A key-recovery timing attack on post-quantum primitives using the Fujisaki-Okamoto transformation and its application on FrodoKEM

Qian Guo, Thomas Johansson, Alexander Nilsson

August 10, 2020

Preliminaries

Implementing Crypto Is Hard

As shown by attacks on:

Implementing Crypto Is Hard

As shown by attacks on:

- DH / RSA / DSS in 1996 [Koc96]

Implementing Crypto Is Hard

As shown by attacks on:

- DH / RSA / DSS in 1996 [Koc96]
- Openssl in 2002 and 2016 [BB03; YGH16] . . .

Implementing Crypto Is Hard

As shown by attacks on:

- DH / RSA / DSS in 1996 [Koc96]
- Openssl in 2002 and 2016 [BB03; YGH16] ...
    - 212 CVEs currently in NIST's Vulnerability Database

Implementing Crypto Is Hard

As shown by attacks on:

- DH / RSA / DSS in 1996 [Koc96]
- Openssl in 2002 and 2016 [BB03; YGH16] . . .
  - 212 CVEs currently in NIST's Vulnerability Database

Post quantum Schemes?

Implementing Crypto Is Hard

As shown by attacks on:

- DH / RSA / DSS in 1996 [Koc96]
- Openssl in 2002 and 2016 [BB03; YGH16] . . .
  - 212 CVEs currently in NIST's Vulnerability Database

Post quantum Schemes?

- McEliece in 2010 and 2013 [Str10; Str13]

Implementing Crypto Is Hard

As shown by attacks on:

- DH / RSA / DSS in 1996 [Koc96]
- Openssl in 2002 and 2016 [BB03; YGH16] . . .
  - 212 CVEs currently in NIST's Vulnerability Database

Post quantum Schemes?

- McEliece in 2010 and 2013 [Str10; Str13]
- BLISS in 2016 [Bru+16]

Implementing Crypto Is Hard

As shown by attacks on:

- DH / RSA / DSS in 1996 [Koc96]
- Openssl in 2002 and 2016 [BB03; YGH16] . . .
  - 212 CVEs currently in NIST's Vulnerability Database

Post quantum Schemes?

- McEliece in 2010 and 2013 [Str10; Str13]
- BLISS in 2016 [Bru+16]
- LAC & Ramstake in 2019 [D'A+19]

Implementing Crypto Is Hard

As shown by attacks on:

- DH / RSA / DSS in 1996 [Koc96]
- Openssl in 2002 and 2016 [BB03; YGH16] . . .
  - 212 CVEs currently in NIST's Vulnerability Database

Post quantum Schemes?

- McEliece in 2010 and 2013 [Str10; Str13]
- BLISS in 2016 [Bru+16]
- LAC & Ramstake in 2019 [D'A+19]

This presentation: A general attack against the Fujisaki-Okamoto transformation.

Our contribution

The Fujisaki-Okamoto (FO) transform does not directly handle secret data, yet must be implemented in constant time.

Our contribution

The Fujisaki-Okamoto (FO) transform does not directly handle secret data, yet must be implemented in constant time.

Potentially vulnerable NIST PQC candidates:

FrodoKEM, LAC, BIKE (early version), HQC, ROLLO and RQC.

Maybe others?

Our contribution

The Fujisaki-Okamoto (FO) transform does not directly handle secret data, yet must be implemented in constant time.

Potentially vulnerable NIST PQC candidates:

FrodoKEM, LAC, BIKE (early version), HQC, ROLLO and RQC.

Maybe others?

We show the attack for FrodoKEM (Lattice/LWE based).

A quick, lightweight, background

Publik Key Encryption Schemes

$\mathtt{sk}, \mathtt{pk} \leftarrow \mathtt{KeyGen}(\cdot)$ $(\mathtt{sk}, \mathtt{pk}) \Leftrightarrow$ (secret key, public key)

$\mathtt{c} \leftarrow \mathtt{PKE.CPA.Encrypt}(\mathtt{pk},\mathtt{m})$ $(\mathtt{m}, \mathtt{c}) \Leftrightarrow$ (plaintext, ciphertext)

$\mathtt{m} \leftarrow \mathtt{PKE.CPA.Decrypt}(\mathtt{sk},\mathtt{c})$

Publik Key Encryption Schemes

$\mathrm{sk}, \mathrm{pk} \leftarrow \texttt{KeyGen}(\cdot)$ $\qquad (\mathrm{sk}, \mathrm{pk}) \Leftrightarrow (\text{secret key, public key})$

$\mathrm{c} \leftarrow \texttt{PKE.CPA.Encrypt(pk,m)}$ $\qquad (\mathrm{m}, \mathrm{c}) \Leftrightarrow (\text{plaintext, ciphertext})$

$\mathrm{m} \leftarrow \texttt{PKE.CPA.Decrypt(sk,c)}$

Key Encapsulation Mechanisms

$\mathrm{sk}, \mathrm{pk} \leftarrow \texttt{KeyGen}(\cdot)$

$\mathrm{c}, \mathrm{ss} \leftarrow \texttt{KEM.CCA.Encaps(pk)}$ $\qquad \mathrm{ss} \Leftrightarrow (\text{shared secret})$

$\mathrm{ss} \leftarrow \texttt{KEM.CCA.Decaps(sk,c)}$

PKE-schemes are often proven under the IND-CPA model

PKE-schemes are often proven under the IND-CPA model

INDistinguishability under Chosen Plaintext Attack:

Security game with no access to a decryption oracle.

PKE-schemes are often proven under the IND-CPA model

INDistinguishability under Chosen Plaintext Attack:

Security game with no access to a decryption oracle.

Often, IND-CCA is desirable.

PKE-schemes are often proven under the IND-CPA model

INDistinguishability under Chosen Plaintext Attack:

Security game with no access to a decryption oracle.

Often, IND-CCA is desirable.

INDistinguishability under Chosen Ciphertext Attack:

Security game with access to a decryption oracle.

PKE-schemes are often proven under the IND-CPA model

INDistinguishability under Chosen Plaintext Attack:

Security game with no access to a decryption oracle.

Often, IND-CCA is desirable.

INDistinguishability under Chosen Ciphertext Attack:

Security game with access to a decryption oracle.

The Fujisaki-Okamoto (FO) transform can be used to transform a CPA secure PKE-cipher into a CCA secure cipher.

A common property:

A common property:

LWE encoding

$c = g(pk, m; r) + e(r)$

Code-based encoding

$c = mG \oplus e$

LWE and Code-based schemes

A common property:

LWE encoding

$c = g(pk, m; r) + e(r)$

Code-based encoding

$c = mG \oplus e$

e can vary by a small degree without affecting decryption.

A common property:

LWE encoding

$$c = g(pk, m; r) + e(r)$$

Code-based encoding

$$c = mG \oplus e$$

e can vary by a small degree without affecting decryption.

Decryption fails if e varies by a larger degree.

Fujisaki-Okamoto I

The FO-transform can be used to transform a CPA secure PK-cipher into a CCA secure non-malleable KEM:

Fujisaki-Okamoto I

The FO-transform can be used to transform a CPA secure PK-cipher into a CCA secure non-malleable KEM:

Algorithm 1: $\texttt{KEM.CCA.Encaps}$

Input: $\texttt{pk}$

Output: $(\texttt{c}, \texttt{ss})$

pick a random $\texttt{m}$

$(r, k) \leftarrow H(\texttt{m}, \texttt{pk})$

$c \leftarrow \texttt{PKE.CPA.Encrypt(pk,m;r)}$

$\texttt{ss} \leftarrow H(c, k)$

return (c, ss)

The FO-transform can be used to transform a CPA secure PK-cipher into a CCA secure non-malleable KEM:

Algorithm 1: `KEM.CCA.Encaps`

Input: `pk`

Output: $(\mathtt{c}, \mathtt{ss})$

```
pick a random m
(r, k) ← H(m, pk)
c ← PKE.CPA.Encrypt(pk,m;r) /* IND-CPA secure */
ss ← H(c, k)
return (c, ss)
```

Fujisaki-Okamoto II

The decapsulation function decodes and compare the re-encoding with the received ciphertext.

Fujisaki-Okamoto II

The decapsulation function decodes and compare the re-encoding with the received ciphertext.

Algorithm 2: `KEM.CCA.Decaps`

Input: $(\mathtt{sk}, \mathtt{pk}, \mathtt{c})$

Output: $(\mathtt{ss})$

$\mathtt{m}' \leftarrow$ `PKE.CPA.Decrypt(sk,c)`

$(r', k') \leftarrow H(m', \mathtt{pk})$

$\mathtt{c}' \leftarrow$ `PKE.CPA.Encrypt(pk,m';r)`

if $(\mathrm{c}' = \mathrm{c})$ then **return** $ss' \leftarrow H(c, k)$

else **return** $ss' \leftarrow H(c, k')$

end if

return (c, ss)

Fujisaki-Okamoto II

The decapsulation function decodes and compare the re-encoding with the received ciphertext.

Algorithm 2: `KEM.CCA.Decaps`

Input: $(\mathrm{sk}, \mathrm{pk}, \mathrm{c})$

Output: (ss)

$\mathrm{m}' \leftarrow \texttt{PKE.CPA.Decrypt(sk,c)}$

$(r', k') \leftarrow H(m', \mathrm{pk})$

$\mathrm{c}' \leftarrow \texttt{PKE.CPA.Encrypt(pk,m';r)}$

if $(\mathrm{c}' = \mathrm{c})$ then **return** $ss' \leftarrow H(c, k)$

else **return** $ss' \leftarrow H(c, k')$

end if

return (c, ss)

The decapsulation function decodes and compare the re-encoding with the received ciphertext.

Algorithm 2: `KEM.CCA.Decaps`

Input: $(\mathtt{sk}, \mathtt{pk}, \mathtt{c})$

Output: $(\mathtt{ss})$

$\mathtt{m}' \leftarrow \mathtt{PKE.CPA.Decrypt(sk,c)}$

$(r', k') \leftarrow H(m', \mathtt{pk})$

$\mathtt{c}' \leftarrow \mathtt{PKE.CPA.Encrypt(pk,m';r)}$

if $(\mathtt{c}' = \mathtt{c})$ then **return** $ss' \leftarrow H(c, k)$

else **return** $ss' \leftarrow H(c, k')$

end if

return (c, ss)

Fujisaki-Okamoto II

The decapsulation function decodes and compare the re-encoding with the received ciphertext.

Algorithm 2: `KEM.CCA.Decaps`

Input: $(\mathtt{sk}, \mathtt{pk}, \mathtt{c})$

Output: $(\mathtt{ss})$

$\mathtt{m}' \leftarrow \mathtt{PKE.CPA.Decrypt}(\mathtt{sk},\mathtt{c})$ memcmp? Constant time?

$(r', k') \leftarrow H(m', \mathtt{pk})$

$\mathtt{c}' \leftarrow \mathtt{PKE.CPA.Encrypt}(\mathtt{pk},\mathtt{m'};\mathtt{r})$

if $(\mathtt{c}' = \mathtt{c})$ then **return** $ss' \leftarrow H(c, k)$

else **return** $ss' \leftarrow H(c, k')$

end if

return (c, ss)

The Attack, Generalized

The Vulnerability

c:	FF	EE	DD	CC	BB	AA	99	88	77	66
c':	FF	EE	DD	CC	BB	AA	99	88	77	66

} `memcmp`

Assumptions:

1. Not constant time
2. Tiny modification to c $\rightarrow$ no change to c'
3. Large modification to c $\rightarrow$ total change of c'

Strategy:

- Do modifications at the end of c
- Find the exact threshold between case 2 and 3.
- Time `KEM.CCA.Decaps`, repeat as necessary.
- Extract secrets from the KEM-scheme.

The Vulnerability

c:	FF	EE	DD	DD	BB	AA	99	88	77	66
c':	FF	EE	DD	CC	BB	AA	99	88	77	66

} `memcmp`

Assumptions:

1. Not constant time
2. Tiny modification to c $\rightarrow$ no change to c'
3. Large modification to c $\rightarrow$ total change of c'

Strategy:

- Do modifications at the end of c
- Find the exact threshold between case 2 and 3.
- Time `KEM.CCA.Decaps`, repeat as necessary.
- Extract secrets from the KEM-scheme.

The Vulnerability

c:	FF	EE	DD	00	BB	AA	99	88	77	66	memcmp
c':	15	CB	B8	E2	C6	66	79	1A	A1	3F	

Assumptions:

1. Not constant time
2. Tiny modification to c → no change to c'
3. Large modification to c → total change of c'

Strategy:

- Do modifications at the end of c
- Find the exact threshold between case 2 and 3.
- Time `KEM.CCA.Decaps`, repeat as necessary.
- Extract secrets from the KEM-scheme.

The Vulnerability

c:	FF	EE	DD	CC	BB	AA	99	88	77	77
c':	FF	EE	DD	CC	BB	AA	99	88	77	66

} `memcmp`

Assumptions:

1. Not constant time
2. Tiny modification to c $\rightarrow$ no change to c'
3. Large modification to c $\rightarrow$ total change of c'

Strategy:

- Do modifications at the end of c
- Find the exact threshold between case 2 and 3.
- Time `KEM.CCA.Decaps`, repeat as necessary.
- Extract secrets from the KEM-scheme.

The Vulnerability

c:	FF	EE	DD	CC	BB	AA	99	88	77	AA	memcmp
c':	15	CB	B8	E2	C6	66	79	1A	A1	3F	

Assumptions:

1. Not constant time
2. Tiny modification to c → no change to c'
3. Large modification to c → total change of c'

Strategy:

- Do modifications at the end of c
- Find the exact threshold between case 2 and 3.
- Time `KEM.CCA.Decaps`, repeat as necessary.
- Extract secrets from the KEM-scheme.

The Vulnerability

c:	FF	EE	DD	CC	BB	AA	99	88	77	66
c':	FF	EE	DD	CC	BB	AA	99	88	77	66

} memcmp

Assumptions:

1. Not constant time
2. Tiny modification to c $\rightarrow$ no change to c'
3. Large modification to c $\rightarrow$ total change of c'

Strategy:

- Do modifications at the end of c
- Find the exact threshold between case 2 and 3.
- Time `KEM.CCA.Decaps`, repeat as necessary.
- Extract secrets from the KEM-scheme.

The Vulnerability

c:	FF	EE	DD	CC	BB	AA	99	88	77	66
c':	FF	EE	DD	CC	BB	AA	99	88	77	66

} `memcmp`

Assumptions:

1. Not constant time
2. Tiny modification to c $\rightarrow$ no change to c'
3. Large modification to c $\rightarrow$ total change of c'

Strategy:

- Do modifications at the end of c
- Find the exact threshold between case 2 and 3.
- Time `KEM.CCA.Decaps`, repeat as necessary.
- Extract secrets from the KEM-scheme.

Decryption Error Oracle

Algorithm 3: Error.Oracle

Input: m, a ciphertext modification d

Output: b (decryption failure or not)

$(r, k) \leftarrow H_1(m, \mathtt{pk})$

$c \leftarrow$ `PKE.CPA.Encrypt(pk,m;r)`

$c' \leftarrow c + d$

$t \leftarrow$ Measure[`KEM.CCA.Decaps(sk,c')`]

$b \leftarrow F(t)$

return b

where $F(t)$ uses t to determine whether `PKE.CPA.Decrypt` returns $m' = m$ or $m' \neq m$.

Decryption Error Oracle

Algorithm 3: Error.Oracle

Input: m, a ciphertext modification d

Output: b (decryption failure or not)

$(r, k) \leftarrow H_1(m, \mathtt{pk})$

$c \leftarrow \mathtt{PKE.CPA.Encrypt(pk,m;r)}$

$c' \leftarrow c + d$

$t \leftarrow$ Measure[$\mathtt{KEM.CCA.Decaps(sk,c')}$]

$b \leftarrow F(t)$

return b

where $F(t)$ uses t to determine whether $\mathtt{PKE.CPA.Decrypt}$ returns $m' = m$ or $m' \neq m$.

Decryption Error Oracle

Algorithm 3: Error.Oracle

Input: m, a ciphertext modification d

Output: b (decryption failure or not)

$(r, k) \leftarrow H_1(m, \mathtt{pk})$

$c \leftarrow$ `PKE.CPA.Encrypt(pk,m;r)`

$c' \leftarrow c + d$

$t \leftarrow$ Measure[`KEM.CCA.Decaps(sk,c')`]

$b \leftarrow F(t)$

return b

where $F(t)$ uses t to determine whether `PKE.CPA.Decrypt` returns $m' = m$ or $m' \neq m$.

Decryption Error Oracle

Algorithm 3: Error.Oracle

Input: m, a ciphertext modification d

Output: b (decryption failure or not)

$(r, k) \leftarrow H_1(m, \mathtt{pk})$

$c \leftarrow$ `PKE.CPA.Encrypt(pk,m;r)`

$c' \leftarrow c + d$

$t \leftarrow$ Measure[`KEM.CCA.Decaps(sk,c’)`]

$b \leftarrow F(t)$

return b

where $F(t)$ uses t to determine whether `PKE.CPA.Decrypt` returns $m' = m$ or $m' \neq m$.

Decryption Error Oracle

Algorithm 3: Error.Oracle

Input: m, a ciphertext modification d

Output: b (decryption failure or not)

$(r, k) \leftarrow H_1(m, \mathtt{pk})$

$c \leftarrow$ `PKE.CPA.Encrypt(pk,m;r)`

$c' \leftarrow c + d$

$t \leftarrow$ Measure[`KEM.CCA.Decaps(sk,c’)`]

$b \leftarrow F(t)$

return b

where $F(t)$ uses t to determine whether `PKE.CPA.Decrypt` returns $m' = m$ or $m' \neq m$.

Secret Key Recovery

Algorithm 4: Secret Key Recovery

Input: n_1

Output: $\texttt{sk}$

```
for i ← 0;  i < n_1;  i ← i + 1  do
    begin find (m_i, d_i) such that
        Error.Oracle(m_i, d_i) = 0 and
        Error.Oracle(m_i, d_i + 1) = 1
    end
end
Use set {((m_i, d_i), 0 ≤ i < n)} to extract the secret key
return sk
```

Secret Key Recovery

Algorithm 4: Secret Key Recovery

Input: n_1

Output: $\mathtt{sk}$

for $i \leftarrow 0; \quad i < n_1; \quad i \leftarrow i + 1$ **do**
  begin find (m_i, d_i) such that
    $\text{Error.Oracle}(m_i, d_i) = 0$ and
    $\text{Error.Oracle}(m_i, d_i + 1) = 1$
  end
end
Use set $\{((m_i, d_i), 0 \leq i < n)\}$ to extract the secret key
return sk

Secret Key Recovery

Algorithm 4: Secret Key Recovery

Input: n_1

Output: $\mathtt{sk}$

for $i \leftarrow 0; \quad i < n_1; \quad i \leftarrow i + 1$ **do**

  begin find (m_i, d_i) such that

    $\mathrm{Error.Oracle}(m_i, d_i) = 0$ and

    $\mathrm{Error.Oracle}(m_i, d_i + 1) = 1$

  end

end

Use set $\{((m_i, d_i), 0 \leq i < n)\}$ to extract the secret key

return sk

The case of FrodoKEM

Simplified:

$$(r_1, r2, \text{seed}_A, s) \leftarrow \text{uniform random seeds.}$$

$$E \leftarrow \text{Frodo.SampleMatrix}(r_2)$$

Secret Key (S, s)

$$S \leftarrow \text{Frodo.SampleMatrix}(r_1)$$

Public Key $(\textbf{seed}_A, B)$

$$A \leftarrow \text{Frodo.Gen}(\text{seed}_A)$$

$$B \leftarrow AS + E \tag{1}$$

FrodoKEM KeyGen

Simplified:

$$(r_1, r2, \text{seed}_A, s) \leftarrow \text{uniform random seeds.}$$

$$E \leftarrow \text{Frodo.SampleMatrix}(r_2)$$

Secret Key (S, s)

$$S \leftarrow \text{Frodo.SampleMatrix}(r_1)$$

Public Key $(\textbf{seed}_A, B)$

$$A \leftarrow \text{Frodo.Gen}(\text{seed}_A)$$

$$B \leftarrow AS + E \tag{1}$$

FrodoKEM Encaps

Algorithm 5: FrodoKEM.Encaps (simplified)

Input: $\mathtt{pk}$

Output: $\mathtt{ss}$, $\mathtt{c}$

$m \leftarrow$ uniform random plaintext

$(r_1, r_2, r_3, k) \leftarrow H(H(\mathtt{pk})||m)$

$(S', E', E'') \leftarrow$ **for** $i \in \{1, 2, 3\}$ **do** Frodo.SampleMatrix(r_i) **end**

$B' \leftarrow S'A + E'$

$C \leftarrow S'B + E'' + \mathit{Frodo.Encode}(m)$

$\mathtt{c} \leftarrow$ Frodo.Pack$(B'||C)$

return $(H(\mathtt{c}||k), \mathtt{c})$

FrodoKEM Encaps

Algorithm 5: FrodoKEM.Encaps (simplified)

Input: $\texttt{pk}$

Output: $\texttt{ss}$, $\texttt{c}$

$m \leftarrow$ uniform random plaintext

$(r_1, r_2, r_3, k) \leftarrow H(H(\texttt{pk})||m)$

$(S', E', E'') \leftarrow$ **for** $i \in \{1, 2, 3\}$ **do** Frodo.SampleMatrix(r_i) **end**

$B' \leftarrow S'A + E'$

$C \leftarrow S'B + E'' + Frodo.Encode(m)$

$\texttt{c} \leftarrow$ Frodo.Pack$(B'||C)$

return $(H(c||k), c)$

FrodoKEM Encaps

Algorithm 5: FrodoKEM.Encaps (simplified)

Input: `pk`

Output: `ss`, `c`

$m \leftarrow$ uniform random plaintext

$(r_1, r_2, r_3, k) \leftarrow H(H(\texttt{pk})||m)$

$(S', E', E'') \leftarrow$ **for** $i \in \{1, 2, 3\}$ **do** Frodo.SampleMatrix(r_i) **end**

$B' \leftarrow S'A + E'$

$C \leftarrow S'B + E'' + Frodo.Encode(m)$

$\mathrm{c} \leftarrow$ Frodo.Pack$(B'||C)$

return $(H(c||k), c)$

FrodoKEM Encaps

Algorithm 5: FrodoKEM.Encaps (simplified)

Input: `pk`

Output: `ss`, `c`

$m \leftarrow$ uniform random plaintext

$(r_1, r_2, r_3, k) \leftarrow H(H(\mathtt{pk})||m)$

$(S', E', E'') \leftarrow$ **for** $i \in \{1, 2, 3\}$ **do** $\text{Frodo.SampleMatrix}(r_i)$ **end**

$B' \leftarrow S'A + E'$

$C \leftarrow S'B + E'' + Frodo.Encode(m)$

$\mathtt{c} \leftarrow \text{Frodo.Pack}(B'||C)$

return $(H(c||k), c)$

FrodoKEM Encaps

Algorithm 5: FrodoKEM.Encaps (simplified)

Input: $\texttt{pk}$

Output: $\texttt{ss}$, $\texttt{c}$

$m \leftarrow$ uniform random plaintext

$(r_1, r_2, r_3, k) \leftarrow H(H(\texttt{pk})||m)$

$(S', E', E'') \leftarrow$ **for** $i \in \{1, 2, 3\}$ **do** Frodo.SampleMatrix(r_i) **end**

$B' \leftarrow S'A + E'$

$C \leftarrow S'B + E'' + Frodo.Encode(m)$

$\texttt{c} \leftarrow$ Frodo.Pack$(B'||C)$

return $(H(c||k), c)$

FrodoKEM Encaps

Algorithm 5: FrodoKEM.Encaps (simplified)

Input: $\mathtt{pk}$

Output: $\mathtt{ss}$, $\mathtt{c}$

$m \leftarrow$ uniform random plaintext

$(r_1, r_2, r_3, k) \leftarrow H(H(\mathtt{pk})||m)$

$(S', E', E'') \leftarrow$ **for** $i \in \{1, 2, 3\}$ **do** Frodo.SampleMatrix(r_i) **end**

$B' \leftarrow S'A + E'$

$C \leftarrow S'B + E'' + Frodo.Encode(m)$

$\mathsf{c} \leftarrow$ Frodo.Pack$(B'||C)$

return $(H(c||k), c)$

FrodoKEM Decaps

Algorithm 6: FrodoKEM.Decaps (simplified)

Input: `c`, `sk`

Output: `ss`

$(B', C) \Leftarrow \text{Frodo.Unpack}(c)$

$m' \leftarrow \text{Frodo.Decode}(C - B'S)$

$(r_1, r_2, r_3, k') \leftarrow H(H(\mathtt{pk})||m')$

$(S', E', E'') \leftarrow$ **for** $i \in \{1, 2, 3\}$ **do** $\text{Frodo.SampleMatrix}(r_i)$ **end**

$B'' \leftarrow S'A + E'$

$C' \leftarrow S'B + E'' + Frodo.Encode(m')$

if $B'||C = B''||C'$ **then**

  return $H(c||k')$

else

  return $H(c||s)$

end

FrodoKEM Decaps

Algorithm 6: FrodoKEM.Decaps (simplified)

Input: c, sk

Output: ss

$(B', C) \Leftarrow \text{Frodo.Unpack}(c)$

$m' \leftarrow \text{Frodo.Decode}(C - B'S)$

$(r_1, r_2, r_3, k') \leftarrow H(H(\mathtt{pk})||m')$

$(S', E', E'') \leftarrow$ **for** $i \in \{1, 2, 3\}$ **do** $\text{Frodo.SampleMatrix}(r_i)$ **end**

$B'' \leftarrow S'A + E'$

$C' \leftarrow S'B + E'' + Frodo.Encode(m')$

if $B'||C = B''||C'$ **then**

  return $H(c||k')$

else

  return $H(c||s)$

end

FrodoKEM Decaps

Algorithm 6: FrodoKEM.Decaps (simplified)

Input: $\texttt{c}$, $\texttt{sk}$

Output: $\texttt{ss}$

$(B', C) \Leftarrow \text{Frodo.Unpack}(c)$

$m' \leftarrow \text{Frodo.Decode}(C - B'S)$

$(r_1, r_2, r_3, k') \leftarrow H(H(\texttt{pk})||m')$

$(S', E', E'') \leftarrow$ **for** $i \in \{1, 2, 3\}$ **do** $\text{Frodo.SampleMatrix}(r_i)$ **end**

$B'' \leftarrow S'A + E'$

$C' \leftarrow S'B + E'' + Frodo.Encode(m')$

if $B'||C = B''||C'$ **then**

  return $H(c||k')$

else

  return $H(c||s)$

end

FrodoKEM Decaps

Algorithm 6: FrodoKEM.Decaps (simplified)

Input: $\texttt{c}$, $\texttt{sk}$

Output: $\texttt{ss}$

$(B', C) \Leftarrow \text{Frodo.Unpack}(c)$

$m' \leftarrow \text{Frodo.Decode}(C - B'S)$

$(r_1, r_2, r_3, k') \leftarrow H(H(\text{pk})||m')$

$(S', E', E'') \leftarrow$ **for** $i \in \{1, 2, 3\}$ **do** $\text{Frodo.SampleMatrix}(r_i)$ **end**

$B'' \leftarrow S'A + E'$

$C' \leftarrow S'B + E'' + Frodo.Encode(m')$

if $B'||C = B''||C'$ **then**

  return $H(c||k')$

else

  return $H(c||s)$ /* where s is part of secret key */

end

The combined noise matrix E'''

line 2: $m' \leftarrow$ **Frodo.Decode**$(C - B'S)$

$$C - B'S = \text{Frodo.Encode}(m') + \underbrace{S'E - E'S + E''}_{E'''}$$

The combined noise matrix E'''

line 2: $m' \leftarrow$ **Frodo.Decode**$(C - B'S)$

$$C - B'S = \text{Frodo.Encode}(m') + \underbrace{S'E - E'S + E''}_{E'''}$$

Since S', E' and E'' are known and Equation (1) $\Rightarrow E = B - AS$:

The combined noise matrix E'''

line 2: $m' \leftarrow$ **Frodo.Decode**$(C - B'S)$

$$C - B'S = \text{Frodo.Encode}(m') + \underbrace{S'E - E'S + E''}_{E'''}$$

Since S', E' and E'' are known and Equation (1) $\Rightarrow E = B - AS$:

We get linear equations for the values in S, if we know E'''.

Figuring out E'''

Paraphrasing lemma 2.18 from [Nae+18]:

For successfull decryption:

$-2^{D-B_p-1} \leq E'''_{i,j} < 2^{D-B_p-1}$ for all entries i, j in matrix E'''.

Where $B_p \leq D$ and $B_p, D \in \mathbb{Z}$ are FrodoKEM paramters.

Figuring out E'''

Paraphrasing lemma 2.18 from [Nae+18]:

For successfull decryption:

$-2^{D-B_p-1} \leq E'''_{i,j} < 2^{D-B_p-1}$ for all entries i, j in matrix E'''.

Where $B_p \leq D$ and $B_p, D \in \mathbb{Z}$ are FrodoKEM paramters.

Picking $x_0 > 0$ we get decryption failure when $E'''_{i,j} + x_0 \geq 2^{D-B_p-1}$

Paraphrasing lemma 2.18 from [Nae+18]:

For successfull decryption:

$-2^{D-B_p-1} \leq E'''_{i,j} < 2^{D-B_p-1}$ for all entries i, j in matrix E'''.

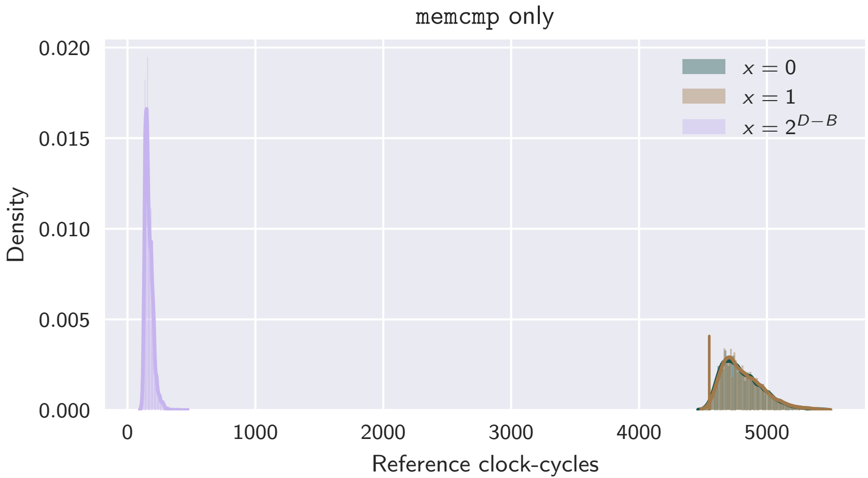
Where $B_p \leq D$ and $B_p, D \in \mathbb{Z}$ are FrodoKEM paramters.

Picking $x_0 > 0$ we get decryption failure when $E'''_{i,j} + x_0 \geq 2^{D-B_p-1}$

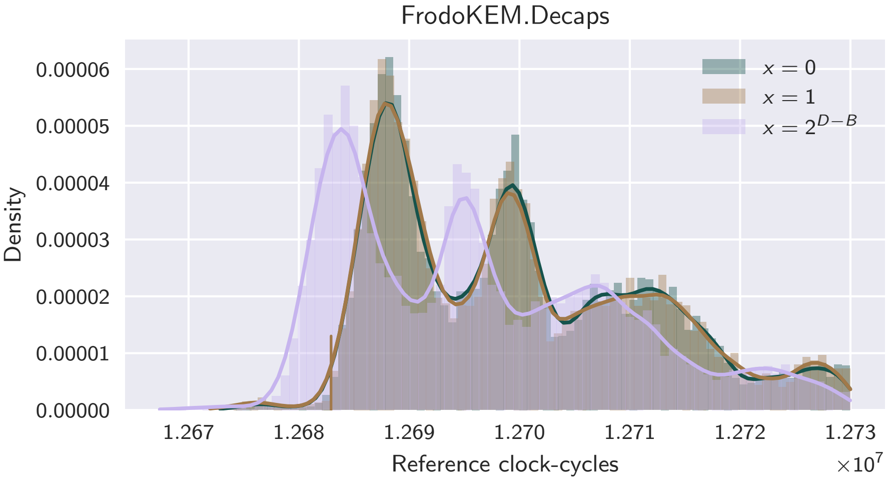
Thus $E'''_{i,j} = 2^{D-B_p-1} - x_0$
if Error.Oracle$(m_i, x_0) = 1$
and Error.Oracle$(m_i, x_0 - 1) = 0$.

Graphs, numbers and such

Results


memcmp only
0.020
0.015
0.010
0.005
0.000
Density
$x = 0$
$x = 1$
$x = 2^{D-B}$
0
1000
2000
3000
4000
5000
Reference clock-cycles

Results II


FrodoKEM.Decaps
Density
0.00006
0.00005
0.00004
0.00003
0.00002
0.00001
0.00000
1.267
1.268
1.269
1.270
1.271
1.272
1.273
Reference clock-cycles
$\times 10^7$
$x = 0$
$x = 1$
$x = 2^{D-B}$

Results III

Tiny differences

$$\frac{4800}{12700000} \approx 0.04\%$$

Results III

Tiny differences

$\frac{4800}{12700000} \approx 0.04\%$

Binary search

- One binary search ≈ 97000 decapsulations
- Size of combined noice matrix 1344×8

Results III

Tiny differences

$\frac{4800}{12700000} \approx 0.04\%$

Binary search

- One binary search $\approx$ 97000 decapsulations
- Size of combined noice matrix 1344×8

Complete Key Recovery

$97000 \times 1344 \times 8 \approx 2^{30}$ queries for FrodoKEM-1344-AES on a Intel i5-4200U CPU running at 1.6GHz.

Summary

Conclusions

“All our implementations avoid the use of secret address accesses and secret branches and, hence, are protected against timing and cache attacks.”
— FrodoKEM Specification.

Very good, but still not enough

Thank you!

References

[BB03] David Brumley and Dan Boneh. "Remote Timing Attacks Are Practical". In: 2003.

[Bru+16] Leon Groot Bruinderink et al. "Flush, Gauss, and Reload - A Cache Attack on the BLISS Lattice-Based Signature Scheme". In: 2016, pp. 323–345. DOI: 10.1007/978-3-662-53140-2_16.

[D'A+19] Jan-Pieter D'Anvers et al. "Timing attacks on Error Correcting Codes in Post-Quantum Secure Schemes.". In: *IACR Cryptology ePrint Archive* 2019 (2019), p. 292.

[GJN20] Qian Guo, Thomas Johansson, and Alexander Nilsson. "A key-recovery timing attack on post-quantum primitives using the Fujisaki-Okamoto transformation and its application on FrodoKEM". In: *Crypto* (2020).

[Koc96] Paul C. Kocher. "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems". In: 1996, pp. 104–113. DOI: 10.1007/3-540-68697-5_9.

[Nae+18] M Naehrig et al. *FrodoKEM: Learning With Errors Key Encapsulation–Algorithm Specifications And Supporting Documentation*. Tech. rep. tech. rep., National Institute of Standards and Technology, 2019. https, 2018.

[Str10] Falko Strenzke. "A Timing Attack against the Secret Permutation in the McEliece PKC". In: 2010, pp. 95–107. DOI: 10.1007/978-3-642-12929-2_8.

[Str13] Falko Strenzke. "Timing Attacks against the Syndrome Inversion in Code-Based Cryptosystems". In: 2013, pp. 217–230. DOI: 10.1007/978-3-642-38616-9_15.

[YGH16] Yuval Yarom, Daniel Genkin, and Nadia Heninger. *CacheBleed: A Timing Attack on OpenSSL Constant Time RSA*. Cryptology ePrint Archive, Report 2016/224. https://eprint.iacr.org/2016/224. 2016.