Cryptographic Hardware and Embedded Systems
20th September, 2022

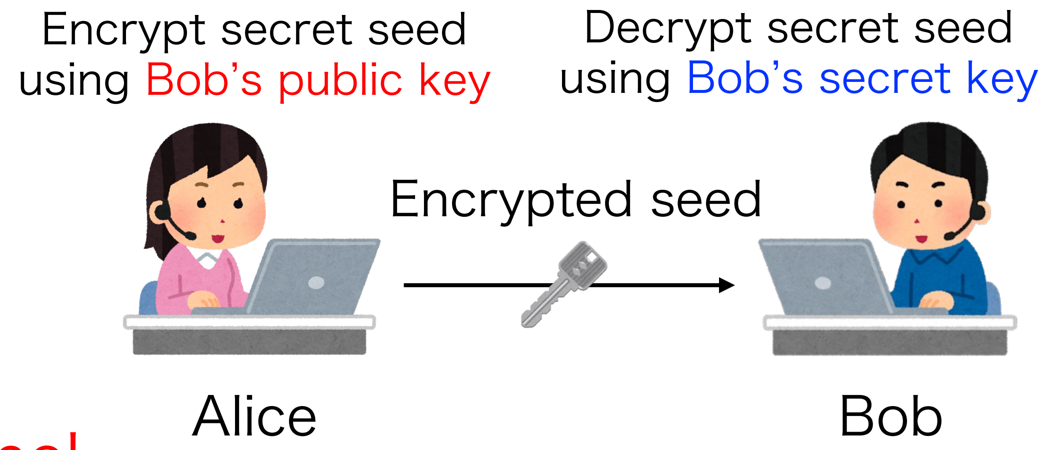
Curse of Re-encryption: A Generic Power/EM Analysis on Post-Quantum KEMs

Rei Ueno[1,2,3], Keita Xagawa[4], Yutaro Tanaka[1,2], Akita Ito[1,2], Junko Takahashi[4], and Naofumi Homma[1,2]

[1] Tohoku University, [2] JST CREST, [3] JST PRESTO, [4] NTT Social Informatics Laboratories

Post-Quantum KEMs

- Essential public key primitive
  - CCA-secure PKE, (authenticated) KX, hybrid cryptography with DEM, etc.
  - Based on quantum-resistant problems
    - Lattice, code, and isogeny
  - This talk is about power/EM side-channel attack on them



Key encapsulation mechanism (KEM)

- Post-quantum KEMs usually employ re-encryption
  - Quite difficult to construct CCA-secure PKE directly
  - Most KEMs are realized by combining CPA-secure PKE and equality (validity) check with re-encryption

This study: *Curse of re-encryption*

- Power/EM analysis *generally* applicable to post-quantum KEMs
- Focus on re-encryption leakage instead of PKE decryption to implement plaintext-checking (PC) oracle
  - Key recovery of eight out of nine KEMs at NIST PQC third-round

Applicability of SCAs focusing on re-encryption-related leakage

		[GTN20]	[PP21]	[RRCB20]	**This work**
	Attack type	Timing	Fault	Power/EM	
Lattice	Kyber	Yes	Yes	Yes	**Yes**
	Saber	Yes	Yes	Yes	**Yes**
	FrodoKEM	Yes	No	Yes	**Yes**
	NTRU	No	No	No	**Yes**
	NTRU Prime	Partially yes†	No	No	**Yes**
Code	HQC	Yes	No	No	**Yes**
	BIKE	Yes	No	No	**Yes***
	Classic McEliece	Unknown	No	No	**Unknown**
Isogeny	SIKE	No	No	No	**Yes**
Countermeasure/mitigation		Constant-time	Redundancy	Masking	

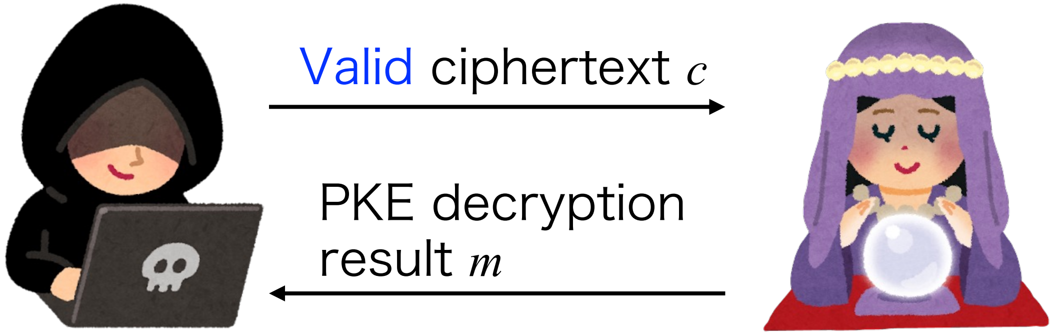
† Applicable to NTRU LPRime, but not to Streamlined NTRU Prime.
* Partial-key recovery, not full-key recovery.

This study: *Curse of re-encryption*

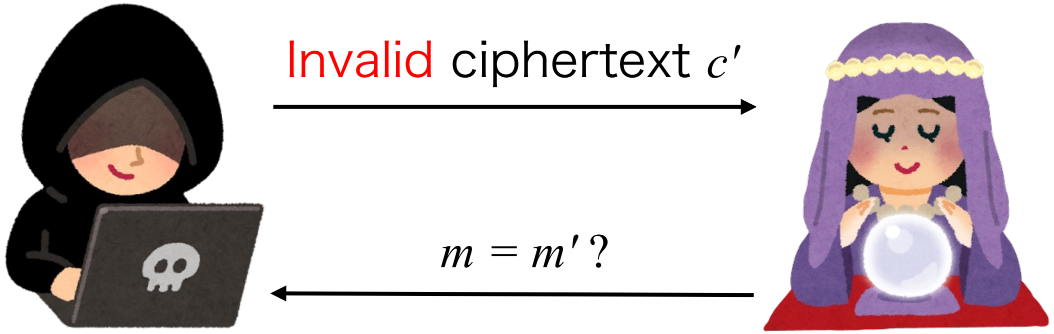
- Power/EM analysis *generally* applicable to post-quantum KEMs
- Focus on re-encryption leakage instead of PKE decryption to implement plaintext-checking (PC) oracle
  - Key recovery of eight out of nine KEMs at NIST PQC third-round
- We also propose deep learning (DL)-based distinguisher to efficiently implement PC oracle
  - Profiled attack, but no need for profiling device
  - Directly applicable to protected (e.g., masked) implementations
- Perform experimental attack on various PRF implementations
  - Key recovery is feasible for non-masked hw/sw and masked sw
  - Masked hw based on threshold implementation would be effective as countermeasure

Plaintext-checking (PC) oracle

- Decryption oracle which returns binary information on PKE decryption result



Attacker can generate valid ciphertext for any plaintext using public key

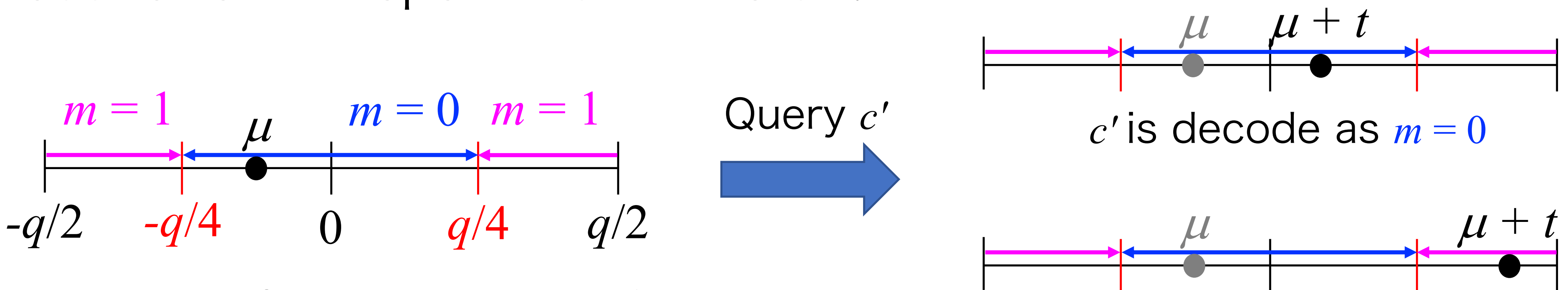


PC oracle returns binary information of whether $m = m'$ for invalid ciphertext

- PC oracle is one of major oracles used for CCA on PKE
  - Key-recovery PC attack (KR-PCA) is known for most CPA-secure post-quantum PKEs
  - KR-PCA on PKEs at 3rd-round NIST PQC KEM candidates is known except for Classic McEliece

KR-PCA on lattice-based PKEs [GJN20]

- Lattice-based PKE decryption employs decode (e.g., rounding) to remove noise incurred by PKE encryption
  - PKE decryption result value μ before decode is secret-key dependent
  - If querying invalid ciphertext $c' = c + t$, μ is changed to $\mu' = \mu + t$
  - Decode result depends on value of t



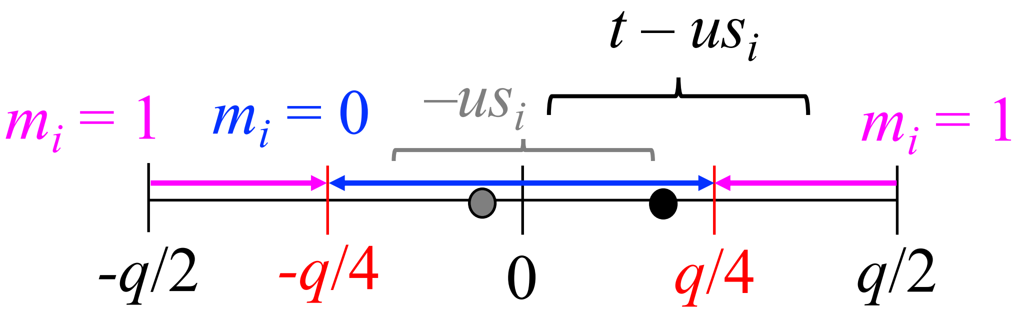
A decode of lattice-based PKE
(q: modulus)

- If attacker find border value of t that changes m, then he/she can recover secret key by solving linear equations

Kyber-512 PKE and KR-PCA [HV20]

- Gen()
  - $A \leftarrow \mathcal{R}_q^{2\times 2}$
  - $s, d \leftarrow_\$ \Psi^2$
  - $B \leftarrow As + d$
  - $\mathrm{ek} \leftarrow (A, B), \mathrm{dk} \leftarrow s$
- Enc(ek = (A, B), m; t, e, f)
  - $U \leftarrow tA + e$
  - $V \leftarrow tB + f + [q/2]m$
  - $\mathrm{ct} \leftarrow (U, V)$
- Dec(dk = s, ct)
  - $W \leftarrow V - Us$
  - $m \leftarrow \mathrm{near}(2W/q) \bmod 2$

- Queries invalid ciphertext: $U = (u, 0), V = tx^i$
  - Check m_i to determine s_i
  - Repeat checking for different t and i



Kyber.Decode

Value of m_i given s_i and t

s_i \ t	-3	-2	-1	0	1	2	3
-3	1	1	1	0	0	0	0
-2	1	1	0	0	0	0	0
-1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1
2	0	0	0	0	0	1	1
3	0	0	0	0	1	1	1

KR-PCAs on PKE in NIST PQC third-round KEMs

KEM type	Scheme	Instance	# Oracle accesses	Ref. to KR-PCA
Lattice	Kyber	Kyber-512	$1536\ (= 3 \times 512)$	[HV20, XIU+21]
		Kyber-1024	$3072\ (= 3 \times 1024)$	[HV20, XIU+21]
	Saber	LightSaber-KEM	$3072\ (= 4 \times 512 + 2 \times 512)$	[HV20]
		FireSaber-KEM	$3072\ (= 3 \times 1024)$	[OUKT21]
	FrodoKEM	FrodoKEM-640	$25600\ (= 5 \times 5120)$	[GTN20, BDL+19]
		FrodoKEM-1344	$43008\ (= 4 \times 10752)$	[GTN20, BDL+19]
	NTRU	ntruhrss701	$\approx 2804\ (= 4 \times 701)$	[DDS+19, ZCQD21]
		ntruhps2048509	$\approx 1018\ (= 2 \times 509)$	[DDS+19, ZCQD21]
		ntruhps4096821	$\approx 1642\ (= 2 \times 821)$	[DDS+19, ZCQD21]
	NTRU Prime	ntrulpr653	$1306\ (= 2 \times 653)$	[XIU+21]
		ntrulpr1277	$2554\ (= 2 \times 1277)$	[XIU+21]
		sntrup653	2712 in avg. $(= 100/1 + 4 \times 653)$	[JJ00, REB+21]
		sntrup1277	5175 in avg. $(= 100/1.5 + 4 \times 1277)$	[JJ00, REB+21]
Code	HQC	hqc128	$\approx 18111\ (= 46 + \log(46) + 46 \times (384 + \log(384)))$	[HV20, XIU+21]
		hqc256	$\approx 58536\ (= 90 + \log(90) + 90 \times (640 + \log(640)))$	[HV20, XIU+21]
	BIKE†	Level 1	3M $(= 2000 \times 1500)$	[GJS16, XIU+21]
		Level 5	N/A	None
	Classic McEliece	Any	N/A	None
Isogeny	SIKE	SIKEp434	$274\ (= 2 \times 137)$	[GPST16]
		SIKEp751	$478\ (= 2 \times 239)$	[GPST16]

† Partial key recovery for BIKE

※ SIKE currently no longer requires side-channels…😭 [CD22]

Fujisaki–Okamoto (FO) transform

- Quite difficult to directly construct CCA-secure PKE
- FO transform realizes CCA-secure KEM from CPA-secure PKE
- Most post-quantum KEMs employ FO transform and its variants

Algorithm 1 CCA-secure KEM based on FO transformation $(\mathsf{KeyGen}, \mathsf{Encaps}, \mathsf{Decaps})$

Input: 1^λ
Output: $\mathsf{sk}, \mathsf{pk}, s$

Function $\textsc{KeyGen}(1^\lambda)$
  $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{PKE.Gen}(1^\lambda)$;
  $s \leftarrow_\$ \mathcal{M}$;
  return $(\mathsf{sk}, \mathsf{pk}, s)$;
end Function

Input: pk
Output: c, k

Function $\textsc{Encaps}(\mathsf{pk})$
  $m \leftarrow_\$ \mathcal{M}$;
  $r \leftarrow G(m[, \mathsf{pk}])$;
  $c \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, m; r)$;
  $k \leftarrow \mathsf{KDF}(m, c)$;
  return (c, k);
end Function

Input: $c, \mathsf{sk}, \mathsf{pk}, s$
Output: k

Function $\textsc{Decaps}(c, \mathsf{sk}, \mathsf{pk}, s)$
  $m' \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}, c)$;
  $r' \leftarrow G(m'[, \mathsf{pk}])$;
  $c' \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, m'; r')$;
  if $c = c'$ **then**
    return $\mathsf{KDF}(m, c)$;
  else
    return $\mathsf{KDF}(s, c)$;
  end if
end Function

Fujisaki–Okamoto (FO) transform

- Quite difficult to directly construct CCA-secure PKE
- FO transform realizes CCA-secure KEM from CPA-secure PKE
- Most post-quantum KEMs employ FO transform and its variants

Algorithm 1 CCA-secure KEM based on FO transformation $(\mathsf{KeyGen}, \mathsf{Encaps}, \mathsf{Decaps})$

Input: 1^λ
Output: $\mathsf{sk}, \mathsf{pk}, s$

```
Function KeyGen(1^λ)
    (sk, pk) ← PKE.Gen(1^λ);
    s ←$ M;
    return (sk, pk, s);
end Function
```

Input: pk
Output: c, k

```
Function Encaps(pk)
    m ←$ M;
    r ← G(m[, pk]);
    c ← PKE.Enc(pk, m; r);
    k ← KDF(m, c);
    return (c, k);
end Function
```

Input: $c, \mathsf{sk}, \mathsf{pk}, s$
Output: k

```
Function Decaps(c, sk, pk, s)
    m' ← PKE.Dec(sk, c);
    r' ← G(m'[, pk]);
    c' ← PKE.Enc(pk, m'; r');
    if c = c' then
        return KDF(m, c);
    else
        return KDF(s, c);
    end if
end Function
```

PKE decryption

Fujisaki–Okamoto (FO) transform

- Quite difficult to directly construct CCA-secure PKE
- FO transform realizes CCA-secure KEM from CPA-secure PKE
- Most post-quantum KEMs employ FO transform and its variants

Algorithm 1 CCA-secure KEM based on FO transformation $(\mathsf{KeyGen}, \mathsf{Encaps}, \mathsf{Decaps})$

Input: 1^λ
Output: $\mathsf{sk}, \mathsf{pk}, s$

Function $\textsc{KeyGen}(1^\lambda)$
    $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{PKE.Gen}(1^\lambda)$;
    $s \leftarrow_\$ \mathcal{M}$;
    return $(\mathsf{sk}, \mathsf{pk}, s)$;
end Function

Input: pk
Output: c, k

Function $\textsc{Encaps}(\mathsf{pk})$
    $m \leftarrow_\$ \mathcal{M}$;
    $r \leftarrow G(m[, \mathsf{pk}])$;
    $c \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, m; r)$;
    $k \leftarrow \mathsf{KDF}(m, c)$;
    return (c, k);
end Function

Input: $c, \mathsf{sk}, \mathsf{pk}, s$
Output: k

Function $\textsc{Decaps}(c, \mathsf{sk}, \mathsf{pk}, s)$
    $m' \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}, c)$;
    $r' \leftarrow G(m'[, \mathsf{pk}])$;
    $c' \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, m'; r')$;
    if $c = c'$ **then**
        return $\mathsf{KDF}(m, c)$;
    else
        return $\mathsf{KDF}(s, c)$;
    end if
end Function

Re-encryption

Fujisaki–Okamoto (FO) transform

- Quite difficult to directly construct CCA-secure PKE
- FO transform realizes CCA-secure KEM from CPA-secure PKE
- Most post-quantum KEMs employ FO transform and its variants

Algorithm 1 CCA-secure KEM based on FO transformation $(\mathsf{KeyGen}, \mathsf{Encaps}, \mathsf{Decaps})$

Input: 1^λ
Output: $\mathsf{sk}, \mathsf{pk}, s$

Function $\textsc{KeyGen}(1^\lambda)$
  $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{PKE.Gen}(1^\lambda)$;
  $s \leftarrow_\$ \mathcal{M}$;
  return $(\mathsf{sk}, \mathsf{pk}, s)$;
end Function

Input: pk
Output: c, k

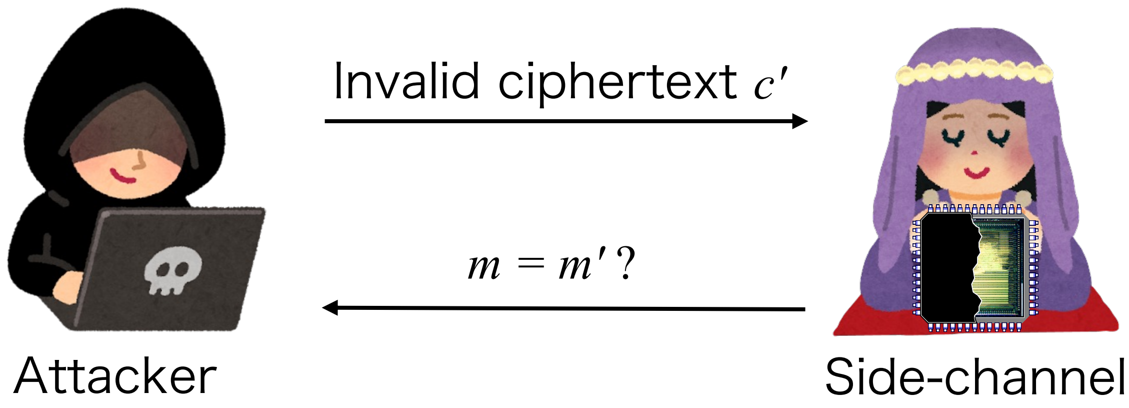
Function $\textsc{Encaps}(\mathsf{pk})$
  $m \leftarrow_\$ \mathcal{M}$;
  $r \leftarrow G(m[, \mathsf{pk}])$;
  $c \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, m; r)$;
  $k \leftarrow \mathsf{KDF}(m, c)$;
  return (c, k);
end Function

Input: $c, \mathsf{sk}, \mathsf{pk}, s$
Output: k

Function $\textsc{Decaps}(c, \mathsf{sk}, \mathsf{pk}, s)$
  $m' \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}, c)$;
  $r' \leftarrow G(m'[, \mathsf{pk}])$;
  $c' \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, m'; r')$;
  if $c = c'$ **then** Equality check
    return $\mathsf{KDF}(m, c)$;
  else
    return $\mathsf{KDF}(s, c)$;
  end if
end Function

Key idea: IND–SCA game

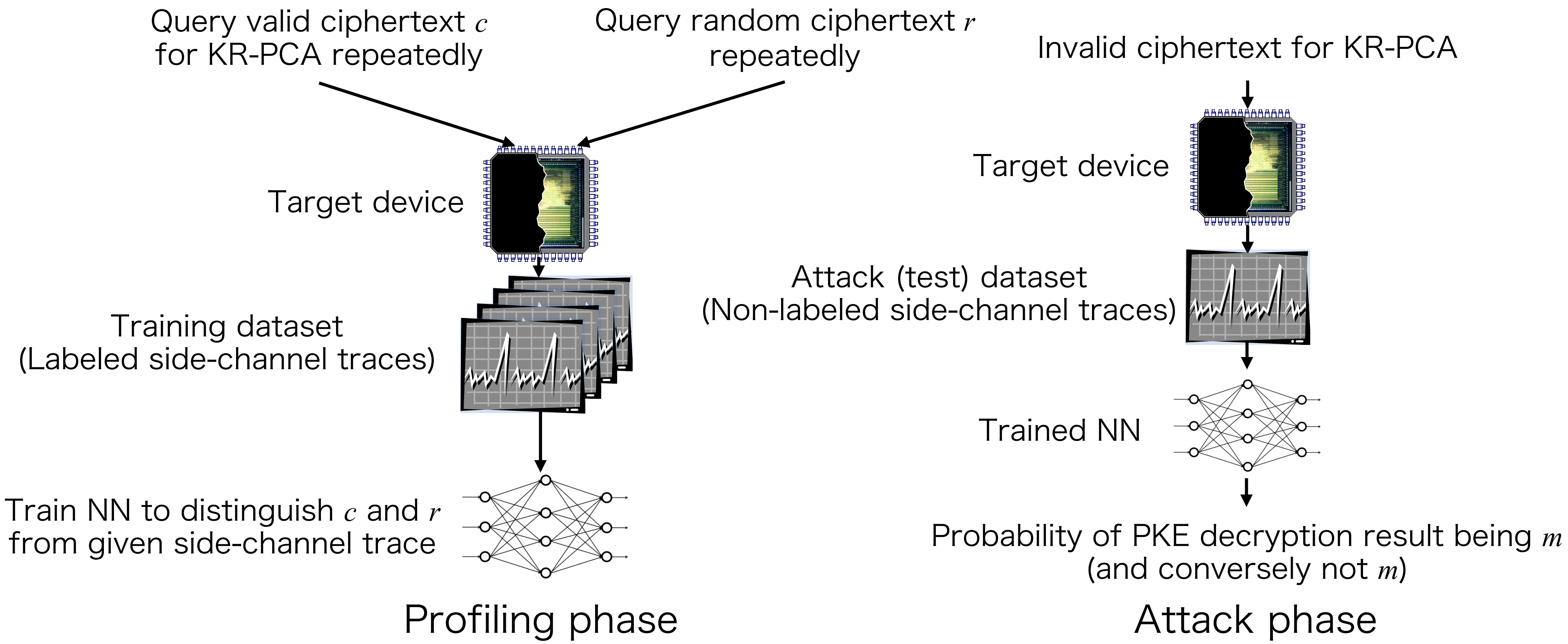
- Exploit leakage during re-encryption to implement PC oracle
  - PRF input fully depends on PKE decryption result m'
  - Distinguish two cases from side-channels:
    - If PKE decryption results are identical for c and c' (i.e., $m = m'$), PRF leakage for c' is meaningfully similar to that for c
    - Otherwise (i.e., $m \neq m'$), they are different



Input: c, sk, pk, s
Output: k

```
Function DECAPS(c, sk, pk, s)
    m' ← PKE.Dec(sk, c);
    r' ← G(m'[, pk]);        PRF (e.g., SHA-3)
    c' ← PKE.Enc(pk, m'; r');
    if c = c' then
        return KDF(m, c);
    else
        return KDF(s, c);
    end if
end Function
```

Side-channel distinguisher based on DL



- Neural network (NN) is used for distinguisher (PC oracle impl.)
  - Train NN to distinguish whether PRF input is m or others (Imitate PC oracle as conditional probability distribution given trace)
  - Side-channel traces for PRF input m can be acquired without secret key
    - Profiling is performed using target device, no need for profiling device

Experimental attack

- Perform proposed attack on various PRF implementations
  - Non-protected software: AES and SHAKE in pqm4
  - Non-protected hardware: AES for SASEBO
  - Protected software: Bit-sliced masked AES
  - Protected hardware: Masked AES based on threshold implementation
- We need 100% accuracy for key recovery
  - Use multiple traces for one PC oracle implementation
  - Majority voting or likelihood ratio test from inference results

NN performance evaluated using 10,000 test traces

	Non-protected software	Non-protected hardware	Protected software	Protected hardware
NN accuracy	0.998	0.999	0.960	0.515
# Traces for 100% accuracy	2	2	5	1000 >

Traces required for key recovery of NIST PQC third-round KEM candidates

Traces required for 100% accuracy

Non-protected software	Non-protected hardware	Masked software	Masked hardware
2	2	5	1000 >

- Total # traces for key recovery is given by (# traces for one PC oracle) × (# PC oracle accesses)
- Threshold implementation would be especially effective as countermeasure

KEM type	Scheme	Instance	# Traces for attack phase: Non-masked implementations	# Traces for attack phase: Masked software
Lattice	Kyber	Kyber-512	3,072	7,680
		Kyber-1024	6,144	15,360
	Saber	LightSaber-KEM	6,144	15,360
		FireSaber-KEM	6,144	15,360
	FrodoKEM	FrodoKEM-640	51,200	128,000
		FrodoKEM-1344	86,016	215,040
	NTRU	ntruhrss701	5,608	14,020
		ntruhps2048509	2,036	5,090
		ntruhps4096821	3,284	8,210
	NTRU Prime	ntrulpr653	2,612	6,530
		ntrulpr1277	5,108	12,770
		sntrup653	5,424	13,560
		sntrup1277	10,350	25,875
Code	HQC	hqc128	36,222	90,555
		hqc256	117,072	292,680
	BIKE	Level 1	6M	15M
		Level 5	N/A	N/A
	Classic McEliece	Any	N/A	N/A
Isogeny	SIKE	SIKEp434	548	1,370
		SIKEp751	956	2,390

Concluding remarks

- Re-encryption is used for CCA security, but its leakage is exploited to mount CCA
- End-to-end protection is mandatory as countermeasure
  - PKE decryption, PRF, PKE encryption, and equality/validity check
  - If PKE decryption is not masked, leakage of initial masking allows for key-recovery SCA even on threshold implementation
- More efficient SCA based on multiple-valued PC oracle

  [TUX+22] Y. Tanaka et al., “Multiple-valued Plaintext-checking Side-Channel Attacks on Post-Quantum KEMs,” IACR ePrint Archive, https://eprint.iacr.org/2022/940
- Key-recovery attacks with fault injection

  [XIU+21] K. Xagawa et al., “Fault-injection attacks against NIST’s post-quantum cryptography round 3 KEM candidates,” ASIACRYPT 2021, pp. 33–61, https://eprint.iacr.org/2021/840

Experimental condition

	Non-protected AES/SHAKE software	Non-protected AES hardware	Masked bit-sliced AES software	Masked AES hardware based on TI
Reference	`pqm4` [KRSS19, pqm21]	SASEBO IP [Toh]	Schwabe and Stoffelen [SS16, git21]	Ueno *et al.* [UHA17]
Device	STM32F415RGT6	Xilinx Kintex-7	STM32F407VGT6U	Xilinx Kintex-7
Board	NewAE Technology STM32F	SAKURA-X	STM32F407G-DISC1	SAKURA-X
Side-channel trace	Supply voltage current	Supply voltage current	EM radiation	Supply voltage current
Measurement interface	NewAE technology chip-whisperer CW308	On-board coaxial connector	Langer EMV-Technik RF-U T-2 probe	On-board coaxial connector
Oscilloscope	Keysight Technologies MSOX6004A			
# Training traces	30,000	30,000	900,000	980,000
# Validation traces	10,000			
# Test traces	10,000			

PC oracle realization with multiple traces

- PC oracle accuracy of 99% is insufficient for key recovery
  - Key recovery requires completely correct PC oracles
    - Requires 300–3M PC oracle accesses
- Use t traces for one PC oracle access to improve accuracy
  - Simplest method: Majority voting using multiple NN inference outputs
    - Resulting accuracy is easily and analytically derived
    - But it cannot fully exploit NN feature which outputs probability
  - Likelihood ratio test: Compute negative log-likelihood (NLL) for $b \in \{0, 1\}$ to determine *more likely* value of b

$$\mathrm{NLL}_b(\boldsymbol{X}^t, \hat{\theta}) = -\frac{1}{t}\sum_{i=1}^{t} \log q_{B|\boldsymbol{X}}(b \mid \boldsymbol{X}_i; \hat{\theta})$$

    - $q_{B|\boldsymbol{X}}(b \mid \boldsymbol{X}_i; \hat{\theta})$: NN output for b given trace with trained parameter $\hat{\theta}$
    - $\boldsymbol{X}_i$: i-th trace

On optimality of distinguishing attack

[Theorem 1, TUX+22] Optimal distinguishing attack

Let $p_{B\mid X}$ be the true conditional probability distribution of PC oracle output B given a side-channel trace $\boldsymbol{X}$. Distinguisher with t traces defined as

$$d(\boldsymbol{X}^t) = \underset{b\in\{0,1\}}{\arg\max} \sum_{i=1}^{t} \log p_{B\mid\boldsymbol{X}}(b \mid \boldsymbol{X}_i),$$

maximizes the success rate of distinguishing attack.

- DL goal is to imitate true conditional probability distribution $p_{B\mid\boldsymbol{X}}$
  - Right hand side is equivalent to argmin of NLL with $p_{B\mid\boldsymbol{X}}$
  - Proposed distinguisher is optimal if NN completely imitates $p_{B\mid\boldsymbol{X}}$
- Maximum success rate = Minimum number of traces
  - Proposed distinguisher may yield most efficient SCA with PC oracle

[TUX+22] Y. Tanaka et al., "Multiple-valued Plaintext-checking Side-Channel Attacks on Post-Quantum KEMs," IACR ePrint Archive, https://eprint.iacr.org/2022/940