Memory-Tight Reductions for Practical Key Encapsulation Mechanisms

Rishiraj Bhattacharyya

NISER, HBNI, India.
rishiraj.bhattacharyya@gmail.com

Abstract. The efficiency of a black-box reduction is an important goal of modern cryptography. Traditionally, the time complexity and the success probability were considered as the main aspects of efficiency measurements. In CRYPTO 2017, Auerbach *et al* introduced the notion of memory-tightness in cryptographic reductions and showed a memory-tight reduction of the existential unforgeability of the RSA-FDH signature scheme. Unfortunately, their techniques do not extend directly to the reductions involving intricate RO-programming. The problem seems to be inherent as all the other existing results on memory-tightness are lower bounds and impossibility results. In fact, Auerbach *et al* conjectured that a memory-tight reduction for IND-CCA security of Hashed-ElGamal KEM is impossible.

- We refute the above conjecture. Using a simple RO simulation technique, we provide memory-tight reductions of IND-CCA security of the Cramer-Shoup and the ECIES version of Hashed-ElGamal KEM.
- We prove memory-tight reductions for different variants of Fujisaki-Okamoto Transformation. We analyze the modular transformations introduced by Hofheinz, Hövermanns and Kiltz (TCC 2017). In addition to the constructions involving implicit rejection, we present a memory-tight reduction for the IND-CCA security of the transformation $\mathrm{QFO}_{\mathsf{m}}^{\perp}$. Our techniques can withstand correctness-errors, and applicable to several lattice-based KEM candidates.

Keywords: Memory-tight Reduction; Hashed-ElGamal; FO transformation

1 Introduction

Memory Efficiency of Black-box Reductions Black-box reduction is an imperative tool in modern cryptography. The security of any scheme S is typically argued by an algorithm $\mathtt{R}$. Given an adversary, $\mathcal{A}_S$ against S, $\mathtt{R}$ with black-box access to $\mathcal{A}$ is shown to solve some underlying hard problem $\mathcal{P}$. The efficiency of a black-box reduction is measured by the resources $\mathtt{R}$ uses, typically in terms of $\mathcal{A}$. Traditionally the reductions aimed at optimizing the time complexity and/or the success probability [5,4,10]. However, Auerbach *et al* [3] observed that some reductions which are *tight* in success probability and time complexity, require a large amount of memory. If the underlying problem is memory sensitive (easier to solve with larger memory), then a memory loose reduction does not rule out the existence of an efficient adversary. They noted further that many of the standard assumptions including LPN, SVP, Discrete Logarithm Problem in prime fields, factoring are memory sensitive. Hence it is imperative to find memory-efficient reductions when the security is based on the hardness of these problems.

Unfortunately, most of the existing results on memory-tight reductions are lower bounds. In [3], authors ruled out memory-tight, restricted black-box reductions for the security of multi-signatures from unique signatures, and multicollision resistance from collision resistance. In [21], Wang *et al* showed lower bounds for a larger class of black-box reductions including the security of public-key encryption and signature schemes in the multi-user setting. In [13], Demay *et al* considered the indifferentiability notion in the memory restricted setting, and proved the impossibility of domain extension of hash functions (even by one bit).

On the other hand, to the best of our knowledge, the only positive result so far is the memory-efficient reduction for RSA FDH in the Random Oracle model [3]. The authors introduced new techniques for the random oracle model and showed, using pseudo-random functions and the power of rewinding the adversary once, one can prove a memory-tight reduction of the existential unforgeability of RSA-FDH from RSA assumption. Their technique seems to be generally applicable for hash and sign paradigm, where the domain of the underlying trapdoor permutation enjoys some form of homomorphism (required for applying Coron's technique [11]).

Key Encapsulation Mechanisms. A Key Encapsulation Mechanism (KEM) is a fundamental primitive to construct efficient public-key cryptosystem. Research in KEM design has been rejuvenated in the last few years due to the ongoing effort to standardize post-quantum cryptographic algorithms. While constructions of IND-CCAsecure KEM in the "classical" setting have been known for years (see [14] for a comprehensive treatment), the reductions were non-tight, and required perfect correctness from the underlying public-key encryption scheme. There are numerous recent works on KEM in the quantum setting [9,20,16,19,17]. However, not much progress has been made in the classical setting until the work of Hofheinz, Hövermanns and Kiltz [16]. HHK revisited the KEM version of Fujisaki Okamoto transformations and presented a modular analysis of multiple variants. Their results, notably include, *tight* reduction (traditional sense) even when underlying public-key encryption scheme has some correctness error.

1.1 Our Contributions

In this paper, we present memory-efficient reductions of the IND-CCA security of hashed-ElGamal and other variants of Fujisaki-Okamoto transformations.

Memory-tight Reduction for Hashed-ElGamal Our starting point is the following conjecture of Auerbach *et al* [3].

Conjecture 1 *[3] Memory-tight Reduction for Hashed-ElGamal does not exist.*

In this paper, **we refute the above conjecture**. We introduce a simple "map-then-prf" technique to simulate the random oracle in a memory-efficient way. Our technique programs the Random Oracle non-adaptively, avoiding the need to tabulate the Random Oracle queries. We consider two versions of Hashed-ElGamal KEM, ECIES [1,2] and HEG[12]. We summarize these results in the following two informal theorems.

Theorem 2 (Informal). *Let $\mathbb{G}$ be a prime-order cyclic group. Let $F : \{0,1\}^{\lambda+1} \times \mathbb{G} \times \mathbb{G} \to \mathcal{K}$ be a prf. There exists a memory-tight reduction, in the random oracle model, of the* IND-CCA *security of HEG over $\mathbb{G}$ and $\mathcal{K}$ from the gap-Diffie-Hellman problem over $\mathbb{G}$.*

Theorem 3 (Informal). *Let $\mathbb{G}, \mathbb{G}_T$ be prime-order cyclic groups and $\hat{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be a bilinear map. Let $F : \{0,1\}^{\lambda} \times \mathbb{G}_T \to \mathcal{K}$ be a prf. There exists a memory-tight reduction, in the random oracle model, of the* IND-CCA *security of ECIES over $\mathbb{G}$ and $\mathcal{K}$ from the Computational-Diffie-Hellman problem over $\mathbb{G}$.*

Memory-tight reduction for variants of Fujisaki-Okamoto Transformations. Fujisaki-Okamoto transformation and other related KEM constructions have gained particular importance in recent years for their applications in constructing post-quantum KEM schemes. In particular, the modular analysis in [16] has been applied widely in constructing lattice-based candidates. In this paper, we prove memory-tight reduction for three variants of Fujisaki-Okamoto transformations (described in Table 1).

We revisit the analysis in [16] and show techniques for memory-tight reductions for all the modules, even withstanding the correctness errors. We summarize the results below.

- **Transformations** $\mathrm{U}^{\not\perp}, \mathrm{U}^{\not\perp}_m, \mathrm{U}^{\perp}, \mathrm{U}^{\perp}_m$**.** In [16], the authors presented four closely related modules to construct an IND-CCA secure KEM from a public-key encryption scheme PKE. The security requirement from PKE depends on the specific variant of U. In this paper, we show new RO simulation techniques for all the four variants to convert corresponding the reductions in [16] into memory-tight ones.
- **Preprocessing Module** T**.** In [16], the transformation T was presented as the preprocessing module to convert (with a tight reduction) an IND-CPA secure public-key encryption scheme $\overline{\mathsf{PKE}}$ to a deterministic OW-PCVA secure public-key encryption scheme. We observe that the RO simulation technique of Auerbach *et al* [3], is sufficient for a memory-tight reduction for OW-PCA security of $T[\overline{\mathsf{PKE}}]$. When applied with the new reductions for $\mathrm{U}^{\not\perp}$ and $\mathrm{U}^{\not\perp}$, this gives a memory-tight reduction for the IND-CCA security of $\mathrm{KEM}^{\not\perp}$ and $\mathrm{KEM}^{\not\perp}_m$ respectively.
- **A new intermediate module** V**.** The modules with explicit reject, (namely $U^{\perp}_m$ and $U^{\perp}$) require security relative to a ciphertext verification oracle. Unfortunately, our technique only proves OW-PCA security of T. To bridge

the gap, we present a transformation V to convert a OW-PCA deterministic public-key encryption scheme to a OW-PCVA deterministic public-key encryption scheme via a memory-efficient reduction. When applied with T and $U_m^{\perp}$, we get a memory efficient reduction (in the *classical* setting) for the scheme $\mathrm{QKEM}_m^{\perp}$ of [16] (Table 4 in [14]).

Constructions	$\mathtt{Encap}(pk)$	$\mathtt{Decap}(sk', c)$

$\mathrm{KEM}^{\not\perp} = U^{\not\perp}\left[T[\overline{\mathsf{PKE}}, \mathtt{G}], \mathtt{H}\right]$	$m \xleftarrow{\$} \mathcal{M}$
	$c = \overline{\mathtt{Enc}}(pk, m, \mathtt{G}(m))$
	$K = \mathtt{H}(m, c)$
		$m' = \overline{\mathtt{Dec}}(sk, c)$
		if $m' \neq \perp \wedge c = \overline{\mathtt{Enc}}(pk, m', \mathtt{G}(m'))$ then $\quad K = \mathtt{H}(m', c)$
		else $\quad K = \mathtt{H}(s, c)$

$\mathrm{KEM}_m^{\not\perp} = U_m^{\not\perp}\left[T[\overline{\mathsf{PKE}}, \mathtt{G}], \mathtt{H}\right]$	$m \xleftarrow{\$} \mathcal{M}$
	$c = \overline{\mathtt{Enc}}(pk, m, \mathtt{G}(m))$
	$K = \mathtt{H}(m)$
		$m' = \overline{\mathtt{Dec}}(sk, c)$
		if $m' \neq \perp \wedge c = \overline{\mathtt{Enc}}(pk, m', \mathtt{G}(m'))$ then $\quad K = \mathtt{H}(m')$
		else $\quad K = \mathtt{H}(s, c)$

$\mathrm{QKEM}_m^{\perp}$ $= U_m^{\perp}\left[V\left[T[\overline{\mathsf{PKE}}, \mathtt{G}], \mathtt{H}'\right], \mathtt{H}\right]$	$m \xleftarrow{\$} \mathcal{M}$ $c_1 = \overline{\mathtt{Enc}}(pk, m, \mathtt{G}(m))$ $c_2 = \mathtt{H}'(m)$ $c = c_1 \Vert c_2$ $K = \mathtt{H}(m)$	Parse $c = c_1 \Vert c_2$, $m' = \overline{\mathtt{Dec}}(sk, c_1)$ if $m' \neq \perp \wedge c_1 = \overline{\mathtt{Enc}}(pk, m', \mathtt{G}(m')) \wedge c_2 = \mathtt{H}'(m')$ $K = \mathtt{H}(m')$ else $K = \perp$

Table 1. Considered variants of Fujisaki-Okamoto Transformations. $\overline{\mathsf{PKE}} = (\overline{\mathtt{Keygen}}, \overline{\mathtt{Enc}}, \overline{\mathtt{Dec}})$ is an IND-CPA secure public-key encryption scheme. In the column $\mathtt{Decap}$, s is a random string, $sk' = sk \Vert s$.

Other Implications. Besides memory efficiency, we found two additional implications of our work. This result refutes the folklore idea that the additional hash present in the $\mathrm{QKEM}_m^{\perp}$ transformation is redundant in the classical setting [14,16,17]. The second implication is that V composed with T gives a OW-PCVA secure encryption scheme from an IND-CPA secure encryption scheme without the γ-spread requirement of [16].

Concurrent Related Work In a concurrent and independent work, Ghoshal and Tessaro [15] prove a memory-efficiency lower bound for ECIES in the generic group model where the pairing is not available. To the best of our knowledge, this work is the first in proving memory lower bound for any particular construction. Interestingly, the authors develop new techniques for proving memory-efficiency lower bounds. We remind the reader that our reduction for ECIES works on pairing friendly groups and does not contradict their results.

1.2 Overview of our Techniques.

Challenges with existing technique The memory-efficient technique to simulate an RO in [3] (and later suggested in [8] in the context of KEM) is to evaluate a PRF on the input. However, in the IND-CCA security reduction for key encapsulation mechanisms, the reduction often needs to adaptively program the output of the RO. Evaluating the prf directly on the query input does not provide the required programming capability.

For example, consider the basic construction of a Key Encapsulation Mechanism from a *deterministic* public-key encryption scheme $\mathsf{PKE} = (\mathtt{Gen}, \mathtt{Enc}, \mathtt{Dec})$. The public-key, secret-key of the KEM would be a key pair $(pk, sk) \leftarrow \mathtt{Gen}$. An encapsulation involves choosing a random message m, and computing

$$c = \mathtt{Enc}(pk, m), \qquad k = \mathtt{H}(m, c).$$

The output of the encapsulation is (c, k). A traditional security proof assuming $\mathtt{H}$ to be a random oracle would be to maintain a table containing the queries and corresponding responses of $\mathtt{H}$ queries. Whenever the adversary makes a decapsulation query on $\hat{c}$, the reduction will check the table whether it contains an entry $(\hat{m}, \hat{c}, \hat{h})$ such that $\mathtt{Enc}(pk, \hat{m})$ returns $\hat{c}$. If such an entry exists, the answer to the decapsulation query would be $\hat{h}$. Otherwise the reduction would return a randomly sampled element $\hat{h}'$, and save $(-, \hat{c}, \hat{h}')$ in the list. The first entry will be filled up when, in a future hash query, the adversary submits $(\hat{m}, \hat{c})$ where $\hat{c} = \mathtt{Enc}(pk, \hat{m})$.

Now consider a memory-efficient reduction where simulation of $\mathtt{H}$ is performed using a prf $F(k, .)$. A hash query on $(\hat{m}, \hat{c})$ is returned with $F(k, \hat{m}, \hat{c})$. The problem arises when simulating the decapsulation query $\hat{c}$. As the entries are no longer saved in a table, the reduction cannot find the required $\hat{m}$ to complete the prf evaluation! One may attempt to solve the issue by answering the hash query with $F(k, \hat{c})$. In that case, the decapsulation queries can be answered. However, two hash queries with the same $\hat{c}$ but different $\hat{m}$ would result in a collision! Hence, this idea fails as well.

Core of our Idea: “injectively map and prf”. Our method originates from the following observation. Let us call $(\hat{m}, \hat{c})$ a good pair if $\hat{c} = \mathtt{Enc}(pk, \hat{m})$. In the IND-CCA security game, the answer to a decapsulation query $\hat{c}$ needs to match with the response of a hash query $(\hat{m}, \hat{c})$ only when $(\hat{m}, \hat{c})$ is a good pair. When answering hash queries on a good pair $(\hat{m}, \hat{c})$, we can “program” the output to be $F(k, m_0, \hat{c})$ (m_0 being any fixed message). For pairs which are not good, we can query an independent prf $F'(k, \hat{m}, \hat{c})$ to compute the responses. Answer to a decapsulation query on (a valid ciphertext) $\hat{c}$ will simply be $F(k, m_0, \hat{c})$. The idea can be generalized as “ Apply an appropriate injective function ϕ on the input, and then apply the prf”. As the composition of an injective function with a prf results into a prf, we can use the arguments of [3]. This basic technique can readily be applied to the Cramer-Shoup version of Hashed-ElGamal, as well as the modules $\mathrm{U}^{\not\perp}$, and $\mathrm{U}^{\perp}$.

Technique for $\mathbf{U}^{\not\perp}_{\mathbf{m}}, \mathbf{U}^{\perp}_{\mathbf{m}}$. In these cases, the hash function is evaluated only on m. Thus, the above idea is not applicable directly. However, as PKE is deterministic, the reduction can still construct a good pair by simply computing $\hat{c} = \mathtt{Enc}(pk, \hat{m})$, and respond a hash query on $\hat{m}$ by $F(k, \hat{c})$. We no longer need to use the independent prf F', as the hash query only contains the message.

Interestingly, the technique works even if PKE has amall correctness errors. Although, $\mathtt{Enc}(pk, .)$ is no longer injective, finding a collision in the output of

$\texttt{Enc}(pk, .)$ implies finding a correctness error. Conditioned on no collision in the output of $\texttt{Enc}(pk, .)$, the argument of [3] goes through. However, one needs to be careful here, as pointed out in [8]. In some definition of deterministic encryption, it is easy to come up with a scheme where a ciphertext decrypts to a message which in turn encrypts to a different ciphertext. To solve the issue, we require that for every message $\hat{m}$ there exists a single ciphertext $\hat{c}$ that decrypts to $\hat{m}$. Our definition of deterministic encryption is carefully considered to maintain this property. Moreover, the schemes generated by the transformation T of [16] satisfies the definition.

Technique for ECIES. In the case of ECIES, we have a group $\mathbb{G}$ of prime order q with a generator $g \in \mathbb{G}$. A public-key is a random element X with the corresponding secret-key x such that $X = g^x$. The encapsulation involves choosing a random $y \overset{\$}{\leftarrow} \mathbb{Z}_q$ and computing

$$Y = g^y \qquad Z = Y^x \qquad k = \mathtt{H}(Z)$$

The output of the encapsulation is (Y, k). While ECIES is analogous to $U^{\not\perp}_m$, we cannot find Y from Z! Hence, we cannot "map to ciphertext space" and apply F.

Fortunately, the "map-then-prf" technique is not limited to mapping to the ciphertext space. We note, when ECIES is implemented using a pairing friendly curve, there exists a bilinear map $\hat{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ for some $\mathbb{G}_T$. Moreover, by the bilinear property, $\hat{e}(g^x, g^y) = \hat{e}(g, g^{xy})$. We simulate the random oracle using $F(k, \hat{e}(g, .))$. The decapsulation oracle can maintain consistency by using $F(k, \hat{e}(X, .))$.

2 Notations and Preliminaries

If S is a set $|S|$ denotes the size of S. $x \overset{\$}{\leftarrow} S$ denotes the process of choosing x uniformly at random from S. $[n]$ denotes the set of first n natural numbers. Composition of two functions is denoted by $\circ$. If $\hat{F} = F \circ \phi$, then $\hat{F}(x) = F(\phi(x))$.

Algorithms and Security Games. The algorithms and complexities considered in the papers are in the RAM model. The algorithms have access to memory and constant number of registers, each having size of one word. For a deterministic (resp. probabilistic) algorithm $\mathcal{A}$, $y = \mathcal{A}(x)$ (resp $y \overset{\$}{\leftarrow} \mathcal{A}(x)$) denotes y is the (resp. uniformly sampled) output of $\mathcal{A}$ on input x. $\mathcal{A}^{\mathcal{O}}$ denotes that $\mathcal{A}$ has access to $\mathcal{O}$ as an oracle. The oracles in this paper may be *stateful*; $st_{\mathcal{O}}$ denotes the state of the RAM $\mathcal{O}$. As followed in [3], $\mathcal{A}$ with oracle access to $\mathcal{O}$ cannot access $st_{\mathcal{O}}$.

SECURITY GAMES The results are proven in the framework of code based games of [6]. A game G consists an algorithm consists of a `main` oracle, and zero or more stateful oracles $O_1, O_2, \cdots, O_n$. If a game G is implemented using a function f, we write $G[f]$ to denote the game.

Complexity Measures. In this paper, we consider the following three complexity measures of an algorithm.

Success Probability. The success probability of an algorithm $\mathcal{A}$ in game G is defined by $\mathbf{Succ}_{\mathcal{A},G} \overset{def}{=} \text{Prob}[G^{\mathcal{A}} = 1]$.
Time Complexity. The time complexity of an algorithm $\mathcal{A}$, denoted by $\text{Time}_\lambda(\mathcal{A})$, is the number of computation steps performed by $\mathcal{A}$ in the worst case over all possible input of size λ. When $\mathcal{A}$ plays a security game G, the time complexity of the game, denoted by $\text{LocalTime}_\lambda(G^{\mathcal{A}})$, is the time complexity of $\mathcal{A}$ plus the number of queries $\mathcal{A}$ makes to the oracle. [1]
Memory Complexity. Following [3,21], we define the memory complexity of an algorithm $\mathcal{A}$ to be the size of the code plus the worst-case number of registers used in memory at any step in computation, over all possible input of size λ and random coins. $\text{LocalMem}_\lambda(G^{\mathcal{A}})$ denotes the memory complexity of $\mathcal{A}$ (not the oracles) in the security game G.
Reductions and Efficiency. We follow the definition of black-box reductions proposed in [18]. A cryptographic primitive $\mathcal{P}$ is a family of efficiently computable functions $f : \{0,1\}^* \to \{0,1\}^*$. Security of $\mathcal{P}$ is described using a game G. An adversary $\mathcal{A}$ is said to *$\mathcal{P}$-break* f with probability ϵ, if

$$\mathbf{Succ}_{\mathcal{A},G[f]} = \epsilon.$$

We follow the following definition of a cryptographic reduction.

Definition 1. *Let $\mathcal{P}, \mathcal{Q}$ be cryptographic primitives and G_P and G_Q be the corresponding security games respectively. A* reduction *from $\mathcal{P}$ to $\mathcal{Q}$ is a pair of algorithms $\mathtt{C}, \mathtt{R}$ such that*

- *$\mathtt{C}^f \in \mathcal{Q}$ for all $f \in \mathcal{P}$*
- *For all $f \in \mathcal{P}$, for all adversary $\mathcal{A}$ that $\mathcal{Q}$-breaks $\mathtt{C}^f$, the algorithm $\mathtt{R}^{\mathcal{A}}$ $\mathcal{P}$-breaks f.*

Memory-Tight Reductions. Following [3,21], we define memory-tight reductions as follows.

Definition 2. *A Cryptographic reduction $(\mathtt{C}, \mathtt{R})$ from $\mathcal{P}$ to $\mathcal{Q}$ is called* memory-tight*, if for all $f \in \mathcal{P}$,*

$$\begin{aligned}
\mathbf{Succ}_{\mathcal{A},G_Q[\mathtt{C}^f]} &\approx \mathbf{Succ}_{\mathtt{R}^{\mathcal{A}},G_P[f]} \\
\mathbf{LocalTime}_\lambda(\mathtt{R}^{\mathcal{A}}) &\approx \mathbf{LocalTime}_\lambda(\mathcal{A}) \\
\mathbf{LocalMem}_\lambda(\mathtt{R}^{\mathcal{A}}) &\approx \mathbf{LocalMem}_\lambda(\mathcal{A})
\end{aligned}$$

Hardness Assumptions The security proofs of Hashed-ElGamal variants are reduced from the Computational Diffie-Hellman and gap-Diffie-Hellman assumption. Consider the CDH game described in figure 1.

[1] In [3], authors defined the local time of the game only by the number of computations of $\mathcal{A}$. In this paper we explicitly include the number of queries made to the oracle.

```
Game CDH($q, g, \mathbb{G}$)
  $x \xleftarrow{\$} \mathbb{Z}_q^*$
  $y \xleftarrow{\$} \mathbb{Z}_q^*$
  $z \leftarrow \mathcal{A}(g^x, g^y)$
  if $z = g^{xy}$   return 1
  else   return 0
```

```
Oracle DDH($X, Y, Z$)
  if $\exists y$ such that $Y = g^y$ and $Z = X^y$
  return 1
  else
  return 0
```

Fig. 1. CDH game and gap-DH game. In gap-DH game, $\mathcal{A}$ has oracle access to $\mathtt{DDH}(\cdot, \cdot, \cdot)$

Definition 3. *(gap-Diffie-Hellman Assumption) Let q be a prime. Let $\mathbb{G} = \langle g \rangle$ be a cyclic group of order q. The (t, μ, ϵ) gap-Diffie-Hellman (*gap-DH*) assumption states that for all adversary $\mathcal{A}$ that runs in times t and uses μ bites of memory,*

$$\mathbf{Succ}_{\mathcal{A}^{\mathtt{DDH}},\mathsf{CDH}} \leq \epsilon$$

The Computational Diffie-Hellman assumption is defined in the same way, except the condition that $\mathcal{A}$ has no access to the DDH oracle.

Key Encapsulation Mechanism A key encapsulation mechanism KEM consists of three algorithms; $\mathtt{Gen}, \mathtt{Encap}, \mathtt{Decap}$. The key generation algorithm Gen takes a security parameter 1^λ as input and outputs a public key pk and a secret key sk. The encapsulation algorithm Encap, on input pk, outputs a key-ciphertext pair (c, K), where $K \in \mathcal{K}$ for some non-empty set $\mathcal{K}$. c is said to be the encapsulation of K. The deterministic decapsulation algorithm Decap takes an encapsulation c as input along with sk, and outputs a key $K \in \mathcal{K}$. A KEM is called δ-correct if

$$\text{Prob}[\mathtt{Decap}(sk, c) \neq K | (pk, sk) \leftarrow \mathtt{Gen}; (c, K) \leftarrow \mathtt{Encap}(pk)] \leq \delta$$

IND-CCA security of a Key Encapsulation Mechanism We recall the IND-CCA security game for a Key Encapsulation Mechanism in Figure 2.
The IND-CCA advantage of an adversary $\mathcal{A}$ against KEM is defined as

$$\mathbf{Adv}^{\text{IND-CCA}}_{\mathcal{A},\mathsf{KEM}} \stackrel{def}{=} \left| \mathbf{Succ}_{\mathcal{A},\text{IND-CCA}} - \frac{1}{2} \right|.$$

Public-Key Encryption A public-key encryption scheme consists of three algorithms, $\mathsf{PKE} = (\mathtt{Gen}, \mathtt{Enc}, \mathtt{Dec})$. There are three sets associated with PKE, the message space $\mathcal{M}$, the randomness space $\mathcal{R}$, and the ciphertext space $\mathcal{C}$. The key generation algorithm takes the security parameter as input and outputs a public-key, secret-key pair (pk, sk). The encryption algorithm takes the public key pk, and a message $m \in \mathcal{M}$ as input, samples a random string $r \xleftarrow{\$} \mathcal{R}$, and outputs a ciphertext.$c \leftarrow \mathtt{Enc}(pk, m, r)$. The decryption algorithm Dec, on input

```
Game IND-CCA
$(pk, sk) \leftarrow \mathtt{Gen}(1^\lambda)$
$b \xleftarrow{\$} \{0,1\}$
$(c^*, K_0^*) \leftarrow \mathtt{Encap}(pk)$
$K_1^* \xleftarrow{\$} \mathcal{K}$
$b' \leftarrow \mathcal{A}^{\mathrm{Decap}}(c^*, K_b^*)$
if $b = b'$ return $1$
else return $0$
```

```
Oracle Decap($c$)
if $c = c^*$return $\perp$
$K \leftarrow \mathtt{Decap}(sk, c)$
return $K$
```

Fig. 2. IND-CCA game for KEM

```
Game COR
$(pk, sk) \leftarrow \mathtt{Gen}(1^\lambda)$
$m \leftarrow \mathcal{A}(pk, sk)$
$c \leftarrow \mathtt{Enc}(pk, m)$
if $m \neq \mathtt{Dec}(sk, c)$ return $1$
else return $0$
```

Fig. 3. Correctness game for PKE

sk and a ciphertext c, outputs a message $m = \mathtt{Dec}(sk, c) \in \mathcal{M}$ or a special symbol $\perp \notin \mathcal{M}$. We say, c is an invalid ciphertext, if $\mathtt{Dec}(sk, c) = \perp$.

Deterministic Public Key Encryption. We call a public-key encryption scheme PKE *deterministic*, if the algorithm $\mathtt{Enc}$ is deterministic and for every message $m \in \mathcal{M}$, there exists a unique $c \in \mathcal{C}$ such that $\mathtt{Dec}(sk, c) = m$. We write $c \leftarrow \mathtt{Enc}(pk, m)$ for deterministic encryption.

Correctness. Following [16], we define the correctness of a public-key encryption scheme by the security game COR in Figure 3.

Definition 4. *Let $\delta : \mathbb{N} \rightarrow [0, 1]$ be an increasing function. Consider the game COR in Figure 3. A public-key encryption scheme PKE is called δ-correct, if for all adversary $\mathcal{A}$ with running time bounded by t,*

$$\mathbf{Succ}_{\mathcal{A},\mathsf{COR[PKE]}} \leq \delta(t)$$

where the probability is taken over the randomness of $\mathtt{Gen}$ and $\mathcal{A}$. Moreover, we say PKE is strongly $\overline{\delta}$ correct, if $\forall\ t, \delta(t) \leq \overline{\delta}$.

```
Game OW-PCVA
$(pk, sk) \leftarrow \mathtt{Gen}(1^\lambda)$
$m \xleftarrow{\$} \mathcal{M}$
$c \leftarrow \mathtt{Enc}(pk, m)$
$m' \leftarrow \mathcal{A}^{\mathtt{PCO},\mathtt{CVO}}(pk, c)$
if $m' = \mathtt{Dec}(sk, c)$ return $1$
else return $0$
```

```
Procedure $\mathtt{PCO}(m, c)$
if $m = \mathtt{Dec}(sk, c)$ return $1$
else return $0$
```

```
Procedure $\mathtt{CVO}(c)$
$m \leftarrow \mathtt{Dec}(sk, c)$
if $m \in \mathcal{M}$ return $1$
else return $0$
```

Fig. 4. Game OW-PCVA. In the game OW-PCA (resp. OW-VA), $\mathcal{A}$ has oracle access to only $\mathtt{PCO}$ (resp. $\mathtt{CVO}$).

Security. Following [16], we define three security games for a public-key encryption scheme, OW-PCA, OW-VA, and OW-PCVA in Figure 4. In OW-PCA game, the adversary has oracle access to PCO. In the OW-VA game, the adversary has oracle access to CVO. In OW-PCVA game, the adversary has oracle access to both PCO and CVO. For ATK $\in$ {PCA, VA, PCVA}, we define the corresponding advantages of an adversary $\mathcal{A}$ against PKE as

$$\mathbf{Adv}^{\text{OW-ATK}}_{\mathcal{A},\mathsf{PKE}} \overset{def}{=} \text{Prob}[\text{OW-ATK}[\mathsf{PKE}]^{\mathcal{A}} = 1]$$

Random Oracles. An (idealized) function $\mathcal{F} :; \{0,1\}^{\delta} \rightarrow \{0,1\}^{\rho}$ is said to be a *Random Oracle*, if for all $x \in \{0,1\}^{\delta}$, the output $\mathcal{F}(x)$ is independently and uniformly distributed over $\{0,1\}^{\rho}$.

Pseudo-random Functions

Definition 5. *Let $F : \{0,1\}^{\lambda} \times \{0,1\}^{\delta} \rightarrow \{0,1\}^{\rho}$ be a deterministic algorithm and let $\mathcal{A}$ be an algorithm. The prf advantage of $\mathcal{A}$ is defined as*

$$\mathbf{Adv}^{\text{prf}}_{\mathcal{A},F} \overset{def}{=} |\mathbf{Succ}(\mathtt{Real}^{\mathcal{A}}) - \mathbf{Succ}(\mathtt{Random}^{\mathcal{A}})|.$$

F is said to implement a family of (t,d,ϵ)-pseudo-random functions if for all adversary $\mathcal{A}$ that runs in time t and uses memory d,

$$\mathbf{Adv}^{\text{prf}}_{\mathcal{A},F} \leq \epsilon$$

Simulating Random Oracle using PRF. If a game G is defined in the random oracle model, then one procedure of the game defines the random oracle $\mathtt{H} : \{0,1\}^{\delta} \rightarrow \{0,1\}^{\rho}$. The standard technique to implement the random oracle procedure is via lazy sampling. However, the lazy sampling technique requires $\mathcal{O}(q_h \cdot \lambda)$ additional memory where q_h is the number of H queries made by the adversary. In [3], the authors formalized the technique, originally suggested in [7], of simulating the Random Oracle using a prf. Let $G[\mathtt{H}]$ be a game where H is a random oracle used in G. Let $G[F]$ be the same game where the random oracle is implemented using a prf F. Specifically, the oracle H is implemented using $F(k,.)$ for a randomly sampled key k.

Lemma 1 (RO simulation using prf [3]). *For all adversary $\mathcal{A}$ against G making at most q_h queries to the random oracle, there exists a $\mathcal{B}_F$ against F in the* prf *game such that*

$$\left|\mathbf{Succ}_{\mathcal{A}^{\mathtt{H}},G[\mathtt{H}]} - \mathbf{Succ}_{\mathcal{A}^{\mathtt{H}},G[F]}\right| \leq \mathbf{Adv}^{\text{prf}}_{\mathcal{B}_F,F}$$

Moreover, it holds that

$$\mathbf{LocalTime}(\mathcal{B}_F) = \mathbf{LocalTime}(\mathcal{A}) + \mathbf{LocalTime}(G) + q_h$$
$$\mathbf{LocalMem}(\mathcal{B}_F) = \mathbf{LocalMem}(\mathcal{A}) + \mathbf{LocalMem}(G)$$

```
Game Real
Procedure main
    $k \xleftarrow{\$} \{0,1\}^\lambda$
    $b \leftarrow \mathcal{A}^{O_F}$
    if $b = 0$
        return 1
    else
        return 0
    endif

Procedure $O_F(x)$
    return $F(k, x)$

Game Random
Procedure main
    $b \leftarrow \mathcal{A}^{O_F}$
    if $b = 0$
        return 1
    else
        return 0
    endif

Procedure $O_F(x)$
    $y \xleftarrow{\$} \{0,1\}^\rho$
    return $y$
```

Fig. 5. PRF security game

```
RO simulation by lazy sampling
Procedure main

Procedure H($x$)
    if H($x$) =$\perp$
        H($x$) $\xleftarrow{\$} \{0,1\}^\rho$
    endif
    return H($x$)

RO simulation using PRF
Procedure main
    $k \xleftarrow{\$} \{0,1\}^\kappa$

Procedure H($x$)
    return $F(k, x)$
```

Fig. 6. Memory Efficient simulation of Random Oracle

```
Procedure Gen($1^\lambda$)
    $(q, g, \mathbb{G}) \leftarrow \mathtt{DH}(1^\lambda)$
    $x \xleftarrow{\$} \mathbb{Z}_q^*$
    $\mathrm{pk} = (g, g^x)$
    $\mathrm{sk} = x$
    return $(\mathrm{pk}, \mathrm{sk})$

Procedure Encap(pk)
    $(g, h) = \mathrm{pk}$
    $y \xleftarrow{\$} \mathbb{Z}_q^*$
    $Y = g^y$
    $Z = h^y$
    $K = \mathtt{H}(Y, Z)$
    return $(Y, K)$

Procedure Decap(sk, $Y$)
    $x = sk$
    $Z = Y^x$
    $K = \mathtt{H}(Y, Z)$
    return $K$
```

Fig. 7. HEG: Cramer-Shoup Version of Hashed-ElGamal KEM. $\mathtt{H} : \mathbb{G} \times \mathbb{G} \to \mathcal{K}$ is a cryptographic hash function

3 Memory-tight Reductions for Hashed-ElGamal

3.1 Cramer-Shoup Variant

In this section we present a memory-tight reduction of Cramer-Shoup version of hashed-ElGamal Key Encapsulation mechanism [12]. We describe the scheme in Figure 7. $\mathbb{G}$ is a cyclic group of prime order q. Let $\mathtt{H} : \mathbb{G} \times \mathbb{G} \to \mathcal{K}$ be a hash function. Our main result in this section is the following theorem.

Theorem 4. *Let q be a prime and $\mathbb{G}$ be any gap group of order q. Let* DDH *be the Decisional Diffie Hellman oracle on $\mathbb{G}$. Let* DH *be the Diffie Hellman instance generation algorithm over $\mathbb{G}$. Let $F : \{0,1\}^\lambda \times \{0,1\} \times \mathbb{G} \times \mathbb{G} \to \mathcal{K}$ be a prf. Let Π be the HEG KEM scheme over $\mathbb{G}$ and $\mathcal{K}$, with security parameter λ.*

Let $\mathcal{A}$ be any adversary in the IND-CCA*game of Π. Suppose $\mathcal{A}$ makes q_H hash queries and q_D decapsulation queries. Then, in the random oracle model, there exists an adversary $\mathcal{B}_{DH}$ in the* gap-DH *game, and an adversary $\mathcal{B}_F$ such that*

$$\mathbf{Adv}^{\text{IND-CCA}}_{\mathcal{A},\Pi} \leq \mathbf{Adv}^{\mathsf{gap\text{-}DH}}_{\mathcal{B}_{DH},\mathbb{G}} + \mathbf{Adv}^{\text{prf}}_{\mathcal{B}_F,F}$$

Moreover, it holds that

$$\begin{aligned}
\mathbf{LocalTime}(\mathcal{B}_{DH}) \approx& \mathbf{LocalTime}(\mathcal{A}) + (q_H + q_D) \cdot \mathbf{LocalTime}(F) + q_H \\
\mathbf{LocalMem}(\mathcal{B}_{DH}) \approx& \mathbf{LocalMem}(\mathcal{A}) + \mathbf{LocalMem}(F) + 7\lambda + 1 \\
\mathbf{LocalTime}(\mathcal{B}_F) \approx& \mathbf{LocalTime}(\mathcal{A}) + \mathbf{LocalTime}(\mathtt{DH}) + (q_H + q_D) \\
\mathbf{LocalMem}(\mathcal{B}_F) \approx& \mathbf{LocalMem}(\mathcal{A}) + \mathbf{LocalMem}(\mathtt{DH}) + 11\lambda + 2
\end{aligned}$$

Before proving the Theorem 4, we construct a prf $\hat{F} : \{0,1\}^\lambda \times \mathbb{G} \times \mathbb{G} \to \mathcal{K}$ that we shall use in the proof.

Construction of $\hat{F}$. Let DDH be the decisional Diffie-Hellman oracle such that $\mathtt{DDH}(X,Y,Z) = 1$, if (X,Y,Z) is a valid Diffie-Hellman tuple.

Construction 5 *Let $\mathbb{G}$ be a group of prime order q and let g be a generator of $\mathbb{G}$. Fix $X \in \mathbb{G}$. Let $F : \{0,1\}^\lambda \times \{0,1\} \times \mathbb{G} \times \mathbb{G} \to \mathcal{K}$. We define $\hat{F}_X : \{0,1\}^\lambda \times \mathbb{G} \times \mathbb{G} \to \mathcal{K}$ as follows*

$$\hat{F}_X(k,Y,Z) = \begin{cases} F(k,0,Y,Z) & \textit{if } \mathtt{DDH}(X,Y,Z) = 0 \\ F(k,1,Y,g) & \textit{if } \mathtt{DDH}(X,Y,Z) = 1 \end{cases}$$

In order to use the map then prf technique, we need the following lemma.

Lemma 2. *If F is a prf, then $\hat{F}_X$ is a prf. Moreover, for every adversary $\mathcal{B}_{\hat{F}}$ against $\hat{F}_X$, there exists a $\mathcal{B}_F$ against F such that,*

$$\begin{aligned}
\mathbf{Adv}^{\text{prf}}_{\mathcal{B}_F,F} &= \mathbf{Adv}^{\text{prf}}_{\mathcal{B}_{\hat{F}},\hat{F}} \\
\mathbf{LocalTime}(\mathcal{B}_F) &= \mathbf{LocalTime}(\mathcal{B}_{\hat{F}}) + q \\
\mathbf{LocalMem}(\mathcal{B}_F) &= \mathbf{LocalMem}(\mathcal{B}_{\hat{F}}) + 2\lambda.
\end{aligned}$$

where q is the number of queries made by $\mathcal{B}_{\hat{F}}$.

Proof. Fix $X \in \mathbb{G}$. Note that for every $Y \in \mathbb{G}$, there exists a unique $Z \in G$ such that $\mathtt{DDH}(X,Y,Z) = 1$. We define $\psi_X : \mathbb{G} \times \mathbb{G} \to \{0,1\} \times \mathbb{G} \times \mathbb{G}$ as

$$\psi_X(Y,Z) = \begin{cases} (0,Y,Z) & \text{if } \mathtt{DDH}(X,Y,Z) = 0 \\ (1,Y,0^\lambda) & \text{if } \mathtt{DDH}(X,Y,Z) = 1 \end{cases}$$

It is easy to verify that ψ_X is an injective function. Moreover, $\hat{F}_X = F \circ \psi_X$.

Let O be the oracle of $\mathcal{B}_F$. $\mathcal{B}_F$ chooses $x \in \mathbb{Z}_q^*$, set $X = g^x$ and invokes $\mathcal{B}_{\hat{F}}$. For every query (Y,Z) of $\mathcal{B}_{\hat{F}}$, $\mathcal{B}_F$ checks whether $Y^x = Z$, computes $\psi_X(Y,Z)$

accordingly and queries O. The response of the oracle is passed to $\mathcal{B}_{\hat{F}}$. When $\mathcal{B}_{\hat{F}}$ outputs a bit b, $\mathcal{B}_F$ outputs b. This perfectly simulates the prf game of $\hat{F}_X$.

We assume the computation time of ψ_X is constant. In order to simulate the prf game of $\hat{F}_X$, $\mathcal{B}_F$ needs to compute ψ_X for q many times. Moreover, $\mathcal{B}_F$ needs store x and a temporary variable for passing the values. The lemma follows. □

The Reduction. Theorem 4 is proven via a sequence of games. Formal description of the games are given in Figure 8, and Figure 9.

```
$\mathbf{G_0}$ $\boxed{\mathbf{G_1}}$
1 :  $(pk, sk) \leftarrow \texttt{Gen}(1^\lambda)$
2 :  Parse $pk = (g, X)$
3 :  Parse $sk = x$
4 :  $y^* \xleftarrow{\$} \mathbb{Z}_q^*$
5 :  $b \xleftarrow{\$} \{0,1\}$
6 :  $Y^* = g^{y^*}$
7 :  $Z^* = Y^{*x}$
8 :  $K_0^* = \texttt{H}(Y^*, Z^*)$ $\boxed{K_0^* \xleftarrow{\$} \mathcal{K}}$
9 :  $K_1^* \xleftarrow{\$} \mathcal{K}$
10 :  $b^* \leftarrow \mathcal{A}^{\texttt{Decap},\texttt{H}}(pk, Y^*, K_b^*)$
11 :  if $b = b^*$ return 1
12 :  else return 0
13 :  endif
```

```
Procedure $\texttt{H}(Y, Z)$ in $\mathbf{G_0}$
1 :  if $\texttt{H}(Y, Z)$ is undefined
2 :    $\texttt{H}(Y, Z) \xleftarrow{\$} \mathcal{K}$
3 :  endif
4 :  return $\texttt{H}(Y, Z)$
```

```
Procedure $\texttt{Decap}(Y)$ in $\mathbf{G_0}, \mathbf{G_1}$
1 :  if $Y = Y^*$ return $\perp$
2 :  $Z = Y^x$
3 :  $K = \texttt{H}(Y, Z)$
4 :  return $K$
```

```
Procedure $\texttt{H}(Y, Z)$ in $\mathbf{G_1}$
1 :  if $Z = Y^x \wedge Y = Y^*$
2 :    return $K_0^*$
3 :  else
4 :    if $\texttt{H}(Y, Z)$ is undefined
5 :      $\texttt{H}(Y, Z) \xleftarrow{\$} \mathcal{K}$
6 :    endif
7 :  return $\texttt{H}(Y, Z)$
8 :  endif
```

Fig. 8. The games $\mathbf{G}_0$ and $\mathbf{G}_1$. In game $\mathbf{G}_1$, Line 8 in replaced by the boxed statement

Game $\mathbf{G}_0$. The game $\mathbf{G}_0$ is the original IND-CCA game.

$$\mathbf{Adv}_{\mathcal{A},\Pi}^{\text{IND-CCA}} \overset{def}{=} \left| \text{Prob}[\mathbf{G}_0^{\mathcal{A}} = 1] - \frac{1}{2} \right| .$$

Game $\mathbf{G}_1$: We predefine $K_0^* = \texttt{H}(Y^*, Z^*)$ by sampling a random element from the keyspace $\mathcal{K}$. Y^* is the challenge ciphertext sent in the KEM game and $Z^* = Y^{*x}$. The hash oracle is modified to return K_0^* for the input (Y^*, Z^*). As K_0^* is still uniformly chosen at random, and the hash oracle output is consistent, there is no change in the distribution of adversary's view.

$$\text{Prob}[\mathbf{G}_0^{\mathcal{A}} = 1] = \text{Prob}[\mathbf{G}_1^{\mathcal{A}} = 1]$$

Game $\mathbf{G}_2$. In this game the oracles H and Decap are changed. We replace the random oracle by a prf $\hat{F}_X : \{0,1\}^\lambda \times \mathbb{G} \times \mathbb{G} \to \mathcal{K}$. By Lemma 1, there exists an adversary $\mathcal{B}_{\hat{F}}$ such that

$$\left|\text{Prob}[\mathbf{G}_1^{\mathcal{A}} = 1] - \text{Prob}[\mathbf{G}_2^{\mathcal{A}} = 1]\right| \leq \mathbf{Adv}_{\mathcal{B}_{\hat{F}}, \hat{F}_X}^{\text{prf}}$$

Game $\mathbf{G}_3$. We rewrite the prf evaluation of $\hat{F}_X$ using a prf F as defined in Construction 5. In the procedure $\texttt{Decap}$ of the game $\mathbf{G}_2$, Step 2 ($Z = Y^x$) ensures that $\hat{F}_X(k, Y, Z)$ in that procedure always evaluates to $F(k, 1, Y, g)$. As the view of the adversary remains unchanged,

$$\mathrm{Prob}[\mathbf{G}_2^{\mathcal{A}} = 1] = \mathrm{Prob}[\mathbf{G}_3^{\mathcal{A}} = 1]$$

Game $\mathbf{G}_4$: In this game, we set a flag $\textsc{Flag}$ and abort on the event that $\mathcal{A}$ queries $\texttt{H}$ on (Y^*, Z^*) where Y^* is the challenge in the KEM game and (X, Y^*, Z^*) is a valid diffie hellman tuple. By the fundamental lemma of game playing proofs

$$\left|\mathrm{Prob}[\mathbf{G}_3^{\mathcal{A}} = 1] - \mathrm{Prob}[\mathbf{G}_4^{\mathcal{A}} = 1]\right| \leq \mathrm{Prob}[\textsc{Flag} = 1].$$

In the game $\mathbf{G}_4$, the adversary $\mathcal{A}$ is unable to compute $\texttt{H}(Y^*, Z^*)$ using either the hash oracle or the decapsulation oracle. The decapsulation oracle outputs $\perp$ whenever the input Y is equal to Y^*. The hash oracle aborts for the input (Y^*, Z^*). This implies that the bit b is independent from the adversary's view. Hence

$$\mathrm{Prob}[\mathbf{G}_3^{\mathcal{A}}] = \frac{1}{2}.$$

To bound $\mathrm{Prob}[\textsc{Flag} = 1]$, we construct an algorithm $\mathcal{B}_{DH}$ against the $\mathsf{gap\text{-}DH}$ security of $\mathbb{G}$. $\mathcal{B}_{DH}$ simulates game $\mathbf{G}_4$ for $\mathcal{A}$.

$\mathsf{gap\text{-}DH}$ adversary $\mathcal{B}_{DH}$. Formal code of $\mathcal{B}_{DH}$ is given in Figure 10. $\mathcal{B}_{DH}$ simulates $\mathbf{G}_4$. In order to execute line 1 of the game $\mathbf{G}_4$, $\mathcal{B}_{DH}$ uses the $\texttt{DDH}$ oracle. By the definition of $\mathsf{gap\text{-}DH}$ game, X and Y^* are uniformly and independently distributed. Hence the simulation of $\mathbf{G}_4$ is perfect. $\textsc{Flag} = 1$ implies that $\mathcal{A}$ queried $\texttt{H}(Y, Z)$ where $Y = Y^*$ and $\texttt{DDH}(X, Y^*, Z) = 1$. $\mathcal{B}_{DH}$ returns that Z and wins the $\mathsf{gap\text{-}DH}$ game. Hence,

$$\mathrm{Prob}[\textsc{Flag} = 1] = \mathbf{Adv}_{\mathcal{B}_{DH}, G}^{\mathsf{gap\text{-}DH}}$$

Collecting the probabilities, we get

$$\mathbf{Adv}_{\mathcal{A}, \Pi}^{\text{IND-CCA}} \leq \mathbf{Adv}_{\mathcal{B}_{DH}, \mathbb{G}}^{\mathsf{gap\text{-}DH}} + \mathbf{Adv}_{\mathcal{B}_{\hat{F}}, \hat{F}}^{\text{prf}}$$

Efficiency of $\mathcal{B}_{DH}$. $\mathcal{B}_{DH}$ runs $\mathcal{A}$, queries $\texttt{DDH}$ oracle for q_H many times, computes the prf F for $(q_H + q_D)$ many times. $\mathcal{O}(\mathrm{poly}(\lambda))$ is the cost of other operations in $\mathbf{G}_4$.

$$\mathbf{LocalTime}(\mathcal{B}_{DH}) \approx \mathbf{LocalTime}(\mathcal{A}) + (q_H + q_D)\mathbf{LocalTime}(F) + q_H$$

The last q_H term in the right-hand side of the above equation is to denote the number of queries made to the $\texttt{DDH}$ oracle.

```
G_2
$(pk, sk) \leftarrow \mathtt{Gen}(1^\lambda)$
Parse $pk = (g, X)$
Parse $sk = x$
$k \xleftarrow{\$} \{0,1\}^\lambda$
$y^* \xleftarrow{\$} \mathbb{Z}_q^*$
$b \xleftarrow{\$} \{0,1\}$
$Y^* = g^{y^*}$
$Z^* = Y^{*x}$
$K_0^* \xleftarrow{\$} \mathcal{K}$
 $K_1^* \xleftarrow{\$} \mathcal{K}$
 $b^* \leftarrow \mathcal{A}^{\mathtt{Decap},\mathtt{H}}(pk, Y^*, K_b^*)$
 if $b = b^*$return 1
 else return 0
 endif

Procedure $\mathtt{H}(Y, Z)$
if $Z = Y^x \wedge Y = Y^*$
  return $K_0^*$
else
   $K = \hat{F}_X(k, Y, Z)$
return $K$
endif

Procedure $\mathtt{Decap}(Y)$
if $Y = Y^*$return $\perp$
$Z = Y^x$
$K = \hat{F}_X(k, Y, Z)$
return $K$
```

```
G_3
$(pk, sk) \leftarrow \mathtt{Gen}(1^\lambda)$
Parse $pk = (g, X)$
Parse $sk = x$
$k \xleftarrow{\$} \{0,1\}^\lambda$
$y^* \xleftarrow{\$} \mathbb{Z}_q^*$
$b \xleftarrow{\$} \{0,1\}$
$Y^* = g^{y^*}$
$Z^* = Y^{*x}$
$K_0^* \xleftarrow{\$} \mathcal{K}$
 $K_1^* \xleftarrow{\$} \mathcal{K}$
 $b^* \leftarrow \mathcal{A}^{\mathtt{Decap},\mathtt{H}}(pk, Y^*, K_b^*)$
 if $b = b^*$return 1
 else return 0
 endif

Procedure $\mathtt{H}(Y, Z)$
if $Z = Y^x$
  if $Y = Y^*$
    return $K_0^*$
  else
    $K = F(k, 1, Y, g)$
  endif
else
  $K = F(k, 0, Y, Z)$
endif
 return $K$

Procedure $\mathtt{Decap}(Y)$
if $Y = Y^*$return $\perp$
$Z = Y^x$
$K = F(k, 1, Y, g)$
return $K$
```

```
G_4
$(pk, sk) \leftarrow \mathtt{Gen}(1^\lambda)$
Parse $pk = (g, X)$
Parse $sk = x$
$k \xleftarrow{\$} \{0,1\}^\lambda$
$y^* \xleftarrow{\$} \mathbb{Z}_q^*$
$b \xleftarrow{\$} \{0,1\}$
$Y^* = g^{y^*}$
$Z^* = Y^{*x}$
$K_0^* \xleftarrow{\$} \mathcal{K}$
 $K_1^* \xleftarrow{\$} \mathcal{K}$
 $b^* \leftarrow \mathcal{A}^{\mathtt{Decap},\mathtt{H}}(pk, Y^*, K_b^*)$
 if $b = b^*$return 1
 else return 0
 endif

Procedure $\mathtt{H}(Y, Z)$
if $Z = Y^x$
  if $Y = Y^*$
    FLAG=1
    Abort
  endif
  $K = F(k, 1, Y, g)$
else
  $K = F(k, 0, Y, Z)$
endif
 return $K$

Procedure $\mathtt{Decap}(Y)$
if $Y = Y^*$return $\perp$
-
$K = F(k, 1, Y, g)$
return $K$
```

Fig. 9. IND-CCA game of HEG: highlighted statements are the modifications from the previous game

Memory Efficiency of $\mathcal{B}_{DH}$. $\mathcal{B}_{DH}$ needs to save the code of $\mathcal{A}$, and F. In addition, counting the registers in $\mathbf{G}_4$,

$$\mathbf{LocalMem}(\mathcal{B}_{DH}) \approx \mathbf{LocalMem}(\mathcal{A}) + \mathbf{LocalMem}(F) + 7\lambda + 1$$

So far, we have proven that there exist adversaries $\mathcal{B}_{DH}$ and $\mathcal{B}_{\hat{F}}$

$$\mathbf{Adv}^{\text{IND-CCA}}_{\mathcal{A},\Pi} \leq \mathbf{Adv}^{\mathsf{gap\text{-}DH}}_{\mathcal{B}_{DH},G} + \mathbf{Adv}^{\mathrm{prf}}_{\mathcal{B}_{\hat{F}},\hat{F}}$$

```
Algorithm $\mathcal{B}_{DH}(g, X, Y^*)$
  Set $pk = (g, X)$
  $k \xleftarrow{\$} \{0,1\}^\lambda$
  $K^* \xleftarrow{\$} \mathcal{K}$
  $b^* \leftarrow \mathcal{A}^{\mathtt{Decap},\mathtt{H}}(pk, Y^*, K^*)$
  output $\perp$ .

Procedure $\mathtt{Decap}(Y)$
  if $Y = Y^*$ return $\perp$
  $K = F(k, 1, Y, g)$
  return $K$

Procedure $\mathtt{H}(Y, Z)$
  if $\mathtt{DDH}(X, Y, Z) = 1$
    if $Y = Y^*$
      FLAG $= 1$
      Output $Z$
    else
      $K = F(k, 1, Y, g)$
    endif
  else
    $K = F(k, 0, Y, Z)$
  endif
  return $K$
```

Fig. 10. Diffie Hellman adversary $\mathcal{B}_{DH}$

Applying Lemma 2, we get the adversary $\mathcal{B}_F$ such that

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathcal{B}_{\hat{F}},\hat{F}} = \mathbf{Adv}^{\mathrm{prf}}_{\mathcal{B}_F,F}$$

Hence, there exist adversaries $\mathcal{B}_{DH}$ and $\mathcal{B}_F$ such that

$$\mathbf{Adv}^{\text{IND-CCA}}_{\mathcal{A},\Pi} \leq \mathbf{Adv}^{\textsf{gap-DH}}_{\mathcal{B}_{DH},G} + \mathbf{Adv}^{\mathrm{prf}}_{\mathcal{B}_F,F}$$

The following lemma finds the efficiency of $\mathcal{B}_F$

Lemma 3.

$$\begin{aligned}\mathbf{LocalTime}(\mathcal{B}_F) &\approx \mathbf{LocalTime}(\mathcal{A}) + \mathbf{LocalTime}(\mathtt{DH}) + 2(q_H + q_D)\\ \mathbf{LocalMem}(\mathcal{B}_F) &\approx \mathbf{LocalMem}(\mathcal{A}) + \mathbf{LocalMem}(\mathtt{DH}) + 11\lambda + 2\end{aligned}$$

3.2 ECIES

Let $\mathbb{G} = \langle g \rangle$ be a cyclic group of prime order q, equipped with a pairing $\hat{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. Let $\mathtt{H} : \mathbb{G} \to \mathcal{K}$ be a hash function. In this section, we present a memory tight reduction of the underlying Key Encapsulation Mechanism of ECIES from the Computational Diffie-Hellman assumption over $\mathbb{G}$. We describe the ECIES KEM scheme in Figure 11. Our main result in this section is the following theorem.

Theorem 6. *Let q be a prime and $\mathbb{G}$ be a group of order q equipped with a pairing $\hat{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. Let $\mathtt{DH}$ be the Diffie Hellman instance generation algorithm over $\mathbb{G}$. Let $F : \{0,1\}^\lambda \times \mathbb{G}_T \to \mathcal{K}$ be a prf. Let $\hat{\Pi}$ be the ECIES-KEM scheme over $\mathbb{G}$ and $\mathcal{K}$, with security parameter λ.*

```
Procedure Gen$(1^\lambda)$
    $(q, g, \mathbb{G}) \leftarrow \mathtt{DH}(1^\lambda)$
    $x \xleftarrow{\$} \mathbb{Z}_p^*$
    $pk = (g, g^x)$
    $sk = x$
    return (pk, sk)
```

```
Procedure Encap(pk)
    $(g, X) = \mathrm{pk}$
    $y \xleftarrow{\$} \mathbb{Z}_p^*$
    $Y = g^y$
    $Z = X^y$
    $K = \mathtt{H}(Z)$
    return $(Y, K)$
```

```
Procedure Decap(sk, $Y$)
    $x = sk$
    $Z = Y^x$
    $K = \mathtt{H}(Z)$
    return $K$
```

Fig. 11. ECIES KEM. $\mathtt{H} : \{0,1\}^\lambda \times \mathbb{G} \to \mathcal{K}$ is a cryptographic hash function

Let $\mathcal{A}$ be an adversary in the IND-CCA*game of $\hat{\Pi}$. Suppose $\mathcal{A}$ makes q_h hash queries and q_D decapsulation queries. Then, in the random oracle model, there exists an adversary $\mathcal{B}_{DH}$ in the* CDH *game, and an adversary $\mathcal{B}_F$ such that*

$$\mathbf{Adv}^{\text{IND-CCA}}_{\mathcal{A},\hat{\Pi}} \leq \mathbf{Adv}^{\mathsf{CDH}}_{\mathcal{B}_{DH},\mathbb{G}} + \mathbf{Adv}^{\text{prf}}_{\mathcal{B}_F,F}$$

Moreover, it holds that

$$\begin{aligned}
\mathbf{LocalTime}(\mathcal{B}_{DH}) \approx & \mathbf{LocalTime}(\mathcal{A}) + (q_H + q_D)\mathbf{LocalTime}(F) + \\
& (q_D + 3q_H)\mathbf{LocalTime}(\hat{e}) \\
\mathbf{LocalMem}(\mathcal{B}_{DH}) \approx & \mathbf{LocalMem}(\mathcal{A}) + \mathbf{LocalMem}(F) + 7\lambda + 1 \\
\mathbf{LocalTime}(\mathcal{B}_F) \approx & \mathbf{LocalTime}(\mathcal{A}) + \mathbf{LocalTime}(\mathtt{DH}) + (q_H + q_D) \\
& (q_H + q_D)\mathbf{LocalTime}(\hat{e}) \\
\mathbf{LocalMem}(\mathcal{B}_F) \approx & \mathbf{LocalMem}(\mathcal{A}) + \mathbf{LocalMem}(\mathtt{DH}) + 12\lambda + 2
\end{aligned}$$

The reduction to prove Theorem 6 is almost the same as in the previous section. The only difference is in the construction of the intermediate prf $\hat{F}$ and the reduced CDH-adversary $\mathcal{B}_{DH}$. As the details are almost similar to the reduction of HEG, we only describe $\hat{F}$ and $\mathcal{B}_{DH}$ here. The rest of the reduction including the games and the analysis is moved to Section 8.

Construction of $\hat{F}$.

Construction 7 *Let $\mathbb{G}$ be a group of prime order q and let g be a generator of $\mathbb{G}$. Let $\hat{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be a bilinear map. Let $F : \{0,1\}^\lambda \times \mathbb{G}_T \to \mathcal{K}$. We define $\hat{F} : \{0,1\}^\lambda \times \mathbb{G} \to \mathcal{K}$ as follows*

$$\hat{F}(k, Z) = F(k, \hat{e}(g, Z))$$

Lemma 4. *If F is a prf, then $\hat{F}$ is a prf. Moreover, for every adversary $\mathcal{B}_{\hat{F}}$ against $\hat{F}$, there exists a $\mathcal{B}_F$ against F such that,*

$$\begin{aligned}\mathbf{Adv}^{\mathrm{prf}}_{\mathcal{B}_F,F} &= \mathbf{Adv}^{\mathrm{prf}}_{\mathcal{B}_{\hat{F}},\hat{F}}\\ \mathbf{LocalTime}(\mathcal{B}_F) &= \mathbf{LocalTime}(\mathcal{B}_{\hat{F}}) + q \cdot \mathbf{LocalTime}(\hat{e})\\ \mathbf{LocalMem}(\mathcal{B}_F) &= \mathbf{LocalMem}(\mathcal{B}_{\hat{F}}) + 2\lambda.\end{aligned}$$

where q is the number of queries made by $\mathcal{B}_{\hat{F}}$ to its oracle.

Proof. Let O be the oracle in the prf game of F. The adversary $\mathcal{B}_F$ fixes g, the generator of $\mathbb{G}$, and invokes $\mathcal{B}_{\hat{F}}$. For every query Z of $\mathcal{B}_{\hat{F}}$, $\mathcal{B}_F$ queries O with $\hat{e}(g, Z)$. The response of the oracle is passed to $\mathcal{B}_{\hat{F}}$. This perfectly simulates the prf game of $\hat{F}$. When $\mathcal{B}_{\hat{F}}$ outputs a bit b, $\mathcal{B}_F$ outputs the same bit b.

To simulate the prf game of $\hat{F}$, $\mathcal{B}_F$ computes the pairing for q times, where q is the number of queries made by $\mathcal{B}_{\hat{F}}$. $\mathcal{B}_F$ needs store g and a temporary variable for passing the values. The lemma follows. □

Description of $\mathcal{B}_{DH}$: the adversary to game CDH. Formal code of $\mathcal{B}_{DH}$ is given in Figure 12. $\mathcal{B}_{DH}$ gets (g, X, Y^*) as input, where X, Y^* are distributed uniformly over $\mathbb{G}$. FLAG $= 1$ implies that $\mathcal{A}$ queried $\mathtt{H}(Z)$ where (X, Y^*, Z) is a valid Diffie Hellman tuple. If FLAG is set for some query made by $\mathcal{A}$, $\mathcal{B}_{DH}$ returns that corresponding Z and wins the CDH game.

```
Algorithm $\mathcal{B}_{DH}((g, X, Y^*))$
  Set $pk = (g, X)$
  $k \xleftarrow{\$} \{0,1\}^\lambda$
  $K^* \xleftarrow{\$} \mathcal{K}$
  $b^* \leftarrow \mathcal{A}^{\mathtt{Decap},\mathtt{H}}(pk, Y^*, K^*)$
  output $\perp$ .

Procedure Decap$(Y)$
  if $Y = Y^*$return $\perp$
  $K = F(k, \hat{e}(X, Y))$
  return $K$

Procedure H$(Z)$
  if $\hat{e}(g, Z) = \hat{e}(X, Y^*)$
    FLAG $= 1$
    Output $Z$
  else
    $K = F(k, \hat{e}(g, Z))$
  return $K$
  endif
```

Fig. 12. Diffie Hellman adversary $\mathcal{B}_{DH}$

Efficiency of $\mathcal{B}_{DH}$. $\mathcal{B}_{DH}$ runs $\mathcal{A}$, computes the pairing $\hat{e}(.,.)$ oracle for $q_D + 3q_H$ many times, computes the prf F for $(q_H + q_D)$ many times. As the rest of the steps in the algorithm takes $\mathcal{O}(\mathrm{poly}(\lambda))$ time,

$$\begin{aligned}\mathbf{LocalTime}(\mathcal{B}_{DH}) \approx &\mathbf{LocalTime}(\mathcal{A}) + (q_H + q_D)\mathbf{LocalTime}(F) +\\ &(q_D + 3q_H)\mathbf{LocalTime}(\hat{e})\end{aligned}$$

Memory Efficiency of $\mathcal{B}_{DH}$. $\mathcal{B}_{DH}$ needs to save the code of $\mathcal{A}$, $\hat{e}$, and F. Counting the registers, we get

$$\textbf{LocalMem}(\mathcal{B}_{DH}) = \textbf{LocalMem}(\mathcal{A}) + \textbf{LocalMem}(F) + 7\lambda + 1$$

4 Transformation V: OW-PCA PKE to OW-PCVA PKE

In this section, we introduce a transformation V to construct OW-PCVA secure deterministic PKE from a OW-PCA secure PKE. Our main result is a memory-tight reduction of V. The main application of V will be in Section 5, where we shall use V to get a memory-tight reductions of the IND-CCA security of QKEM$^{\perp}$ and QKEM$_{m}^{\perp}$.

4.1 The Transformation

We start with a deterministic δ-correct OW-PCA secure public key encryption scheme, $\mathsf{PKE} = (\mathtt{Gen}, \mathtt{Enc}, \mathtt{Dec})$. Let $\mathcal{M} = \{0,1\}^n$ be the message space, and $\mathcal{C}$ be the ciphertext space. Let $\mathtt{H}' : \mathcal{M} \rightarrow \{0,1\}^{\eta}$ be a hash function. The transformed scheme is described as $\mathsf{PKE}_1 = (\mathtt{Gen}, \mathtt{Enc}_1, \mathtt{Dec}_1)$.

```
Procedure $\mathtt{Enc}_1(pk, m)$
    $c_1 = \mathtt{Enc}(pk, m)$
    $c_2 = \mathtt{H}'(m)$
    $c = c_1 || c_2$
    return $c$

Procedure $\mathtt{Dec}_1(sk, c)$
    Parse $c = (c_1, c_2)$
    $m' = \mathtt{Dec}(sk, c_1)$
    if $m' = \perp \vee \mathtt{H}'(m') \neq c_2 \ \vee \mathtt{Enc}(pk, m') \neq c_1$
        return $\perp$
    else return $m'$
```

Fig. 13. OW-PCVA secure encryption scheme $\mathsf{PKE}_1 = V[\mathsf{PKE}]$

Our main result of this section is the following theorem.

Theorem 8. *Let* $\mathsf{PKE} = (\mathtt{Gen}, \mathtt{Enc}, \mathtt{Dec})$ *be a deterministic* δ *correct* OW-PCA *secure public key encryption scheme. Let* $\mathcal{M}$ *be the message space, and* $\mathcal{C}$ *be the ciphertext space of* PKE. *Let* PKE_1 *be the transformed public encryption scheme. Let* $F' : \{0,1\}^{\lambda} \times \mathcal{C} \rightarrow \{0,1\}^{\eta}$ *be a prf. Let* $\mathcal{A}$ *be any adversary in the* OW-PCVA *game of* PKE_1. *Suppose* $\mathcal{A}$ *makes* $q_{h'}$ *queries to* $\mathtt{H}'$. *Let* q_P *denote the number of plaintext checking queries and* q_V *denote the number of validity checking queries made by* $\mathcal{A}$.

PKE_1 *is* δ*-correct. Moreover, in the random oracle model, there exists an adversary* $\mathcal{B}$ *in the* OW-PCA *game of* PKE_1, *and an adversary* $\mathcal{B}_{F'}$ *in the* prf *game of* F', *such that*

$$\mathbf{Adv}^{\text{OW-PCVA}}_{\mathcal{A},\mathsf{PKE}_1} \leq \mathbf{Adv}^{\text{OW-PCA}}_{\mathcal{B},\mathsf{PKE}} + 2 \cdot \mathbf{Adv}^{\text{prf}}_{\mathcal{B}_{F'},F'} + \frac{q_V}{2^\eta} + 2\delta(1 + q_{h'} + q_P + q_V)$$

Moreover it holds that

$$\begin{aligned}
\mathbf{LocalTime}(\mathcal{B}) \approx & \mathbf{LocalTime}(\mathcal{A}) + q_{h'}\mathbf{LocalTime}(\texttt{Enc}) \\
& + (1 + q_{h'} + q_V + q_P)\mathbf{LocalTime}(F') + q_P \\
\mathbf{LocalMem}(\mathcal{B}) \approx & \mathbf{LocalMem}(\mathcal{A}) + \mathbf{LocalMem}(F') \\
& + \mathbf{LocalMem}(\texttt{Enc}) + 8\lambda \\
\mathbf{LocalTime}(\mathcal{B}_{F'}) \approx & \mathbf{LocalTime}(\mathcal{A}) + \mathbf{LocalTime}(\texttt{Gen}) + (q_V + q_P)\mathbf{LocalTime}(\texttt{Dec}) \\
& + (1 + q_V + q_P + q_{h'})(1 + 2 \cdot \mathbf{LocalTime}(\texttt{Enc})) \\
\mathbf{LocalMem}(\mathcal{B}_{F'}) \approx & \mathbf{LocalMem}(\mathcal{A}) + \mathbf{LocalMem}(\texttt{Gen}) + +\mathbf{LocalMem}(\texttt{Enc}) \\
& + \mathbf{LocalMem}(\texttt{Dec}) + 11\lambda + 1
\end{aligned}$$

Similar to previous section, we first construct a prf $\hat{F}$.

4.2 Construction of $\hat{F}$

Construction 9 *Fix a public key pk of* PKE*. Let $F' : \{0,1\}^\lambda \times \mathcal{C} \to \{0,1\}^\eta$. We define $\hat{F}$ as*

$$\hat{F}(k, m) = F'(k, \texttt{Enc}(pk, m))$$

In order to use the map then prf technique, we need the following lemma.

Lemma 5. *Fix pk. For every* prf*-adversary $\mathcal{B}_{\hat{F}}$ against $\hat{F}$, there exists a $\mathcal{B}_{F'}$ against F' such that,*

$$\begin{aligned}
\mathbf{Adv}^{\text{prf}}_{\mathcal{B}_{\hat{F}},\hat{F}} &\leq \mathbf{Adv}^{\text{prf}}_{\mathcal{B}_{F'},F'} + \delta(q) \\
\mathbf{LocalTime}(\mathcal{B}_{F'}) &= \mathbf{LocalTime}(\mathcal{B}_{\hat{F}}) + q \cdot \mathbf{LocalTime}(\texttt{Enc}) \\
\mathbf{LocalMem}(\mathcal{B}_{F'}) &= \mathbf{LocalMem}(\mathcal{B}_{\hat{F}}) + 3\lambda.
\end{aligned}$$

where q is the number of queries made by $\mathcal{B}_{\hat{F}}$.

The main difference in Lemma 5 with the ones in the previous section is the decryption error of PKE. In other words, we can not claim that $\texttt{Enc}(pk, .)$ is an injective function. However, if $\mathcal{B}_{\hat{F}}$ can query with messages m_1, m_2 such that $\texttt{Enc}(pk, m_1) = \texttt{Enc}(pk, m_2)$, implying a decryption error for either m_1 or m_2.

Proof. First, we prove that if F' is a prf, then $\hat{F}$ is a prf. Let $\texttt{O}$ be the oracle of $\mathcal{B}_{F'}$. $\mathcal{B}_{F'}$ runs $\texttt{Gen}$ to receive pk, sk, and invokes $\mathcal{B}_{\hat{F}}$. For every query m of $\mathcal{B}_{\hat{F}}$, $\mathcal{B}_{F'}$, computes $c = \texttt{Enc}(pk, m)$, and checks whether $m = \texttt{Dec}(sk, c)$. If the check fails $\mathcal{B}_{F'}$ aborts. If the check succeeds, $\mathcal{B}_{F'}$ queries $\texttt{O}(c)$, and the response of the oracle is passed to $\mathcal{B}_{\hat{F}}$. When $\mathcal{B}_{\hat{F}}$ outputs a bit b, $\mathcal{B}_F$ outputs b.

If $\mathcal{B}_{F'}$ aborts on input m, then correctness error occurs in $\mathtt{Dec}(sk, \mathtt{Enc}(pk, m))$. By assumption, probability of this event is bounded by $\delta(q)$. Conditioned on that $\mathcal{B}_{F'}$ does not abort, the output of $\mathtt{Enc}(pk, m)$ are unique for all m queried by $\mathcal{B}_{\hat{F}}$. In that case, $\mathcal{B}_{F'}$ perfectly simulates the prf game of $\hat{F}$. When $\mathtt{O}$ is a random function, the simulation implements a random function. When $\mathtt{O}$ is implemented by F', $\mathcal{B}_{F'}$ implements $\hat{F}$. Thus we get,

$$\mathbf{Succ}_{\mathcal{B}_{\hat{F}},\mathrm{prf}[\hat{F}]} = \mathbf{Succ}_{\mathcal{B}_{F'},\mathrm{prf}[F']} + \mathrm{Prob}[\mathcal{B}_{F'} \text{ aborts}] \leq \mathbf{Succ}_{\mathcal{B}_{F'},\mathrm{prf}[F']} + \delta(q)$$

$$\implies \mathbf{Adv}^{\mathrm{prf}}_{\mathcal{B}_{\hat{F}},\hat{F}} \leq \mathbf{Adv}^{\mathrm{prf}}_{\mathcal{B}_{F'},F'} + \delta(q)$$

In order to simulate the prf game of $\hat{F}$, $\mathcal{B}_F$ needs to run $\mathtt{Enc}$ for q many times. Moreover, $\mathcal{B}_F$ needs store pk, sk and a temporary variable for passing the values. The lemma follows.

4.3 Proof of Theorem 8

It is obvious that the correctness holds. We prove rest of Theorem 8 via a sequence of games. Formal description of the games are given in the Figure 14 and Figure 15.

```
$\mathbf{G_0}$, $\boxed{\mathbf{G_1\text{-}G_7}}$
$(pk, sk) \xleftarrow{\$} \mathtt{Gen}$
$m^* \xleftarrow{\$} \mathcal{M}$
$\boxed{k' \xleftarrow{\$} \{0,1\}^\lambda}$
$c_2 = \mathtt{H}'(m^*)$
$c_1 = \mathtt{Enc}(pk, m^*)$
$c^* = (c_1, c_2)$
$m \leftarrow \mathcal{A}^{\mathtt{PCO},\mathtt{CVO},\mathtt{H}'}(pk, c^*)$
if $m^* = m$ return $1$
else return $0$
```

```
Game $\mathbf{G_0}$

Procedure $\mathtt{PCO}(m, c)$
Parse $c = c_1||c_2$
$m' = \mathtt{Dec}(sk, c_1)$
$c_1' = \mathtt{Enc}(pk, m')$
$c_2' = \mathtt{H}'(m')$
$c' = c_1'||c_2'$
if $m' = m$ and $c' = c$
    return $1$
else
    return $0$
```

```
Procedure $\mathtt{H}'(m)$
if $\mathtt{H}'(m)$ is undefined
    $\mathtt{H}'(m) \xleftarrow{\$} \mathcal{M}$
endif
return $\mathtt{H}'(m)$
```

```
Procedure $\mathtt{CVO}(c)$
Parse $c = c_1||c_2$
$m' = \mathtt{Dec}(sk, c_1)$
$c_1' = \mathtt{Enc}(pk, m')$
$c_2' = \mathtt{H}'(m')$
$c' = c_1'||c_2'$
if $m' \in \mathcal{M}$ and $c' = c$
    return $1$
else
    return $0$
```

Fig. 14. The main function of games $\mathbf{G_0}$-$\mathbf{G_7}$. The boxed statement is not executed in $\mathbf{G_0}$. Right hand side figure describes the oracles in $\mathbf{G_0}$

Game $\mathbf{G_0}$. G_0 is the OW-PCVA security game of PKE_1.

$$\mathbf{Adv}^{\text{OW-PCVA}}_{\mathcal{A},\mathsf{PKE}_1} = \mathrm{Prob}[G_0^{\mathcal{A}} = 1]$$

Game $\mathbf{G_1}$. In this game, we replace $\mathtt{H}'$ by prf $\hat{F}$. By Lemma 1, there exists adversary, $\mathcal{B}_{\hat{F}}$ such that

$$\left|\text{Prob}[G_1^{\mathcal{A}} = 1] - \text{Prob}[G_0^{\mathcal{A}} = 1]\right| \leq \mathbf{Adv}^{\text{prf}}_{\mathcal{B}_{\hat{F}},\hat{F}} \tag{1}$$

Game $\mathbf{G_1}$

```
Procedure H'(m)
  $h' = \hat{F}(k', m)$
  return $h'$

Procedure PCO(m, c)
  Parse $c = c_1||c_2$
  $m' = \mathtt{Dec}(sk, c_1)$
  $c_1' = \mathtt{Enc}(pk, m')$
  $c_2' = \hat{F}(k', m')$
  $c' = c_1'||c_2'$
  if $m' = m$ and $c' = c$
    return 1
  else
    return 0

Procedure CVO(c)
  Parse $c = c_1||c_2$
  $m' = \mathtt{Dec}(sk, c_1)$
  $c_1' = \mathtt{Enc}(pk, m')$
  $c_2' = \hat{F}(k', m')$
  $c' = c_1'||c_2'$
  if $m' \in \mathcal{M}$ and $c' = c$
    return 1
  else
    return 0
```

Game $\mathbf{G_2}$

```
Procedure H'(m)
  $h' = \hat{F}(k', m)$
  return $h'$

Procedure PCO(m, c)
  Parse $c = c_1||c_2$
  $c_2' = \hat{F}(k', m)$
  if $c_2' = c_2 \wedge \mathtt{Enc}(pk, m) = c_1$
    return 1
  else
    return 0

Procedure CVO(c)
  Parse $c = c_1||c_2$
  $m' = \mathtt{Dec}(sk, c_1)$
  $c_1' = \mathtt{Enc}(pk, m')$
  $c_2' = \hat{F}(k', m')$
  $c' = c_1'||c_2'$
  if $m' \in \mathcal{M}$ and $c' = c$
    return 1
  else
    return 0
```

Game $\mathbf{G_3}$

```
Procedure H'(m)
  $c = \mathtt{Enc}(pk, m)$
  $h' = F'(k', c)$
  return $h'$

Procedure PCO(m, c)
  Parse $c = c_1||c_2$
  $c_1' = \mathtt{Enc}(pk, m)$
  $c_2' = F'(k', c_1')$
  if $c_2' = c_2 \wedge c_1' = c_1$
    return 1
  else
    return 0

Procedure CVO(c)
  Parse $c = c_1||c_2$
  $m' = \mathtt{Dec}(sk, c_1)$
  $c_1' = \mathtt{Enc}(pk, m')$
  $c_2' = F'(k', c_1')$
  $c' = c_1'||c_2'$
  if $m' \in \mathcal{M}$ and $c' = c$
    return 1
  else
    return 0
```

Fig. 15. The oracles in $\mathbf{G_1}, \mathbf{G_2}, \mathbf{G_3}$

Game $\mathbf{G_2}$. In this game, we modify the $\mathtt{PCO}(m, c = (c_1, c_2))$ oracle simulation. Instead of the decryption, $m' = \mathtt{Dec}(sk, c_1)$, and equality check $m = m'$, we only check whether, $c_1 = \mathtt{Enc}(pk, m)$. Notice, the condition $c_2 = \hat{F}(k', m)$ remains unchanged. Conditioned on correctness error does not happen, $c_1' = c_1 = \mathtt{Enc}(pk, m)$ implies that $m' = \mathtt{Dec}(sk, c_1') = m$. Hence, this change does not affect the transcript distribution until correctness error occurs in PKE.

$$\left|\text{Prob}[G_1^{\mathcal{A}} = 1] - \text{Prob}[G_2^{\mathcal{A}} = 1]\right| \leq \delta(q_P)$$

Game $\mathbf{G_3}$. In this game we replace $\hat{F}$ as defined. The change is syntactical and does not change the distribution of any output.

$$\text{Prob}[G_2^{\mathcal{A}} = 1] = \text{Prob}[G_3^{\mathcal{A}} = 1]$$

Game $\mathbf{G_4}$. In this game, we change how the oracles PCO and CVO responds. For a $\texttt{PCO}(m, c)$ query, we no longer encrypt m to compute c_2'. Instead, we run the plaintext checking oracle $\overline{\texttt{PCO}}$, provided for PKE, to check correctness of (m, c_1). If c_1 is indeed a valid ciphertext of m, then by deterministic property of PKE, $F'(k, \texttt{Enc}(pk, m))$ is equal to $F'(k, c_1)$. Hence we only check whether $F'(k, c_1) = c_2$. The change in PCO is syntactical, and does not change output distribution of the oracle.

Similarly, in CVO, we change the computation of c_2', which is now computed as $F(k', c_1)$. If $c_1 = c_1'$, then the change is syntactical and has no effect in the check in Step 5. If $c_1 \neq c_1'$, the condition in Step 5 rejects irrespective of the value of c_2'. Hence, this change does not change the output distribution of the oracles as well.

$$\text{Prob}[G_3^{\mathcal{A}} = 1] = \text{Prob}[G_4^{\mathcal{A}} = 1]$$

Game $\mathbf{G_5}$. We change the description of the oracle $\texttt{CVO}(c)$. We raise a flag BAD, if $c_2' = c_2$ but c_1 is not a valid ciphertext of PKE, i.e $m' \notin \mathcal{M}$ or $c_1 \neq \texttt{Enc}(pk, m')$ where $m' = \texttt{Dec}(c_1)$. However, we do not change the output of the oracle. $\texttt{CVO}(c)$ still return 0 when BAD is set.

$$\text{Prob}[G_4^{\mathcal{A}} = 1] = \text{Prob}[G_5^{\mathcal{A}} = 1]$$

Game $\mathbf{G_6}$. In game $\mathbf{G_6}$, $\texttt{CVO}(c)$ returns 1, when BAD is set. Rest of the games remain unchanged. By the fundamental lemma of game playing proofs,

$$\left|\text{Prob}[G_5^{\mathcal{A}} = 1] - \text{Prob}[G_6^{\mathcal{A}} = 1]\right| \leq \text{Prob}[\text{BAD}]$$

Note, in the game $\mathbf{G_6}$, the oracle CVO returns 1, if and only if $c_2 = F'(k', c_1)$. Game $\mathbf{G_7}$. We rewrite the description of $\texttt{CVO}(c)$. We no longer run Dec and Enc. The oracle $\texttt{CVO}(c)$ parses c as $c_1||c_2$, and returns 1 if $c_2 = F'(k', c_1)$ and returns 0 otherwise. Rest of the game remain unchanged. As the output distribution of all the procedures in $\mathbf{G_7}$ is same as that in $\mathbf{G_6}$.

$$\text{Prob}[G_6^{\mathcal{A}} = 1] = \text{Prob}[G_7^{\mathcal{A}} = 1]$$

Bounding $\mathbf{Prob}[G_7^{\mathcal{A}} = 1]$. In Figure 17, we construct an adversary $\mathcal{B}$ against OW-PCA security of PKE. $\mathcal{B}$ receives (pk, c^*), invokes $\mathcal{A}(pk, c^*)$ and perfectly simulates the game $\mathbf{G_7}$ for $\mathcal{A}$. When $\mathcal{A}$ returns a message m, $\mathcal{B}$ returns m.

$$\text{Prob}[G_7^{\mathcal{A}} = 1] = \mathbf{Adv}_{\mathcal{B},PKE}^{\text{OW-PCA}}$$

```
Game $\mathbf{G_4}$

Procedure $\mathtt{H}'(m)$
    $c = \mathtt{Enc}(pk, m)$
    $h' = F'(k', c)$
    return $h'$

Procedure $\mathtt{PCO}(m, c)$
    Parse $c = c_1 || c_2$
    if $\overline{\mathtt{PCO}}(m, c_1) = 1$
        $c_2' = F'(k', c_1)$
        if $c_2' = c_2$
            return 1
    endif
    endif
    return 0

Procedure $\mathtt{CVO}(c)$
    Parse $c = c_1 || c_2$
    $m' = \mathtt{Dec}(sk, c_1)$
    $c_1' = \mathtt{Enc}(pk, m')$
    $c_2' = F'(k', c_1)$
    if $c_2' = c_2 \wedge m' \in \mathcal{M} \wedge c_1' = c_1$
        return 1
    else
        return 0
```

```
Game $\mathbf{G_5}$ $\boxed{\mathbf{G_6}}$

Procedure $\mathtt{H}'(m)$
    $c = \mathtt{Enc}(pk, m)$
    $h' = F'(k', c)$
    return $h'$

Procedure $\mathtt{PCO}(m, c)$
    Parse $c = c_1 || c_2$
    if $\overline{\mathtt{PCO}}(m, c_1) = 1$
        $c_2' = F'(k', c_1)$
        if $c_2' = c_2$
            return 1
    endif
    endif
    return 0

Procedure $\mathtt{CVO}(c)$
    Parse $c = c_1 || c_2$
    $m' = \mathtt{Dec}(sk, c_1)$
    $c_1' = \mathtt{Enc}(pk, m')$
    $c_2' = F'(k', c_1)$
    if $c_2' = c_2$
        if $m' \notin \mathcal{M}$ or $c_1' \neq c_1$
            $Bad = 1$
            return 0 $\boxed{\textbf{return } 1}$
        else
            return 1
        endif
    else
        return 0
```

```
Game $\mathbf{G_7}$

Procedure $\mathtt{H}'(m)$
    $c = \mathtt{Enc}(pk, m)$
    $h' = F'(k', c)$
    return $h'$

Procedure $\mathtt{PCO}(m, c)$
    Parse $c = c_1 || c_2$
    if $\overline{\mathtt{PCO}}(m, c_1) = 1$
        $c_2' = F'(k', c_1)$
        if $c_2' = c_2$
            return 1
    endif
    endif
    return 0

Procedure $\mathtt{CVO}(c)$
    Parse $c = c_1 || c_2$
    $c_2' = F'(k', c_1)$
    if $c_2' = c_2$
        return 1
    else
        return 0
```

Fig. 16. The oracles in $\mathbf{G_4}, \mathbf{G_5}, \mathbf{G_6}, \mathbf{G_7}$. $\overline{\mathtt{PCO}}$ is the plaintext checking oracle for PKE.

Efficiency of $\mathcal{B}$. Algorithm $\mathcal{B}$ runs $\mathcal{A}$, queries $\mathtt{PCO}$ for q_P many times, runs $\mathtt{Enc}$ for $q_{h'}$ many times, and computes F' for $(1 + q_{h'} + q_V + q_P)$ many times. Rest of the steps take $\mathcal{O}(\text{poly}(\lambda))$ time.

$$\begin{aligned}\textbf{LocalTime}(\mathcal{B}) =& \textbf{LocalTime}(\mathcal{A}) + q_{h'}\textbf{LocalTime}(\mathtt{Enc}) \\ & + (1 + q_{h'} + q_V + q_P)\textbf{LocalTime}(F') + \mathcal{O}(\text{poly}(\lambda)) + q_P\end{aligned}$$

The last q_P term in the right hand side denotes the number of queries made to $\mathtt{PCO}$.

```
Algorithm $\mathcal{B}^{\overline{\texttt{PCO}}(.)}(pk, c)$
  $k' \xleftarrow{\$} \{0,1\}^\lambda$
  $c_2 = F'(k', c')$
  $c^* = c||c_2$
  $m \leftarrow \mathcal{A}^{\texttt{PCO}(.),\texttt{CVO}(.),\texttt{H}'}(pk, c^*)$
  return $m$
```

```
Procedure $\texttt{CVO}(c)$
  Parse $c = c_1||c_2$
  $c_2' = F'(k', c_1)$
  if $c_2' = c_2$
    return 1
  else
    return 0
```

```
Procedure $\texttt{H}'(m)$
  $c = \texttt{Enc}(pk, m)$
  $h' = F'(k', c)$
  return $h'$
```

```
Procedure $\texttt{PCO}(m, c)$
  Parse $c = c_1||c_2$
  if $\overline{\texttt{PCO}}(m, c_1) = 1$
    $c_2' = F'(k', c_1)$
    if $c_2' = c_2$
      return 1
    endif
  endif
  return 0
```

Fig. 17. OW-PCA adversary $\mathcal{B}$

Memory Efficiency of $\mathcal{B}$. $\mathcal{B}$ needs to save the code of $\mathcal{A}$, $\texttt{Enc}$, and F'. In addition, there are following λ size registers, $c^*, c_1, c_2, k', m, c, c_2', h'$.

$$\begin{aligned}\textbf{LocalMem}(\mathcal{B}) =& \textbf{LocalMem}(\mathcal{A}) + \textbf{LocalMem}(F') \\ &+ \textbf{LocalMem}(\texttt{Enc}) + 8\lambda\end{aligned}$$

Bounding Prob[Bad]**.** To bound Prob[Bad], we construct a prf adversary $\mathcal{B}_{F'}^{(1)}$ against F'. Recall that Bad occurs when for a $\texttt{CVO}(c)$ query, we get

$$c_2' = c_2 \text{ and } (m' \notin \mathcal{M} \text{ or } c_1' \neq c_1)$$

where $c = c_1||c_2$, $m' = \texttt{Dec}(sk, c_1)$, $c_1' = \texttt{Enc}(pk, m')$, and $c_2' = F'(k', c_1)$.

Case $m' \in \mathcal{M}$ and $c_1' \neq c_1$. In this case correctness error occurs in PKE. Probability of this event is bounded by $\delta(q_V)$.

Case $m' \notin \mathcal{M}$. In this case, for an invalid ciphertext c_1 in PKE, $\mathcal{A}$ can produce a c_2 such that $c_2 = F'(k', c_1)$. As $\mathcal{A}$ has no direct access to $F'(k', .)$ evaluation, and c_1 is an invalid ciphertext, there is no $H'(m)$ or $\texttt{PCO}(m, c)$ query in the transcript for which $F'(k', c_1)$ was evaluated. Notice that, in $\texttt{PCO}(m, c)$ evaluates $F'(k', c_1)$ only when $\texttt{PCO}(m, c_1) = 1$, which can not occur here. So, Bad $= 1$ implies that $\mathcal{A}$ can "guess" the output of $F'(k', c_1)$ for some $c_1 \in \mathcal{C}$. For random function this can happen with probability $\frac{q_V}{2^\eta}$. If Bad happens in significantly more probability in $\mathcal{G}_5$, that can be used to break the prf security of F'.

Formal description of $\mathcal{B}_{F'}^{(1)}$ is given in Figure 18. $\mathcal{B}_{F'}^{(1)}$ perfectly simulates game G_5 with the help of its oracle $\texttt{O}_{F'}$. If $\mathcal{A}$ ever submits a $\texttt{CVO}(c)$ query for which Bad occurs, $\mathcal{B}_{F'}^{(1)}$ outputs 1 and halts. If no such query is made, then at the end of the simulation, $\mathcal{B}_{F'}^{(1)}$ outputs 0. If $\texttt{O}_{F'}$ is a random function, then for a fixed

```
$\mathcal{B}^{(1)}_{F'}$
$(pk, sk) \xleftarrow{\$} \texttt{Gen}$
$m^* \xleftarrow{\$} \mathcal{M}$
$c_2 = \texttt{O}_{F'}(m^*)$
$c_1 = \texttt{Enc}(pk, m^*)$
$c^* = (c_1, c_2)$
$\textsc{Bad} = 0$
$m \leftarrow \mathcal{A}^{\texttt{PCO},\texttt{CVO},\texttt{H}'}(pk, c^*)$
if $\textsc{Bad} = 1$
  Output $1$
else
  Output $0$
```

```
Procedure $\texttt{H}'(m)$
$c = \texttt{Enc}(pk, m)$
$h' = \texttt{O}_{F'}(c)$
return $h'$
```

```
Procedure $\texttt{PCO}(m, c)$
Parse $c = c_1||c_2$
if $\texttt{Enc}(pk, m) = c_1$
  $c_2' = \texttt{O}_{F'}(c_1)$
  if $c_2' = c_2$
    return $1$
  endif
endif
return $0$
```

```
Procedure $\texttt{CVO}(c)$
Parse $c = c_1||c_2$
$m' = \texttt{Dec}(sk, c_1)$
$c_1' = \texttt{Enc}(pk, m')$
$c_2' = \texttt{O}_{F'}(c_1)$
if $c_2' = c_2$
  if $m' \notin \mathcal{M}$ or $c_1' \neq c_1$
    $\textsc{Bad} = 1$
    return $0$
  else
    return $1$
  endif
else
  return $0$
```

Fig. 18. The PRF adversary $\mathcal{B}^{(1)}_{F'}$

$\texttt{CVO}(c)$ query, $\text{Prob}[\mathcal{B}^{(1)}_{F'} = 1]$ is at most $\frac{1}{2^\eta}$. Taking union bound over all the $\texttt{CVO}(c)$ queries made by $\mathcal{A}$, when $\texttt{O}_{F'}$ is a random function, $\text{Prob}[\mathcal{B}^{(1)}_{F'} = 1]$ is at most $\frac{q_V}{2^\eta}$. When $\texttt{O}_{F'}$ is the prf F', $\text{Prob}[\mathcal{B}^{(1)}_{F'} = 1]$ is exactly $\text{Prob}[\textsc{Bad}]$ in G_5.

$$\mathbf{Adv}^{\text{prf}}_{\mathcal{B}^{(1)}_{F'}, F'} \geq \left|\text{Prob}[\textsc{Bad}] - \frac{q_V}{2^\eta} - \delta(q_V)\right|$$
$$\implies \text{Prob}[\textsc{Bad}] \leq \mathbf{Adv}^{\mathsf{prf}}_{\mathcal{B}^{(1)}_{F'}, F'} + \frac{q_V}{2^\eta} + \delta(q_V)$$

Efficiency of $\mathcal{B}^{(1)}_{F'}$. $\mathcal{B}^{(1)}_{F'}$ runs $\mathcal{A}$ once, algorithm $\texttt{Gen}$ once, algorithm $\texttt{Enc}$ for $(1 + q_{h'} + q_P + q_V)$ times, and $\texttt{Dec}$ for q_V times. Additionally $\mathcal{B}^{(1)}_{F'}$ queries the oracle $\texttt{O}_{F'}$ for $(1 + q_{h'} + q_P + q_V)$ times.

$$\begin{aligned}\mathbf{LocalTime}(\mathcal{B}^{(1)}_{F'}) \approx & \mathbf{LocalTime}(\mathcal{A}) + \mathbf{LocalTime}(\texttt{Gen}) + q_V \cdot \mathbf{LocalTime}(\texttt{Dec}) \\ & + (1 + q_{h'} + q_P + q_V)(1 + \mathbf{LocalTime}(\texttt{Enc}))\end{aligned}$$

$\mathcal{B}^{(1)}_{F'}$ needs to save the code of $\mathcal{A}, \texttt{Gen}, \texttt{Enc}$, and $\texttt{Dec}$. In addition, it needs to save eight λ size and a flag of a single bit. registers.

$$\begin{aligned}\mathbf{LocalMem}(\mathcal{B}^{(1)}_{F'}) \approx & \mathbf{LocalMem}(\mathcal{A}) + \mathbf{LocalMem}(\texttt{Gen}) + \mathbf{LocalMem}(\texttt{Enc}) \\ & + \mathbf{LocalMem}(\texttt{Dec}) + 8\lambda + 1\end{aligned}$$

Finishing the proof of Theorem 8. Collecting the probabilities of the games, we have proven so far, there exist adversaries $\mathcal{B}, \mathcal{B}_{\hat{F}}$, and $\mathcal{B}^{(1)}_{F'}$, such that

$$\mathbf{Adv}^{\text{OW-PCVA}}_{\mathcal{A},\mathsf{PKE}_1} \leq \mathbf{Adv}^{\text{OW-PCA}}_{\mathcal{B},\mathsf{PKE}} + \mathbf{Adv}^{\text{prf}}_{\mathcal{B}_{\hat{F}},\hat{F}} + \mathbf{Adv}^{\text{prf}}_{\mathcal{B}^{(1)}_{F'},F'} + \frac{q_V}{2^\eta} + \delta(q_V) + \delta(q_p)$$

Applying Lemma 5, we get a $\mathcal{B}_{F'}^{(2)}$ such that,

$$\begin{aligned}\mathbf{Adv}_{\mathcal{A},\mathsf{PKE}_1}^{\text{OW-PCVA}} \leq & \mathbf{Adv}_{\mathcal{B},\mathsf{PKE}}^{\text{OW-PCA}} + \mathbf{Adv}_{\mathcal{B}_{F'}^{(2)},F'}^{\text{prf}} + \mathbf{Adv}_{\mathcal{B}_{F'}^{(1)},F'}^{\text{prf}} + \frac{q_V}{2^\eta} + \\ & \delta(q_V) + \delta(q_p) + \delta(1 + q_{h'} + q_P + q_V)\end{aligned}$$

Efficiency of $\mathcal{B}_{F'}^{(2)}$ is bounded using following lemma. See Section 7.3 for a proof.

Lemma 6.

$$\begin{aligned}\mathbf{LocalTime}(\mathcal{B}_{F'}^{(2)}) \approx & \mathbf{LocalTime}(\mathcal{A}) + \mathbf{LocalTime}(\texttt{Gen}) + (q_V + q_P)\mathbf{LocalTime}(\texttt{Dec}) \\ & + (2 + 2q_V + 2q_P + q_{h'})\mathbf{LocalTime}(\texttt{Enc}) + (1 + q_{h'} + q_P + q_V) \\ \mathbf{LocalMem}(\mathcal{B}_{F'}^{(2)}) \approx & \mathbf{LocalMem}(\mathcal{A}) + \mathbf{LocalMem}(\texttt{Gen}) + \mathbf{LocalMem}(\texttt{Enc}) \\ & + \mathbf{LocalMem}(\texttt{Dec}) + 11\lambda\end{aligned}$$

Merging $\mathcal{B}_{F'}^{(1)}$ and $\mathcal{B}_{F'}^{(2)}$ into one adversary $\mathcal{B}_{F'}$, and taking upper bound of their efficiencies, we get Theorem 8.

5 Memory-tight Reductions for Fujisaki-Okamoto Transformation and Variants

In this section, we prove memory-tight reduction of the IND-CCA security of four different variants of the Fujisaki-Okamoto transformation, following the modular approach of [16]. Before describing the exact transformations we consider, first we recall the modules introduces in [16].

5.1 Brief Overview of Modules from [16]

We recall the modules in the top-down fashion. First we describe the transformations from a public key encryption scheme to a key encapsulation mechanisms.

Outer Modules: $\mathbf{U}^{\not\perp}$, $\mathbf{U}_{\boldsymbol{m}}^{\not\perp}$, $\mathbf{U}^{\perp}$, $\mathbf{U}_{\boldsymbol{m}}^{\perp}$ Let $\mathsf{PKE}_1 = (\texttt{Gen}_1, \texttt{Enc}_1, \texttt{Dec}_1)$ be a public key encryption scheme with the message space $\mathcal{M}$ and let $\texttt{H} : \mathcal{M} \to \mathcal{K}$ be a hash function. Table 2 describes the variants of module U to construct a KEM using PKE_1 and $\texttt{H}$. The transformations yield KEM of two categories. Transformations $\mathrm{U}^{\not\perp}$ and $\mathrm{U}_m^{\not\perp}$ are in the category of implicit rejection, as the decapsulation algorithms in these transformations do not output $\perp$, when queried with an invalid ciphertext. Transformation $\mathrm{U}^{\perp}$, $\mathrm{U}_m^{\perp}$ are in the category of explicit rejection, implying that the decapsulation algorithms, given any invalid ciphertext, indeed output $\perp$.

Transformations & Security Implications	$\texttt{Encap}(pk)$	$\texttt{Decap}(sk', c)$
$\mathrm{U}^{\not\perp}$(OW-PCA ⇒ IND-CCA)	$(c = \texttt{Enc}_1(pk, m), K = \texttt{H}(m, c))_{m \xleftarrow{\$} \mathcal{M}}$	$\texttt{H}(m, c)$ if $m \neq \perp$ $\texttt{H}(s, c)$ if $m = \perp$
$\mathrm{U}^{\not\perp}_m(\texttt{det} + \text{OW-CPA} \Rightarrow \text{IND-CCA})$	$(c = \texttt{Enc}_1(pk, m), K = \texttt{H}(m))_{m \xleftarrow{\$} \mathcal{M}}$	$\texttt{H}(m)$ if $m \neq \perp$ $\texttt{H}(s, c)$ if $m = \perp$
$\mathrm{U}^{\perp}$(OW-PCVA ⇒ IND-CCA)	$(c = \texttt{Enc}_1(pk, m), K = \texttt{H}(m, c))_{m \xleftarrow{\$} \mathcal{M}}$	$\texttt{H}(m, c)$ if $m \neq \perp$ $\perp$ if $m = \perp$
$\mathrm{U}^{\perp}_m(\texttt{det} + \text{OW-VA} \Rightarrow \text{IND-CCA})$	$(c = \texttt{Enc}_1(pk, m), K = \texttt{H}(m))_{m \xleftarrow{\$} \mathcal{M}}$	$\texttt{H}(m)$ if $m \neq \perp$ $\perp$ if $m = \perp$

Table 2. Variants of transformation U. In the column $\texttt{Decap}$, s is a random string, $sk' = sk||s$, and $m = \texttt{Dec}_1(sk, c)$.

Inner Module: *T* Let $\overline{\mathsf{PKE}} = (\texttt{Gen}, \overline{\texttt{Enc}}, \overline{\texttt{Dec}})$ be an IND-CPA secure public key encryption scheme. Let $\mathcal{M}= \{0, 1\}^n$ be the message space, $\mathcal{C}$ be the ciphertext space, and $\mathcal{R}$ be the randomness space. Let $\texttt{G} : \mathcal{M} \to \mathcal{R}$ be a hash function. The transformation T results in a deterministic public key encryption scheme $\mathsf{PKE} = T[\overline{\mathsf{PKE}}, \texttt{G}]$. Formal description of T is given in Figure 19.

```
Procedure Enc(pk, m)
  $c = \overline{\texttt{Enc}}(pk, m; \texttt{G}(m))$
  return $c$

Procedure Dec(sk, c)
  $m' = \overline{\texttt{Dec}}(sk, c)$
  if $m' = \perp \vee \overline{\texttt{Enc}}(pk, m'; \texttt{G}(m')) \neq c$
    return $\perp$
  else return $m'$
```

Fig. 19. Encryption scheme $\mathsf{PKE} = T[\overline{\mathsf{PKE}}]$

5.2 Considered Variants and the reductions

We consider three other variants of FO transformations. The variants and their modular decomposition are listed in Table 3. For each transformation we start with an IND-CPA secure public key encryption $\overline{PKE}$. We prove memory-tight reduction for each of the modules next.

Category	Transformation	Modular Decomposition
Implicit Rejection	$\mathrm{KEM}^{\not\perp}$	$\mathrm{U}^{\not\perp}\left[T[\overline{\mathsf{PKE}}, \texttt{G}], \texttt{H}\right]$
	$\mathrm{KEM}^{\not\perp}_m$	$\mathrm{U}^{\not\perp}_m\left[T\left[\overline{\mathsf{PKE}}, \texttt{G}\right], \texttt{H}\right]$
Explicit Rejection	$\mathrm{QKEM}^{\perp}_m$	$\mathrm{U}^{\perp}_m\left[V\left[T[\overline{\mathsf{PKE}}, \texttt{G}], \texttt{H}'\right], \texttt{H}\right]$

Table 3. Variants of FO transformations and their modular breakup

Memory-tight Reduction for T: IND-CPA $\Rightarrow$ OW-PCA

Theorem 10. *Let $\mathcal{A}$ be any adversary in the* OW-PCA *game of* PKE. *Suppose $\mathcal{A}$ makes q_g queries to* $\mathtt{G}$. *Let q_p denote the number of plaintext checking queries made by $\mathcal{A}$. Then, in the random oracle model, there exists adversaries $\mathcal{B}$ in the* IND-CPA *game against* $\overline{\mathsf{PKE}}$, *and $\mathcal{B}_F$ in the* prf *game, such that*

$$\mathbf{Adv}^{\text{OW-PCA}}_{\mathcal{A},\mathsf{PKE}} \leq 3 \cdot \mathbf{Adv}^{\text{IND-CPA}}_{\mathcal{B},\overline{\mathsf{PKE}}} + \mathbf{Adv}^{\text{prf}}_{\mathcal{B}_F,F} + \frac{2q_g+1}{|\mathcal{M}|} + \delta(q_p+q_g)$$

$$\begin{aligned}\mathbf{LocalTime}(\mathcal{B}) \approx& \mathbf{LocalTime}(\mathcal{A}) + (q_g+q_p)\mathbf{LocalTime}(F)\\ \mathbf{LocalMem}(\mathcal{B}) \approx& \mathbf{LocalMem}(\mathcal{A}) + \mathbf{LocalMem}(F)\end{aligned}$$

The proof of the above theorem follows exactly from the proof of analogous Theorem 3.2 of [16] and using the random oracle simulation by a prf F. In Section 9, we include a sketch of the proof. Finally, from [16], we get that, if PKE is strongly $\overline{\delta}$ correct, then PKE is $\delta(q_g+q_p)$ correct where $\delta(x) = x\overline{\delta}$.

Transformation	Key Derivation	RO simulation in Hash Query	RO Simulation in $\mathtt{Decap}$ query

$\mathrm{U}^{\not\perp}$	$K = \mathtt{H}(m,c)$	if $\mathtt{PCO}(m,c) = 1$
		$K = F(k,0,0^\mu,c)$
		else
		$K = F(k,1,m,c)$
			$K = F(k,0,0^\mu,c)$

$\mathrm{U}^{\perp}$	$K = \mathtt{H}(m,c)$	if $\mathtt{PCO}(m,c) = 1$
		$K = F(k,0,0^\mu,c)$
		else
		$K = F(k,1,m,c)$
			if $\mathtt{CVO}(c) = 0$
			$K =\perp$
			else
			$K = F(k,0,0^\mu,c)$

$\mathrm{U}^{\not\perp}_m$	$K = \mathtt{H}(m)$	$K = F(k,\mathtt{Enc}_1(pk,m))$	$K = F(k,c)$

$\mathrm{U}^{\perp}_m$	$K = \mathtt{H}(m)$	$K = F(k,\mathtt{Enc}_1(pk,m))$	if $\mathtt{CVO}(c) = 0$
			$K =\perp$
			else
			$K = F(k,c)$

Table 4. Random Oracle Simulation for $\mathrm{U}^{\not\perp}$, $\mathrm{U}^{\not\perp}_m$, $\mathrm{U}^{\perp}$, $\mathrm{U}^{\perp}_m$. We assume $\mathcal{M} = \{0,1\}^\mu$ is the message space of the underlying encryption scheme

Memory-tight Reduction for V: OW-PCA $\Rightarrow$ OW-PCVA. It follows from Theorem 8.

Memory-tight Reduction for variants of U Table 2 lists four variants of U with different security implications. The memory-efficient reductions of these implications are in principle same as the proofs presented in [16]. The only difference is in the simulation of the Random Oracle $\mathtt{H}$. In Table 4, we write the precise functions to be used to simulate the random oracles in the reductions. We assume the message space of the underlying encryption scheme to be $\{0,1\}^{\mu}$. $\mathtt{PCO}(m,c)$ returns 1 if c decrypts to m. $\mathtt{CVO}(c)$ returns 0 if c decrypts to $\perp$.
In Section 10, we present a detailed reduction for the transformation $\mathrm{U}_m^{\perp}$.

6 Conclusion

Memory efficiency of blackbox reduction is of paramount importance as many of the standard assumptions are memory sensitive. Unfortunately, most of the existing results on memory-tightness of reductions are negative. In this paper, we presented a memory-tight reduction of IND-CCA security of Hashed ElGamal from gap Diffie Hellman assumption in the random oracle model. We also show memory-tight reductions for different modules of Fujisaki-Okamoto transformation and its variants. As a future work, we believe, it will be interesting to find lower bounds of memory efficiency in the random oracle model.

Acknowledgements. We thank Eike Kiltz for encouraging us to write up and submit the work. We are thankful to the reviewers for their comments on this and the previous versions of the paper. The author is supported by *SERB ECR/2017/001974*.

References

1. Michel Abdalla, Mihir Bellare, and Phillip Rogaway. DHIES: An encryption scheme based on the Diffie-Hellman problem. Contributions to IEEE P1363a, September 1998.
2. Michel Abdalla, Mihir Bellare, and Phillip Rogaway. The oracle Diffie-Hellman assumptions and an analysis of DHIES. In *CT-RSA 2001*, volume 2020 of *LNCS*, pages 143–158. Springer, Heidelberg, April 2001.
3. Benedikt Auerbach, David Cash, Manuel Fersch, and Eike Kiltz. Memory-tight reductions. In *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 101–132. Springer, Heidelberg, August 2017.
4. Mihir Bellare, Alexandra Boldyreva, and Silvio Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 259–274. Springer, Heidelberg, May 2000.
5. Mihir Bellare and Phillip Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. In *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 399–416. Springer, Heidelberg, May 1996.
6. Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426. Springer, Heidelberg, May / June 2006.

7. Daniel Bernstein. Extending the salsa20 nonce. Workshop Record of Symmetric Key Encryption Workshop 2011, 2011.

8. Daniel J. Bernstein and Edoardo Persichetti. Towards KEM unification. Cryptology ePrint Archive, Report 2018/526, 2018. `https://eprint.iacr.org/2018/526`.

9. Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. In *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 41–69. Springer, Heidelberg, December 2011.

10. Sanjit Chatterjee, Alfred Menezes, and Palash Sarkar. Another look at tightness. In *SAC 2011*, volume 7118 of *LNCS*, pages 293–319. Springer, Heidelberg, August 2012.

11. Jean-Sébastien Coron. On the exact security of full domain hash. In *CRYPTO 2000*, volume 1880 of *LNCS*, pages 229–235. Springer, Heidelberg, August 2000.

12. Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003.

13. Gregory Demay, Peter Gaži, Martin Hirt, and Ueli Maurer. Resource-restricted indifferentiability. In *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 664–683. Springer, Heidelberg, May 2013.

14. Alexander W. Dent. A designer's guide to KEMs. In *9th IMA International Conference on Cryptography and Coding*, volume 2898 of *LNCS*, pages 133–151. Springer, Heidelberg, December 2003.

15. Ashrujit Ghoshal and Stefano Tessaro. On the memory-tightness of hashed elgamal. In *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part II*, pages 33–62, 2020.

16. Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 341–371. Springer, Heidelberg, November 2017.

17. Haodong Jiang, Zhenfeng Zhang, Long Chen, Hong Wang, and Zhi Ma. IND-CCA-secure key encapsulation mechanism in the quantum random oracle model, revisited. In *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 96–125. Springer, Heidelberg, August 2018.

18. Omer Reingold, Luca Trevisan, and Salil P. Vadhan. Notions of reducibility between cryptographic primitives. In *TCC 2004*, volume 2951 of *LNCS*, pages 1–20. Springer, Heidelberg, February 2004.

19. Tsunekazu Saito, Keita Xagawa, and Takashi Yamakawa. Tightly-secure key-encapsulation mechanism in the quantum random oracle model. In *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 520–551. Springer, Heidelberg, April / May 2018.

20. Ehsan Ebrahimi Targhi and Dominique Unruh. Post-quantum security of the Fujisaki-Okamoto and OAEP transforms. In *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 192–216. Springer, Heidelberg, October / November 2016.

21. Yuyu Wang, Takahiro Matsuda, Goichiro Hanaoka, and Keisuke Tanaka. Memory lower bounds of reductions revisited. In *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 61–90. Springer, Heidelberg, April / May 2018.

Appendix

7 Leftout Proofs

7.1 Proof of Lemma 1

Proof. We construct $\mathcal{B}_F$ against F in the prf game. Let $\mathtt{O}_F$ be the oracle provided to $\mathcal{B}_F$. $\mathcal{B}_F$ simulates game G, invokes $\mathcal{A}$, and answers the query. Whenever there is a $\mathtt{H}$ query in G or a $\mathtt{H}$ query made by $\mathcal{A}$, $\mathcal{B}$ queries its oracle $\mathcal{O}_F$ and returns the output. Finally, $\mathcal{B}_F$ outputs 1 if $G^{\mathcal{A}}$ outputs 1.
If $\mathtt{O}_F$ is a random function, $\mathcal{B}_F$ simulates $G[\mathtt{H}]$ perfectly. On the other hand, if $\mathtt{O}_F$ is implemented by F, $\mathcal{B}_F$ simulates $G[F]$. Hence,

$$\left|\mathbf{Succ}_{\mathcal{A}^{\mathtt{H}},G[\mathtt{H}]} - \mathbf{Succ}_{\mathcal{A}^{\mathtt{H}},G[F]}\right| \leq \mathbf{Adv}^{\mathrm{prf}}_{\mathcal{B}_F,F}$$

To calculate the efficiency, we note that $\mathcal{B}_F$ runs $\mathcal{A}$ once, and simulates game G (but not the hash oracle), queries $\mathtt{O}_F$ for q_h times.

$$\mathbf{LocalTime}(\mathcal{B}_F) = \mathbf{LocalTime}(\mathcal{A}) + \mathbf{LocalTime}(G) + q_h$$

The last q_h term in the right hand side denotes the queries $\mathcal{B}_F$ makes to the $\mathcal{O}_F$ oracle.
$\mathcal{B}_F$ needs to save the code of $\mathcal{A}$ and the code and the variables of the game G.

$$\mathbf{LocalMem}(\mathcal{B}_F) = \mathbf{LocalMem}(\mathcal{A}) + \mathbf{LocalMem}(G)$$

7.2 Proof of Lemma 3

Game G_2 runs the key generation algorithm $\mathtt{Gen}$ once which has the same time complexity as $\mathtt{DH}$. In addition, G_2 queries the $\mathtt{O}_F$ oracle for $q_H + q_D$ many times during $\mathtt{H}$ and $\mathtt{Decap}$ queries. Assuming , rest of G_2 takes $\mathcal{O}(\mathrm{poly}(\lambda))$ time in total

$$\begin{aligned}\mathbf{LocalTime}(G_2) \approx& \mathbf{LocalTime}(\mathtt{DH}) + (q_H + q_D)\\ \mathbf{LocalMem}(G_2) \approx& \mathbf{LocalMem}(\mathtt{DH}) + 9\lambda + 2\end{aligned}$$

Using Lemma 1,

$$\begin{aligned}\mathbf{LocalTime}(\mathcal{B}_{\hat{F}}) \approx& \mathbf{LocalTime}(\mathcal{A}) + \mathbf{LocalTime}(\mathtt{DH}) + (q_H + q_D)\\ \mathbf{LocalMem}(\mathcal{B}_{\hat{F}}) =& \mathbf{LocalMem}(\mathcal{A}) + \mathbf{LocalMem}(\mathtt{DH}) + 9\lambda + 2\end{aligned}$$

Finally, applying Lemma 2, we get the efficiency of $\mathcal{B}_F$ as

$$\begin{aligned}\mathbf{LocalTime}(\mathcal{B}_F) \approx& \mathbf{LocalTime}(\mathcal{A}) + \mathbf{LocalTime}(\mathtt{DH}) + 2(q_H + q_D)\\ \mathbf{LocalMem}(\mathcal{B}_F) \approx& \mathbf{LocalMem}(\mathcal{A}) + \mathbf{LocalMem}(\mathtt{DH}) + 11\lambda + 2\end{aligned}$$

7.3 Proof of Lemma 6

In order to bound the efficiency of $\mathcal{B}_{F'}^{(2)}$, we observe that the time and memory complexity of $\mathcal{B}_{\hat{F}}$ is same as of the game $\mathbf{G}_1$. By observation

$$\begin{aligned}\mathbf{LocalTime}(\mathbf{G_1}) \approx & \mathbf{LocalTime}(\texttt{Gen}) + (1 + q_P + q_V) \cdot \mathbf{LocalTime}(\texttt{Enc}) \\ & + (q_V + q_P)\mathbf{LocalTime}(\texttt{Dec}) + (1 + q_{h'} + q_P + q_V) \\ \mathbf{LocalMem}(\mathbf{G_1}) \approx & \mathbf{LocalMem}(\texttt{Gen}) + +\mathbf{LocalMem}(\texttt{Enc}) \\ & + \mathbf{LocalMem}(\texttt{Dec}) + 8\lambda\end{aligned}$$

Then applying Lemma 1 and Lemma 5, we can bound the efficiency of $\mathcal{B}_{F'}^{(2)}$ as

$$\begin{aligned}\mathbf{LocalTime}(\mathcal{B}_{F'}^{(2)}) \approx & \mathbf{LocalTime}(\mathcal{A}) + \mathbf{LocalTime}(\texttt{Gen}) + (q_V + q_P)\mathbf{LocalTime}(\texttt{Dec}) \\ & + (2 + 2q_V + 2q_P + q_{h'})\mathbf{LocalTime}(\texttt{Enc}) + (1 + q_{h'} + q_P + q_V) \\ \mathbf{LocalMem}(\mathcal{B}_{F'}^{(2)}) \approx & \mathbf{LocalMem}(\mathcal{A}) + \mathbf{LocalMem}(\texttt{Gen}) + \mathbf{LocalMem}(\texttt{Enc}) \\ & + \mathbf{LocalMem}(\texttt{Dec}) + 11\lambda\end{aligned}$$

8 Proof of Theorem 6

Theorem 6 is proven via a sequence of games. Formal description of the games are given in Figure 20, and Figure 21.

```
$\mathbf{G_0}$ $\boxed{\mathbf{G_1}}$
1 :  $(pk, sk) \leftarrow \texttt{Gen}(1^\lambda)$
2 :  Parse $pk = (g, X)$
3 :  Parse $sk = x$
4 :  $y^* \xleftarrow{\$} \mathbb{Z}_q^*$
5 :  $b \xleftarrow{\$} \{0, 1\}$
6 :  $Y^* = g^{y^*}$
7 :  $Z^* = {Y^*}^x$
8 :  $K_0^* = \texttt{H}(Z^*)$ $\boxed{K_0^* \xleftarrow{\$} \mathcal{K}}$
9 :  $K_1^* \xleftarrow{\$} \mathcal{K}$
10 : $b^* \leftarrow \mathcal{A}^{\mathsf{Decap},\mathtt{H}}(pk, Y^*, K_b^*)$
11 : if $b = b^*$ return 1
12 : else return 0
13 : endif
```

```
Procedure $\texttt{H}(Z)$ in $\mathbf{G_0}$
1 :  if $\texttt{H}(Z)$ is undefined
2 :    $\texttt{H}(Z) \xleftarrow{\$} \mathcal{K}$
3 :  endif
4 :  return $\texttt{H}(Z)$
```

```
Procedure $\texttt{Decap}(Y)$ in $\mathbf{G_0}, \mathbf{G_1}$
1 :  if $Y = Y^*$ return $\perp$
2 :  $Z = Y^x$
3 :  $K = \texttt{H}(Z)$
4 :  return $K$
```

```
Procedure $\texttt{H}(Z)$ in $\mathbf{G_1}$
1 :  if $Z = Z^*$
2 :    return $K_0^*$
3 :  else
4 :    if $\texttt{H}(Z)$ is undefined
5 :      $\texttt{H}(Z) \xleftarrow{\$} \mathcal{K}$
6 :    endif
7 :  return $\texttt{H}(Z)$
8 :  endif
```

Fig. 20. The games G_0 and G_1. In game G_1, the boxed statement in Step 8 is executed.

Game $\mathbf{G}_0$. The game G_0 is the original IND-CCA game. Y^* is the challenge ciphertext for the KEM game. $Z^* = Y^{*x}$ where x is the secret key.

$$\mathbf{Adv}_{\mathcal{A},\hat{\Pi}}^{\text{IND-CCA}} \overset{def}{=} \left|\text{Prob}[\mathbf{G}_0^{\mathcal{A}} = 1] - \frac{1}{2}\right|.$$

Game $\mathbf{G}_1$: We predefine $K_0^* = \mathtt{H}(Z^*)$ by sampling a random element from the keyspace $\mathcal{K}$. The hash oracle is modified to return K_0^* for the input (Z^*). As K_0^* is still uniformly chosen at random, and the hash oracle output is consistent, there is no change in the distribution of adversary's view.

$$\text{Prob}[\mathbf{G}_0^{\mathcal{A}} = 1] = \text{Prob}[\mathbf{G}_1^{\mathcal{A}} = 1]$$

Game $\mathbf{G}_2$. In this game the oracles `H` and `Decap` are changed. We simulate the random oracle by a prf $\hat{F} : \{0,1\}^\lambda \times \mathbb{G} \to \mathcal{K}$. In the `Decap` procedure we do not make the hash query anymore. Instead, the hash value is determined by a direct evaluation of $\hat{F}$. By Lemma 1, there exists an adversary $\mathcal{B}_{\hat{F}}$ such that

$$\left|\text{Prob}[\mathbf{G}_1^{\mathcal{A}} = 1] - \text{Prob}[\mathbf{G}_2^{\mathcal{A}} = 1]\right| \leq \mathbf{Adv}_{\mathcal{B}_{\hat{F}},\hat{F}}^{\text{prf}}$$

Game $\mathbf{G}_3$. We rewrite the prf evaluation of $\hat{F}$ using the prf F as defined in Construction 7. While simulating the hash oracle, Step 1 of procedure `H`, the check whether the input Z is equal to Z^*, is performed using the bilinear map. Specifically, we check, whether $\hat{e}(g, Z) = \hat{e}(X, Y^*)$. By the property of bilinear map, the equality holds only if (X, Y^*, Z) is a Diffie Hellman tuple,i.e. $Z = Z^*$. Hence the checking is performed correctly and there is no change in adversary's view.

$$\text{Prob}[\mathbf{G}_2^{\mathcal{A}} = 1] = \text{Prob}[\mathbf{G}_3^{\mathcal{A}} = 1]$$

Game $\mathbf{G}_4$: In this game, we make three changes. As Z^* is no longer used in any of the procedure, we drop Step 8. The second change is in the procedure `Decap`. We do not compute $Z = Y^x$ (Step 2) anymore. We compute the value of $K = F(k, \hat{e}(g, Y^x))$ by computing $F(k, \hat{e}(X, Y))$. By the bilinear property of $\hat{e}$, the computation is correct. Hence, these two changes do not incurr any change in the distribution of the adversary's view.
The final change is that we set a flag FLAG and abort on the event that $\mathcal{A}$ queries $\mathtt{H}(Z^*)$. By the fundamental lemma of game playing proofs

$$\left|\text{Prob}[\mathbf{G}_3^{\mathcal{A}} = 1] - \text{Prob}[\mathbf{G}_4^{\mathcal{A}} = 1]\right| \leq \text{Prob}[\text{FLAG} = 1].$$

In the game $\mathbf{G}_4$, the adversary $\mathcal{A}$ is unable to compute $\mathtt{H}(Z^*)$ using either the hash oracle or the decapsulation oracle. The decapsulation oracle outputs $\perp$ whenever the input Y is equal to Y^*. The hash oracle aborts for the input Z^*. This implies that the bit b is independent from the adversaries view. Hence

$$\text{Prob}[\mathbf{G}_4^{\mathcal{A}}] = \frac{1}{2}.$$

```
$\mathbf{G_2}$
$(pk, sk) \leftarrow \texttt{Gen}(1^\lambda)$
Parse $pk = (g, X)$
Parse $sk = x$
$k \xleftarrow{\$} \{0,1\}^\lambda$
$y^* \xleftarrow{\$} \mathbb{Z}_q^*$
$b \xleftarrow{\$} \{0,1\}$
$Y^* = g^{y^*}$
$Z^* = Y^{*x}$
$K_0^* \xleftarrow{\$} \mathcal{K}$
  $K_1^* \xleftarrow{\$} \mathcal{K}$
  $b^* \leftarrow \mathcal{A}^{\texttt{Decap},\texttt{H}}(pk, Y^*, K_b^*)$
  if $b = b^*$ return $1$
  else return $0$
  endif

Procedure $\texttt{H}(Z)$
if $Z = Z^*$
  return $K_0^*$
else
$K = \hat{F}(k, Z)$
return $K$
endif

Procedure $\texttt{Decap}(Y)$
if $Y = Y^*$ return $\perp$
$Z = Y^x$
$K = \hat{F}(k, Z)$
return $K$
```

```
$\mathbf{G_3}$
$(pk, sk) \leftarrow \texttt{Gen}(1^\lambda)$
Parse $pk = (g, X)$
Parse $sk = x$
$k \xleftarrow{\$} \{0,1\}^\lambda$
$y^* \xleftarrow{\$} \mathbb{Z}_q^*$
$b \xleftarrow{\$} \{0,1\}$
$Y^* = g^{y^*}$
$Z^* = Y^{*x}$
$K_0^* \xleftarrow{\$} \mathcal{K}$
  $K_1^* \xleftarrow{\$} \mathcal{K}$
  $b^* \leftarrow \mathcal{A}^{\texttt{Decap},\texttt{H}}(pk, Y^*, K_b^*)$
  if $b = b^*$ return $1$
  else return $0$
  endif

Procedure $\texttt{H}(Z)$
if $\hat{e}(g, Z) = \hat{e}(X, Y^*)$
  return $K_0^*$
else
  $K = F(k, \hat{e}(g, Z))$
return $K$
endif

Procedure $\texttt{Decap}(Y)$
if $Y = Y^*$ return $\perp$
$Z = Y^x$
$K = F(k, \hat{e}(g, Z))$
return $K$
```

```
$\mathbf{G_4}$
$(pk, sk) \leftarrow \texttt{Gen}(1^\lambda)$
Parse $pk = (g, X)$
Parse $sk = x$
$k \xleftarrow{\$} \{0,1\}^\lambda$
$y^* \xleftarrow{\$} \mathbb{Z}_q^*$
$b \xleftarrow{\$} \{0,1\}$
$Y^* = g^{y^*}$

$K_0^* \xleftarrow{\$} \mathcal{K}$
  $K_1^* \xleftarrow{\$} \mathcal{K}$
  $b^* \leftarrow \mathcal{A}^{\texttt{Decap},\texttt{H}}(pk, Y^*, K_b^*)$
  if $b = b^*$ return $1$
  else return $0$
  endif

Procedure $\texttt{H}(Z)$
if $\hat{e}(g, Z) = \hat{e}(X, Y^*)$
  FLAG=1
  Abort
else
  $K = F(k, \hat{e}(g, Z))$
return $K$
endif

Procedure $\texttt{Decap}(Y)$
if $Y = Y^*$ return $\perp$

$K = F(k, \hat{e}(X, Y))$
return $K$
```

Fig. 21. IND-CCA game of ECIES: highlighted statements are the modifications from the previous game

To bound Prob[FLAG = 1], we use algorithm $\mathcal{B}_{DH}$ (Figure 12) against the hardness of CDH problem of $\mathbb{G}$. $\mathcal{B}_{DH}$ simulates game $\mathbf{G_4}$ for $\mathcal{A}$.
So far, we have proven that there exists adversaries $\mathcal{B}_{DH}$ and $\mathcal{B}_{\hat{F}}$ such that

$$\mathbf{Adv}_{\mathcal{A},\hat{\Pi}}^{\text{IND-CCA}} \leq \mathbf{Adv}_{\mathcal{B}_{DH},G}^{\mathsf{CDH}} + \mathbf{Adv}_{\mathcal{B}_{\hat{F}},\hat{F}}^{\text{prf}}$$

By applying Lemma 4, we get the adversary $\mathcal{B}_F$ such that

$$\mathbf{Adv}_{\mathcal{B}_{\hat{F}},\hat{F}}^{\text{prf}} = \mathbf{Adv}_{\mathcal{B}_F,F}^{\text{prf}}$$

Hence, there exist adversaries $\mathcal{B}_{DH}$ and $\mathcal{B}_F$ such that

$$\mathbf{Adv}_{\mathcal{A},\hat{\Pi}}^{\text{IND-CCA}} \leq \mathbf{Adv}_{\mathcal{B}_{DH},G}^{\mathsf{CDH}} + \mathbf{Adv}_{\mathcal{B}_F,F}^{\text{prf}}$$

The following lemma finishes the proof of Theorem 6.

Lemma 7.

$$\begin{aligned}\mathbf{LocalTime}(\mathcal{B}_F) \approx& \mathbf{LocalTime}(\mathcal{A}) + \mathbf{LocalTime}(\mathtt{DH}) + (q_H + q_D)\\ & (q_H + q_D)\mathbf{LocalTime}(\hat{e})\\ \mathbf{LocalMem}(\mathcal{B}_F) \approx& \mathbf{LocalMem}(\mathcal{A}) + \mathbf{LocalMem}(\mathtt{DH}) + 12\lambda + 2\end{aligned}$$

Proof. To find the efficiency of $\mathcal{B}_F$, we first observe that game G_2 runs the key generation algorithm `Gen` once (same complexity as `DH`). In addition, G_2 queries the $\mathtt{O}_F$ oracle for q_D many times during `Decap` queries. Assuming rest of G_2 takes $\mathcal{O}(\text{poly}(\lambda))$ time in total,

$$\mathbf{LocalTime}(G_2) = \mathbf{LocalTime}(\mathtt{DH}) + q_D + \mathcal{O}(\text{poly}(\lambda))$$

Now, we use Lemma 1, and Lemma 4.

$$\begin{aligned}\mathbf{LocalTime}(\mathcal{B}_F) =& \mathbf{LocalTime}(\mathcal{B}_{\hat{F}}) + q\mathbf{LocalTime}(\hat{e}) \qquad \text{[Lemma 4]}\\ =& \mathbf{LocalTime}(\mathcal{A}) + \mathbf{LocalTime}(G_2)\\ & + q_H\mathbf{LocalTime}(\hat{e}) + q_H \qquad \text{[Lemma 1]}\\ =& \mathbf{LocalTime}(\mathcal{A}) + \mathbf{LocalTime}(\mathtt{DH}) + (q_H + q_D)\\ & (q_H + q_D)\mathbf{LocalTime}(\hat{e}) + \mathcal{O}(\text{poly}(\lambda))\end{aligned}$$

Similarly,

$$\mathbf{LocalMem}(G_2) = \mathbf{LocalMem}(\mathtt{DH}) + 10\lambda + 2$$

Thus, using Lemma 1,

$$\mathbf{LocalMem}(\mathcal{B}_{\hat{F}}) = \mathbf{LocalMem}(\mathcal{A}) + \mathbf{LocalMem}(\mathtt{DH}) + 10\lambda + 2$$

Now, applying Lemma 4

$$\mathbf{LocalMem}(\mathcal{B}_F) = \mathbf{LocalMem}(\mathcal{A}) + \mathbf{LocalMem}(\mathtt{DH}) + 12\lambda + 2$$

□

9 Proof of Theorem 10

We prove Theorem 10 via a sequence of games. Formal description of the games are given in the Figure 22 and Figure 23.
Game $\mathbf{G_0}$. G_0 is the OW-PCA security game of PKE.

$$\mathbf{Adv}^{\text{OW-PCA}}_{\mathcal{A},\mathsf{PKE}} = \text{Prob}[G_0^{\mathcal{A}} = 1]$$

```
$\mathbf{G_0, G_1, G_2}$
$(pk, sk) \xleftarrow{\$}$ Gen
$\boxed{k \xleftarrow{\$} \{0,1\}^\lambda}$
$m^* \xleftarrow{\$} \mathcal{M}$
$r^* = \mathtt{G}(m^*)$
$c^* = \mathtt{Enc}(pk, m^*, r^*)$
$m \leftarrow \mathcal{A}^{\mathtt{PCO},\mathtt{G}}(pk, c^*)$
if $m^* = m$ return 1
else return 0
```

Fig. 22. The main function of the games, G_0 to G_2. The boxed statement is not executed in the game G_0

```
Game $G_0$

Procedure $\mathtt{G}(m)$
if $\mathtt{G}(m)$ is undefined
    $\mathtt{G}(m) \xleftarrow{\$} \mathcal{R}$
endif
return $\mathtt{G}(m)$

Procedure $\mathtt{PCO}(m, c)$
$m' = \mathtt{Dec}(sk, c)$
$r' = \mathtt{G}(m')$
$c' = \mathtt{Enc}(pk, m', r')$
if $m' = m$ and $c' = c$
    return 1
else
    return 0
```

```
Game $G_1$

Procedure $\mathtt{G}(m)$
$h = F(k, m)$
return $h$

Procedure $\mathtt{PCO}(m, c)$
$r = F(k, m)$
$c' = \mathtt{Enc}(pk, m, r)$
if $c' = c$
    return 1
else
    return 0
```

```
Game $G_2$

Procedure $\mathtt{G}(m)$
if $m = m^*$
    $Flag = 1$
    return $\perp$
endif
$h = F(k, m)$
return $h$

Procedure $\mathtt{PCO}(m, c)$
$r = F(k, m)$
$c' = \mathtt{Enc}(pk, m, r)$
if $c' = c$
    return 1
else
    return 0
```

Fig. 23. Procedures G and PCO in Games G_1, G_2. The main function of the games are same as in Game G_0

Game $\mathbf{G_1}$. In this game we simulate Random Oracle G by a prf $F : \{0,1\}^\lambda \times \mathcal{M} \to \mathcal{R}$. Moreover, we modify the $\mathtt{PCO}(m, c)$ oracle simulation. Instead of the decryption, $m' = \mathtt{Dec}(sk, c)$, and equality check $m = m'$, we only check whether c is the correctly computed ciphertext of m, $c = \mathtt{Enc}(pk, m, r)$ where $r = F(k, m)$.

When G is a random oracle, the probability that $\mathcal{A}$ queries oracles G or PCO with a message m such that decryption error occurs, is at most . As replacing the random oracle by a prf is computationally indistinguishable, the probability of the same event in Game G_1 is at most negligibly more than $(q_g + q_p)\delta$. Conditioned on the correctness error does not occur, $c_1' = c_1 = \mathtt{Enc}(pk, m, r)$ implies that $m' = \mathtt{Dec}(sk, c_1') = m$. In that case, the change in the description of PCO does not affect the transcript distribution. Formally, we can claim the following lemma.

Lemma 8. *There exists adversary $\mathcal{B}_F$ such that*

$$\left|Prob[G_1^{\mathcal{A}} = 1] - Prob[G_0^{\mathcal{A}} = 1]\right| \leq \mathbf{Adv}^{\mathrm{prf}}_{\mathcal{B}_F, F} + \delta(q_p + q_g)$$

Game $\mathbf{G_2}$. In this game we raise a flag $Flag$ (and abort) if $\mathcal{A}$ makes a $\mathtt{G}(m^*)$ query. Rest of the game remain unchanged. By the fundamental lemma of game based proof,

$$\left|\mathrm{Prob}[G_1^{\mathcal{A}} = 1] - \mathrm{Prob}[G_2^{\mathcal{A}} = 1]\right| \leq \mathrm{Prob}[Flag = 1]$$

To bound $\mathrm{Prob}[G_2^{\mathcal{A}} = 1]$, we first construct an adversary $\overline{\mathcal{B}}$ against the IND-CPAsecurity of PKE. C on input pk, chooses a uniformly random message $m_0 \xleftarrow{\$} \mathcal{M}$ and a fixed message m_1. $\overline{\mathcal{B}}$ submits m_0, m_1 to the IND-CPA challenger. On receiving the challenge ciphertext c^*, $\mathcal{B}$ invokes $\mathcal{A}(pk, c^*)$, simulates G_3 perfectly. When $\mathcal{A}$ returns m', $\overline{\mathcal{B}}$ returns 0 if $m' = m_0$ and returns 1 otherwise. Simple analysis shows that

$$\mathrm{Prob}[G_3^{\mathcal{A}} = 1] \leq \mathbf{Adv}^{\text{IND-CPA}}_{\overline{\mathcal{B}}, \mathsf{PKE}} + \frac{1}{|\mathcal{M}|}$$

We observe that,

$$\mathbf{LocalTime}(\overline{\mathcal{B}}) = \mathbf{LocalTime}(\mathcal{A})$$
$$\mathbf{LocalMem}(\overline{\mathcal{B}}) = \mathbf{LocalMem}(\mathcal{A}) + 91^{\lambda} + 1$$

To bound $\mathrm{Prob}[Flag = 1]$, we use the adversary described in the proof of Theorem 3.2 of [16].

Bounding $\mathbf{Prob}[\boldsymbol{Flag} = \mathbf{1}]$ We construct an adversary $\mathcal{B}'$ against IND-CPAsecurity of PKE that wins if Flag is raised in the game G_3. $\mathcal{B}'$ on input pk, chooses two uniformly random message m_0, m_1 and submits them to the challenger of IND-CPAgame. Upon receiving the challenge ciphertext c^*, $\mathcal{B}'$ invokes $\mathcal{A}(pk, c^*)$ and simulates G_3. By construction, m_b^* is uniformly distributed.

Following the analysis of [16], we can prove

$$\mathrm{Prob}[Flag = 1] \leq 2 \cdot \mathbf{Adv}^{\text{IND-CPA}}_{\mathcal{B}', \mathsf{PKE}} + \frac{2q_G}{|\mathcal{M}|}$$

As $\mathcal{B}'$ chooses two random messages and simulates G_3, we get

$$\mathbf{LocalTime}(\mathcal{B}') = \mathbf{LocalTime}(\mathcal{A})$$
$$\mathbf{LocalMem}(\mathcal{B}') = \mathbf{LocalMem}(\mathcal{A}) + 11\lambda + 1$$

Merging $\overline{\mathcal{B}}$ and $\mathcal{B}'$, and taking upper bound of the efficiencies, we get the required adversary $\mathcal{B}$.

10 Transformation $U_m^{\perp}$[16]

Let $\mathsf{PKE}_1 = (\mathtt{Gen}_1,\ \mathtt{Enc}_1, \mathtt{Dec}_1)$ be a deterministic OW-PCVA secure public key encryption scheme. Let $\mathcal{M}$ be the message space, and $\mathcal{C}_1$ be the ciphertext space. Let $\mathtt{H} : \mathcal{M} \to \mathcal{K}$ be a hash function. The transformed scheme $U_m^{\perp}[\mathsf{PKE}_1] = (\mathtt{Gen}, \mathtt{Encap}, \mathtt{Decap})$ is described in Figure 24. The key generation algorithm $\mathtt{Gen}$ is same as $\mathtt{Gen}_1$ Our main theorem is the following.

```
Procedure Encap(pk)
    $m \xleftarrow{\$} \mathcal{M}$
    $c = \mathtt{Enc}_1(pk, m)$
    $K = \mathtt{H}(m)$
    return $(c, K)$

Procedure Decap(sk, c)
    $m' = \mathtt{Dec}_1(sk, c)$
    if $m' = \perp$
        return $\perp$
    else
        $K = \mathtt{H}(m')$
        return $K$
    endif
```

Fig. 24. Transformation $U_m^{\perp}$: IND-CCAsecure KEM

Theorem 11. *Let* $\mathsf{PKE}_1 = (\mathtt{Gen}_1,\ \mathtt{Enc}_1, \mathtt{Dec}_1)$ *be a* δ_1*-correct deterministic public key encryption scheme. Let* $\mathcal{M}$ *be the message space, and* $\mathcal{C}_1$ *be the ciphertext space of* PKE_1*. Let* Π *be the Key Encapsulation Scheme* $U_m^{\perp}[\mathsf{PKE}_1]$*. Let* $F : \{0,1\}^{\lambda} \times \mathcal{C}_1 \to \mathcal{K}$ *be a prf.*

Let $\mathcal{A}$ *be any adversary in the* IND-CCA*game of* Π*. Suppose* $\mathcal{A}$ *makes* q_h *hash queries and* q_D *decapsulation queries. Then, in the random oracle model, there exists an adversary* $\mathcal{B}_{PKE}$*, and an adversary* $\mathcal{B}_F$ *such that*

$$\mathbf{Adv}^{\text{IND-CCA}}_{\mathcal{A},\Pi} \leq \mathbf{Adv}^{\text{OW-VA}}_{\mathcal{B}_{PKE},\mathsf{PKE}_1} + \mathbf{Adv}^{\text{prf}}_{\mathcal{B}_F,F} + 2\delta_1(q_H + q_D)$$

Moreover it holds that

$$\begin{aligned}
\mathbf{LocalTime}(\mathcal{B}_{PKE}) \approx & \mathbf{LocalTime}(\mathcal{A}) + (q_h + q_D)\mathbf{LocalTime}(F) \\
& + q_h\mathbf{LocalTime}(\mathtt{Enc}_1) + q_D \\
\mathbf{LocalMem}(\mathcal{B}_{PKE}) \approx & \mathbf{LocalMem}(\mathcal{A}) + \mathbf{LocalMem}(F) \\
& + \mathbf{LocalMem}(\mathtt{Enc}_1) + 5\lambda + 1
\end{aligned}$$

$$\begin{aligned}
\mathbf{LocalTime}(\mathcal{B}_F) \approx & \mathbf{LocalTime}(\mathcal{A}) + \mathbf{LocalTime}(\mathtt{Gen}) + q_D\mathbf{LocalTime}(\mathtt{Dec}_1) \\
& + (q_h + q_D)(1 + \mathbf{LocalTime}(\mathtt{Enc}_1)) \\
\mathbf{LocalMem}(\mathcal{B}_F) \approx & \mathbf{LocalMem}(\mathcal{A}) + \mathbf{LocalMem}(\mathtt{Gen}) \\
& + \mathbf{LocalMem}(\mathtt{Dec}_1) + 11\lambda + 2
\end{aligned}$$

10.1 Proof of Theorem 11

Our first step is to we construct a prf $\hat{F} : \{0,1\}^\lambda \times \mathcal{M} \to \mathcal{K}$ that we shall use in the proof.

10.2 Construction of $\hat{F}$

Construction 12 *Fix a public key pk of* PKE_1. *Let* $F : \{0,1\}^\lambda \times \mathcal{C}_1 \to \mathcal{K}$. *We define* $\hat{F}$ *as*

$$\hat{F}_{pk}(k, m) = F(k, \mathtt{Enc}_1(pk, m))$$

In order to use the map then prf technique, we need the following lemma.

Lemma 9. *Fix pk. If F is a prf, then* $\hat{F}_{pk}$ *is a prf. Moreover, for every adversary* $\mathcal{B}_{\hat{F}}$ *against* $\hat{F}_{pk}$, *there exists a* $\mathcal{B}_F$ *against F such that,*

$$\begin{aligned}\mathbf{Adv}^{\mathrm{prf}}_{\mathcal{B}_{\hat{F}},\hat{F}} &\leq \mathbf{Adv}^{\mathrm{prf}}_{\mathcal{B}_F,F} + \delta_1(q)\\ \mathbf{LocalTime}(\mathcal{B}_F) &= \mathbf{LocalTime}(\mathcal{B}_{\hat{F}}) + q\mathbf{LocalTime}(\mathtt{Enc}_1)\\ \mathbf{LocalMem}(\mathcal{B}_F) &= \mathbf{LocalMem}(\mathcal{B}_{\hat{F}}) + 3\lambda.\end{aligned}$$

where q is the number of queries made by $\mathcal{B}_{\hat{F}}$.

Proof. Let $\mathcal{O}$ be the oracle of $\mathcal{B}_F$. $\mathcal{B}_F$ runs $\mathtt{Gen}$ to receive pk, sk, and invokes $\mathcal{B}_{\hat{F}}$. For every query m of $\mathcal{B}_{\hat{F}}$, $\mathcal{B}_F$, computes $c = \mathtt{Enc}_1(pk, m)$ and and checks whether $m = \mathtt{Dec}_1(sk, c)$. If the equality does not hold, $\mathcal{B}_F$ aborts. Otherwise, it queries $\mathcal{O}(c)$. The response of the oracle is passed to $\mathcal{B}_{\hat{F}}$. When $\mathcal{B}_{\hat{F}}$ outputs a bit b, $\mathcal{B}_F$ outputs b.

Probability that $\mathcal{B}_F$ aborts is bounded by $\delta_1(q)$. Conditioned on the event that$\mathcal{B}_F$ does not abort, the simulation of the prf game for $\mathcal{B}_{\hat{F}}$ is perfect. Hence,

$$\mathbf{Succ}_{\mathcal{B}_{\hat{F}},\mathrm{prf}[\hat{F}]} = \mathbf{Succ}_{\mathcal{B}_F,\mathrm{prf}[F]} + \mathrm{Prob}[\mathcal{B}_F \text{ aborts}] \leq \mathbf{Succ}_{\mathcal{B}_F,\mathrm{prf}[F]} + \delta_1(q)$$

Hence, we get,

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathcal{B}_{F'},F'} = \mathbf{Adv}^{\mathrm{prf}}_{\mathcal{B}_{\hat{F}},\hat{F}} + \delta_1(q)$$

In order to simulate the prf game of $\hat{F}_{pk}$, $\mathcal{B}_F$ needs to run $\mathtt{Enc}_1$ for q many times. Moreover, $\mathcal{B}_F$ needs store pk, sk and a temporary variable for passing the values. The lemma follows.

10.3 The reduction

Theorem 11 is proven via a sequence of games. Formal description of the games are given in Figure 25, and Figure 26.

```
$\mathbf{G_0}$ $\boxed{\mathbf{G_1}}$
1 :  $(pk, sk) \leftarrow \mathtt{Gen}(1^\lambda)$
2 :  $b \xleftarrow{\$} \{0,1\}$3 :  $m^* \xleftarrow{\$} \mathcal{M}$
4 :  $c^* = \mathtt{Enc}_1(pk, m^*)$
5 :  $K_0^* = \mathtt{H}(m^*)$ $\boxed{K_0^* \xleftarrow{\$} \mathcal{K}}$
6 :  $K_1^* \xleftarrow{\$} \mathcal{K}$
7 :  $b^* \leftarrow \mathcal{A}^{\mathtt{Decap},\mathtt{H}}(pk, c^*, K_b^*)$
8 :  if $b = b^*$return 1
9 :  else return 0
10 :  endif
```

```
Procedure $\mathtt{H}(m)$ in $\mathbf{G_0}$
1 :  if $\mathtt{H}(m)$ is undefined
2 :    $\mathtt{H}(m) \xleftarrow{\$} \mathcal{K}$
3 :  endif
4 :  return $\mathtt{H}(m)$
```

```
Procedure $\mathtt{Decap}(c)$ in $\mathbf{G_0}, \mathbf{G_1}$
1 :  if $c = c^*$return $\perp$
2 :  $m' = \mathtt{Dec}_1(sk, c)$
3 :  if $m' = \perp$
4 :    return $\perp$
5 :  else
6 :    $K = \mathtt{H}(m')$
7 :  return $K$
```

```
Procedure $\mathtt{H}(m)$ in $\mathbf{G_1}$
1 :  if $m = m^*$
2 :    return $K_0^*$
3 :  else
4 :    if $\mathtt{H}(m)$ is undefined
5 :      $\mathtt{H}(m) \xleftarrow{\$} \mathcal{K}$
6 :    endif
7 :  return $\mathtt{H}(m)$
8 :  endif
```

Fig. 25. The games G_0 and G_1. In game G_1, Line 5 is replaced by the boxed statement

Game G_0. The game G_0 is the original IND-CCA game.

$$\mathbf{Adv}^{\text{IND-CCA}}_{\mathcal{A},\Pi} \overset{def}{=} \left|\text{Prob}[G_0^{\mathcal{A}} = 1] - \frac{1}{2}\right|.$$

Game G_1: We predefine $K_0^* = \mathtt{H}(m^*)$ by sampling a random element from the keyspace $\mathcal{K}$. m^* is the randomly sampled element from $\mathcal{M}$ where $c^* = \mathtt{Enc}_1(pk, m^*)$ is the challenge ciphertext, sent in the KEM game. The hash oracle is modified to return K_0^* for the input (m^*). As K_0^* is still uniformly chosen at random, and the hash oracle output is consistent, there is no change in the distribution of adversary's view.

$$\text{Prob}[G_0^{\mathcal{A}} = 1] = \text{Prob}[G_1^{\mathcal{A}} = 1]$$

Game G_2. In this game the oracles $\mathtt{H}$ and $\mathtt{Decap}$ are changed. We replace the random oracle by a prf $\hat{F}_{pk} : \{0,1\}^\lambda \times \mathcal{M} \to \mathcal{K}$. The following lemma bounds the distinguishing advantage of G_1 and G_2.

Lemma 10. *There exists an adversary $\mathcal{B}_{\hat{F}}$ such that*

$$\left|Prob[G_1^{\mathcal{A}} = 1] - Prob[G_2^{\mathcal{A}} = 1]\right| \leq \mathbf{Adv}^{\text{prf}}_{\mathcal{B}_{\hat{F}}, \hat{F}_{pk}}$$

Game G_3. We rewrite the prf evaluation of $\hat{F}_{pk}$ using prf F as defined in Construction 12. In the procedure $\mathtt{Decap}$ of the game G_2, we no longer need to decrypt c for evaluating the hash. Conditioned on the event that correctness error does not occur for $\mathtt{Enc}_1(pk, \mathtt{Dec}_1(sk, c)$, for all queried, valid c, and using the fact that PKE_1 is deterministic, we get that $F(k, \mathtt{Enc}_1(pk, \mathtt{Dec}_1(sk, c))$ always evaluates to $F(k, c)$. As the view of the adversary remains unchanged until correctness error occurs,

$$\left|\mathrm{Prob}[G_2^{\mathcal{A}} = 1] - \mathrm{Prob}[G_3^{\mathcal{A}} = 1]\right| \leq \delta_1(q_D).$$

Game G_4: In this game, we set a flag Flag and abort on the event that $\mathcal{A}$ queries $\mathtt{H}$ on (m^*). By the fundamental lemma of game playing proofs

$$\left|\mathrm{Prob}[G_3^{\mathcal{A}} = 1] - \mathrm{Prob}[G_4^{\mathcal{A}} = 1]\right| \leq \mathrm{Prob}[\textsc{Flag} = 1].$$

In the game G_4, the adversary $\mathcal{A}$ is unable to compute $\mathtt{H}(m^*)$ using either the hash oracle or the decapsulation oracle. The decapsulation oracle outputs $\perp$ whenever the input c is equal to c^*. The hash oracle aborts for the input (m^*). This implies that the bit b is independent from the adversaries view. Hence

$$\mathrm{Prob}[G_3^{\mathcal{A}}] = \frac{1}{2}.$$

To bound Prob[Flag = 1], we construct an algorithm $\mathcal{B}_{PKE}$ against the OW-VA security of PKE_1. $\mathcal{B}_{PKE}$ simulates game G_4 for $\mathcal{A}$.

The adversary to game OW-VA Description of $\mathcal{B}_{PKE}$ Formal code of $\mathcal{B}_{PKE}$ is given in Figure 27. In order to simulate Step 2 and Step 3 in the $\mathtt{Decap}$ subroutine of the game G_4, $\mathcal{B}_{PKE}$ uses the $\mathtt{CVO}$ oracle in Step 2 the $\mathtt{Decap}$ subroutine (see Figure 27). It is straightforward to check that the simulation is perfect. Flag = 1 implies that $\mathcal{A}$ queried $\mathtt{H}(m^*)$ where $c^* = \mathtt{Enc}_1(pk, m^*)$ and c^* was the challenge ciphertext received by $\mathcal{B}_{PKE}$ and is passed on as the challenge ciphertext to $\mathcal{A}$ in the IND-CCA game. $\mathcal{B}_{PKE}$ returns that m^* and wins the OW-VA game. Hence,

$$\mathrm{Prob}[\textsc{Flag} = 1] = \mathbf{Adv}^{\text{OW-VA}}_{\mathcal{B}_{PKE}, \mathsf{PKE}_1}$$

Assuming Lemma 10 holds, collecting the probabilities, we get

$$\mathbf{Adv}^{\text{IND-CCA}}_{\mathcal{A}, \mathit{\Pi}} \leq \mathbf{Adv}^{\text{OW-VA}}_{\mathcal{B}_{PKE}, \mathsf{PKE}_1} + \mathbf{Adv}^{\text{prf}}_{\mathcal{B}_{\hat{F}}, \hat{F}}$$

Efficiency of $\mathcal{B}_{PKE}$
$\mathcal{B}_{PKE}$ runs $\mathcal{A}$, queries $\mathtt{CVO}$ oracle for q_D many times, runs $\mathtt{Enc}_1$ for q_H many times, computes the prf F for $(q_H + q_D)$ many times.

$$\begin{aligned}\mathbf{LocalTime}(\mathcal{B}_{PKE}) =& \mathbf{LocalTime}(\mathcal{A}) + (q_H + q_D)\mathbf{LocalTime}(F) \\ &+ q_H \mathbf{LocalTime}(\mathtt{Enc}_1) + q_D\end{aligned}$$

The last q_D term in the right hand side of the above equation is to denote the number of queries made to the $\mathtt{CVO}$ oracle.
Memory Efficiency of $\mathcal{B}_{PKE}$
$\mathcal{B}_{PKE}$ needs to save the code of $\mathcal{A}$, $\mathtt{Enc}_1$, and F. In addition, there are the following λ size registers, c^*, k, K^*, m, c, and a single bit b^*.

```
$\mathbf{G_2}$
$(pk, sk) \leftarrow \mathtt{Gen}$
$k \xleftarrow{\$} \{0,1\}^\lambda$
$b \xleftarrow{\$} \{0,1\}$
$m^* \xleftarrow{\$} \mathcal{M}$
$c^* = \mathtt{Encrypt}_1(pk, m^*)$
$K_0^* \xleftarrow{\$} \mathcal{K}$
$K_1^* \xleftarrow{\$} \mathcal{K}$
$b^* \leftarrow \mathcal{A}^{\mathtt{Decap},\mathtt{H}}(pk, c^*, K_b^*)$
if $b = b^*$return $1$
  else
  return $0$
  endif

Procedure $\mathtt{H}(m)$
if $m = m^*$
  return $K_0^*$
else
   $K = \hat{F}_{pk}(k, m)$
   return $K$
endif

Procedure $\mathtt{Decap}(c)$
if $c = c^*$return $\perp$
$m' = \mathtt{Decrypt}_1(sk, c)$
if $m' =\perp$ return $\perp$
else
   $K = \hat{F}_{pk}(k, m')$
return $K$
```

```
$\mathbf{G_3}$
$(pk, sk) \leftarrow \mathtt{Gen}$
$k \xleftarrow{\$} \{0,1\}^\lambda$
$b \xleftarrow{\$} \{0,1\}$
$m^* \xleftarrow{\$} \mathcal{M}$
$c^* = \mathtt{Encrypt}_1(pk, m^*)$
$K_0^* \xleftarrow{\$} \mathcal{K}$
$K_1^* \xleftarrow{\$} \mathcal{K}$
$b^* \leftarrow \mathcal{A}^{\mathtt{Decap},\mathtt{H}}(pk, c^*, K_b^*)$
if $b = b^*$return $1$
  else
  return $0$
  endif

Procedure $\mathtt{H}(m)$
if $m = m^*$
  return $K_0^*$
else
   $c = \mathtt{Encrypt}_1(pk, m)$
   $K = F(k, c)$
   return $K$
endif

Procedure $\mathtt{Decap}(c)$
if $c = c^*$return $\perp$
$m' = \mathtt{Decrypt}_1(sk, c)$
if $m' =\perp$ return $\perp$
else
   $K = F(k, c)$
   return $K$
```

```
$\mathbf{G_4}$
$(pk, sk) \leftarrow \mathtt{Gen}$
$k \xleftarrow{\$} \{0,1\}^\lambda$
$b \xleftarrow{\$} \{0,1\}$
$m^* \xleftarrow{\$} \mathcal{M}$
$c^* = \mathtt{Encrypt}_1(pk, m^*)$
$K_0^* \xleftarrow{\$} \mathcal{K}$
$K_1^* \xleftarrow{\$} \mathcal{K}$
$b^* \leftarrow \mathcal{A}^{\mathtt{Decap},\mathtt{H}}(pk, c^*, K_b^*)$
if $b = b^*$return $1$
  else
  return $0$
  endif

Procedure $\mathtt{H}(m)$
if $m = m^*$
    FLAG=1
    Abort
else
   $c = \mathtt{Encrypt}_1(pk, m)$
   $K = F(k, c)$
   return $K$
  endif

Procedure $\mathtt{Decap}(c)$
if $c = c^*$return $\perp$
$m' = \mathtt{Decrypt}_1(sk, c)$
if $m' =\perp$ return $\perp$
else
   $K = F(k, c)$
return $K$
```

Fig. 26. IND-CCA game of $U_m^{\perp}$: highlighted statements are the modifications from the previous game

$$\begin{aligned}\mathbf{LocalMem}(\mathcal{B}_{PKE}) =&\mathbf{LocalMem}(\mathcal{A}) + \mathbf{LocalMem}(F)\\ &+ \mathbf{LocalMem}(\mathtt{Enc}_1) + 5\lambda + 1\end{aligned}$$

The PRF adversary: Proof of Lemma 10 We construct the PRF adversary $\mathcal{B}_{\hat{F}}$ in Figure 28. $\mathcal{B}_{\hat{F}}$ samples the values pk, sk using Gen. Next, it chooses $m^* \xleftarrow{\$} \mathcal{M}$, $b \xleftarrow{\$} \{0,1\}$, and $K_0^*, K_1^* \xleftarrow{\$} \mathcal{K}$. Then $\mathcal{B}_{\hat{F}}$ computes $c^* = \mathtt{Enc}_1(pk, m^*)$ and runs $\mathcal{A}(pk, c^*, K_b^*)$. If $\mathcal{A}$ queries the hash oracle with m^*, $\mathcal{B}_{\hat{F}}$ return K_0^*. Similarly,

```
Algorithm $\mathcal{B}_{PKE}^{\mathtt{CVO}(.)}(pk, c^*)$
  $k \xleftarrow{\$} \{0,1\}^\lambda$
  $K^* \xleftarrow{\$} \mathcal{K}$
  $b^* \leftarrow \mathcal{A}^{\mathtt{Decap},\mathtt{H}}(pk, c^*, K^*)$

Procedure $\mathtt{Decap}(Y)$
  if $c = c^*$return $\perp$
  if $\mathtt{CVO} = 0$ return $\perp$
  else
    $K = F(k, c)$
  return $K$

Procedure $\mathtt{H}(m)$
  if $\mathtt{Enc}_1(pk, m) = c^*$
      $\textsc{Flag} = 1$
      $Outputm$
  else
     $c = \mathtt{Enc}_1(pk, m)$
     $K = F(k, c)$
     return $K$
    endif
```

Fig. 27. OW-VA adversary $\mathcal{B}_{PKE}$

if $\mathcal{A}$ queries the decapsulation oracle `Decap` with input c^*, $\mathcal{B}_{\hat{F}}$ returns $\perp$. Fo all other queries, $\mathcal{B}_{\hat{F}}$ simulates the `Decap` and `H` oracle by invoking its own oracle $\mathtt{O}_{\hat{\mathtt{F}}}$. When $\mathcal{A}$ reurns a bit b', $\mathcal{B}_{\hat{F}}$ outputs 1 if $b = b'$ and outputs 0 otherwise.

When $\mathtt{O}_{\hat{\mathtt{F}}}$ is a random function (Game Random), $\mathcal{B}_{\hat{F}}$ simulates game G1 perfectly. On the other hand when $\mathtt{O}_{\hat{\mathtt{F}}}$ is the pseudorandom function $\hat{F}_{pk}$ (Game Real), $\mathcal{B}_{\hat{F}}$ provides perfect simulation of the game G_2. Hence

$$\left|\text{Prob}[G_1^{\mathcal{A}} = 1] - \text{Prob}[G_2^{\mathcal{A}} = 1]\right| \leq \mathbf{Adv}^{\text{prf}}_{\mathcal{B}_{\hat{F}}, \hat{F}_{pk}}$$

Efficiency of $\mathcal{B}_{\hat{F}}$.
$\mathcal{B}_{\hat{F}}$ runs $\mathcal{A}$ once, runs the key generation algorithm `Gen` once, the decryption algorithm $\mathtt{Dec}_1$ for q_D times. In addition, $\mathcal{B}_{\hat{F}}$ queries the oracle $\mathtt{O}_{\hat{\mathtt{F}}}$ for $(q_H + q_D)$ many times.

$$\begin{aligned}\mathbf{LocalTime}(\mathcal{B}_{\hat{F}}) =& \mathbf{LocalTime}(\mathcal{A}) + \mathbf{LocalTime}(\mathtt{Gen}) \\ &+ q_D\mathbf{LocalTime}(\mathtt{Dec}_1) + (q_H + q_D)\end{aligned}$$

$\mathcal{B}_{\hat{F}}$ needs to save the code of $\mathcal{A}, \mathtt{Gen}$, and $\mathtt{Dec}_1$. In addition, there are nine λ size registers, $pk, sk, m^*, m, c, c^*K, K_0^*, K_1*$, and two single bits b, b'.

$$\begin{aligned}\mathbf{LocalMem}(\mathcal{B}_{\hat{F}}) =& \mathbf{LocalMem}(\mathcal{A}) + \mathbf{LocalMem}(\mathtt{Gen}) \\ &+ \mathbf{LocalMem}(\mathtt{Dec}_1) + 9\lambda + 2\end{aligned}$$

□

```
$\mathcal{B}_{\hat{F}}$
$(pk, sk) \leftarrow \mathtt{Gen}$
$k \xleftarrow{\$} \{0,1\}^{\lambda}$
$b \xleftarrow{\$} \{0,1\}$
$m^* \xleftarrow{\$} \mathcal{M}$
$c^* = \mathtt{Enc}_1(pk, m^*)$
$K_0^* \xleftarrow{\$} \mathcal{K}$
$K_1^* \xleftarrow{\$} \mathcal{K}$
$b^* \leftarrow \mathcal{A}^{\mathtt{Decap},\mathtt{H}}(pk, c^*, K_b^*)$
if $b = b^*$ return $1$
 else
 return $0$
 endif
```

```
Procedure $\mathtt{H}(m)$
if $m = m^*$
  return $K_0^*$
else
   $K = \mathtt{O}_{\hat{\mathtt{F}}}(k, m)$
   return $K$
endif
```

```
Procedure $\mathtt{Decap}(c)$
if $c = c^*$ return $\perp$
$m' = \mathtt{Dec}_1(sk, c)$
if $m' = \perp$ return $\perp$
else
   $K = \mathtt{O}_{\hat{\mathtt{F}}}(k, m')$
return $K$
```

Fig. 28. The PRF adversary $\mathcal{B}_{\hat{F}}$

Finishing the proof of Theorem 11 So far, we have proven that there exists adversaries $\mathcal{B}_{PKE}$ and $\mathcal{B}_{\hat{F}}$

$$\mathbf{Adv}^{\text{IND-CCA}}_{\mathcal{A},\Pi} \leq \mathbf{Adv}^{\text{OW-VA}}_{\mathcal{B}_{PKE},PKE} + \mathbf{Adv}^{\text{prf}}_{\mathcal{B}_{\hat{F}},\hat{F}} + \delta_1(q_D).$$

Note, $\mathcal{B}_{\hat{F}}$ queries the oracle for $(q_H + q_D)$ many times. By applying Lemma 9, we get the adversary $\mathcal{B}_F$ such that

$$\mathbf{Adv}^{\text{prf}}_{\mathcal{B}_{\hat{F}},\hat{F}} = \mathbf{Adv}^{\text{prf}}_{\mathcal{B}_F,F} + \delta_1(q_H + q_D)$$

Hence

$$\mathbf{Adv}^{\text{IND-CCA}}_{\mathcal{A},\Pi} \leq \mathbf{Adv}^{\text{OW-VA}}_{\mathcal{B}_{PKE},PKE} + \mathbf{Adv}^{\text{prf}}_{\mathcal{B}_F,F} + 2\delta_1(q_H + q_D)$$

Further, using the relations in Lemma 9, we get that

$$\begin{aligned}\mathbf{LocalTime}(\mathcal{B}_F) =& \mathbf{LocalTime}(\mathcal{A}) + \mathbf{LocalTime}(\mathtt{Gen})\\ &+ q_D\mathbf{LocalTime}(\mathtt{Dec}_1) + (q_H + q_D)(1 + \mathbf{LocalTime}(\mathtt{Enc}_1))\\ \mathbf{LocalMem}(\mathcal{B}_F) =& \mathbf{LocalMem}(\mathcal{A}) + \mathbf{LocalMem}(\mathtt{Gen})\\ &+ \mathbf{LocalMem}(\mathtt{Dec}_1) + 12\lambda + 2\end{aligned}$$

This finishes the proof of Theorem 11.