FrodoKEM
Learning With Errors Key Encapsulation

Annex on FrodoKEM updates

April 18, 2023

Contents

1 Introduction

The FrodoKEM preliminary standardization proposal (2023/03/14) [2] differs from the NIST Round 3 version of FrodoKEM [1] in the following ways:

- The NIST Round 3 version is now called eFrodoKEM. This version is suitable for applications in which the number of ciphertexts produced relative to any single public key is fairly small.
- A modified version of eFrodoKEM, simply called FrodoKEM, is suitable for applications in which many ciphertexts might be produced relative to a single public key. More specifically, to address certain multi-ciphertext attacks that were outside the scope of FrodoKEM's original IND-CCA security target, this version of FrodoKEM doubles the length of the $\mathsf{seed}_{\mathbf{SE}}$ value, and incorporates a public random salt value into encapsulation.

This short annex provides a summary of the algorithmic changes introduced to the FrodoKEM variant, and a security analysis of these changes. This document is intended to be read in conjunction with the FrodoKEM preliminary standardization proposal (2023/03/14) [2] and the FrodoKEM NIST Round 3 specification document (June 4, 2021 update) [1].

2 Multi-target and multi-ciphertext attacks

Multi-target security. Multi-target attacks aim to break security against one of N public keys. FrodoKEM's primary security target of IND-CCA security considers only a single public key, hence multi-target attacks fall outside of its scope (though multi-target security provably follows by a routine hybrid argument, with looseness linear in the number of keys). However, multi-target security can be a desirable feature in some settings. The security analysis in the FrodoKEM specification document (Section 5.1) does not formally cover security in the multi-target setting. However, in order to reduce the risk of batch attacks targeting multiple keys, FrodoKEM includes the hashed value of the public key **pkh** in the computation of the random bit strings **r** (Algorithm 2, line 3).

Multi-ciphertext security. Multi-ciphertext attacks target a single public key, but aim to break one of N ciphertexts produced under that key. FrodoKEM's primary security target of IND-CCA security considers only a single challenge ciphertext, hence multi-ciphertext attacks fall outside of its scope (though multi-ciphertext security provably follows by a routine hybrid argument, with looseness linear in the number of ciphertexts). However, multi-ciphertext security can be a desirable feature in some settings.

A multi-ciphertext attack was identified against earlier versions of FrodoKEM, which we summarize here.[1] The pseudorandom values $\mathsf{seed}_{\mathbf{SE}}$ and **k** computed on line 3 of FrodoKEM.Encaps (Algorithm 2) were determined entirely by the public key hash **pkh** and seed value (message) μ. Consequently, the encapsulation secret key $\mathbf{S}'$ and $\mathbf{E}'$ were also determined entirely by **pkh** and μ (lines 4–6 of Algorithm 2). Since μ is from a space having bit length len_μ, an adversary could run a multi-ciphertext attack for a specific public key via a brute-force search on this space.

More specifically, an attacker could collect N challenge ciphertexts encrypted under a given public key, and do a brute-force search through M different values of μ, seeking a match with one of the ciphertexts. This would reveal the corresponding shared secret of the ciphertext. For each of the N challenge ciphertexts, there is approximately an $M/2^{\mathsf{len}_\mu}$ probability that this ciphertext used the same seed μ as one of the M the attacker generated. Therefore, with work proportional to roughly $M + N$, the attacker succeeds in breaking at least one of the challenge ciphertexts with probability roughly $MN/2^{\mathsf{len}_\mu}$ (when $MN \leq 2^{\mathsf{len}_\mu}$, as is typical). For, e.g., the Level-1 value $\mathsf{len}_\mu = 128$ and large but technologically feasible values of N and M, this probability may be considered unacceptably large.

While none of the above breaks FrodoKEM at any of the targeted NIST security levels, it comes closer to those levels than the LWE security estimates given in the FrodoKEM specification document (Section 5.2.4, Table 11).

To mitigate the risk of multi-ciphertext attacks, we revised FrodoKEM to concatenate a fresh, uniformly random, public value salt of bit length $\ell = \mathsf{len}_{\mathsf{salt}}$ to the message μ (Algorithm 2, line 3). The salt value is made public as part of the ciphertext output by encapsulation. This change has negligible effect on performance (about 1% or less overhead in runtime and ciphertext size).

[1] Thanks to Ray Perlner of NIST for pointing out this attack and suggesting the salt-based countermeasure (personal communication, August 2021).

3 Updated FrodoKEM algorithm specification

This section gives the algorithm specifications for FrodoKEM, a key encapsulation mechanism that is derived from FrodoPKE by applying the $\mathsf{FO}^{\not\perp\prime}$ transform. Compared with the NIST Round 3 version of FrodoKEM, this version includes a salt of bit length $\mathsf{len}_{\mathsf{salt}}$, as highlighted in red in the FrodoKEM.Encaps and FrodoKEM.Decaps algorithms shown below.

Algorithm 1 FrodoKEM.KeyGen.

Input: None.
Output: Key pair (pk, sk') with $pk \in \{0,1\}^{\mathsf{len}_{\mathsf{seed}_\mathbf{A}}+D\cdot n\cdot\overline{n}}$, $sk' \in \{0,1\}^{\mathsf{len}_\mathbf{s}+\mathsf{len}_{\mathsf{seed}_\mathbf{A}}+D\cdot n\cdot\overline{n}} \times \mathbb{Z}_q^{\overline{n}\times n} \times \{0,1\}^{\mathsf{len}_{\mathbf{pkh}}}$.

1: Choose uniformly random seeds $\mathbf{s}\|\mathsf{seed}_{\mathbf{SE}}\|\mathbf{z} \leftarrow\!\!\$\ U(\{0,1\}^{\mathsf{len}_\mathbf{s}+\mathsf{len}_{\mathsf{seed}_{\mathbf{SE}}}+\mathsf{len}_\mathbf{z}})$
2: Generate pseudorandom seed $\mathsf{seed}_\mathbf{A} \leftarrow \mathrm{SHAKE}(\mathbf{z}, \mathsf{len}_{\mathsf{seed}_\mathbf{A}})$
3: Generate the matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times n}$ via $\mathbf{A} \leftarrow \mathsf{Frodo.Gen}(\mathsf{seed}_\mathbf{A})$
4: Generate pseudorandom bit string $(\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, \ldots, \mathbf{r}^{(2n\overline{n}-1)}) \leftarrow \mathrm{SHAKE}(\mathtt{0x5F}\|\mathsf{seed}_{\mathbf{SE}}, 2n\overline{n}\cdot\mathsf{len}_\chi)$
5: Sample error matrix $\mathbf{S}^\mathrm{T} \leftarrow \mathsf{Frodo.SampleMatrix}((\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, \ldots, \mathbf{r}^{(n\overline{n}-1)}), \overline{n}, n, T_\chi)$
6: Sample error matrix $\mathbf{E} \leftarrow \mathsf{Frodo.SampleMatrix}((\mathbf{r}^{(n\overline{n})}, \mathbf{r}^{(n\overline{n}+1)}, \ldots, \mathbf{r}^{(2n\overline{n}-1)}), n, \overline{n}, T_\chi)$
7: Compute $\mathbf{B} \leftarrow \mathbf{AS} + \mathbf{E}$
8: Compute $\mathbf{b} \leftarrow \mathsf{Frodo.Pack}(\mathbf{B})$
9: Compute $\mathbf{pkh} \leftarrow \mathrm{SHAKE}(\mathsf{seed}_\mathbf{A}\|\mathbf{b}, \mathsf{len}_{\mathbf{pkh}})$
10: **return** public key $pk \leftarrow \mathsf{seed}_\mathbf{A}\|\mathbf{b}$ and secret key $sk' \leftarrow (\mathbf{s}\|\mathsf{seed}_\mathbf{A}\|\mathbf{b}, \mathbf{S}^\mathrm{T}, \mathbf{pkh})$

Algorithm 2 FrodoKEM.Encaps.

Input: Public key $pk = \mathsf{seed}_\mathbf{A}\|\mathbf{b} \in \{0,1\}^{\mathsf{len}_{\mathsf{seed}_\mathbf{A}}+D\cdot n\cdot\overline{n}}$.
Output: Ciphertext $\mathbf{c}_1\|\mathbf{c}_2\|\mathsf{salt} \in \{0,1\}^{(\overline{m}\cdot n+\overline{m}\cdot\overline{n})D+\mathsf{len}_{\mathsf{salt}}}$ and shared secret $\mathbf{ss} \in \{0,1\}^{\mathsf{len}_{\mathbf{ss}}}$.

1: Choose uniformly random values $\mu \leftarrow\!\!\$\ U(\{0,1\}^{\mathsf{len}_\mu})$ and $\mathsf{salt} \leftarrow\!\!\$\ U(\{0,1\}^{\mathsf{len}_{\mathsf{salt}}})$
2: Compute $\mathbf{pkh} \leftarrow \mathrm{SHAKE}(pk, \mathsf{len}_{\mathbf{pkh}})$
3: Generate pseudorandom values $\mathsf{seed}_{\mathbf{SE}}\|\mathbf{k} \leftarrow \mathrm{SHAKE}(\mathbf{pkh}\|\mu\|\mathsf{salt}, \mathsf{len}_{\mathsf{seed}_{\mathbf{SE}}} + \mathsf{len}_\mathbf{k})$
4: Generate pseudorandom bit string $(\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, \ldots, \mathbf{r}^{(2\overline{m}n+\overline{m}\overline{n}-1)}) \leftarrow \mathrm{SHAKE}(\mathtt{0x96}\|\mathsf{seed}_{\mathbf{SE}}, (2\overline{m}n + \overline{m}\overline{n})\cdot \mathsf{len}_\chi)$
5: Sample error matrix $\mathbf{S}' \leftarrow \mathsf{Frodo.SampleMatrix}((\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, \ldots, \mathbf{r}^{(\overline{m}n-1)}), \overline{m}, n, T_\chi)$
6: Sample error matrix $\mathbf{E}' \leftarrow \mathsf{Frodo.SampleMatrix}((\mathbf{r}^{(\overline{m}n)}, \mathbf{r}^{(\overline{m}n+1)}, \ldots, \mathbf{r}^{(2\overline{m}n-1)}), \overline{m}, n, T_\chi)$
7: Generate $\mathbf{A} \leftarrow \mathsf{Frodo.Gen}(\mathsf{seed}_\mathbf{A})$
8: Compute $\mathbf{B}' \leftarrow \mathbf{S}'\mathbf{A} + \mathbf{E}'$
9: Compute $\mathbf{c}_1 \leftarrow \mathsf{Frodo.Pack}(\mathbf{B}')$
10: Sample error matrix $\mathbf{E}'' \leftarrow \mathsf{Frodo.SampleMatrix}((\mathbf{r}^{(2\overline{m}n)}, \mathbf{r}^{(2\overline{m}n+1)}, \ldots, \mathbf{r}^{(2\overline{m}n+\overline{m}\overline{n}-1)}), \overline{m}, \overline{n}, T_\chi)$
11: Compute $\mathbf{B} \leftarrow \mathsf{Frodo.Unpack}(\mathbf{b}, n, \overline{n})$
12: Compute $\mathbf{V} \leftarrow \mathbf{S}'\mathbf{B} + \mathbf{E}''$
13: Compute $\mathbf{C} \leftarrow \mathbf{V} + \mathsf{Frodo.Encode}(\mu)$
14: Compute $\mathbf{c}_2 \leftarrow \mathsf{Frodo.Pack}(\mathbf{C})$
15: Compute $\mathbf{ss} \leftarrow \mathrm{SHAKE}(\mathbf{c}_1\|\mathbf{c}_2\|\mathsf{salt}\|\mathbf{k}, \mathsf{len}_{\mathbf{ss}})$
16: **return** ciphertext $\mathbf{c}_1\|\mathbf{c}_2\|\mathsf{salt}$ and shared secret $\mathbf{ss}$

Algorithm 3 FrodoKEM.Decaps.

Input: Ciphertext $\mathbf{c}_1 \| \mathbf{c}_2 \| \mathsf{salt} \in \{0,1\}^{(\overline{m} \cdot n + \overline{m} \cdot \overline{n})D + \mathsf{len}_{\mathsf{salt}}}$, secret key $sk' = (\mathbf{s} \| \mathsf{seed}_{\mathbf{A}} \| \mathbf{b}, \mathbf{S}^{\mathrm{T}}, \mathbf{pkh}) \in \{0,1\}^{\mathsf{len}_{\mathbf{s}} + \mathsf{len}_{\mathsf{seed}_{\mathbf{A}}} + D \cdot n \cdot \overline{n}} \times \mathbb{Z}_q^{\overline{n} \times n} \times \{0,1\}^{\mathsf{len}_{\mathbf{pkh}}}$.
Output: Shared secret $\mathbf{ss} \in \{0,1\}^{\mathsf{len}_{\mathbf{ss}}}$.

1: $\mathbf{B}' \leftarrow \mathsf{Frodo.Unpack}(\mathbf{c}_1, \overline{m}, n)$
2: $\mathbf{C} \leftarrow \mathsf{Frodo.Unpack}(\mathbf{c}_2, \overline{m}, \overline{n})$
3: Compute $\mathbf{M} \leftarrow \mathbf{C} - \mathbf{B}'\mathbf{S}$
4: Compute $\mu' \leftarrow \mathsf{Frodo.Decode}(\mathbf{M})$
5: Parse $pk \leftarrow \mathsf{seed}_{\mathbf{A}} \| \mathbf{b}$
6: Generate pseudorandom values $\mathsf{seed}_{\mathbf{SE}}{}' \| \mathbf{k}' \leftarrow \mathrm{SHAKE}(\mathbf{pkh} \| \mu' \| \mathsf{salt}, \mathsf{len}_{\mathsf{seed}_{\mathbf{SE}}} + \mathsf{len}_{\mathbf{k}})$
7: Generate pseudorandom bit string $(\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, \ldots, \mathbf{r}^{(2\overline{m}n + \overline{m}\overline{n} - 1)}) \leftarrow \mathrm{SHAKE}(\mathtt{0x96} \| \mathsf{seed}_{\mathbf{SE}}{}', (2\overline{m}n + \overline{m}\overline{n}) \cdot \mathsf{len}_{\chi})$
8: Sample error matrix $\mathbf{S}' \leftarrow \mathsf{Frodo.SampleMatrix}((\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, \ldots, \mathbf{r}^{(\overline{m}n-1)}), \overline{m}, n, T_\chi)$
9: Sample error matrix $\mathbf{E}' \leftarrow \mathsf{Frodo.SampleMatrix}((\mathbf{r}^{(\overline{m}n)}, \mathbf{r}^{(\overline{m}n+1)}, \ldots, \mathbf{r}^{(2\overline{m}n-1)}), \overline{m}, n, T_\chi)$
10: Generate $\mathbf{A} \leftarrow \mathsf{Frodo.Gen}(\mathsf{seed}_{\mathbf{A}})$
11: Compute $\mathbf{B}'' \leftarrow \mathbf{S}'\mathbf{A} + \mathbf{E}'$
12: Sample error matrix $\mathbf{E}'' \leftarrow \mathsf{Frodo.SampleMatrix}((\mathbf{r}^{(2\overline{m}n)}, \mathbf{r}^{(2\overline{m}n+1)}, \ldots, \mathbf{r}^{(2\overline{m}n+\overline{m}\overline{n}-1)}), \overline{m}, \overline{n}, T_\chi)$
13: Compute $\mathbf{B} \leftarrow \mathsf{Frodo.Unpack}(\mathbf{b}, n, \overline{n})$
14: Compute $\mathbf{V} \leftarrow \mathbf{S}'\mathbf{B} + \mathbf{E}''$
15: Compute $\mathbf{C}' \leftarrow \mathbf{V} + \mathsf{Frodo.Encode}(\mu')$
16: (in constant time) $\overline{\mathbf{k}} \leftarrow \mathbf{k}'$ if $(\mathbf{B}' \| \mathbf{C} = \mathbf{B}'' \| \mathbf{C}')$ else $\overline{\mathbf{k}} \leftarrow \mathbf{s}$
17: Compute $\mathbf{ss} \leftarrow \mathrm{SHAKE}(\mathbf{c}_1 \| \mathbf{c}_2 \| \mathsf{salt} \| \overline{\mathbf{k}}, \mathsf{len}_{\mathbf{ss}})$
18: **return** shared secret $\mathbf{ss}$

3.1 Summary of parameters

The lengths of the salt and seed parameters for FrodoKEM and eFrodoKEM are shown in Table 1. In eFrodoKEM (which is identical to the NIST Round 3 version of FrodoKEM), there is no salt (i.e., $\mathsf{len}_{\mathsf{salt}} = 0$) and the $\mathsf{seed}_{\mathbf{SE}}$ value has length equal to the security parameter ℓ. In the new version of FrodoKEM, the salt and the $\mathsf{seed}_{\mathbf{SE}}$ value have lengths equal to twice the security parameter. Table 2 shows the resulting input/output sizes, in bytes.

Table 1: **Salt and seed bitlength parameters**

	Frodo-640	Frodo-976	Frodo-1344
$\mathsf{len}_{\mathsf{salt}} = 2\ell$	256	384	512
$\mathsf{len}_{\mathsf{seed}_{\mathbf{SE}}} = 2\ell$	256	384	512
	eFrodo-640	eFrodo-976	eFrodo-1344
$\mathsf{len}_{\mathsf{salt}} = 0$	0	0	0
$\mathsf{len}_{\mathsf{seed}_{\mathbf{SE}}} = \ell$	128	192	256

Table 2: **Size (in bytes) of inputs and outputs.**

Scheme	**secret key** sk	**public key** pk	**ciphertext** c	**shared secret** **ss**
FrodoKEM-640	19,888	9,616	9,752	16
FrodoKEM-976	31,296	15,632	15,792	24
FrodoKEM-1344	43,088	21,520	21,696	32
eFrodoKEM-640	19,888	9,616	9,720	16
eFrodoKEM-976	31,296	15,632	15,744	24
eFrodoKEM-1344	43,088	21,520	21,632	32

4 Security analysis

Since eFrodoKEM is identical to the NIST Round 3 version of FrodoKEM, the analysis given in the FrodoKEM NIST Round 3 specification document continues to apply for eFrodoKEM.

The new version FrodoKEM requires additional analysis. FrodoKEM is constructed from the IND-CPA-secure FrodoPKE public key encryption scheme by applying a variant of the Fujisaji–Okamoto transform. The NIST Round 3 version of FrodoKEM used a variant of the $\mathsf{FO}^{\not\perp}$ transform by Hofheinz, Hövelmanns, and Kiltz (HHK) [3]. The new version of FrodoKEM can be viewed as being constructed from a "salted" version of the previously used transform. Section 4.1 presents this modification to the FO transform, and Section 4.2 and Section 4.3 describe how the relevant security theorems from [3, 4] adapt to the salted version. FrodoKEM continues to use the same LWE parameters as before, so the analysis of the lattice attacks given in the FrodoKEM NIST Round 3 specification document continues to apply.

Our analysis shows that the salted version of the FO transform yields IND-CCA security, just as in the previous version of FrodoKEM, and with the same concrete security bounds. We caution, however, that we have not *proved* that the salted version achieves *stronger* security against multi-ciphertext attacks than the unsalted version, even though the salted version appears to thwart the specific multi-ciphertext attack described in Section 2. Obtaining a (tighter) proof of multi-ciphertext security seems to require adapting the results of [3] to the multi-ciphertext setting, which looks non-trivial. In summary, while the results in Section 4.2 do not prove any stronger IND-CCA security against multi-ciphertext attacks, they at least show that the salted version maintains the prior level of IND-CCA security in the single-ciphertext setting.

4.1 Salted FO transform

Figure 1 shows the "salted" version of the FO transform, which we denote $\mathsf{FO}^{\not\perp\prime}$, highlighting in red the changes compared to the version of the transform used in the FrodoKEM NIST Round 3 specification.

$\mathsf{KEM}^{\not\perp\prime}.\mathrm{KeyGen}()$:

1: $(pk, sk) \leftarrow_{\$} \mathsf{PKE.KeyGen}()$
2: $\mathbf{s} \leftarrow_{\$} \{0,1\}^{\mathsf{len_s}}$
3: $\mathbf{pkh} \leftarrow G_1(pk)$
4: $sk' \leftarrow (sk, \mathbf{s}, pk, \mathbf{pkh})$
5: **return** (pk, sk')

$\mathsf{KEM}^{\not\perp\prime}.\mathrm{Encaps}(pk)$:

1: $\mu \leftarrow_{\$} \mathcal{M}$, $\mathsf{salt} \leftarrow_{\$} \{0,1\}^{\mathsf{len_{salt}}}$
2: $(\mathbf{r}, \mathbf{k}) \leftarrow G_2(G_1(pk) \| \mu \| \mathsf{salt})$
3: $c \leftarrow \mathsf{PKE.Enc}(\mu, pk; \mathbf{r})$
4: $\mathbf{ss} \leftarrow F(c \| \mathsf{salt} \| \mathbf{k})$
5: **return** $(c \| \mathsf{salt}, \mathbf{ss})$

$\mathsf{KEM}^{\not\perp\prime}.\mathrm{Decaps}(c \| \mathsf{salt}, (sk, \mathbf{s}, pk, \mathbf{pkh}))$:

1: $\mu' \leftarrow \mathsf{PKE.Dec}(c, sk)$
2: $(\mathbf{r}', \mathbf{k}') \leftarrow G_2(\mathbf{pkh} \| \mu' \| \mathsf{salt})$
3: $\mathbf{ss}'_0 \leftarrow F(c \| \mathsf{salt} \| \mathbf{k}')$
4: $\mathbf{ss}'_1 \leftarrow F(c \| \mathsf{salt} \| \mathbf{s})$
5: (in constant time) $\mathbf{ss}' \leftarrow \mathbf{ss}'_0$ if $c = \mathsf{PKE.Enc}(\mu', pk; \mathbf{r}')$ else $\mathbf{ss}' \leftarrow \mathbf{ss}'_1$
6: **return** $\mathbf{ss}'$

Figure 1: Salted version $\mathsf{FO}^{\not\perp\prime}$ of the FO transform.

Hofheinz, Hövelmanns, and Kiltz [3] show how to obtain the $\mathsf{FO}^{\not\perp}$ transform in a modular way as the composition of two transforms, $\mathsf{FO}^{\not\perp} = \mathsf{U}^{\not\perp} \circ \mathsf{T}$. The T transform converts an IND-CPA-secure public-key encryption scheme into an OW-PCA-secure PKE, and the $\mathsf{U}^{\not\perp}$ transform converts an OW-PCA-secure PKE into an IND-CCA-secure KEM. We define the analogous "salted" transform $\mathsf{FO}^{\not\perp\prime}$ from Figure 1 as $\mathsf{FO}^{\not\perp\prime} = \mathsf{U}^{\not\perp} \circ \mathsf{T}'$, where T' is a salted version of the T transform as defined in Figure 2. We show in the subsequent section that T' similarly converts an IND-CPA-secure public key encryption scheme into an OW-PCA-secure PKE (with the same tight security bounds as for the T transform), allowing us to rely on existing results from HHK to complete the IND-CCA security analysis.

$\underline{\mathsf{PKE}_1.\mathrm{KeyGen}()}$:
1: **return** $\mathsf{PKE}.\mathrm{KeyGen}()$

$\underline{\mathsf{PKE}_1.\mathrm{Enc}(\mu, pk)}$:
1: $\mathsf{salt} \leftarrow\!\!\$\ U(\{0,1\}^{\mathsf{len}_{\mathsf{salt}}})$
2: $r \leftarrow G_2(\mu \| \mathsf{salt})$
3: $c \leftarrow \mathsf{PKE}.\mathrm{Enc}(\mu, pk; r)$
4: **return** $c \| \mathsf{salt}$

$\underline{\mathsf{PKE}_1.\mathrm{Dec}(c \| \mathsf{salt}, sk)}$:
1: $\mu' \leftarrow \mathsf{PKE}.\mathrm{Dec}(c, sk)$
2: **if** $\mu' = \bot$ or $c \neq \mathsf{PKE}.\mathrm{Enc}(\mu', pk; G_2(\mu' \| \mathsf{salt}))$ **then**
3: **return** $\bot$
4: **else**
5: **return** μ'

Figure 2: Salted T' construction of a public-key encryption scheme $\mathsf{PKE}_1 = \mathsf{T}'[\mathsf{PKE}, G_2]$ from a public-key encryption scheme PKE and hash function G_2.

4.2 IND-CCA security of $\mathsf{FO}^{\not\perp\prime}$ in the classical random-oracle model

Our main goal here is to give a slight extension of HHK's Theorem 3.2, showing that the "salted" version of their T transform, which we call T', converts the IND-CPA-secure public-key encryption scheme PKE_Q into an OW-PCA-secure public-key encryption scheme (in the random-oracle model).

To complete the IND-CCA analysis, we then apply the remaining steps exactly as in Section 5.1.1 of the FrodoKEM NIST Round 3 specification:

2. We apply distribution substitution for the OW-PCA security experiment (which represents a search problem), to switch from distribution Q to P.
3. Finally, we apply HHK's Theorem 3.4, which shows that their $\mathsf{U}^{\not\perp}$ transform converts the OW-PCA-secure public-key encryption scheme from the previous step into an IND-CCA-secure KEM (in the random-oracle model).

Step 1: IND-CPA PKE to OW-PCA PKE_1. For self-containment, we recall the definition of OW-PCA, following the presentation of Hofheinz et al. [3].

Definition 4.1 (OW-PCA for PKE [5]). Let PKE be a public-key encryption scheme with message space $\mathcal{M}$ and let $\mathcal{A}$ be an algorithm. The OW-PCA security experiment for $\mathcal{A}$ attacking PKE is $\mathrm{Exp}^{\mathsf{ow\text{-}pca}}_{\mathsf{PKE}}(\mathcal{A})$ from Figure 3. The advantage of $\mathcal{A}$ in the experiment is

$$\mathrm{Adv}^{\mathsf{ow\text{-}pca}}_{\mathsf{PKE}}(\mathcal{A}) := \Pr\left[\mathrm{Exp}^{\mathsf{ow\text{-}pca}}_{\mathsf{PKE}}(\mathcal{A}) \Rightarrow 1\right].$$

Experiment $\mathrm{Exp}^{\mathsf{ow\text{-}pca}}_{\mathsf{PKE}}(\mathcal{A})$:
1: $(pk, sk) \leftarrow \mathsf{PKE}.\mathrm{KeyGen}()$
2: $m \leftarrow\!\!\$\ \mathcal{M}$
3: $c^* \leftarrow \mathsf{PKE}.\mathrm{Enc}(m, pk)$
4: $m' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{Pco}}(\cdot,\cdot)}(pk, c^*)$
5: **return** $\mathcal{O}_{\mathrm{Pco}}(m', c^*)$

Oracle $\mathcal{O}_{\mathrm{Pco}}(m, c)$:
1: **if** $\mathsf{PKE}.\mathrm{Dec}(c, sk) = m$ **then**
2: **return** 1
3: **else**
4: **return** 0

Figure 3: Security experiment for OW-PCA.

The T transform of HHK converts a public-key encryption scheme PKE to a (deterministic) public-key encryption scheme. Figure 2 defines a slight extension of this transform, called T', which includes a random public "salt" in the ciphertext and hash input.[2] HHK's Theorem 3.2 tightly establishes the OW-PCVA-security of $\mathsf{T}[\mathsf{PKE}, G_2]$ under, among others, the assumption that PKE is IND-CPA secure and γ-spread. (In the OW-PCVA security game, the attacker additionally has a ciphertext-validity oracle, which checks whether a queried ciphertext has a valid decryption.) However, they note that OW-PCA security follows (tightly) *without*

[2]The T' transform specializes to the T transform simply by taking $\mathsf{len}_{\mathsf{salt}} = 0$. Unlike the T transform, the T' transform does not yield a *deterministic* encryption algorithm. However, this has no effect on the cited security theorems.

the γ-spread assumption, because in the security bounds γ-spreadness is relevant only to ciphertext-validity queries. We observe that these claims also hold (with the same security bounds) for the "salted" version $\mathsf{PKE}_1 = \mathsf{T}'[\mathsf{PKE}, G_2]$, by straightforward adaptation of the proof; see Appendix A for details. The formal statement is as follows.

Lemma 4.2 ([3], Theorem 3.2; salted, OW-PCA version). *Let* PKE *be a δ-correct public-key encryption scheme with message space $\mathcal{M}$. For any* OW-PCA *adversary $\mathcal{A}$ that issues at most q_G queries to the random oracle G_2 and q_P queries to the plaintext-checking oracle, there exists an* IND-CPA *adversary $\mathcal{B}$ such that*

$$\mathrm{Adv}^{\mathsf{ow\text{-}pca}}_{\mathsf{PKE}_1}(\mathcal{A}) \leq q_G \cdot \delta + \frac{2q_G + 1}{|\mathcal{M}|} + 3 \cdot \mathrm{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathsf{PKE}}(\mathcal{B}) \ ,$$

and the running time of $\mathcal{B}$ is about that of $\mathcal{A}$ plus the time needed to simulate the random oracle.

4.3 IND-CCA security of $\mathsf{FO}^{\not\perp\prime}$ in the quantum random-oracle model

Jiang et al. [4] show that the $\mathsf{FO}^{\not\perp}$ transform yields an IND-CCA-secure KEM from an OW-CPA-secure public-key encryption scheme, in the *quantum* random oracle model. As noted above, we apply a slight ("salted") variant $\mathsf{FO}^{\not\perp\prime}$ of the $\mathsf{FO}^{\not\perp}$ transform. The relevant security theorem and proof from [4] adapts straightforwardly to this variant (with the same, but loose, security bounds); see Appendix A for details.

Theorem 4.3 ([4, Theorem 1], salted version). *Let* $\mathsf{PKE} = (\mathrm{KeyGen}, \mathrm{Enc}, \mathrm{Dec})$ *be a δ-correct public-key encryption scheme with message space $\mathcal{M}$. Let G_2 and F be independent random oracles. Let $\mathsf{KEM}^{\not\perp\prime} = \mathsf{FO}^{\not\perp\prime}[\mathsf{PKE}, G_2, F]$ be the KEM obtained by applying the $\mathsf{FO}^{\not\perp\prime} = \mathsf{U}^{\not\perp} \circ \mathsf{T}'$ transform to* PKE*. For any quantum algorithm $\mathcal{A}$ against the* IND-CCA *security of $\mathsf{KEM}^{\not\perp\prime}$ that makes q_F quantum oracle queries to F and q_G quantum oracle queries to G_2, there exists a quantum algorithm $\mathcal{B}$ against the* OW-CPA *security of* PKE *such that*

$$\mathrm{Adv}^{\mathsf{ind\text{-}cca}}_{\mathsf{KEM}^{\not\perp\prime}}(\mathcal{A}) \leq \frac{2q_F}{\sqrt{|\mathcal{M}|}} + 4q_G\sqrt{\delta} + 2(q_G + q_F)\sqrt{\mathrm{Adv}^{\mathsf{ow\text{-}cpa}}_{\mathsf{PKE}}(\mathcal{B})}.$$

Moreover, the running time of $\mathcal{B}$ is about that of $\mathcal{A}$.

References

[1] E. Alkim, J. W. Bos, L. Ducas, P. Longa, I. Mironov, M. Naehrig, V. Nikolaenko, C. Peikert, A. Raghunathan, and D. Stebila. FrodoKEM: Learning With Errors Key Encapsulation, 2017–2023. Specification document available at https://frodokem.org/.

[2] E. Alkim, J. W. Bos, L. Ducas, P. Longa, I. Mironov, M. Naehrig, V. Nikolaenko, C. Peikert, A. Raghunathan, and D. Stebila. FrodoKEM Preliminary Standardization Proposal (submitted to ISO), Mar. 2023. https://frodokem.org/files/FrodoKEM-standard_proposal-20230314.pdf.

[3] D. Hofheinz, K. Hövelmanns, and E. Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In Y. Kalai and L. Reyzin, editors, *TCC 2017: 15th Theory of Cryptography Conference, Part I*, volume 10677 of *Lecture Notes in Computer Science*, pages 341–371. Springer, Heidelberg, Nov. 2017.

[4] H. Jiang, Z. Zhang, L. Chen, H. Wang, and Z. Ma. IND-CCA-secure key encapsulation mechanism in the quantum random oracle model, revisited. In H. Shacham and A. Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 96–125. Springer, Heidelberg, Aug. 2018.

[5] T. Okamoto and D. Pointcheval. REACT: Rapid Enhanced-security Asymmetric Cryptosystem Transform. In D. Naccache, editor, *Topics in Cryptology – CT-RSA 2001*, volume 2020 of *Lecture Notes in Computer Science*, pages 159–175. Springer, Heidelberg, Apr. 2001.

A Proofs for salted FO transform

As noted in Section 4, Lemma 4.2 and Theorem 4.3 were respectively proved in [3] and [4] for the "unsalted" T and $\mathsf{FO}^{\not\perp}$ transforms, in the classical and quantum random-oracle models. Here we show that these results extend to the "salted" versions of the transforms (with no changes to the security bounds), via straightforward adaptations of the original proofs. We describe these changes in a way that is meant to be read alongside the proofs from [3, 4].

A.1 Salted T' transform in the ROM

The proof of Lemma 4.2 for the (salted) T' transform (in the random-oracle model) closely follows that of [3, Theorems 3.1 and 3.2] for the (unsalted) T transform, with the following mechanical, purely syntactic changes.

First, the attack games G_0 through G_3 are modified in the following way. In summary, every input to the random oracle G is of the form $m\|\mathsf{salt}$ (instead of just m), and every ciphertext of the transformed scheme (i.e., challenge ciphertext or adversarially generated oracle query) is of the form $c\|\mathsf{salt}$ (instead of just c), where c is a ciphertext of the underlying encryption scheme. In detail:

- Following the definition of the T' transform (Figure 2), along with m^*, a $\mathsf{salt}^* \in \mathcal{S}$ (the salt domain) is chosen independently and uniformly at random, and $r^* = \mathrm{G}(m^*\|\mathsf{salt}^*)$ is used as the random coins for $c^* = \mathrm{Enc}(pk, m^*; r^*)$. The adversary is given the challenge ciphertext $c^*\|\mathsf{salt}^*$ (instead of just c^*).
- Every input to G is of the form $m\|\mathsf{salt}$ (instead of just m), where the two components are unambiguously parseable and $\mathsf{salt} \in \mathcal{S}$; similarly, $\mathcal{L}_\mathrm{G}$ is a growing set of input-output pairs $(m\|\mathsf{salt}, r)$ for G.
- In game G_3, on query $\mathrm{G}(m\|\mathsf{salt})$, the test "if $m = m^*$" on line 08 (that conditionally triggers the QUERY event, and termination of the game) is replaced with the test "if $m\|\mathsf{salt} = m^*\|\mathsf{salt}^*$".
- PCO inputs (i.e., plaintext-checking queries) are of the form $(m, c\|\mathsf{salt})$, and the inputs for the induced calls to G are augmented with salt.
- CVO inputs (i.e., ciphertext-validity queries) are of the form $c\|\mathsf{salt} \neq c^*\|\mathsf{salt}^*$, and the inputs for the induced calls to G and the inspected pairs in $\mathcal{L}_\mathrm{G}$ are augmented with salt.

The analysis of the games proceeds essentially identically, with only mechanical syntactic changes. Each occurrence of a message m (respectively, c) associated with a query to any of the oracles is replaced by $m\|\mathsf{salt}$ (resp., $c\|\mathsf{salt}$) for the value of salt from the query. Similarly, each occurrence of m^* is replaced by $m^*\|\mathsf{salt}^*$ (where recall that salt^* is chosen at random alongside m^*), and likewise for m_0^*, m_1^* in the tight IND-CPA analysis. Finally, in place of $\mathcal{L}_\mathrm{G}(m)$, the set $\mathcal{L}_\mathrm{G}(m\|\mathsf{salt})$ is defined and used in the natural way.

With these changes, all the same probabilistic analyses and bounds apply to the modified games, mutatis mutandis. In particular, the games G_2 and G_3 are identical until and unless the event QUERY occurs, i.e., the query $\mathrm{G}(m^*\|\mathsf{salt}^*)$ is made. This enables reductions based on the OW-CPA or IND-CPA security of the underlying encryption scheme. Lemma 4.2 therefore follows.

A.2 Salted $\mathsf{FO}^{\not\perp\prime}$ transform in the QROM

The proof of Theorem 4.3 for the (salted) $\mathsf{FO}^{\not\perp\prime}$ transform (in the quantum random-oracle model) closely follows that of [4, Theorem 1] for the (unsalted) $\mathsf{FO}^{\not\perp}$ transform, with the following mechanical, purely syntactic changes. The attack games all are modified in essentially the same way as above. In particular, a $\mathsf{salt}^* \in \mathcal{S}$ is chosen independently and uniformly at random, and used in the challenge ciphertext; every input to the random oracle G is of the form $m\|\mathsf{salt}$ (instead of just m), and in particular $m^*\|\mathsf{salt}^*$ takes the place of the challenge message m^* throughout; every ciphertext of the transformed scheme is of the form $c\|\mathsf{salt}$ (instead of just c); and the entire $c\|\mathsf{salt}$ is hashed when deriving the shared secret.

With these changes, all the same probabilistic and quantum analyses apply to the modified games, mutatis mutandis. Theorem 4.3 therefore follows.