The Fujisaki-Okamoto Transform

Advanced Topics in ~~Cybersecurity~~Cryptography (7CCSMATC)

Martin R. Albrecht

Main Reference

Eiichiro Fujisaki and Tatsuaki Okamoto. **Secure Integration of Asymmetric and Symmetric Encryption Schemes**. In: *Journal of Cryptology* 26.1 (Jan. 2013), pp. 80–101. DOI: `10.1007/s00145-011-9114-1`

- Alexander W. Dent. **A Designer's Guide to KEMs**. In: *9th IMA International Conference on Cryptography and Coding*. Ed. by Kenneth G. Paterson. Vol. 2898. LNCS. Springer, Berlin, Heidelberg, Dec. 2003, pp. 133–151. DOI: `10.1007/978-3-540-40974-8_12`
- Martin R. Albrecht, Emmanuela Orsini, Kenneth G. Paterson, Guy Peer, and Nigel P. Smart. **Tightly Secure Ring-LWE Based Key Encapsulation with Short Ciphertexts**. In: *ESORICS 2017, Part I*. ed. by Simon N. Foley, Dieter Gollmann, and Einar Snekkenes. Vol. 10492. LNCS. Springer, Cham, Sept. 2017, pp. 29–46. DOI: `10.1007/978-3-319-66402-6_4`
- Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. **A Modular Analysis of the Fujisaki-Okamoto Transformation**. In: *TCC 2017, Part I*. ed. by Yael Kalai and Leonid Reyzin. Vol. 10677. LNCS. Springer, Cham, Nov. 2017, pp. 341–371. DOI: `10.1007/978-3-319-70500-2_12`

Public-Key Encryption (PKE) with Explicit Randomness

A Public-Key Encryption (PKE) scheme is a triple of PPT algorithms (KeyGen, Enc, Dec) with the following syntax and operation:

KeyGen The key generation algorithm is a randomised algorithm taking as input a security parameter 1^λ and outputs a public/secret key-pair $(\mathsf{pk}, \mathsf{sk})$, the public key and the secret key respectively.

Enc The encryption algorithm is a **deterministic** algorithm taking as input a public-key pk and a message m **and some randomness** r and outputs an encryption of m under pk.

Dec The decryption algorithm is a deterministic algorithm taking as input a ciphertext c and a secret-key sk, and outputs a message m (or an error message $\perp$ indicating a decryption failure).

Key Encapsulation Mechanism (KEM)

A Key Encapsulation Mechanism (KEM) is a triple of PPT algorithms $(\mathsf{KeyGen}, \mathsf{Encap}, \mathsf{Decap})$ with the following syntax and operation:

KeyGen The key generation algorithm is a randomised algorithm taking as input a security parameter 1^λ and outputs a public/secret key-pair $(\mathsf{pk}, \mathsf{sk})$, the **public key** and the **secret key** respectively.

Encap The encapsulation algorithm is a randomised algorithm taking as input a public-key pk and outputs a key $k \in \mathcal{K}$ and an encryption of k under pk.

Decap The decapsulation algorithm is a deterministic algorithm taking as input a ciphertext c and a secret-key sk, and outputs a key k (or an error message $\perp$ indicating a decryption failure).

IND-CPA PKE (Recap)

```
$\text{IND-CPA}_{\text{PKE}}$
$\text{pk}, \text{sk} \leftarrow\!\$\ \text{KeyGen}(1^{\lambda})$
$b \leftarrow\!\$\ \{0, 1\}$
$b' \leftarrow \mathcal{D}^{\text{C}}(\text{pk})$
$\textbf{return } b = b'$
```

```
$\text{C}(m_0, m_1)$
$\textbf{if } |m_0| \neq |m_1| \textbf{ then}$
    $\textbf{return } \perp$
$c \leftarrow\!\$\ \text{Enc}(\text{pk}, m_b)$
$\textbf{return } c$
```

Figure 1: IND-CPA Security Game (PKE).

$$\text{Adv}^{\text{ind-cpa}}_{PKE}(\mathcal{D}) = |\Pr[\text{IND-CPA}^{\mathcal{D}} = 1] - 1/2|$$

IND-CCA KEM

```
$\text{IND-CCA}_{\text{KEM}}$
    $\mathcal{C} \leftarrow \emptyset$
    $\mathsf{pk}, \mathsf{sk} \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$
    $b \xleftarrow{\$} \{0, 1\}$
    $b' \leftarrow \mathcal{D}^{\mathsf{C},\mathsf{D}}(\mathsf{pk})$
    return $b = b'$

$\mathsf{C}()$
    $k_0 \xleftarrow{\$} \mathcal{K}$
    $\mathsf{c}, k_1 \xleftarrow{\$} \mathsf{Encap}(\mathsf{pk})$
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathsf{c}\}$
    return $c, k_b$

$\mathsf{D}(c)$
    if $c \in \mathcal{C}$ then
        return $\bot$
    return $\mathsf{Decap}(\mathsf{sk}, c)$
```

$$\mathrm{Adv}^{\text{ind-cca}}_{KEM}(\mathcal{D}) = |\Pr[\text{IND-CCA}^{\mathcal{D}} = 1] - 1/2|.$$

IND-CCA KEM: Construction I

```
$\mathsf{KeyGen}(1^\lambda)$
    $\mathsf{pk}, \mathsf{sk} \leftarrow\!\!\$\ \mathsf{PKE.KeyGen}(1^\lambda)$
    return $\mathsf{pk}, \mathsf{sk}$
```

```
$\mathsf{Encap}(\mathsf{pk})$
    $x \leftarrow\!\!\$\ \{0,1\}^\lambda$
    $r \leftarrow \mathsf{Hash}(x)$    // RO
    $c \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, x, r)$
    $k \leftarrow \mathsf{KDF}(x)$    // different RO
    return $c, k$
```

```
$\mathsf{Decap}(\mathsf{sk}, c)$
    $y \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}, c)$
    $s \leftarrow \mathsf{Hash}(y)$
    $d \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, y, s)$
    if $d \neq c$ return $\perp$
    $k \leftarrow \mathsf{KDF}(y)$
    return $k$
```

IND-CCA KEM: Construction II

- Encap(pk) is essentially PKE.Enc() for a random x with the randomness r computed as the output of the random oracle Hash(x).
- Decap(sk, c) is essentially PKE.Dec() but with an additional check after, called the “re-encryption check”.
    - It uses the fact that if we know x then we also know r that was used to call PKE.Enc(pk, x, r) so we can “recompute” and check if we get c again.
    - This seems pointless for now, but it will be critical in the proof.

ROs all the way down

It is critical to keep in mind that in the above construction, we have used two **random oracles**: Hash() and KDF().

IND-CCA KEM: Proof Idea I

- We will show that if there is an adversary $\mathcal{D}$ that can break our CCA-secure KEM in the IND-CCA security game then there is also an adversary $\mathcal{A}$ that breaks our CPA-secure PKE in the IND-CPA game.
- Then, if we assume that our IND-CPA PKE is secure, i.e. that there is no $\mathcal{A}$ in the IND-CPA game, this implies that our IND-CCA KEM is secure in the IND-CCA game, i.e. there is no $\mathcal{D}$.

Punchline

$\mathcal{D}$ implies $\mathcal{A}$, but $\mathcal{A}$ does not exist $\Rightarrow$ $\mathcal{D}$ does not exist.

IND-CCA KEM: Proof Idea II

We will build $\mathcal{A}$ from $\mathcal{D}$.

- We're going put $\mathcal{D}$ in a box, where it will start interacting with what it believes to be the IND-CCA secure KEM.
- But we simulate all aspects of this little world such that when $\mathcal{D}$ succeeds we can use this to break the IND-CPA PKE in the IND-CPA game.
- The problem is that $\mathcal{D}$ and $\mathcal{A}$ live in different worlds: IND-CCA (KEM) game and IND-CPA (PKE) game.
- The key problem is that $\mathcal{D}$ expects a decapsulation oracle, it is a CCA attacker.
- But in the world where we are building $\mathcal{A}$ (the IND-CPA game) this decapsulation oracle does not exist: all we have is
  - our challenge ciphertext c,
  - two messages m_0 and m_1 (that we can choose) and
  - the ability to encrypt whatever we want using the provided public key pk.
- We need to simulate the decapsulation oracle for $\mathcal{D}$ somehow.

IND-CCA KEM: Proof Idea III

- We could attempt to always return garbage or a random plaintext whenever $\mathcal{D}$ calls it.
- But $\mathcal{D}$ could easily detect this deception:
  - It prepares a (ciphertext, key) pair as $c, k = \text{Encap}(\text{pk})$ and calls the decapsulation oracle on c. If it does not get k back, it knows we are cheating.
- So we must implement the decapsulation oracle correctly without having access to the secret key for decapsulation
  - We are tying to build $\mathcal{A}$ which is pointless if we already know the secret key.
- To accomplish this feat we make use of the random oracles.

The Trick

The hash functions Hash() and KDF() that $\mathcal{D}$ calls are oracles: **we** provide them to $\mathcal{D}$!

IND-CCA KEM: Game Hops I

```
$\mathsf{Game}_0$
$\mathcal{C} \leftarrow \emptyset$
$\mathsf{pk}, \mathsf{sk} \leftarrow\!\!\$\ \mathsf{KeyGen}(1^\lambda)$
$b \leftarrow\!\!\$\ \{0, 1\}$
$b' \leftarrow \mathcal{D}^{\mathsf{C},\mathsf{D},\mathsf{H},\mathsf{K}}(\mathsf{pk})$
return $b = b'$
```

```
$\mathsf{H}(x)$
return $\mathsf{Hash}(x)$
```

```
$\mathsf{C}()$
$k_0 \leftarrow\!\!\$\ \mathcal{K}$
$c, k_1 \leftarrow\!\!\$\ \mathsf{Encap}(\mathsf{pk})$
$\mathcal{C} \leftarrow \mathcal{C} \cup \{c\}$
return $c, k_b$
```

```
$\mathsf{K}(x)$
return $\mathsf{KDF}(x)$
```

```
$\mathsf{D}(c)$
if $c \in \mathcal{C}$ then
    return $\perp$
return $\mathsf{Decap}(\mathsf{sk}, c)$
```

IND-CCA KEM: Game Hops II

```
$\mathsf{Game}_1$
  $\mathcal{C} \leftarrow \emptyset$
  $\mathsf{pk}, \mathsf{sk} \leftarrow\!\!\$\ \mathsf{KeyGen}(1^\lambda)$
  $b \leftarrow\!\!\$\ \{0,1\}$
  $b' \leftarrow \mathcal{D}^{\mathsf{C},\mathsf{D},\mathsf{H},\mathsf{K}}(\mathsf{pk})$
  return $b = b'$
```

```
$\mathsf{H}(x)$
  return $\mathsf{Hash}(x)$
```

```
$\mathsf{C}()$
  $x \leftarrow\!\!\$\ \{0,1\}^\lambda$
  $r \leftarrow \mathsf{H}(x)$
  $c \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, x, r)$
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{c\}$
  $k_0 \leftarrow\!\!\$\ \mathcal{K}$; $k_1 \leftarrow \mathsf{K}(x)$
  return $c, k_b$
```

```
$\mathsf{K}(x)$
  return $\mathsf{KDF}(x)$
```

```
$\mathsf{D}(c)$
  if $c \in \mathcal{C}$ then
    return $\perp$
  $y \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}, c)$
  $s \leftarrow \mathsf{H}(y)$
  $d \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, y, s)$
  if $d \neq c$ return $\perp$
  $k \leftarrow \mathsf{K}(y)$
  return $k$
```

IND-CCA KEM: Game Hops III

```
$\mathsf{Game}_2$
  $\mathcal{C}, \mathcal{T}_\mathsf{H}, \mathcal{T}_\mathsf{K} \leftarrow \emptyset, \emptyset, \emptyset$
  $\mathsf{pk}, \mathsf{sk} \leftarrow\$ \mathsf{KeyGen}(1^\lambda)$
  $b \leftarrow\$ \{0, 1\}$
  $b' \leftarrow \mathcal{D}^{\mathsf{C},\mathsf{D},\mathsf{H},\mathsf{K}}(\mathsf{pk})$
  return $b = b'$
```

```
$\mathsf{H}(x)$
  if $x \notin \mathcal{T}_\mathsf{H}.\mathsf{keys}$
    $\mathcal{T}_\mathsf{H}[x] \leftarrow\$ \{0, 1\}^\lambda$
  return $\mathcal{T}_\mathsf{H}[x]$
```

```
$\mathsf{C}()$
  $x \leftarrow\$ \{0, 1\}^\lambda$
  $r \leftarrow \mathsf{H}(x)$
  $c \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, x, r)$
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{c\}$
  $k_0 \leftarrow\$ \mathcal{K}; k_1 \leftarrow \mathsf{K}(x)$
  return $c, k_b$
```

```
$\mathsf{K}(x)$
  if $x \notin \mathcal{T}_\mathsf{K}.\mathsf{keys}$
    $\mathcal{T}_\mathsf{K}[x] \leftarrow\$ \{0, 1\}^\lambda$
  return $\mathcal{T}_\mathsf{K}[x]$
```

```
$\mathsf{D}(c)$
  if $c \in \mathcal{C}$ then
    return $\bot$
  $y \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}, c)$
  $s \leftarrow \mathsf{H}(y)$
  $d \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, y, s)$
  if $d \neq c$ return $\bot$
  $k \leftarrow \mathsf{K}(y)$
  return $k$
```

IND-CCA KEM: Game Hops IV

```
$\mathsf{Game}_3$
$\mathcal{C}, \mathcal{T}_\mathsf{H}, \mathcal{T}_\mathsf{K}, \mathcal{T}_\mathsf{c} \leftarrow \emptyset, \emptyset, \emptyset, \emptyset$
$\mathsf{pk}, \mathsf{sk} \leftarrow\!\!\$ \ \mathsf{KeyGen}(1^\lambda)$
$b \leftarrow\!\!\$ \ \{0,1\}$
$b' \leftarrow \mathcal{D}^{\mathsf{C},\mathsf{D},\mathsf{H},\mathsf{K}}(\mathsf{pk})$
return $b = b'$
```

```
$\mathsf{H}(x)$
if $x \notin \mathcal{T}_\mathsf{H}.\mathsf{keys}$
    $\mathcal{T}_\mathsf{H}[x] \leftarrow\!\!\$ \ \{0,1\}^\lambda$
$\mathcal{T}_c[\mathsf{PKE.Enc}(\mathsf{pk}, x, \mathcal{T}_\mathsf{H}[x])] \leftarrow x$
return $\mathcal{T}_\mathsf{H}[x]$
```

```
$\mathsf{C}()$
$x \leftarrow\!\!\$ \ \{0,1\}^\lambda$
$r \leftarrow \mathsf{H}(x)$
$c \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, x, r)$
$\mathcal{C} \leftarrow \mathcal{C} \cup \{c\}$
$k_0 \leftarrow\!\!\$ \ \mathcal{K}; k_1 \leftarrow \mathsf{K}(x)$
return $c, k_b$
```

```
$\mathsf{K}(x)$
if $x \notin \mathcal{T}_\mathsf{K}.\mathsf{keys}$
    $\mathcal{T}_\mathsf{K}[x] \leftarrow\!\!\$ \ \{0,1\}^\lambda$
$\mathcal{T}_c[\mathsf{PKE.Enc}(\mathsf{pk}, x, \mathsf{H}(x))] \leftarrow x$
return $\mathcal{T}_\mathsf{K}[x]$
```

```
$\mathsf{D}(c)$
if $c \in \mathcal{C}$ then
    return $\bot$
$y \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}, c)$
$s \leftarrow \mathsf{H}(y)$
$d \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, y, s)$
if $d \neq c$ return $\bot$
$k \leftarrow \mathsf{K}(y)$
return $k$
```

IND-CCA KEM: Game Hops V

Game_4

```
$\mathcal{C}, \mathcal{T}_{\mathsf{H}}, \mathcal{T}_{\mathsf{K}}, \mathcal{T}_c \leftarrow \emptyset, \emptyset, \emptyset, \emptyset$
$\mathsf{pk}, \mathsf{sk} \leftarrow\$ \mathsf{KeyGen}(1^\lambda)$
$b \leftarrow\$ \{0,1\}$
$b' \leftarrow \mathcal{D}^{\mathsf{E},\mathsf{D},\mathsf{H},\mathsf{K}}(\mathsf{pk})$
return $b = b'$
```

$\mathsf{H}(x)$

```
if $x \notin \mathcal{T}_{\mathsf{H}}.\mathsf{keys}$
    $\mathcal{T}_{\mathsf{H}}[x] \leftarrow\$ \{0,1\}^\lambda$
$c \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, x, \mathcal{T}_{\mathsf{H}}[x])$
$\mathcal{T}_c[c] \leftarrow x$
return $\mathcal{T}_{\mathsf{H}}[x]$
```

$\mathsf{C}()$

```
$x \leftarrow\$ \{0,1\}^\lambda$
$r \leftarrow \mathsf{Hash}(x)$
$c \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, x, r)$
$\mathcal{C} \leftarrow \mathcal{C} \cup \{c\}$
$k_0 \leftarrow\$ \mathcal{K};\ k_1 \leftarrow \mathsf{K}(x)$
return $c, k_b$
```

$\mathsf{K}(x)$

```
if $x \notin \mathcal{T}_{\mathsf{K}}.\mathsf{keys}$
    $\mathcal{T}_{\mathsf{K}}[x] \leftarrow\$ \{0,1\}^\lambda$
$c \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, x, \mathsf{H}(x))$
$\mathcal{T}_c[c] \leftarrow x$
return $\mathcal{T}_{\mathsf{K}}[x]$
```

$\mathsf{D}(c)$

```
if $c \in \mathcal{C}$ then
    return $\perp$
if $c \in \mathcal{T}_c.\mathsf{keys}$
    return $\mathsf{K}(\mathcal{T}_c[c])$
else return $\perp$
```

Analysing the Changes

1. If $\mathcal{D}$ calls $(c, \cdot) \leftarrow \mathsf{Encap}()$ and thus $\mathsf{H}(\cdot)$ before calling $\mathsf{D}(c)$ then we can simulate decapsulation correctly. All good here.
2. If $\mathcal{D}$ sends anything else then our simulation returns $\perp$, but so does the real $\mathsf{Decap}()$ function with high probability:
    - It runs the re-encryption check to see if c is the output of $\mathsf{PKE.Enc}(\mathsf{pk}, x, \mathsf{Hash}(x))$ for the x output by $\mathsf{PKE.Dec}(\mathsf{sk}, c)$.

Task Ahead

$\mathcal{D}$ can only detect that we are cheating if it manages to produce such a c without ever having called $\mathsf{Hash}(x)$ for the matching x.

What can go wrong?

1. $\mathcal{D}$ could have guessed the output of $\text{Hash}(x)$. But since $\text{Hash}(x)$ is a random oracle, the probability of that happening is $1/2^\lambda$ per query
2. A different (x, r') pair with $r' \neq H(x)$ produced c, i.e. we have a collision on c.

To bound this, we need an additional property of our IND-CPA PKE:

Definition (γ-uniformity)

Let $(\text{KeyGen}, \text{Enc}, \text{Dec})$ be a PKE with $\text{Enc} : \mathcal{M} \times \mathcal{R} \to \mathcal{C}$ being the encryption function mapping messages and randomness to ciphertexts. PKE is γ-uniform if for all public keys pk output by KeyGen, all $m \in \mathcal{M}$ and all $c \in \mathcal{C}$, we have $\Pr[r \in \mathcal{R} : c = \text{Enc}(\text{pk}, m, r)] \leq \gamma$.

Applying the union bound, we get $\leq \gamma \cdot q_D$ for the probability of our simulation going wrong, where q_D is the number of decapsulation queries made.

Status

- We have managed to simulate a decapsulation oracle without access to the secret key.
- Our CPA to CCA upgrade is almost complete, we only need to spell out what we actually do with this simulation.
- $\mathcal{D}$ will be on an attack rampage, how do we turn that into a successful $\mathcal{A}$?

Breaking IND-CPA PKE I

- We intend to build $\mathcal{A}$ (a successful IND-CPA PKE adversary) from $\mathcal{D}$ (a successful IND-CCA KEM adversary).
- The IND-CPA PKE game consists of the adversary picking two messages m_0 and m_1 and handing them to the challenger. The challenger will then randomly select on and encrypt it as ciphertext c.
- The adversary must figure out which one it was.
- $\mathcal{A}$ picks two random messages m_0 and m_1 and submit them to the challenger. We get back a challenge ciphertext c^*.
- $\mathcal{D}$ will expect a challenge ciphertext and a key k.
    - It then has to decide if the key k is the one encapsulated under the ciphertext or not.
- Thus, $\mathcal{A}$ picks a random $k^* \xleftarrow{\$} \{0,1\}^\lambda$ and passes (c^*, k^*) to $\mathcal{D}$.

Breaking IND-CPA PKE II

- While we have passed something of the right form to $\mathcal{D}$, the ciphertext c^* is never a correct encapsulation of k^* even if $k^* = \mathrm{KDF}(m_b)$:
  - In the IND-CPA game, $\mathrm{PKE.Enc}(\mathrm{pk}, m_b, r)$ is called with some randomness r that is independent of the message m_b.
  - In the IND-CCA game we expect $r = \mathrm{Hash}(m_b)$.
- $\mathrm{PKE.Enc}(\mathrm{pk}, m_b, r)$ will not have the correct r and we will need to account for this in our analysis.
- For now, $\mathcal{D}$ does not know that, it only sees c^* and k^* and does its attack thing.
- At some point it outputs "yep" that c^* encapsulates k^*" or "nope".
- Or, it might crash and output nothing.
- In any event, we ignore its output.
- Rather, we check if it ever queried m_0 to the random oracles H() or K().
- If it did, we decide that c^* encrypts m_0 otherwise we decide that it encrypts m_1.

Breaking IND-CPA PKE III

```
$\mathcal{A}(\mathsf{pk})$
$\mathcal{C}, \mathcal{T}_{\mathsf{H}}, \mathcal{T}_{\mathsf{K}}, \mathcal{T}_c \leftarrow \emptyset, \emptyset, \emptyset, \emptyset$
$m_0, m_1 \leftarrow\$ \{0,1\}^\lambda, \{0,1\}^\lambda$
$b \leftarrow\$ \{0,1\}; b' \leftarrow \mathcal{D}^{\mathsf{C,D,H,K}}(\mathsf{pk})$
if $m_b \in \mathcal{T}_{\mathsf{H}}.\mathsf{keys} \lor m_b \in \mathcal{T}_{\mathsf{K}}.\mathsf{keys}$
    return $b$
else return $1 - b$
```

```
$\mathsf{H}(x)$
if $x \notin \mathcal{T}_{\mathsf{H}}.\mathsf{keys}$
    $\mathcal{T}_{\mathsf{H}}[x] \leftarrow\$ \{0,1\}^\lambda$
$c \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, x, \mathcal{T}_{\mathsf{H}}[x])$
$\mathcal{T}_c[c] \leftarrow x$
return $\mathcal{T}_{\mathsf{H}}[x]$
```

```
$\mathsf{C}()$
// IND-CPA Oracle
$c^* \leftarrow \mathsf{PKE.C}(m_0, m_1)$
$\mathcal{C} \leftarrow \mathcal{C} \cup \{c^*\}$
$k^* \leftarrow\$ \{0,1\}^\lambda$
return $c^*, k^*$
```

```
$\mathsf{K}(x)$
if $x \notin \mathcal{T}_{\mathsf{K}}.\mathsf{keys}$
    $\mathcal{T}_{\mathsf{K}}[x] \leftarrow\$ \{0,1\}^\lambda$
$c \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, x, \mathsf{H}(x))$
$\mathcal{T}_c[c] \leftarrow x$
return $\mathcal{T}_{\mathsf{K}}[x]$
```

```
$\mathsf{Decap}(c)$
if $c \in \mathcal{C}$ then
    return $\perp$
if $c \in \mathcal{T}_c.\mathsf{keys}$
    return $\mathsf{K}(\mathcal{T}_c[c])$
else return $\perp$
```

Breaking IND-CPA PKE iv

- Note that if $\mathcal{D}$ ever queries m_0 to either RO, it must have gotten it from somewhere.
- We have to account for the probability that it queried it by chance but that probability is low if the space from which we draw m_0 and m_1 is sufficiently large.
- If c^* encrypts m_1 then no information related to m_0 is given to $\mathcal{D}$.
- Thus, if $\mathcal{D}$ queried m_0 it must have gotten it from c^*. Nothing else depends on m_0.

Breaking IND-CPA PKE V

We also know that $\mathcal{D}$ will query m_0 if c^* encrypts it.

- $\mathcal{D}$ is a successful adversary in the IND-CCA KEM game, which requires it to touch m_0 at some point.
- It must check if k^* matches $\mathsf{K}(m_0)$ to decide if k^* is the key encapsulated under c^*.
- In the random oracle model k^* is just a random string and it can only check by querying $\mathsf{K}(m_0)$.
- Instead of querying $\mathsf{K}(m_0)$, $\mathcal{D}$ might realise that we are cheating as $\mathsf{PKE.Enc}(\mathsf{pk}, m_0, H(m_0))$ will not match c^*, the randomness is not right.
- It might then get really angry, flip over tables, whatnot.
- It does not matter, at that point it already solved our problem: $\mathcal{D}$ queried $\mathsf{H}(m_0)$.
- The moment it is able to detect that we are cheating is the moment we win.

Breaking IND-CPA PKE vi

Summary

If $\mathcal{D}$ is a successful adversary then it must either query $\mathsf{K}(m_0)$ to check if k^* is correct or $\mathsf{H}(m_0)$ to check if we are cheating. In either case we win: we've built $\mathcal{A}$ from $\mathcal{D}$. Now, since we assumed $\mathcal{A}$ does not exist, $\mathcal{D}$ does not exist.

Fin

In the Random Oracle Model the adversary sends us its randomness.

References I

[AOPPS17] Martin R. Albrecht, Emmanuela Orsini, Kenneth G. Paterson, Guy Peer, and Nigel P. Smart. Tightly Secure Ring-LWE Based Key Encapsulation with Short Ciphertexts. In: *ESORICS 2017, Part I*. Ed. by Simon N. Foley, Dieter Gollmann, and Einar Snekkenes. Vol. 10492. LNCS. Springer, Cham, Sept. 2017, pp. 29–46. DOI: `10.1007/978-3-319-66402-6_4`.

[Den03] Alexander W. Dent. A Designer's Guide to KEMs. In: *9th IMA International Conference on Cryptography and Coding*. Ed. by Kenneth G. Paterson. Vol. 2898. LNCS. Springer, Berlin, Heidelberg, Dec. 2003, pp. 133–151. DOI: `10.1007/978-3-540-40974-8_12`.

[FO13] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure Integration of Asymmetric and Symmetric Encryption Schemes. In: *Journal of Cryptology* 26.1 (Jan. 2013), pp. 80–101. DOI: `10.1007/s00145-011-9114-1`.

References II

[HHK17] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A Modular Analysis of the Fujisaki-Okamoto Transformation. In: *TCC 2017, Part I*. Ed. by Yael Kalai and Leonid Reyzin. Vol. 10677. LNCS. Springer, Cham, Nov. 2017, pp. 341–371. DOI: `10.1007/978-3-319-70500-2_12`.