
    $s_0, s_i \leftarrow D_\sigma$ s.t. $s_0 + s_i h_i = \tilde{c}$ ▷ Using the FALCON sampler with T_i
  until $\|(s_0, s_1, \ldots, s_N)\| \leq \beta$
  return $\rho, s_1, \ldots, s_N$

Algorithm 10 Ring-FALCON-verify

Input: $M \in \{0,1\}^*, s_1, \ldots, s_N, \rho \in \{0,1\}^{320}$
Output: True / False
  $c \leftarrow \mathcal{H}(\rho, M)$
  $s_0 \leftarrow c - \sum_{j=1}^{N} h_j s_j$
  if $\|(s_0, s_1, \ldots, s_N)\| > \beta$ **then return** False
  end if
  return True

[5]The security of FALCON and its ring version depends on the value of β – the smaller β is, the more secure the scheme. As the number of parties N grows, the value of β also grows, making the scheme less secure if all the other parameters remain the same. For N=2, which is the case most interesting for Signal, the security loss is only several bits and so one can use the same parameters as FALCON for this instantiation. See [2] and [3] for more details.

Algorithm 11 Ring-FALCON-MRM-sign

Input: $M = (M_1, M_2) \in (\mathbb{Z}_q^{n-\lambda-\gamma}, \{0,1\}^*), i \in [N], T_i$
Output: $s_0, s_1, \ldots, s_N, \rho$
  repeat
    $\rho \leftarrow \mathbb{Z}_q^{\gamma}$
    $c = (c_1, c_2) \leftarrow (\mathcal{H}_1(\rho, M), \mathcal{H}_2(c_1) + (\rho, M_1) \bmod q)$
    $\forall j \in [N] \setminus i, s_j \leftarrow D_\sigma$ ▷ Using a discrete Gaussian sampler
    $\tilde{c} \leftarrow c - \sum_{j \in [N], j \neq i} h_j s_j$
    $s_0, s_i \leftarrow D_\sigma$ s.t. $s_0 + s_i h_i = \tilde{c}$ ▷ Using the FALCON sampler with T_i
  until $\|(s_0, s_1, \ldots, s_N)\| \leq \beta$
  return $s_0, s_1, \ldots, s_N, M_2$

Algorithm 12 Ring-FALCON-MRM-verify

Input: $M_2, s_0, s_1, \ldots, s_N$
Output: M_1 or $\perp$ (reject)
  if $\|(s_0, s_1, \ldots, s_N)\| > \beta$ **then return** $\perp$
  end if
  $(c_1, c_2) = c \leftarrow s_0 + \sum_{j \in [N]} h_j s_j$ ▷ $(c_1, c_2) \in \left(\mathbb{Z}_q^{\lambda}, \mathbb{Z}_q^{n-\lambda}\right)$
  $(\rho, M_1) \leftarrow c_2 - \mathcal{H}_2(c_1) \bmod q$ ▷ $(\rho, M_1) \in \left(\mathbb{Z}_q^{\gamma}, \mathbb{Z}_q^{n-\lambda-\gamma}\right)$
  if $\mathcal{H}_1(\rho, (M_1, M_2)) \neq c_1$ **then return** $\perp$
  end if
  return M_1

References

[1] R. del Pino, V. Lyubashevsky, and D. Pointcheval. The whole is less than the sum of its parts: Constructing more efficient lattice-based AKEs. In V. Zikas and R. De Prisco, editors, *SCN 16*, volume 9841 of *LNCS*, pages 273–291, Amalfi, Italy, Aug. 31 – Sept. 2, 2016. Springer, Cham, Switzerland.

[2] P. Gajland, J. Janneck, and E. Kiltz. Ring signatures for deniable AKEM: Gandalf's fellowship. In L. Reyzin and D. Stebila, editors, *CRYPTO 2024, Part I*, volume 14920 of *LNCS*, pages 305–338, Santa Barbara, CA, USA, Aug. 18–22, 2024. Springer, Cham, Switzerland.

[3] K. Hashimoto, S. Katsumata, and T. Wiggers. Bundled authenticated key exchange: A concrete treatment of (post-quantum) Signal's handshake protocol. In *USENIX*, 2025.

[4] T. Prest, P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Ricosset, G. Seiler, W. Whyte, , and Z. Zhang. FALCON. Technical report, 2017. https://falcon-sign.info/falcon.pdf.