Efficient Compilers for After-the-Fact Leakage: from CPA to CCA-2 secure PKE to AKE

Suvradip Chakraborty[1], Goutam Paul[2] and C. Pandu Rangan[1]

[1] Department of Computer Science and Engineering,
Indian Institute of Technology Madras, India
{suvradip1111, prangan55}@gmail.com
[2] Cryptology and Security Research Unit (CSRU),
R. C. Bose Centre for Cryptology and Security,
Indian Statistical Institute, Kolkata, India
goutam.paul@isical.ac.in

Abstract. The goal of leakage-resilient cryptography is to construct cryptographic algorithms that are secure even if the adversary obtains side-channel information from the real world implementation of these algorithms. Most of the prior works on leakage-resilient cryptography consider leakage models where the adversary has access to the leakage oracle before the challenge-ciphertext is generated (before-the-fact leakage). In this model, there are generic compilers that transform any leakage-resilient CPA-secure public key encryption (PKE) scheme to its CCA-2 variant using Naor-Yung type of transformations. In this work, we give an efficient *generic compiler* for transforming a leakage-resilient CPA-secure PKE to leakage-resilient CCA-2 secure PKE in presence of *after-the-fact split-state* (bounded) *memory* leakage model, where the adversary has access to the leakage oracle even after the challenge phase. The *salient* feature of our transformation is that the leakage rate (defined as the ratio of the amount of leakage to the size of secret key) of the transformed after-the-fact CCA-2 secure PKE is same as the leakage rate of the underlying after-the-fact CPA-secure PKE, which is $1 - o(1)$.

We then present another generic compiler for transforming an after-the-fact leakage-resilient CCA-2 secure PKE to a leakage-resilient authenticated key exchange (AKE) protocol in the bounded after-the-fact leakage-resilient eCK (BAFL-eCK) model proposed by Alawatugoda et al. (ASIACCS'14). To the best of our knowledge, this gives the *first* compiler that transform any leakage-resilient CCA-2 secure PKE to an AKE protocol in the leakage variant of the eCK model.

1 Introduction and Related Works

Most of the real-world attacks on a cryptosystem target the physical implementation of the device in which it is implemented. Such "physical attacks" are usually based on the *side-channel information* about the internals of the cryptographic device, which the adversary may get via myriads of side-channel attacks like timing measurements, power analysis, fault injection attacks, electromagnetic measurements, microwave attacks, memory attacks and many more [KJJ99, Koc96, HSH+09]. Leakage-resilient cryptography was introduced to deal with this problem from a theoretical standpoint. It guarantees the security of the cryptosystems even in the face of side-channel attacks and analyzes the effectiveness of side-channel countermeasures in a mathematically rigorous way. The broad idea is that in addition to the usual interfaces with which the adversary can interact with the cryptographic primitive, he/she can choose arbitrary leakage functions (subject to some technical constraints) and get back the result of applying these functions on the secret state of the system.

Based on the restrictions on the leakage functions, various leakage models have evolved in the literature over the past decade. In their pioneering work named "physically observable cryptography", Micali and Reyzin [MR04] put up a comprehensive framework to

model side-channel attacks called *only computation leaks information* (OCLI). Their axiom relies on the assumption that leakage happens as a result of computation and there is no leakage in the absence of computation. Inspired by the "cold-boot attack" Halderman [HSH+09], Akavia and Goldwasser [AGV09] formalized the notion of "bounded memory leakage" model. This model removes the restriction that leakage only happens from computation. Instead it allows the adversary to learn any arbitrary information about the secret state of the system stored in memory, with the only restriction that the amount of leakage is bounded. A generalization of the above model called the continuous leakage model was proposed by Dodis et.al [DHLAW10a] and Brakerski et al. [BKKV]. This model places no bound on the overall size of the leakage. The secret key of the cryptosystem is refreshed periodically (erasing the old one) keeping the public key same, and the adversary can obtain bounded leakage in between any two successive key refreshes.

After-The-Fact Leakage. Most of the prior formulations of leakage-resilient PKE [AGV09, NS09, BKKV, QL13, QL14] considered leakage before the challenge ciphertext is made available to the adversary. So, even if one bit of the secret key leaks in the post-challenge phase, the security of the previously encrypted messages may not be guaranteed. This severely restricts the meaning and applicability of this security notion and also the resulting constructions. However, this seems to be a necessary restriction, as otherwise an adversary may design a leakage function by simply encoding the decryption function along with the challenge ciphertext and the two messages (submitted in the challenge phase) to leak exactly the bit that we are trying to hide using encryption.

Halevi and Lin [HL11] proposed the *first* meaningful security notion of after-the-fact leakage(AFL) in the context of PKE schemes. Since achieving security against after-the-fact leakage in its full generality is impossible, they considered the *split-state* leakage restriction, where it is assumed that the secret key is split into two parts (in general can be multiple) and each of them is stored in separate memory locations. The adversary can get leakages from each of this memory locations, but independently of each other. Then they showed how to construct an AFL-CPA-secure PKE scheme under their new security model. The leakage rate (defined as the ratio of the leakage tolerated by the scheme to the size of the secret key) of their construction approached $1 - o(1)$ under appropriate choice of parameters. Later, Dziembowski and Faust [DF11] gave a construction of a AFL-CCA-2 secure PKE scheme in the continuous leakage model under the split-state assumption, with the further restriction that leakage only happens from computation (OCLI axiom). The leakage rate tolerated by their construction is also far from the optimal $1 - o(1)$ rate obtained by the CPA-secure construction of Halevi and Lin [HL11]. Zhang et al. [ZCC15] proposed a generic transformation from AFL-CPA-secure PKE scheme to a AFL-CCA-2-secure PKE scheme. Their transformation preserves the leakage rate of the AFL-CPA-secure PKE scheme, and hence achieves a leakage rate of $1 - o(1)$. However, the main drawback of their transformation is that it is very *inefficient* since it uses simulation-sound non-interactive zero-knowledge proof system, which is far from practical. Fujisaki et al. [FKN+15] constructed a multiple-challenge CCA-secure PKE that simultaneously tolerates post-challenge secret key and sender-randomness leakage in the split-state leakage model. However, in their construction that randomness is also split into two parts, unlike ours, where we consider only the secret key to be spitted, and *not* the randomness. Also, the scheme of [FKN+15] cannot support split-state decryption as defined in [HL11] and also in this work. Hence, the two approaches are incomparable.

Leakage-resilient AKE. Authenticated Key Exchange (AKE) protocols allow two parties to jointly compute a unique shared secret key and also to mutually authenticate each other with the assurance that the shared key is known only to them. In our work, we consider the case of 2-party AKE setting. The traditional security models for AKE protocols [BR94, Sho99, CK01, Kra05, LLM07, MU08, SEVB10, Cre11] do not incorporate the

possibility of side-channels and hence the AKE protocols analyzed in these models may be completely insecure in the face of side-channel attacks.

Alwen et al. [ADW09] gave the first construction of leakage-resilient AKE (LR-AKE) protocol in the RO model. However, the protocol requires three passes and also does not capture after-the-fact leakage. Later, Moriyama and Okamoto [MO11] proposed a two-pass (one round) LR-AKE protocol by extending the eCK model to the setting of bounded memory leakage introduced in [AGV09]. However, it also does not capture after-the-fact leakage. In the context of key exchange, after-the-fact leakage was first modeled by Alawatugoda *et al.* [ASB14] in both the bounded and continuous leakage setting. They also gave somewhat generic constructions of LR-AKE protocols in their new models [ASB14, ASB15]. Unfortunately, both these protocols have been shown insecure in their respective models in the subsequent works (see [YL16], [Too15]). Recently, Chen *et al.* [CMY+16] gave a generic framework for constructing LR-AKE protocols in the presence of after-the-fact leakage in the bounded memory leakage model (they called their model challenge-dependent eCK (CLR-eCK) model).

2 Our Contributions

In this work we continue the study of after-the-fact leakage in the context of CCA-2 secure public key encryption (PKE) schemes and authenticated key exchange (AKE) protocols. Our contributions are two-fold and described below.

1. As our *first* contribution, we give a *generic compiler* from a AFL-CPA-secure PKE scheme to to a AFL-CCA-2 secure PKE scheme. The *salient* feature of our compiler is that it *preserves the leakage rate* in the CPA to CCA transformation mentioned above. In other words, the amount of leakage than can be tolerated by our AFL-CCA-2 secure PKE scheme is the *same* as the amount of leakage tolerated by the underlying AFL-CPA secure PKE scheme. Besides, our compiler is also much more *efficient* than the compiler proposed in [ZCC15]. So, on one hand our AFL-CCA-2 secure PKE scheme achieves the optimal leakage rate of $1 - o(1)$, and on the other hand is much more efficient than the state-of-the-art AFL-CCA-2 secure PKE constructions.
2. As our *second* contribution, we propose a *generic compiler* from AFL-CCA-2 secure PKE scheme to an after-the-fact leakage-resilient AKE protocol in the BAFL-eCK security model (which is leakage analogue of the eCK model for AKE protocols) proposed in [ASB14]. Note that such a compiler from a CCA-2 secure PKE to a eCK-secure AKE protocol in the standard (non-leakage) model was already proposed by Alawatugoda [Ala15a]. They left such a transformation in the context of leakage as a future open problem. Our compiler from AFL-CCA-2 secure PKE scheme to BAFL-eCK secure AKE protocol can be seen as a leakage-resilient implementation of the compiler presented in [Ala15a], and hence we solve the above open problem.

We now give the high level ideas for each of these contributions.

2.1 Compiler for after-the-fact leakage-resilient CCA-2 secure PKE schemes.

As our *first* contribution, we give a generic compiler from a AFL-CPA-secure PKE scheme to to a AFL-CCA-2 secure PKE scheme. The main tool used in our transformation is true-simulation extractable non-interactive zero-knowledge (tSE-NIZK) argument system [DHLAW10b]. This notion of tSE-NIZK is similar to the notion of simulation extractable NIZK [Gro06], except that the adversary gets to see proofs of *true* statements only rather than proofs of arbitrary statements as in simulation extractable NIZK. In particular, the adversary can see simulated proofs for true statements in the relation, and in addition there is an extractor that can extract a witness from any proof (with the help of an extraction trapdoor) produced by a malicious prover. We show that the CCA-2 secure

PKE scheme of Dodis et. al [DHLAW10b] secure against before-the-fact[3] leakage is also secure against after-the-fact leakage under the split-state assumption. The starting point of our transformation is the AFL-CPA secure PKE scheme of Halevi and Lin [HL11]. Note that the AFL-CPA secure PKE scheme can already handle pre- and post-challenge leakage queries of the adversary, since it is secure against after-the-fact leakage. So we need to show that adding decryption queries in the pre- and post-challenge phase does not provide any extra leakage to the adversary. We first encrypt the message m using the above AFL-CPA secure PKE scheme and prove knowledge of the underlying plaintext and randomness r (used to encrypt m) using the tSE-NIZK argument system. In this case the plaintext-randomness pair (m, r) acts as our witness. The decryption queries of the adversary is then answered by the extractor of the tSE-NIZK argument system by using the extraction trapdoor, which can extract the witness (in our case the underlying plaintext messages) to answer the decryption queries. Since the extraction trapdoor is never used in the real encryption scheme, the adversary gets no leakage from it. This essentially makes the decryption oracle useless and the adversary learns no extra leakage from the decryption process. So the leakage at this point only happens from memory which can be handled by the underlying AFL-LR-CPA-secure PKE scheme.

Comparison of our work with [ZCC15]. As mentioned in the introduction, Zhang et. al [ZCC15] also proposed a transformation from AFL-CPA secure PKE scheme to a AFL-CCA-2 secure PKE scheme achieving an optimal leakage rate of $1-o(1)$ as ours. However, there are two fundamental differences between our work and the work of [ZCC15]. *Firstly*, in the CPA to CCA transformation of [ZCC15], they use a simulation-sound NIZK (SS-NIZK) proof system. SS-NIZK argument systems are extremely inefficient and far from practical. Instead, we observe that tSE-NIZK argument system is sufficient for the same purpose and hence we use tSE-NIZK argument system for our transformation. As already shown in [DHLAW10b], a tSE-NIZK argument can be constructed relying only on standard (labeled) CCA-2 secure PKE scheme and a regular NIZK argument system, and hence is much more efficient and practical than SS-NIZK argument system. . *Secondly*, the transformation presented in [ZCC15] is not direct, in the sense that they showed a two level transformation. First, they showed a transformation from an entropic CPA-secure PKE scheme (a notion introduced in [HL11] for achieving full-fledged AFL-CPA security) to a entropic CCA-secure PKE scheme and then they showed a construction of AFL-CCA-2 secure PKE from such an entropic CCA-secure PKE scheme using two-source extractors and additionally a (one-time) strongly unforgeable signature scheme. On the other hand, we show a direct (one-level) transformation from an AFL-CPA-secure PKE to a AFL-CCA-2 secure PKE scheme without taking the route of entropic PKE schemes. Hence, our transformation is more *efficient* than the one presented in [ZCC15].

2.2 Compiler for leakage-resilient AKE.

As our second contribution, we propose a *second generic compiler* from AFL-CCA-2 secure PKE scheme to an after-the-fact leakage-resilient AKE protocol in the BAFL-eCK security model (which is leakage analogue of the eCK model for AKE protocols) proposed in [ASB14].

The BAFL-eCK model allows the adversary to obtain leakage from computation apart from all other capabilities of an adversary in the eCK model [LLM07]. Our compiler can be seen as leakage-resilient implementation of the compiler of Alawatugoda et. al [Ala15b]. Each party has a pair of long-term Diffie-Hellman (DH) public and secret keys and a pair of public-private key from the CCA-2 secure PKE scheme. Each party computes a DH ephemeral public key and encrypts it using the AFL-CCA-2 secure PKE scheme.

[3] Recall that in *before-the-fact* leakage model the adversary gets access to the leakage oracle only before the challenge phase, in contrast to after-the-fact leakage, where the adversary has the leakage oracle access even after the challenge phase.

The other party can decrypt the ciphertext using its secret key of the PKE scheme and perform DH type computation to derive the session keys. Since the PKE scheme AFL-CCA-2 secure, the AKE protocol is also secure against after-the-fact leakage in split-state. However, the PKE scheme addresses leakage *only* from the public-secret key pair of the PKE scheme. However, each party also has a DH long-term key pair, and the adversary *also* gets leakage from this key pair. So, we have to store the long-term DH keys in a *leakage-resilient* manner and also compute the shared session key in a leakage-resilient fashion.

How to make the Diffie-Hellman key exchange protocol leakage-resilient? The main idea is to use the leakage-resilient storage (LRS) scheme of [DF11]. The LRS scheme stores a secret value securely in the presence of leakage (in split-state). However, directly using the LRS scheme *does not* work in our case. This is because in DH key exchange the process of session key derivation is done by manipulating the DH keys in the exponent. So, we need to perform DH *exponentiation* in a *leakage-resilient* fashion. For this, we use the ideas used in [ABS14, Ala15b] to perform leakage-resilient exponentiation using the LRS encoding scheme. Combined with the security of the AFL-CCA-2 secure PKE scheme and the security of the LRS scheme, we achieve a generic BAFL-eCK-secure AKE protocol.

3 Organization

The rest of the paper is organized as follows. In Section 4, we provide the necessary preliminaries required for our constructions. In Section 5, we give the security model for after-the-fact leakage resilient CCA-2 secure (AFL-CCA-2) PKE in split-state state (Section 5.1) and present our compiler from CPA to CCA-2 secure PKE in the same model (Section 5.2). In Section 6, we present our generic compiler from CCA-2 secure PKE in the above model to a BAFL-eCK-secure AKE in the standard model. We give the BAFL-eCK model in Section 6.1 and then present our compiler from AFL-CCA-2 secure PKE to BAFL-eCK-secure AKE in Section 6.2. Finally Section 7 concludes the paper.

4 Preliminaries

In this section, we provide some basic notations, definitions and tools needed throughout the paper.

4.1 Notations

Throughout this work, we denote the security parameter by κ. We assume that all the algorithms take as input (implicitly) the security parameter represented in unary, i.e., 1^κ. For an integer $n \in \mathbb{N}$, where $\mathbb{N}$ denotes the set of natural numbers, we use the notation $[n]$ to denote the set $[n] \stackrel{\text{def}}{=} \{1, \ldots, n\}$. For a randomized function f, we write $f(x; r)$ to denote the unique output of f on input x with random coins r. We write $f(x)$ to denote a random variable for the output of $f(x; r)$, over the random coins r. For a set S, we let U_S denote the uniform distribution over S. For an integer $r \in \mathbb{N}$, let U_r denote the uniform distribution over $\{0,1\}^r$, the bit strings of length r. For a distribution or random variable X, we denote by $x \leftarrow X$ the action of sampling an element x according to X. For a set S, we write $s \xleftarrow{\$} S$ to denote sampling s uniformly at random from the S. A function μ is negligible iff $\forall c \in \mathbb{N}$, $\exists n_0 \in \mathbb{N}$ such that $\forall n \geq n_0$, $\mu(n) < n^{-c}$. We sometimes use $\mathrm{negl}(\kappa)$ to denote the set of negligible functions $\mu(\kappa)$. We denote an ensemble $\mathcal{X}$ as a collection of distributions $\{\mathcal{X}_\kappa\}_{\kappa \in \mathbb{N}}$. We sometimes drop the subscript κ when clear from context and write $x \leftarrow \mathcal{X}$ instead of $x \leftarrow \mathcal{X}_\kappa$ to denote sampling an element x from $\mathcal{X}_\kappa$. For two matrices A and B, we denote $A \odot B$ to denote the multiplication of A and B.

Let $\mathbb{G}$ be a group of prime order p such that $\log_2(p) \geq \kappa$. Let g be a generator of $\mathbb{G}$, then for a (column/row) vector $A = (A_1, \cdots, A_n) \in \mathbb{Z}_p^n$, we denote by g^A the vector $C = (g^{A_1}, \cdots, g^{A_n})$. Furthermore, for a vector $B = (B_1, \cdots, B_n) \in \mathbb{Z}_p^n$, we denote by C^B the group element $X = \prod_{i=1}^n g^{A_i B_i} = g^{\sum_{i=1}^n A_i B_i}$.

4.2 Entropy and Randomness Extraction

We begin with some definitions and then state an useful result.

Definition 1. (Min-Entropy). The *min-entropy* of a random variable X, *denoted as* $\mathrm{H}_\infty(X)$ *is defined as* $\mathrm{H}_\infty(X) \stackrel{\text{def}}{=} -\log(\max_x \Pr[X = x])$. This is a standard notion of entropy used in cryptography, since it measures the worst-case predictability of X.

Definition 2. (Average Conditional Min-Entropy). The *average-conditional min-entropy* of a random variable X conditioned on a (possibly) correlated variable Z, denoted as $\widetilde{\mathrm{H}}_\infty(X|Z)$ is defined as

$$\widetilde{\mathrm{H}}_\infty(X|Z) = -\log\left(\mathbb{E}_{z \leftarrow Z}\left[\max_x \Pr[X = x|Z = z]\right)\right) = -\log\left(\mathbb{E}_{z \leftarrow Z}\left[2^{\mathrm{H}_\infty(X|Z=z)}\right]\right).$$

This measures the worst-case predictability of X by an adversary that may observe a correlated variable Z.

The following bound on average min-entropy was proved in [DORS08].

Lemma 1. [DORS08] *For any random variable X, Y and Z, if Y takes on values in $\{0,1\}^l$, then*

$$\widetilde{\mathrm{H}}_\infty(X|Y,Z) \geq \widetilde{\mathrm{H}}_\infty(X|Z) - l \quad \text{and} \quad \widetilde{\mathrm{H}}_\infty(X|Y) \geq \widetilde{\mathrm{H}}_\infty(X) - l.$$

Definition 3. (Randomness Extractor). We say that an efficient randomized function Ext: $\mathcal{X} \times \mathcal{S} \to \mathcal{Y}$ is an (υ, ϵ)-extractor if for all (correlated) random variables X, Z such that the support of X is $\mathcal{X}$ and $\widetilde{\mathrm{H}}_\infty(X|Z) \geq \upsilon$, we get $(Z, S, \mathsf{Ext}(X;S)) \approx_\epsilon (Z, S, U_\mathcal{Y})$, where S is uniform over $\mathcal{S}$, and $U_\mathcal{Y}$ denotes the uniform distribution over the range of the extractor $\mathcal{Y}$.

4.3 Leakage-resilient Storage

We review the definitions of leakage-resilient storage according to Dziembowski and Faust [DF11]. The idea is to *split* the storage of elements into two parts using a randomized encoding function. As long as leakage is limited from each of its two parts, no adversary can learn useful information about an encoded element.

For any $m, n \in \mathbb{N}$, the storage scheme $\Lambda^{n,m}_{\mathbb{Z}_p^*}$ efficiently stores elements $s \in \mathbb{Z}_p^*$ where:

- $\mathsf{Encode}^{n,m}_{\mathbb{Z}_p^*}(s) : s_L \stackrel{\$}{\leftarrow} (\mathbb{Z}_p^*)^n \setminus \{(0^n)\}$, *and* $s_R \leftarrow (\mathbb{Z}_p^*)^{n \times m}$ *such that* $s_L \odot s_R = s$, *where s_L and s_R are interpreted as $(1 \times n)$ and $(n \times m)$ matrices respectively. The function finally outputs (s_L, s_R).*
- $\mathsf{Decode}^{n,m}_{\mathbb{Z}_p^*}(s_L, s_R)$: *outputs* $s_L \odot s_R = s$.

Definition 4. (λ_S-limited adversary). If the amount of leakage obtained by the adversary from each of s_L and s_R is limited to λ_{S_L} and λ_{S_R} bits respectively, the adversary is known as a λ_S*-limited adversary, where* $\lambda_S = (\lambda_{S_L}, \lambda_{S_R})$.

Definition 5. ($\lambda_\Lambda, \epsilon_1$-secure leakage-resilient storage scheme). We say that $\Lambda = (\mathsf{Encode}, \mathsf{Decode})$ is a $(\lambda_\Lambda, \epsilon_1)$-secure leakage-resilient, if for any $s_0, s_1 \stackrel{\$}{\leftarrow} \mathcal{M}$, and any λ_Λ-limited adversary $\mathcal{C}$, the leakage from $\mathsf{Encode}(s_0) = (s_{0_L}, s_{0_R})$ and $\mathsf{Encode}(s_1) = (s_{1_L}, s_{1_R})$ are statistically ϵ_1 close. For an adversary-chosen leakage function $\mathbf{f} = (f_1, f_2)$, and a secret s such that $\mathsf{Encode}(s) = (s_L, s_R)$, the leakage is denoted as $(f_1(s_L), f_2(s_R))$.

Lemma 2. ([DF11]). *Suppose that $m < n/20$. Then $\Lambda^{n,m}_{\mathbb{Z}_p^*} = (\mathsf{Encode}^{n,m}_{\mathbb{Z}_p^*}(s), \mathsf{Decode}^{n,m}_{\mathbb{Z}_p^*}(s_L, s_R))$ is $(\lambda_S, negl(\kappa))$-secure for some negligible function $negl$ and $\lambda_S = (0.3{\cdot}n \log p, 0.3{\cdot}n \log p)$.*

4.4 (Pseudo-random functions).

We say that $F : \Sigma^k \times \Sigma^m \to \Sigma^n$ is a $(\epsilon_{\mathsf{prf}}, s_{\mathsf{prf}}, q_{\mathsf{prf}})$-secure pseudo-random function (PRF) if no adversary of size s_{prf} can distinguish F (instantiated with a random key) from a uniformly random function, i.e., for any $\mathcal{A}$ of size s_{prf} (viewed as a circuit) making q_{prf} oracle queries we have:

$$| \mathsf{Pr}_{K \xleftarrow{\$} \Sigma^K}[\mathcal{A}^{F(K,.)} \to 1] - \mathsf{Pr}_{R_{m,n}}[\mathcal{A}^{R_{m,n}(.)} \to 1] | \leq \epsilon_{\mathsf{prf}}.$$

where $R(m, n)$ is the set of all functions from $\Sigma^m \to \Sigma^n$.

4.5 Complexity Assumption

The complexity assumption required for our AKE construction is the standard Decisional Diffie-Hellman (DDH) problem.

Definition 6. Computation Diffie-Hellman Problem (CDH) - Given $(g, g^a, g^b) \xleftarrow{\$} \mathbb{G}^3$ for unknown $a, b \in \mathbb{Z}_p^*$, where $\mathbb{G}$ is a cyclic prime order multiplicative group with g as a generator and p the order of the group, the CDH problem in $\mathbb{G}$ is to compute g^{ab}.

The advantage of any probabilistic polynomial time algorithm $\mathcal{A}$ in solving the CDH problem in $\mathbb{G}$ is defined as

$$\mathsf{Adv}^{\mathsf{CDH}}_{\mathcal{A}}(\kappa) = \mathsf{Pr}\left[\mathcal{A}(g, g^a, g^b) = g^{ab} \mid a, b \in \mathbb{Z}_p^*\right].$$

The *CDH Assumption* is that, for any probabilistic polynomial time algorithm $\mathcal{A}$, the advantage $\mathsf{Adv}^{\mathsf{CDH}}_{\mathcal{A}}(\kappa)$ is negligibly small.

Definition 7. Decisional Diffie-Hellman Problem (DDH) - Given $(g, g^a, g^b, h) \leftarrow \mathbb{G}^4$ for unknown $a, b \xleftarrow{\$} \mathbb{Z}_p^*$, where $\mathbb{G}$ is a cyclic prime order multiplicative group of order p with g as a generator, the DDH problem in $\mathbb{G}$ is to determine whether $h \stackrel{?}{=} g^{ab}$ or a random group element.

The advantage of any probabilistic polynomial time algorithm $\mathcal{A}$ in solving the DDH problem in $\mathbb{G}$ is defined as

$$\mathsf{Adv}^{\mathsf{DDH}}_{\mathcal{A}}(\kappa) = |\mathsf{Pr}\left[\mathcal{A}(g, g^a, g^b, g^{ab}) = 1\right] - \mathsf{Pr}\left[\mathcal{A}(g, g^a, g^b, h) = 1\right] \mid a, b \in \mathbb{Z}_q^*|$$

The *DDH Assumption* is that, for any probabilistic polynomial time algorithm $\mathcal{A}$, the advantage $\mathsf{Adv}^{\mathsf{DDH}}_{\mathcal{A}}(\kappa)$ is negligibly small.

4.6 True Simulation Extractable Non-interactive Zero Knowledge Argument System

In this section we recall the notion of (same-string) *true-simulation extractable non-interactive zero knowledge argument* (tSE-NIZK) first introduced in [DHLAW10b]. This notion is similar to the notion of simulation-sound extractable NIZKs [Gro06] with the difference that the adversary has oracle access to *simulated* proofs only for *true* statements., in contrast to any arbitrary statement as in simulation-sound extractable NIZK proof system.

Let $\Re$ be an efficiently computable binary relation. For pairs $(y, x) \in \Re$, we call y the statement and x the witness. Let $L = \{y \mid \exists\, x \text{ s.t. } (y, x) \in \Re\}$ be the language consisting of statements in $\Re$. A NIZK argument system consists of three algorithms $(\mathsf{CRSGen}, \mathsf{Prove}, \mathsf{Verify})$ such that: (1) Algorithm CRSGen takes as input 1^κ and generates a common reference string (CRS) crs, a trapdoor TK and an extraction key EK; (2) Algorithm Prove takes as input the statement-witness pair (y, x) and crs and outputs an argument π such that $\Re(y, x) \in 1$; (3) Algorithm Verify takes as input crs, a statement y, and a purported argument π and outputs 1 if the argument is acceptable and 0 otherwise. We require the following properties to hold:

1. **Perfect Completeness:** For all $(y, x) \in \Re$, $(\mathsf{crs}, \mathsf{TK}) \leftarrow \mathsf{CRSGen}(1^\kappa)$, if $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, (y, x))$, then $\mathsf{Verify}(\mathsf{crs}, x, \pi) = 1$.
2. **Soundness:** For all malicious provers $\mathcal{P}^*$ we have,

   $\mathsf{Pr}\big[\mathsf{Verify}(\mathsf{crs}, y, \pi^*) = 1,\ y \notin \Re \mid (\mathsf{crs}, \mathsf{TK}) \leftarrow \mathsf{CRSGen}(1^\kappa), (y, \pi^*) \leftarrow \mathcal{P}^*(1^\kappa, \mathsf{crs})\big] \leq \mathrm{negl}(\kappa)$.
3. **(Composable) Zero-Knowledge:** There exists a PPT simulator Sim such that for all PPT adversaries $\mathcal{A}$ the probability that the experiment below outputs 1 is at most $1/2 + negl(\kappa)$.
   (a) The challenger samples $(\mathsf{crs}, \mathsf{TK}) \leftarrow \mathsf{CRSGen}(1^\kappa)$, gives $(\mathsf{crs}, \mathsf{TK})$ to $\mathcal{A}$.
   (b) The adversary $\mathcal{A}$ chooses $(y, x) \in \Re$ and gives it to the challenger.
   (c) The challenger samples $\pi_0 \leftarrow \mathsf{Prove}(y, x, \mathsf{crs})$, $\pi_1 \leftarrow \mathsf{Sim}(y, \mathsf{TK})$, $b \xleftarrow{\$} \{0, 1\}$, and gives π_b to $\mathcal{A}$.
   (d) The adversary $\mathcal{A}$ outputs a bit b' as guess for b; output 1 if $b' = b$, else output 0.
4. **Strong True-simulation f-Extractability:** We start by defining the simulation oracle $\mathcal{SIM}_{\mathsf{TK}}(.)$. A query to the simulation oracle consists of a statement-witness pair (y, x). The oracle checks if $(y, x) \in \Re$. If true, it outputs a simulated argument $\mathsf{Sim}(\mathsf{TK}, y)$, otherwise it outputs $\perp$. Let f be a fixed efficiently computable function. There exists a PPT algorithm $\mathsf{EXT}(y, \pi, \mathsf{EK})$ such that for all PPT adversaries $\mathcal{P}^*$, we have $\mathsf{Pr}[\mathcal{P}^* \text{ wins}| \leq \mathrm{negl}(\kappa)$ in the following game.
   (a) *The challenger samples* $(\mathsf{crs}, \mathsf{TK}, \mathsf{EK}) \leftarrow \mathsf{CRSGen}(1^\kappa)$*, and gives* crs *to* $\mathcal{P}^*$.
   (b) $\mathcal{P}^{*\mathcal{SIM}_{\mathsf{TK}}(.)}$ *can adaptively access the simulation oracle* $\mathcal{SIM}_{\mathsf{TK}}(.)$ *as defined above.*
   (c) *Finally, the adversary* $\mathcal{P}^*$ *outputs a tuple* (y^*, π^*).
   (d) *The challenger runs* $z^* \leftarrow \mathsf{EXT}(y^*, \pi^*, \mathsf{EK})$
   (e) $\mathcal{P}^*$ *wins if (a)* $(y^*, \pi^*) \neq (y, \pi)$ *for all pairs* (y, π) *returned by the simulation oracle* $\mathcal{SIM}_{\mathsf{TK}}(.)$*; (b)* $\mathsf{Verify}(\mathsf{crs}, y^*, \pi^*) = 1$ *and (c) for all* x' *such that* $f(x') = z^*$*, we have* $\Re(y^*, x') \in 0$*. (,i.e., the adversary* $\mathcal{P}^*$ *wins if the extractor cannot extract a good value* z^* *on at least one valid witness* x'*; i.e.,* $f(x') = z^*$*.)*

5 CPA to CCA-2 transformation in the presence of after-the-fact leakage

In this section we present our generic compiler for transforming a leakage-resilient CPA-secure PKE to leakage-resilient CCA-2 secure PKE in the presence of after-the-fact leakage. We first give our model for after-the-fact CCA-2 secure PKE scheme in Section 5.1, followed by the details of our compiler in Section 5.2.

5.1 CCA-2 security in a split state model

We consider the *bounded split-state leakage* model similar to [HL11]. Here the secret key of the cryptosystem is *split* into two parts, and the adversary can obtain leakage from each of these two parts independently, but not a joint leakage from both the secret key components. Note that the independent leakage assumption may seem to be a strong assumption, since in practice leakage appears to be a global function of the computation's intermediate values, for e.g., the power consumption of a device modeled by Hamming weights. However, it turns out many relevant global leakage functions can in fact be computed based only on local leakages. This holds true for all *affine* leakage functions, which also subsumes the Hamming weight leakage. This was also pointed out in [DF11, HL11]. So the split state model is already powerful enough to capture broad class of practically relevant side-channel attacks. Note that the split-state assumption is necessary for our construction since the starting point of our compiler is the PKE scheme of [HL11], and also there *does not* exist any construction of after-the-fact leakage-resilient CPA secure PKE in the non-split state model as a starting point for the compiler.

Definition 8. (Split state encryption) [HL11]. A 2-split state encryption scheme $\mathcal{E} = (\mathcal{E}.\mathsf{Gen}, \mathcal{E}.\mathsf{Enc}, \mathcal{E}.\mathsf{Dec})$ has the following structure:

- $\mathcal{E}.\mathsf{Gen}(1^\kappa)$: The key generation algorithm comprises of two subroutines namely, $\mathcal{E}.\mathsf{Gen}_1$ and $\mathcal{E}.\mathsf{Gen}_2$. On input the security parameter 1^κ, the key generation subroutine $\mathcal{E}.\mathsf{Gen}_i$ $(i \in \{1,2\})$ generates the public-secret key pair, i.e, $(pk_i, sk_i) \leftarrow \mathcal{E}.\mathsf{Gen}_i(1^\kappa, r_i)$ where $r_i \in \{0,1\}^*$. The public key consists of the pair $pk = (pk_1, pk_2)$ and the secret key consists of the pair $sk = (sk_1, sk_2)$.
- $\mathcal{E}.\mathsf{Enc}_{pk=(pk_1,pk_2)}(m)$: The (randomized) encryption algorithm takes as input a message m and outputs the ciphertext c.
- $\mathcal{E}.\mathsf{Dec}(1^\kappa, c, sk = (sk_1, sk_2))$: The decryption consists of two partial decryption subroutines $\mathcal{E}.\mathsf{Dec}_1$, $\mathcal{E}.\mathsf{Dec}_2$ and a combining subroutine Comb. The decryption subroutine $\mathcal{E}.\mathsf{Dec}_i$ $(i \in \{1,2\})$ takes as input the ciphertext c and the secret key sk_i and outputs the partial decryption t_i, i.e., $t_i \leftarrow \mathcal{E}.\mathsf{Dec}_i(c, sk_i)$. Finally, Comb takes the ciphertext and the pair (t_1, t_2) to recover the plaintext m, i.e., $m \leftarrow \mathsf{Comb}(c, t = (t_1, t_2))$.

We want the usual *correctness* requirement to hold for the 2-split state encryption scheme $\mathcal{E}$, i.e., $\forall (pk_i, sk_i) \leftarrow \mathcal{E}.\mathsf{Gen}_i(1^\kappa)\,(i \in \{1,2\}), \forall m \in \mathcal{M}$, we have, $\mathcal{E}.\mathsf{Dec}(sk = (sk_1, sk_2), c = \mathcal{E}.\mathsf{Enc}_{pk}(m)) = m$.

We now define the notion of CCA-2 security of PKE schemes in the presence of after-the-fact split-state memory leakage.

Definition 9. (CCA-2 security of split state PKE against after-the-fact leakage ($\ell(\kappa)$)-AFL-CCA-2 security)). Let $\kappa \in \mathbb{N}$ be the security parameter and let $\ell_{\mathsf{pre}}(\kappa)$ and $\ell_{\mathsf{post}}(\kappa)$ be the upper bound on the amounts of memory leakage before and after the challenge phase respectively. A 2-split state PKE $\mathcal{E} = (\mathcal{E}.\mathsf{Gen}, \mathcal{E}.\mathsf{Enc}, \mathcal{E}.\mathsf{Dec})$ is resilient to $\ell(\kappa) = \big((\ell_{\mathsf{pre}}(\kappa), (\ell_{\mathsf{post}}(\kappa)\big)$ leakage in the split-state model, if for all PPT adversaries $\mathcal{A}$, the probability that the experiment below outputs 1 is at most $\frac{1}{2} + negl(\kappa)$.

1. **Key Generation:** The challenger chooses $r_1, r_2 \in \{0,1\}^*$ at random and computes $(pk_i, sk_i) \leftarrow \mathcal{E}.\mathsf{Gen}_i(1^\kappa, r_i)$ $(i \in \{1,2\})$ and sends $pk = (pk_1, pk_2)$ to the adversary.
2. **Pre-Challenge Leakage queries:** The adversary makes an arbitrary number of leakage queries $(f^{\mathsf{pre}}_{1,i}, f^{\mathsf{pre}}_{2,i})$ adaptively. Upon receiving the i-th leakage query the challenger sends back $(f^{\mathsf{pre}}_{1,i}(sk_1), f^{\mathsf{pre}}_{2,i}(sk_2))$, provided $\sum_{i=1}^{n(\kappa)} |f^{\mathsf{pre}}_{1,i}(sk_1)| \leq \ell_{\mathsf{pre}}(\kappa)$ and $\sum_{i=1}^{n(\kappa)} |f^{\mathsf{pre}}_{2,i}(sk_2)| \leq \ell_{\mathsf{pre}}(\kappa)$, where $n(\kappa)$ denotes the number of pre-challenge leakage queries made in this phase.
3. **Pre-Challenge Decryption queries:** The adversary $\mathcal{A}$ may ask decryption queries adaptively. The challenger returns the plaintexts m_i corresponding to the queried ciphertexts c_i.
4. **Challenge:** In this phase the challenger chooses $b \xleftarrow{\$} \{0,1\}$ and computes $c^* = \mathcal{E}.\mathsf{Enc}_{pk}(m_b)$ and gives it to $\mathcal{A}$.
5. **Post-Challenge Leakage queries:** The adversary makes an arbitrary number of leakage queries $(f^{\mathsf{post}}_{1,j}, f^{\mathsf{post}}_{2,j})$ adaptively. Upon receiving the j-th leakage query the challenger sends back $(f^{\mathsf{post}}_{1,j}(sk_1), f^{\mathsf{post}}_{2,j}(sk_2))$, provided $\sum_{j=1}^{n'(\kappa)} |f^{\mathsf{post}}_{1,j}(sk_1)| \leq \ell_{\mathsf{post}}(\kappa)$ and $\sum_{j=1}^{n'(\kappa)} |f^{\mathsf{post}}_{2,j}(sk_2)| \leq \ell_{\mathsf{post}}(\kappa)$, where $n'(\kappa)$ denotes the number of post-challenge leakage queries made in this phase.

6. **Post-Challenge Decryption queries:** The adversary may continue querying the decryption oracle adaptively with different ciphertexts c_i with the only restriction that $c_i \neq c^*$.

7. **Guess:** Finally, the adversary outputs a bit b' for a guess of the bit b chosen the challenger. If $b' = b$, output 1, else output 0.

We define the advantage of $\mathcal{A}$ as $\mathsf{Adv}^{\mathsf{AFL\text{-}CCA\text{-}2}}_{\mathcal{A}}(\kappa) = |\mathsf{Pr}[b' = b] - \frac{1}{2}|$.

5.2 The Generic Transformation

In this section, we give the generic transformation from after-the-fact leakage-resilient CPA-secure (AFL-CPA) PKE to after-the-fact leakage-resilient CCA-2 secure (AFL-CCA-2) PKE. The main tool we will be using for our transformation is true-simulation extractable NIZK argument system (tSE-NIZK) as defined as section 4.6.

Let $\mathcal{E} = (\mathcal{E}.\mathsf{Gen}, \mathcal{E}.\mathsf{Enc}, \mathcal{E}.\mathsf{Dec})$ be the $\ell(\kappa) = (\ell_{\mathsf{pre}}(\kappa), \ell_{\mathsf{post}}(\kappa))$-leakage-resilient 2-split state CPA-secure PKE from above, and let $\Pi = (\mathsf{CRSGen}, \mathsf{Prove}, \mathsf{Verify})$ be a one-time, strong f-tSE NIZK argument for the relation

$$\Re_{enc} = \{(m, r), (pk, c) \;\mid\; c = \mathsf{Enc}_{pk}(m; r)\}$$

where $f(m, r) = m$, i.e., the extractor only requires to extract the message m and not the randomness r of encryption. We show how to construct a leakage-resilient CCA-2 secure PKE $\mathcal{E}' = (\mathcal{E}'.\mathsf{Gen}, \mathcal{E}'.\mathsf{Enc}, \mathcal{E}'.\mathsf{Dec})$ secure against after-the-fact leakage.

1. $\mathcal{E}'.\mathsf{Gen}(1^\kappa)$: Output $\hat{pk} = (pk, \mathsf{crs})$, $\hat{sk} = sk$, where $(pk, sk) \leftarrow \mathcal{E}.\mathsf{Gen}$, and $(\mathsf{crs}, \mathsf{TK}, \mathsf{EK}) \leftarrow \mathsf{CRSGen}(1^\kappa)$.

2. $\mathcal{E}'.\mathsf{Enc}(\hat{pk}, m)$: Output the ciphertext $C = (c, \pi)$, where $c \leftarrow \mathcal{E}.\mathsf{Enc}_{pk}(m; r)$ and $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, (pk, c), (m, r))$.

3. $\mathcal{E}'.\mathsf{Dec}(sk, C)$: Parse $C = (c, \pi)$. If $\mathsf{Verify}(\mathsf{crs}, (pk, C), \pi) = 1$, output $\mathcal{E}.\mathsf{Dec}(sk, c)$, else output $\perp$.

Theorem 1. *Assume that $\mathcal{E}$ is a $\ell(\kappa) = \big(\ell_{\mathsf{pre}}(\kappa), \ell_{\mathsf{post}}(\kappa)\big)$-AFL-LR-CPA-secure PKE and Π is a one-time strong f-tSE NIZK argument system for the relation $\Re_{enc}$ where, for any witness (m, r), we define $f(m, r) = m$. Then the scheme $\mathcal{E}'$ defined above is $\big(\ell_{\mathsf{pre}}(\kappa), \ell_{\mathsf{post}}(\kappa)\big)$-AFL-LR-CCA-2-secure PKE.*

Proof. The proof of this theorem follows via series of games argument. All the games are variant of the original $\ell(\kappa)$-AFL-CCA-2 security game. These games differ in how the challenger ciphertext $C^* = (c^*, \pi^*)$ is generated and the answers to the decryption oracle queries are simulated.

Game 0. This is the original $\ell(\kappa)$-AFL-LR-CCA-2 security game. The challenger correctly generates the public key $\hat{pk} = (pk, \mathsf{crs})$ as in the key generation algorithm and gives it to the adversary. When the adversary submits two challenge messages m_0, m_1, the challenger computes the challenge ciphertext correctly as in the construction. The answers to the decryption queries are also answered correctly. In other words the challenger does the following:

1. Compute $(pk, sk) \leftarrow \mathcal{E}.\mathsf{Gen}$, and $(\mathsf{crs}, \mathsf{TK}, \mathsf{EK}) \leftarrow \mathsf{CRSGen}(1^\kappa)$. Sets the secret key as $\hat{sk} = sk$ and gives the public key $\hat{pk} = (pk, \mathsf{crs})$ to the adversary $\mathcal{A}$.

2. Chooses bit $b \xleftarrow{\$} \{0, 1\}$ and compute $c^* \leftarrow \mathcal{E}.\mathsf{Enc}_{pk}(m_b; r)$, $\pi^* \leftarrow \mathsf{Prove}_{\mathsf{crs}}((pk, c^*), (m_b, r))$, and output $C^* = (c^*, \pi^*)$ as challenger ciphertext. Finally give C^* to the adversary.

3. The pre- and post-challenge decryption queries (C_i, π_i) made by $\mathcal{A}$ are answered using $\mathcal{E}'.\mathsf{Dec}(sk, C'_i)$.
4. When the adversary asks pre- and post-challenge leakage queries $(f^{\mathsf{pre}}_{1,i}, f^{\mathsf{pre}}_{2,i})$ and $(f^{\mathsf{post}}_{1,i}, f^{\mathsf{post}}_{2,i})$, the challenger returns $(f^{\mathsf{pre}}_{1,i}(sk_1), f^{\mathsf{pre}}_{2,i}(sk_2))$ and $(f^{\mathsf{post}}_{1,i}(sk_1), f^{\mathsf{post}}_{2,i}(sk_2))$ respectively, provided the leakage does not exceed ℓ_{pre} and ℓ_{post} on both the coordinates in the pre- and post-challenge leakage phase.

Game 1. In this game the CRS for Π is generated along with a simulation trapdoor TK and the argument π^* in the challenge ciphertext is simulated using the zero-knowledge simulator $\mathcal{SIM}_{\mathsf{TK}}$. The pre- and post-challenge decryption and leakage queries are answered as in **Game 0**. In other words the challenger does the following:

1. Compute $(pk, sk) \leftarrow \mathcal{E}.\mathsf{Gen}$, and $(\mathsf{crs}, \mathsf{TK}, \mathsf{EK}) \leftarrow \mathsf{CRSGen}(1^\kappa)$. Sets the secret key as $\hat{sk} = sk$ and gives the public key $\hat{pk} = (pk, \mathsf{crs})$ to the adversary $\mathcal{A}$.
2. Chooses bit $b \xleftarrow{\$} \{0,1\}$ and compute $c^* \leftarrow \mathcal{E}.\mathsf{Enc}_{pk}(m_b; r)$, $\pi^* \leftarrow \mathcal{SIM}_{\mathsf{TK}}(pk, c^*)$, and output $C^* = (c^*, \pi^*)$ as challenger ciphertext. Finally give C^* to the adversary.

The decryption and leakage queries are handled in a similar manner as Game 0. The *indistinguishability* of **Game 0** and **Game 1** follows from the *NIZK* property of the tSE-NIZK argument system Π.

Game 2. In this game the CRS for Π is generated together with a simulation trapdoor TK and an extraction trapdoor EK. The challenge ciphertext is simulated using the zero-knowledge simulator similarly as **Game 1**. However the decryption queries are handled in a different manner. The decryption queries $C_i = (c_i, \pi_i)$ are answered by running the extractor on the arguments π_i to extract $f(m_i, r_i) = m_i$. In other words the challenger does the following:

1. Compute $(pk, sk) \leftarrow \mathcal{E}.\mathsf{Gen}$, and $(\mathsf{crs}, \mathsf{TK}, \mathsf{EK}) \leftarrow \mathsf{CRSGen}(1^\kappa)$. Sets the secret key as $\hat{sk} = sk$ and gives the public key $\hat{pk} = (pk, \mathsf{crs})$ to the adversary $\mathcal{A}$.
2. Chooses bit $b \xleftarrow{\$} \{0,1\}$ and compute $c^* \leftarrow \mathcal{E}.\mathsf{Enc}_{pk}(m_b; r)$, $\pi^* \leftarrow \mathcal{SIM}_{\mathsf{TK}}(pk, c^*)$, and output $C^* = (c^*, \pi^*)$ as challenger ciphertext. Finally give C^* to the adversary.
3. When the adversary queries to the decryption oracle using $C_i = (c_i, \pi_i)$, the challenger runs $\mathsf{EXT}((\hat{pk}, c_i), \pi_i, \mathsf{TK})$ to extract the message m_i.

The answers to the leakage queries are answered similarly as in Game 0. The *indistinguishability* of **Game 1** and **Game 2** follow from the *strong one-time true-simulation extractability* property of the tSE-NIZK argument system Π. The adversary $\mathcal{A}$ sees *only one* simulated proof of a true statement, namely, the argument π^* in the challenge ciphertext $C^* = (c^*, \pi^*)$. Therefore by the strong one-time true simulation extractability property of Π, $\mathcal{A}$ cannot produce any new statement-argument pair $(c_i, \pi_i) \neq (c^*, \pi^*)$ for which the argument π_i verifies but the extractor fails to extract the correct m_i.

Game 3. This is the final game. In which game the challenger changes the way in which the challenge ciphertext c^* is generated. Instead of encrypting the message m_b, the challenger produces the challenge ciphertext as an encryption of 0 (or any fixed message in the message space), i.e., the challenger computes the challenge ciphertext as $C^* = (c^*, \pi^*)$, where $c^* \leftarrow \mathcal{E}.\mathsf{Enc}_{pk}(0; r)$ and $\pi^* \leftarrow \mathcal{SIM}_{\mathsf{TK}}(pk, c^*)$. The decryption queries are still answered using the extraction trapdoor as in Game 2. The leakage queries are answered using the leakage oracle of the AFL-LR-CPA secure scheme $\mathcal{E}$. We show that **Game 2** and **Game 3** are indistinguishable.

Claim. Game 2 and Game 3 are indistinguishable by the $\ell(\kappa) = (\ell_{\mathsf{pre}}(\kappa), \ell_{\mathsf{post}}(\kappa))$-AFL-CPA *security of the PKE scheme* $\mathcal{E}$.

Proof. If **Game 2** and **Game 3** can be distinguished we can build an adversary $\mathcal{A}'$ against the $\ell(\kappa)$-AFL-CPA secure PKE $\mathcal{E}$. The adversary $\mathcal{A}'$ simulates the execution environment for $\mathcal{A}$ as follows:

1. $\mathcal{A}'$ receives the public key pk^* and computes $(\mathsf{crs}, \mathsf{TK}, \mathsf{EK}) \leftarrow \mathsf{CRSGen}(1^\kappa)$. It then sends the public key $\hat{pk} = (pk^*, \mathsf{crs})$ to $\mathcal{A}$.
2. When $\mathcal{A}$ makes pre- and post-challenge decryption oracle queries $C_i = (c_i, \pi_i)$, $\mathcal{A}'$ uses the extraction trapdoor of the tSE-NIZK argument system Π to simulate the response to these queries, i.e., it computes $\mathsf{EXT}((\hat{pk}, c_i), \pi_i, \mathsf{TK})$ to extract m_i. As already argued above (indistinguishability of Game 1 and Game 2) this properly simulates the decryption oracle responses (except with negligible probability) by the strong one-time true simulation extractability property of Π.
3. When $\mathcal{A}$ makes pre- and post-challenge leakages queries, $\mathcal{A}'$ forwards them to the leakage oracle of the challenger of the $\ell(\kappa)$-AFL-LR-CPA-secure PKE scheme $\mathcal{E}$, and returns back the response to $\mathcal{A}$.
4. When $\mathcal{A}$ makes the challenge query with two messages m_0 and m_1, $\mathcal{A}'$ forwards them to its challenger. It gets back the ciphertext c^* and computes $\pi^* \leftarrow \mathcal{SIM}_{\mathsf{TK}}(pk, c^*)$. Finally, it returns $C^* = (c^*, \pi^*)$ to $\mathcal{A}$.
5. When $\mathcal{A}$ output a bit b', $\mathcal{A}'$ also outputs the same bit b'.

With all but negligible probability, the above represents a proper simulation of the environment for $\mathcal{A}$ by $\mathcal{A}'$. Thus if the advantage of $\mathcal{A}$ is negligible, the advantage of $\mathcal{A}'$ is also negligible. This proves the above claim.

The above claim shows that Game 2 and Game 3 are indistinguishable. Now, note that Game 3 is completely independent of the bit b, and hence the advantage of any adversary in Game 3 is exactly 0. So, by the indistinguishability of the Games 0-3, the advantage of any adversary in Game 0 is $negl(\kappa)$. This concludes the proof of the above theorem. □

Remark 1. Note that the leakage tolerance of the AFL-LR-CCA-2 secure PKE $\mathcal{E}'$ is *exactly same* as the leakage tolerance of the underlying AFL-LR-CPA-secure PKE $\mathcal{E}$. This is because in Games 2 and 3 the decryption secret key sk is never used for answering the decryption oracle queries. Instead, the extractor trapdoor of the tSE-NIZK is used for simulating the decryption queries and also it is never used in the real scheme. In other words, the decryption oracle responses do not leak any useful information to the adversary, and the leakage that happens from the construction due to adaptive access of the leakage oracle by the adversary is taken care of by the underlying AFL-CPA-secure PKE. This essentially allows us to tolerate the same amount of leakage, namely $\ell = (\ell_{\mathsf{pre}}, \ell_{\mathsf{post}})$ bits of leakage as the underlying CPA-secure scheme $\mathcal{E}$.

6 Compiler for After-the-Fact leakage-resilient AKE protocols

We give a generic framework for designing a bounded after-the-fact leakage eCK-secure (BAFL-eCK) AKE protocol using an arbitrary AFL-LR-CCA-2 secure public key encryption scheme, an arbitrary pseudo-random function and a leakage-resilient storage scheme as defined in Section 4.3. We prove the security of our protocol in the *standard* model, assuming the hardness of the DDH problem.

6.1 The Bounded After-the-fact Leakage-eCK (BAFL-eCK) Model

The BAFL-eCK model [ASB14] can be seen as a (bounded) leakage analogue of the eCK model [LLM07]. Here, the secret key of the cryptosystem is split into n parts and it is assumed that the adversary gets independent leakage from each split. This is modeled by allowing the adversary to send a tuple leakage function $\mathbf{f} = (f_1, \cdots, f_n)$, where the size n of the tuple is protocol-specific (for our purpose $n = 2$, since we consider 2-split state model). The total amount of leakage from each split of the secret key is bounded by the

leakage parameters. In particular, if the total leakage bound on the i-th split of the secret key is λ_i, then the condition $\sum |f_i(sk_i)| \leq \lambda_i$ should hold, where sk_i denote the i^{th} split of the secret key sk. In the BAFL-eCK model it is also assumed that leakage happens as a result of computation following the "Only Computation leaks" (OCLI) axiom [MR04].

Execution environment. In the execution environment, we fix a set of n honest parties $\mathcal{U} = \{U_1, \cdots, U_n\}_{n \in \mathbb{N}}$, each modeled by a probabilistic polynomial time Turing machine (PPTM). We assume the identities are unique and also lexicographically indexed via variable $i \in [n]$. We further assume that the parties $U_1, \ldots, U_n$ are connected over point-to-point links over which the messages can be exchanged between them. The term *principal* is used to identify a party involved in a protocol instance. Each party U_i, where $i \in [n]$ is associated with long-term key pairs (pk_{U_i}, sk_{U_i}). The term *session* is used to identify a protocol instance at a principal. Each honest party U_i can sequentially and also concurrently execute multiple sessions. The oracle $\Pi^s_{U,V}$ represents the s-th session at the owner principal U with intended partner principal V. The party who sends the first message is called the *initiator* of the session and the party which responds to the first protocol message of a session is called the *responder* of the session.

Adversarial Model. An adversary $\mathcal{A}$ is a PPT Turing Machine taking as input the security parameter 1^κ and the public information (e.g. generic description of above environment). It has full control on the communication network , i.e, it can insert, delete, drop, alter, schedule messages in transit accordingly. It may interact with these oracles by issuing the following queries.

1. $\mathsf{Send}(U, V, s, m, \mathbf{f})$: When this query is issued the oracle $\Pi^s_{U,V}$ computes computes the next protocol message according to the protocol specification and sends it to the adversary $\mathcal{A}$. In addition the adversary also gets the output of the tuple leakage function $\mathbf{f}(sk_U)$ following the OCLI framework as mentioned above. The adversary can also activate a new session with blank m and $\mathbf{f}$.
2. $\mathsf{SessionKeyReveal}(U, V, s)$: The session key of the oracle $\Pi^s_{U,V}$ is given $\mathcal{A}$, provided the session is *complete*, else the session key is not generated yet and return $\perp$ in that case.
3. $\mathsf{EphemeralKeyReveal}(U, V, s)$: The ephemeral secret key of the oracle $\Pi^s_{U,V}$ is given $\mathcal{A}$.
4. $\mathsf{Corrupt}(U)$: The long-term secret key sk_U of principle U is given to $\mathcal{A}$.
5. $\mathsf{Test}(U, V, s)$: If the oracle $\Pi^s_{U,V}$ is complete, the challenger chooses a fair coin $b \xleftarrow{\$} \{0, 1\}$. If $b = 0$, it samples a random element K_0 from the session key space $\mathcal{K}$; if $b = 1$, the actual session key K_1 is returned to $\mathcal{A}$. Finally, the key K_b is returned. The Test query is allowed only once and it captures the notion of semantic security for key exchange protocols.

Remark 2. It may seem paradoxical to consider both leakage and corrupt queries at the same time. However, as argued in [ASB14]*, in case of key compromise impersonation (KCI) attacks the adversary gets the secret key of the owner of the cryptosystem before the activation of the test session and he/she has to impersonate other parties to the owner. In BAFL-eCK model, the adversary additionally gets leakage from the partner of the test session also.*

We now define the freshness condition of a session in the BAFL-eCK model.

Definition 10. (λ-BAFL-eCK-freshness). Let the vector $\lambda = (\lambda_1, \cdots, \lambda_n)$ denote the leakage bound on the secret key of the cryptosystem, i.e., λ_i represents the leakage bound on the i-th split sk_i of the secret key sk. Then the oracle π^s_i is said to be $fresh$ if none of the following holds: An oracle $\Pi^s_{U,V}$ is said to be λ-BAFL-eCK-fresh if and only if:

– The oracle $\Pi^s_{U,V}$ or its partner $\Pi^{s'}_{V,U}$ (if it exists) has not been asked $\mathsf{SessionKeyReveal}$ query.

– The partner $\Pi^{s'}_{V,U}$ exists and the following combinations are not asked:

  a. $\mathsf{Corrupt}(U)$ and $\mathsf{EphemeralKeyReveal}(U, V, s)$.
  b. $\mathsf{Corrupt}(V)$ and $\mathsf{EphemeralKeyReveal}(V, U, s')$.

– The partner $\Pi^{s'}_{V,U}$ does not exist and the following combinations are not asked:

  a. $\mathsf{Corrupt}(V)$.
  b. $\mathsf{Corrupt}(U)$ and $\mathsf{EphemeralKeyReveal}(U, V, s)$.

– For all $\mathsf{Send}(., U, ., ., \mathbf{f})$ queries, $\sum |f_i(sk_{U_i})| \leq \lambda_i$.
– For all $\mathsf{Send}(., V, ., ., \mathbf{f})$ queries, $\sum |f_i(sk_{V_i})| \leq \lambda_i$.

Definition 11. (BAFL-eCK security game). Security of a key exchange protocol in the BAFL-eCK model is defined using the following security game, played between between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

1. At the beginning of the game, the challenger $\mathcal{C}$ implements the collection of oracles $\Pi^k_{U_i,U_j}$, where $i, j \in [n]$ and $k \in [\ell]$, where ℓ is the maximum no. of sessions that can be executed by a party simultaneously. It generates n long-term key pairs (pk_{U_i}, sk_{U_i}) for all honest parties U_i for $i \in [n]$. $\mathcal{C}$ gives adversary $\mathcal{A}$ all identities and public keys $\{(U_1, pk_{U_1}), \cdots, (U_n, pk_{U_n})\}$ as input.
2. $\mathcal{A}$ may issue polynomial number of aforementioned queries adaptively, namely $\mathcal{A}$ makes Send, $\mathsf{EphemeralKeyReveal}$, $\mathsf{Corrupt}$ and $\mathsf{SessionKeyReveal}$.
3. At some point, $\mathcal{A}$ may issue a Test query during the game only once.
4. $\mathcal{A}$ may continue to issue Send, $\mathsf{EphemeralKeyReveal}$, $\mathsf{Corrupt}$ and $\mathsf{SessionKeyReveal}$ queries adaptively, provided the λ-BAFL-eCK freshness conditions are not violated.
5. At the end of the game, the $\mathcal{A}$ may terminate with returning a bit b' as its guess for b of Test query. Return 1, if $b' = b$, otherwise return 0.

Definition 12. *(*BAFL-eCK *security). A protocol* P *is said to be BAFL-eCK-secure if there is no PPT algorithm* $\mathcal{A}$ *that can win the BAFL-eCK security game with non-negligible advantage. The advantage of an adversary* $\mathcal{A}$ *is defined as* $\mathsf{Adv}^{\text{BAFL-eCK}}_{\mathsf{P}}(\mathcal{A}) = |\mathsf{Pr}[b' = b] - \frac{1}{2}|$.

6.2 Generic BAFL-eCK secure AKE protocol in standard model

In this section we present a generic construction of BAFL-eCK secure key exchange protocol P using an arbitrary AFL-LR-CCA-2 secure PKE scheme, a LRS encoding scheme and an arbitrary pseudo-random function. We then prove the security of our AKE protocol in the standard model assuming the security of the above primitives and the hardness of the DDH problem. Suppose κ is the security parameter. Let $\mathbb{G}$ denotes a cyclic multiplicative group of prime order p generated by g. The main building blocks used in our construction of the AKE protocol are as follows:

– $\ell(\kappa)$-after-the-fact leakage-resilient 2-split state CCA-2 (AFL-CCA-2) secure PKE $\mathcal{E} = (\mathcal{E}.\mathsf{Gen},$
$\mathcal{E}.\mathsf{Enc}, \mathcal{E}.\mathsf{Dec})$ with key space $\mathcal{K}$, message space $\mathcal{M}$.
– $\Lambda^{n,1}_{\mathbb{Z}^*_p} = (\mathsf{Encode}^{n,1}_{\mathbb{Z}^*_p}(s), \mathsf{Decode}^{n,1}_{\mathbb{Z}^*_p}(s_L, s_R))$ be a (λ_S, ϵ_1) leakage-resilient storage scheme
– $F : \mathbb{G} \times \{0,1\}^* \to \mathcal{SK}$ be a $(\epsilon_{\mathsf{prf}}, s_{\mathsf{prf}}, q_{\mathsf{prf}})$-secure PRF family

Overview of our Construction. We denote the two protocol participants as U_A and U_B. We assume that U_A is the initiator and U_B is the responder. Alawatugoda [Ala15b] gave a generic transformation of a CCA2-secure PKE scheme to an eCK-secure key exchange protocol in the standard model. Our compiler can be viewed as leakage-resilient implementation of the compiler of Alawatugoda [Ala15b]. We denote the public-secret key pair of parties U_A and U_B as (pk_{U_A}, sk_{U_A}) and (pk_{U_B}, sk_{U_B}) respectively. We denote (a, A) and (b, B) as the long-term Diffie-Hellman (DH) secret and public keys of U_A and U_B respectively. We also denote (esk_A, epk_A) and (esk_B, epk_B) as the ephemeral secret and public keys of U_A and U_B respectively.

The main idea of the construction of [Ala15b] is that it computes epk_A and epk_B as DH public keys and esk_A and esk_B as DH secret keys. It then encrypts the epk_A and epk_B using a CCA-2 secure PKE. The other party who has the secret key can successfully recover epk_A and epk_B respectively. In the session key generation generation phase both the parties perform *two* DH session key derivation. The first session key derivation involves the ephemeral public and secret keys of both the parties, whereas the second DH session key generation involves the DH long-term keys of both the parties. Finally both the parties use a PRF to derive the final session key.

Our generic AKE construction also follows this simple design strategy. However, the adversarial model is *stronger* than the eCK model. This is because the BAFL-eCK model trivially implies the eCK model, but the other way is not true. In particular, apart from all the information the adversary gets in the eCK model, additionally it also obtains leakage from the secret key of the parties, i.e. both the secret key of the cryptosystem as well as the DH long-term secret key of parties. The leakage that happens from the secret key of the CCA-2 secure PKE can be countered by using a AFL-CCA-2 secure encryption scheme. However, apart from this the leakage from the DH long-term secret keys also needs to be accounted. For this, we use *leakage-resilient storage* (LRS) scheme. However, directly using the LRS scheme does not work for our purpose since the secret values in our case are exponents of DH public keys. So we need a way to perform exponentiation in a leakage-resilient fashion. The possibility of leakage-resilient exponentiation was mentioned in [Ala15b]. However, no formal derivation was present. Here, we show the leakage-resilient exponentiation operation explicitly and show its correctness.

Leakage-resilient exponentiation. We use the LRS scheme to perform secure exponentiation in the presence of leakage. Suppose that x is the exponent (DH secret key) and we need to compute the DH public key $X = g^x$. We first encode x using the LRS scheme as $(x_L, x_R) \leftarrow \mathsf{Encode}^{n,1}_{\mathbb{Z}^*_p}(x)$, where $x_L \xleftarrow{\$} (\mathbb{Z}^*_p)^n \setminus \{(0^n)\}$, $x_R \leftarrow (\mathbb{Z}^*_p)^{n\times 1} \setminus \{(0^{n\times 1})\}$ and $x_L \odot x_R = x$. To compute $X = g^x$, we use the two encodings (x_L, x_R), and finally erase x from memory. The computation of exponentiations is also split into two parts. More precisely, we first compute $X' = g^{x_L} = (g^{x_{L_1}}, g^{x_{L_2}}, \cdots, g^{x_{L_n}})$, and then compute $X = (X')^{x_R} = \prod_{i=1}^{n} g^{x_{L_i} \cdot x_{R_i}} = g^{\sum_{i=1}^{n} x_{L_i} \cdot x_{R_i}} = g^x$.

In our AKE protocol the party party U_A ($k \in \{A, B\}$) chooses $a_L \xleftarrow{\$} (\mathbb{Z}^*_p)^n \setminus \{(0^n)\}$, $a_R \xleftarrow{\$} (\mathbb{Z}^*_p)^{n\times 1} \setminus \{(0^{n\times 1})\}$ and the value of the ephemeral DH exponent a is implicitly set as $a_L \odot a_R$. Party U_B also performs similar operation. In the key generation process if we first choose the long-term DH secret key a it must be securely erased from memory after getting the encoded values a_L and a_R of a. However, in practice secure erasure may not be possible always and some traces of the secret key may be leaked to the adversary. In order to avoid such a vulnerability, two values a_L and a_R are picked at random and we use them as the encodings of the long-term DH exponent a. In this way we refrain from using the long-term DH secret key a directly. Note that, this approach is identical to first picking a random element $a \in \mathbb{Z}^*_p$ and then encoding it to obtain a_L and a_R. Thus our approach

avoids the vulnerability of exposing the secret DH exponent and hence avoid leaking directly from the exponents a and b. Since the value a is not available to the adversary, it can get only bounded and independent leakage (under split-state assumption) from a_L and a_R respectively. We can then use the security of the LRS scheme from to argue security of our AKE protocol.

Thus combined with the security of the AFL-CCA-2 secure encryption, LRS scheme and security of DH key exchange (DDH assumption),we obtain a BAFL-eCK-secure AKE protocol in standard model. The details of the protocol is presented in Table 1.

U_A		U_B
Key Generation		
Public parameters: $(\mathbb{G}, p, \langle g \rangle)$		
$a_L \xleftarrow{\$} (\mathbb{Z}_p^*)^n \setminus \{(0^n)\}$,		$b_L \xleftarrow{\$} (\mathbb{Z}_p^*)^n \setminus \{(0^n)\}$,
$a_R \xleftarrow{\$} (\mathbb{Z}_p^*)^{n\times 1} \setminus \{(0^{n\times 1})\}$		$b_R \xleftarrow{\$} (\mathbb{Z}_p^*)^{n\times 1} \setminus \{(0^{n\times 1})\}$
$A' = g^{a_L}$, $A = (A')^{a_R} = g^{a_L \cdot a_R} = g^a$		$B' = g^{b_L}$, $B = (B')^{b_R} = g^{b_L \cdot b_R} = g^b$
$(pk_{U_A}, sk_{U_A}) \leftarrow \mathcal{E}.\mathsf{Gen}(1^\kappa)$,		$(pk_{U_B}, sk_{U_B}) \leftarrow \mathcal{E}.\mathsf{Gen}(1^\kappa)$
Session Execution		
$esk_A \xleftarrow{\$} \mathbb{Z}_p^*$, $epk_A \leftarrow g^{esk_A}$		$esk_B \xleftarrow{\$} \mathbb{Z}_p^*$, $epk_B \leftarrow g^{esk_B}$
$C_A \leftarrow \mathcal{E}.\mathsf{Enc}_{pk_{U_B}}(epk_A)$		$C_B \leftarrow \mathcal{E}.\mathsf{Enc}_{pk_{U_A}}(epk_B)$
	$\xrightarrow{U_A, U_B, C_A}$	
	$\xleftarrow{U_B, U_A, C_B}$	
Session Key Generation		
Set $\mathsf{sid} = (U_A \| U_B \| C_A \| C_B)$		Set $\mathsf{sid} = (U_A \| U_B \| C_A \| C_B)$
$epk_B \leftarrow \mathcal{E}.\mathsf{Dec}_{sk_{U_A}}(C_B)$,		$epk_A \leftarrow \mathcal{E}.\mathsf{Dec}_{sk_{U_B}}(C_A)$,
$Z'_{A_1} = (B)^{a_L}$, $Z_{A_1} = (Z'_{A_1})^{a_R}$,		$Z'_{B_1} = (A)^{b_L}$, $Z_{B_1} = (Z'_{B_1})^{b_R}$,
$Z_{A_2} = epk_B^{esk_A}$,		$Z_{B_2} = epk_A^{esk_B}$,
$SK = F(Z_{A_1}, \mathsf{sid}) \oplus F(Z_{A_2}, \mathsf{sid})$		$SK = F(Z_{B_1}, \mathsf{sid}) \oplus F(Z_{B_2}, \mathsf{sid})$

Table 1. Proposed BAFL-eCK secure AKE protocol P

Correctness: The correctness of the protocol is easy to verify. It is enough to show that $Z_{A_1} = Z_{B_1}$ and $Z_{A_2} = Z_{B_2}$. The correctness of the decrypted values epk_B and epk_A at both the parties U_A and U_B respectively follow from the correctness of the AFL-CCA-2 secure PKE scheme.

We have $Z_{A_1} = (Z'_{A_1})^{a_R} = ((B)^{a_L})^{a_R} = B^{a_L \cdot a_R} = B^a = (g^b)^a = g^{ab}$.
Similarly, $Z_{B_1} = (Z'_{B_1})^{b_R} = ((A)^{b_L})^{b_R} = A^{b_L \cdot b_R} = A^b = (g^a)^b = g^{ab} = Z_{A_1}$.
The value $Z_{A_2} = epk_B^{esk_A} = (g^{esk_B})^{esk_A} = (g^{esk_A})^{esk_B} = epk_A^{esk_B} = Z_{B_2}$.

6.3 Security proof

In this section we proof the following theorem:

Theorem 2. *If $\mathcal{E} = (\mathcal{E}.\mathsf{Gen}, \mathcal{E}.\mathsf{Enc}, \mathcal{E}.\mathsf{Dec})$ is a $\ell(\kappa)$-AFL-CCA-2-secure PKE, $\Lambda^{n,1}_{\mathbb{Z}_p^*}$ be a $(\lambda_\Lambda, \epsilon_1)$ leakage-resilient storage scheme, F is a $(\epsilon_{\mathsf{prf}}, s_{\mathsf{prf}}, q_{\mathsf{prf}})$ PRF, and the DDH assumption holds in $\mathbb{G}$ of prime order p generated by g, then the above AKE protocol P is $(\ell(\kappa), \lambda_\Lambda)$-BAFL-eCK-secure. In particular,*

$$\mathsf{Adv}^{\text{BAFL-eCK}}_{\mathsf{P}}(\mathcal{A}) \leq n^2\ell^2 \max\Big((2\epsilon_1 + \mathsf{Adv}^{\mathsf{DDH}}_{\mathcal{B}}(\kappa) + \epsilon_{\mathsf{prf}}), (\mathsf{Adv}^{\mathsf{AFL\text{-}CCA\text{-}2}}_{\mathcal{S}}(\kappa) + \epsilon_{\mathsf{prf}})\Big).$$

where n is the total no. of protocol principles/parties and ℓ is the maximum no. of sessions that can be executed by a party concurrently.

Proof Sketch. Before giving the detailed proof , we first give an overview of our proof here. According to the freshness condition as in Def. 10 we have to consider the following cases and sub-cases:

1. A partner to the test session *exists*.
   (a) Adversary corrupts both the owner and the partner principals to the test session.
   (b) Adversary corrupts neither the owner nor the partner principal to the test session.
   (c) Adversary corrupts the owner to the test session, but does not corrupt the partner to the test session.
   (d) Adversary corrupts the partner to the test session, but does not corrupt the owner to the test session.
2. A partner to the test session *does not exist*.
   (a) Adversary corrupts the owner to the test session.
   (b) Adversary does not corrupt the owner to the test session.

Case 1(a). In this case, the adversary corrupts both the owner and the peer to the test session. So the adversary knows both the long-term Diffie-Hellman (DH) keys a and b of the parties U_A and U_B respectively. Besides, it also learns the secret keys of the AFL-CCA-2 secure encryption scheme $\mathcal{E}$, namely, sk_{U_A} and sk_{U_B} of U_A and U_B respectively. However, the adversary does not learn the ephemeral secret keys of the test session and its matching session. So, the secrecy of the session key lies in the secrecy of the values Z_{A_2} and Z_{B_2}. Note that the adversary can get both epk_A and epk_B by using the encryption secret keys. So the value $Z_{A_2} = Z_{B_2} = g^{esk_A esk_B}$ is hard to distinguish from a random value by the DDH assumption. In our proof we replace it with a random value. Finally, we replace the session key with a random value from the same space. This change is again oblivious to the adversary by the security of the PRF F used for deriving the final session key. Also note that the leakage queries in this case does not make much sense since the adversary already knows the long-term keys of both the principles. In any case, since the challenger has the secret keys of all the parties it can easily simulate the leakage queries.

Case 1(b). In this case the adversary learns the ephemeral secret keys esk_A and esk_B of parties U_A and U_B respectively corresponding to the test session and its matching session. The adversary does not know the long-term DH keys and the secret keys of the AFL-CCA-2 secure PKE scheme $\mathcal{E}$. However, the adversary may obtain leakage from both of them via Send queries according to the BAFL-eCK security model. In this case, the challenger knows neither of the long-term secrets of U_A and U_B, so it cannot simulate the leakage queries by itself. Instead the challenger uses the leakage oracle of the AFL-CCA-2 secure PKE scheme and the LRS scheme to respond to the leakage queries. The LRS scheme $\Lambda^{n,1}_{\mathbb{Z}_p^*}$ ensures that even if the adversary obtains bounded leakage from the two encodings of the secret key independently, it cannot learn any information about the secret value. Given that the adversary does not learn any information about the long-term DH keys, the security of the DH shared key $Z_{A_1} = Z_{B_1}$ ensures the secrecy of the session key. In particular, given the DH public keys g^a and g^b, it is hard to distinguish the value $Z_{A_1} = Z_{B_1} = g^{ab}$ from random value by the DDH assumption. Similar to the above case, we then replace this value with a random value. Finally, we replace the session key with a random value from the same space. This change is again oblivious to the adversary by the security of the PRF F used for deriving the final session key.

Case 1(c). In this case the adversary $\mathcal{A}$ learns the long-term DH key of U_A, i.e., $a = a_L \cdot a_R$ and the secret key sk_{U_A}. For party U_B, the adversary learns the ephemeral secret key esk_B. In his case, the challenger knows sk_{U_A} and a, so it can simulate the leakage queries for party U_A by itself. However, for party U_B, it does not know its long-term secret

keys. It uses the leakage oracle of the AFL-CCA-2 secure PKE scheme and the LRS scheme to answer leakage queries for U_B. Note that, the adversary can compute the value $Z_{A_1} = Z_{B_1} = g^{ab}$. So the secrecy of the session key in this case depends on the security of the values Z_{A_2} and Z_{B_2}. Also note that the adversary knows the value esk_B. However, the ephemeral public key epk_A is protected by the security of C_A, since the long-term secret key sk_B is not revealed to the adversary $\mathcal{A}$.

Case 1(d). The security in this case is similar to that of Case 1(c). Here the adversary $\mathcal{A}$ learns the long-term DH key of U_B, i.e., $b = b_L \cdot b_R$, the secret key sk_{U_B} and the ephemeral secret key esk_A of party U_A. In this case, challenger uses the leakage oracle of the AFL-CCA-2 PKE scheme and the LRS scheme to answer leakage queries for U_A. As before, the adversary can compute the value $Z_{A_1} = Z_{B_1} = g^{ab}$. So the secrecy of the session key in this case depends on the security of the values Z_{A_2} and Z_{B_2}. The adversary also knows the value esk_A. However, the ephemeral public key epk_B is protected by the security of C_B, since the long-term secret key sk_A is not revealed to the adversary $\mathcal{A}$.

In **Case 2** , the partner to the test session *does not* exist. By the freshness condition, the adversary is *not* allowed to corrupt the peer to the test/challenge session. The proof for Case 2(a) is similar to the proof of Case 1(c). The situation that the ephemeral secret key of the partner to the test session is given to $\mathcal{A}$ is the *same* as the case that the test session has no matching session because $\mathcal{A}$ can decide arbitrary ephemeral key. By a similar argument the proof for Case 2(b) is also similar to the analysis for Case 1(d).

Proof. We split the proof of Theorem 2 into two main cases: when the partner to the test session exists, and when it does not. The proof of this theorem will proceed via the *game hopping technique* [Sho04]: define a sequence of games and relate the adversary's advantage of distinguishing each game from the previous game to the advantage of breaking one of the underlying cryptographic primitive. We use $\mathsf{Adv}_{Game_\delta}(\mathcal{M})$ to denote the advantage of the adversary $\mathcal{M}$ in Game δ. We also assume that there are n honest users/parties denoted as $\{U_1, \cdots, U_n\}$, and ℓ is the maximum number of sessions that can be executed by a party simultaneously with other parties.

6.4 Partner to the test session exists

Let $\Pi^s_{U,V}$ be the oracle involved in the Test session and let $\Pi^{s'}_{V,U}$ denote the partner oracle. This case can be further split into the following sub-cases:
(1) The adversary $\mathcal{A}$ issues $\mathsf{Corrupt}(U)$ and $\mathsf{Corrupt}(V)$.
(2) $\mathcal{A}$ issues $\mathsf{EphemeralKeyReveal}(U, V, s)$ and $\mathsf{EphemeralKeyReveal}(V, U, s')$.
(3) $\mathcal{A}$ issues $\mathsf{Corrupt}(U)$ and $\mathsf{EphemeralKeyReveal}(V, U, s')$.
(4) $\mathcal{A}$ issues $\mathsf{EphemeralKeyReveal}(U, V, s)$ and $\mathsf{Corrupt}(V)$.

Analysis of Case B.1.1.

Game 0. This is the original game. When the Test query is asked, the Game 0 challenger chooses a random bit $b \xleftarrow{\$} \{0, 1\}$. If $b = 1$, the real session key is given to $\mathcal{M}$, otherwise a random value chosen from the same session key space is given. So, we have:

$$\mathsf{Adv}_{Game_0}(\mathcal{M}) = \mathsf{Adv}^{\mathsf{BAFL\text{-}eCK}}_{\mathsf{P}}(\mathcal{A}). \tag{1}$$

Game 1. Same as Game 0 with the following exception: before $\mathcal{M}$ begins, two distinct random principals $U^*, V^* \xleftarrow{\$} \{U_1, U_2, \cdots, U_n\}$ are chosen and two random numbers $s, s' \xleftarrow{\$} [\ell]$ are chosen. If $\mathcal{A}$ poses Test query to an oracle except $\Pi^s_{U^*,V^*}$ and the matching session is not the j-th session of party V^*, i.e. $\Pi^{s'}_{V^*,U^*}$, the experiment halts. The probability of Game 1 to be halted due to incorrect choice of the test session is $(1 - \frac{1}{n(n-1)\ell^2})$. Unless the incorrect choice happens, Game 1 is identical to Game 0. Hence,

$$\mathsf{Adv}_{Game_1}(\mathcal{M}) \geq \frac{1}{n^2\ell^2}\ \mathsf{Adv}_{Game_0}(\mathcal{M}). \tag{2}$$

Game 2. Same as Game 1 with the following exception: the challenger randomly chooses $z \xleftarrow{\$} \mathbb{Z}_q^*$ and computes $SK \leftarrow F(\mathsf{CDH}(U,V), \mathsf{sid}) \oplus F(g^z, \mathsf{sid})$. When the adversary asks the $\mathsf{Test}(U^*, V^*, s^*)$ query, Game 2 challenger will answer with SK.

Let U, V be the two long-term Diffie-Hellman public keys of the protocol principals U^*, V^* respectively, such that $U = g^u$, $V = g^v$ and $\mathsf{CDH}(U,V) = g^{uv}$. We construct an algorithm $\mathcal{B}$ against a DDH challenger, using the adversary $\mathcal{A}$ as a sub routine. $\mathcal{B}$ sets all the long-term secret/public key pairs (Diffie-Hellman and encryption key pairs) to all protocol principals. The DDH challenger sends values (g^x, g^y, g^z) such that $z = xy$ or $z \xleftarrow{\$} \mathbb{Z}_p^*$ as the inputs to the algorithm $\mathcal{B}$. Algorithm $\mathcal{B}$ simulates answers to the adversarial queries as follows:

1. $\mathsf{Send}(U, V, s, m, \mathbf{f})$- If U^* is the initiator, $\mathcal{B}$ sends the ciphertext $C_A \leftarrow (pk_{V^*}, epk_A)$ to $\mathcal{A}$ as the first message of the test session. Upon receiving the second protocol message $C_B \leftarrow (pk_{U^*}, epk_B)$ from V^* to U^*, $\mathcal{B}$ computes the session key as $SK \leftarrow F(\mathsf{CDH}(U,V), U^*||C_A||V^*||C_B)$
   $\oplus\, F(g^z, U^*||C_A||V^*||C_B)$.

   If U^* is the responder, upon receiving the first protocol message $C_A \leftarrow (pk_{U^*}, epk_A)$ from V^* to U^*, $\mathcal{B}$ sends $C_B \leftarrow (pk_{V^*}, epk_B)$ to $\mathcal{A}$ as the second protocol message of the test session, and computes the session key as $SK \leftarrow F(\mathsf{CDH}(U,V), V^*||C_A||U^*||C_B) \oplus F(g^z, V^*||C_A||U^*||C_B)$. For all other cases, $\mathcal{B}$ can decrypt incoming protocol messages and execute the protocol normally. Since B has the long-term secret keys of the all the parties, it can compute the output of the leakage functions submitted by $\mathcal{A}$ for both parties U and V and return back the output to $\mathcal{A}$.
2. $\mathsf{SessionKeyReveal}(U, V, s)$- SessionKeyReveal query is not allowed to the target session or the partner of the target session as per the freshness conditions. $\mathcal{B}$ can compute all the other session keys by executing the protocol normally.
3. $\mathsf{EphemeralKeyReveal}(U, V, s)$- The adversary cannot make this query to the owner of the test session and partner to the test session, i.e., when $U = U^*$, $V = V^*$, $s = s^*$ and $U = V^*$, $V = U^*$, $s = s'^*$, the adversary cannot ask this query. For all other EphemeralKeyReveal queries $\mathcal{B}$ can answer correctly, because $\mathcal{B}$ has the ephemeral keys.
4. $\mathsf{Corrupt}(U)$- The adversary can corrupt queries on all the protocol principles including the owner of the test session and partner to the test session. Since $\mathcal{B}$ has the long-term of all the principles, it can answer the Corrupt query for all the principles.
5. $\mathsf{Test}(U, V, s)$- When $U = U^*$, $V = V^*$, $s = s^*$, $\mathcal{B}$ answers with SK as computed above in the Send query. Otherwise it aborts the game.

If $\mathcal{B}$'s input is a Diffie-Hellman triple, simulation constructed by $\mathcal{B}$ is identical to Game 1, otherwise it is identical to Game 2. If $\mathcal{A}$ can distinguish the difference between games, then $\mathcal{B}$ can answer the DDH challenge. Hence,

$$|\mathsf{Adv}_{Game_2}(\mathcal{A}) - \mathsf{Adv}_{Game_1}(\mathcal{A})| \leq \mathsf{Adv}_{\mathcal{B}}^{\mathsf{DDH}}(\kappa). \tag{3}$$

Game 3. Same as Game 2 with the following exception: the challenger randomly chooses $SK \xleftarrow{\$} \mathcal{SK}$ and sends it to the adversary $\mathcal{A}$ as the answer to the $\mathsf{Test}(U^*, V^*, s^*)$ query.

If $\mathcal{A}$ can distinguish the difference between Game 2 and Game 3, then $\mathcal{A}$ can be used as a subroutine of an algorithm $\mathcal{D}$, which is used to distinguish whether the session key value SK is computed using the real PRF with a hidden key, or using a random function. $\mathcal{D}$ simulates the execution environment to $\mathcal{A}$ as follows:

1. $\mathsf{Send}(U, V, s, m)$- If U^* is the initiator, upon receiving the second protocol message C_B from V^* to U^*, it computes the session key as $SK \leftarrow F(\mathsf{CDH}(U, V), U^*||C_A||V^*||C_B) \oplus \text{Oracle}^{\mathsf{PRF}}(U^*||C_A||V^*||C_B)$.

   If U^* is the responder, upon receiving the first protocol message C_A from V^* to U^*, it computes the session key as $SK \leftarrow F(\mathsf{CDH}(U, V), V^*||C_A||U^*||C_B) \oplus \text{Oracle}^{\mathsf{PRF}}(V^*||C_A||U^*||C_B)$. For all the other cases of Send queries, $\mathcal{D}$ can execute the protocol normally. The leakage queries are also handled in a similar manner as defined above.
2. $\mathsf{SessionKeyReveal}(U, V, s)$- $\mathsf{SessionKeyReveal}$ query is not allowed to the target session or the partner of the target session as per the freshness conditions. $\mathcal{D}$ can compute all the other session keys by executing the protocol normally.
3. $\mathsf{EphemeralKeyReveal}(U, V, s)$- The adversary cannot make this query to the owner of the test session and partner to the test session, i.e., when $U = U^*$, $V = V^*$, $s = s^*$ and $U = V^*$, $V = U^*$, $s = s'^*$, the adversary cannot ask this query. For all other EphemeralKeyReveal queries $\mathcal{D}$ can answer correctly, because $\mathcal{D}$ has the ephemeral keys.
4. $\mathsf{Corrupt}(U)$- The adversary can corrupt queries on all the protocol principles including the owner of the test session and partner to the test session. Since $\mathcal{D}$ has the long-term of all the principles, it can answer the $\mathsf{Corrupt}$ query for all the principles.
5. $\mathsf{Test}(U, V, s)$- When $U = U^*$, $V = V^*$, $s = s^*$, $\mathcal{D}$ answers with SK as explained in the Send query. Otherwise it aborts the game.

For $\mathcal{A}$, the simulation by $\mathcal{D}$ is same as the Game 2 if the oracle is the PRF with hidden key. Otherwise, the simulation by $\mathcal{D}$ is same as Game 3. Thus we have,

$$|\mathsf{Adv}_{Game_2}(\mathcal{A}) - \mathsf{Adv}_{Game_1}(\mathcal{A})| \leq \epsilon_{\mathsf{PRF}}. \tag{4}$$

In this game, i.e., in Game 3, the session key in the Test session is *perfectly randomized*. Thus, $\mathcal{A}$ cannot obtain any advantage from Test query. So we have, $\mathsf{Adv}_{Game_3}(\mathcal{A}) = 0$.

Using equations (1)-(4), we get:

$$\mathsf{Adv}_{\mathsf{P}}^{BAFL\text{-}eCK}(\mathcal{A}) \leq n^2 l^2 \max(\mathsf{Adv}_{\mathcal{B}}^{\mathsf{DDH}}(\kappa) + \epsilon_{\mathsf{prf}}).$$

Analysis of Case B.1.2.

Game 0. Same as Game 0 of **Case B.1.1**.

Game 1. Same as Game 1 of **Case B.1.1**.

Game 2. Same as Game 1 with the following exception: The challenger randomly picks $a, b \xleftarrow{\$} \mathbb{Z}_p^*$ and uses encodings of a and b to simulate the adversarial leakage queries $\mathbf{f} = (f_{1j}, f_{2j})$ of the protocol principles U^* and V^*. We construct an algorithm $\mathcal{B}$ against the LRS protocol using the adversary $\mathcal{A}$ as a subroutine.

The $(\lambda_\Lambda, \epsilon_1)$-LRS protocol challenger chooses $a_0, a_1 \xleftarrow{\$} \mathbb{Z}_p^*$, $b_0, b_1 \xleftarrow{\$} \mathbb{Z}_p^*$ and sends them to the algorithm $\mathcal{B}$. Further, the LRS protocol challenger randomly chooses $a \xleftarrow{\$} \{a_0, a_1\}$, $b \xleftarrow{\$} \{b_0, b_1\}$, and uses a and b as the long-term secrets to compute the leakage from encodings of a and b respectively. Let $\lambda_\Lambda = (\lambda_L, \lambda_R)$ be the leakage bound on the two encodings of the secrets a and b. When the algorithm $\mathcal{B}$ get (a_0, a_1) and (b_0, b_1) as challenge from the LRS challenger, it uses a_0 and b_0 as the long-term DH secret keys of the party U^* and V^* respectively, and computes the corresponding DH public keys. For all other parties, it sets up the ephemeral secret/public keys by itself. $\mathcal{B}$ answers all the leakage queries of all parties by computing the adversarial leakage function $\mathbf{f}$ itself except the parties U^* and V^*. In order to obtain the leakage of the long-term secret key of U^* and V^*, algorithm $\mathcal{B}$ queries the LRS protocol challenger with the adversarial leakage function $\mathbf{f}$, and passes that leakage to $\mathcal{A}$.

If the secret a chosen by the LRS protocol challenger is a_0 and the secret b chosen is b_0, the leakage of the DH long-term secret keys of U^* and V^* simulated by $\mathcal{B}$ (with the aid of the LRS protocol challenger) are the real leakages. Then the simulation is identical to Game 1. Otherwise, the leakage of the long-term DH secret key of U^* and V^* simulated by $\mathcal{B}$ are leakages of random values. Then the simulation is identical to Game 2. Hence,

$$|\mathsf{Adv}_{Game_2}(\mathcal{A}) - \mathsf{Adv}_{Game_1}(\mathcal{A})| \leq 2\,\epsilon_1. \tag{5}$$

Game 3. Same as Game 2 with the following exception: the challenger randomly chooses $c \xleftarrow{\$} \mathbb{Z}_q^*$ and computes $SK \leftarrow F(g^c, \mathsf{sid}) \oplus F(\mathsf{CDH}(epk_A, epk_B), \mathsf{sid})$. When the adversary asks the $\mathsf{Test}(U^*, V^*, s^*)$ query, Game 3 challenger will answer with SK. Besides, the leakage queries are also handled *differently* as described below.

Let epk_A, epk_B be the (unencrypted) ephemeral DH public keys of the protocol principals U^*, V^* respectively, such that $epk_A = g^{esk_A}$, $epk_B = g^{esk_B}$ and $\mathsf{CDH}(epk_A, epk_B) = g^{esk_A esk_B}$. We construct an algorithm $\mathcal{B}$ against a DDH challenger, using the adversary $\mathcal{A}$ as a sub routine. $\mathcal{B}$ sets $U \leftarrow g^a$ as the long term Diffie-Hellman public key of U^* and $V \leftarrow g^b$ as the long term Diffie-Hellman public key of V^*. Moreover,$\mathcal{B}$ sets all the long-term secret/public key pairs (Diffie-Hellman and encryption key pairs) to all protocol principals. The DDH challenger sends values (g^a, g^b, g^c) such that $c = ab$ or $c \xleftarrow{\$} \mathbb{Z}_p^*$ as the inputs to the algorithm $\mathcal{B}$. Algorithm $\mathcal{B}$ simulates answers to the adversarial queries as follows:

1. $\mathsf{Send}(U, V, s, m, \mathbf{f})$- If U^* is the initiator, $\mathcal{B}$ sends the ciphertext $C_A \leftarrow (pk_{V^*}, epk_A)$ to $\mathcal{A}$ as the first message of the test session. Upon receiving the second protocol message $C_B \leftarrow (pk_{U^*}, epk_B)$, $\mathcal{B}$ computes the session key as $SK \leftarrow F(g^c, U^*||C_A||V^*||C_B) \oplus F(\mathsf{CDH}(epk_A, epk_B), U^*||C_A||V^*||C_B)$.

   If U^* is the responder, upon receiving the first protocol message $C_A \leftarrow (pk_{U^*}, epk_A)$ from V^* to U^*, $\mathcal{B}$ sends $C_B \leftarrow (pk_{V^*}, epk_B)$ to $\mathcal{A}$ as the second protocol message of the test session, and computes the session key as $SK \leftarrow F(g^c, U^*||C_A||V^*||C_B) \oplus F(\mathsf{CDH}(epk_A, epk_B), U^*||C_A||V^*||C_B)$. For all other cases, $\mathcal{B}$ can decrypt incoming protocol messages and execute the protocol normally.
   The adversarial leakage queries for principles U^* and V^* are answered using the leakage oracle of the AFL-LR-CCA-2 secure PKE scheme and the leakage oracle of the LRS encoding scheme. In other words, $\mathcal{B}$ forwards the tuple leakage function $\mathbf{f} = (f_1, f_2)$ to the leakage oracle of the AFL-LR-CCA-2 secure PKE scheme $\mathcal{E}$ to obtain leakage from the encryption secret keys sk_{U^*} and sk_{V^*}. It also forwards the leakage function $\mathbf{f}$ to the leakage oracle of the LRS encoding scheme. It then forwards the responses of both these oracles to the adversary $\mathcal{A}$.
2. $\mathsf{SessionKeyReveal}(U, V, s)$- $\mathsf{SessionKeyReveal}$ query is not allowed to the target session or the partner of the target session as per the freshness conditions. $\mathcal{B}$ can compute all the other session keys by executing the protocol normally.
3. $\mathsf{EphemeralKeyReveal}(U, V, s)$- $\mathcal{B}$ can answer all the $\mathsf{EphemeralKeyReveal}$ queries, since it has all the the ephemeral keys.
4. $\mathsf{Corrupt}(U)$- The adversary can corrupt queries on all the protocol principles excluding the principles U^* and V^*. $\mathcal{B}$ can answer to all other $\mathsf{Corrupt}$ queries since it has the long-term of all other principles.
5. $\mathsf{Test}(U, V, s)$- When $U = U^*$, $V = V^*$, $s = s^*$, $\mathcal{B}$ answers with SK as computed above in the Send query. Otherwise it aborts the game.

If $\mathcal{B}$'s input is a Diffie-Hellman triple, simulation constructed by $\mathcal{B}$ is identical to Game 2, otherwise it is identical to Game 3. If $\mathcal{A}$ can distinguish the difference between games, then $\mathcal{B}$ can answer the DDH challenge. Hence,

$$|\mathsf{Adv}_{Game_3}(\mathcal{A}) - \mathsf{Adv}_{Game_2}(\mathcal{A})| \leq \mathsf{Adv}^{\mathsf{DDH}}_{\mathcal{B}}(\kappa). \quad (6)$$

Game 4. Same as Game 3 with the following exception: the challenger randomly chooses $SK \xleftarrow{\$} \mathcal{SK}$ and sends it to the adversary $\mathcal{A}$ as the answer to the $\mathsf{Test}(U^*, V^*, s^*)$ query.

If $\mathcal{A}$ can distinguish the difference between Game 2 and Game 3, then $\mathcal{A}$ can be used as a subroutine of an algorithm $\mathcal{D}$, which is used to distinguish whether the session key value SK is computed using the real PRF with a hidden key, or using a random function. $\mathcal{D}$ simulates the execution environment to $\mathcal{A}$ as follows:

1. $\mathsf{Send}(U, V, s, m)$- If U^* is the initiator, upon receiving the second protocol message C_B from V^* to U^*, it computes the session key as $SK \leftarrow \text{Oracle}^{\mathsf{PRF}}(U^*||C_A||V^*||C_B) \oplus F(\mathsf{CDH}(epk_A, epk_B), U^*||C_A||V^*||C_B)$.

   If U^* is the responder, upon receiving the first protocol message C_A from V^* to U^*, it computes the session key as $SK \leftarrow \text{Oracle}^{\mathsf{PRF}}(U^*||C_A||V^*||C_B) \oplus F(\mathsf{CDH}(epk_A, epk_B), U^*||C_A||V^*||C_B)$. For all the other cases of Send queries, $\mathcal{D}$ can execute the protocol normally. The leakage queries are also handled in a similar manner as defined above.
2. $\mathsf{SessionKeyReveal}(U, V, s)$- $\mathsf{SessionKeyReveal}$ query is not allowed to the target session or the partner of the target session as per the freshness conditions. $\mathcal{D}$ can compute all the other session keys by executing the protocol normally.
3. $\mathsf{EphemeralKeyReveal}(U, V, s)$- $\mathcal{B}$ can answer all the $\mathsf{EphemeralKeyReveal}$ queries, since it has all the the ephemeral keys.
4. $\mathsf{Corrupt}(U)$- The adversary can corrupt queries on all the protocol principles excluding the principles U^* and V^*. $\mathcal{B}$ can answer to all other $\mathsf{Corrupt}$ queries since it has the long-term of all other principles.
5. $\mathsf{Test}(U, V, s)$- When $U = U^*$, $V = V^*$, $s = s^*$, $\mathcal{B}$ answers with SK as computed above in the Send query. Otherwise it aborts the game.

For $\mathcal{A}$, the simulation by $\mathcal{D}$ is same as the Game 3 if the oracle is the PRF with hidden key. Otherwise, the simulation by $\mathcal{D}$ is same as Game 4. Thus we have,

$$|\mathsf{Adv}_{Game_2}(\mathcal{A}) - \mathsf{Adv}_{Game_1}(\mathcal{A})| \leq \epsilon_{\mathsf{PRF}}. \quad (7)$$

In this game, i.e., in Game 3, the session key in the Test session is *perfectly randomized*. Thus, $\mathcal{A}$ cannot obtain any advantage from Test query. So we have, $\mathsf{Adv}_{Game_3}(\mathcal{A}) = 0$. Using the above equations in this case, we get:

$$\mathsf{Adv}^{BAFL\text{-}eCK}_{\mathsf{P}}(\mathcal{A}) \leq n^2\ell^2 \max(2\epsilon_1 + \mathsf{Adv}^{\mathsf{DDH}}_{\mathcal{B}}(\kappa) + \epsilon_{\mathsf{prf}}).$$

Analysis of Case B.1.3.

Game 0. Same as Game 0 of **Case B.1.1**.

Game 1. Same as Game 1 of **Case B.1.1**.

Game 2. Same as Game 1 with the following exception: the challenger randomly chooses a ciphertext C from the ciphertext space as encryption of the ephemeral public key epk_A of the test session $\Pi^s_{U^*,V^*}$, and sends it to the session $\Pi^{s'}_{V^*,U^*}$.

We construct an AFL-CCA-2 adversary $\mathcal{S}$ from $\mathcal{A}$ in Game 1 or Game 2 if $\mathcal{A}$ can distinguish between these games. $\mathcal{S}$ picks two random strings $epk_{A_0}, epk_{A_1} \xleftarrow{\$} \mathbb{Z}^*_p$, and passes them to the challenger of the AFL-CCA-2 security game. From the AFL-CCA2 challenger, $\mathcal{S}$ receives a challenge ciphertext C such that $C \leftarrow (pk_{V^*}, epk_{A_b})$, where $b \xleftarrow{\$} \{0, 1\}$. $\mathcal{S}$ uses epk_{A_1} as the decryption of C when answering queries. $\mathcal{S}$ performs the following steps to simulate the execution environment to $\mathcal{A}$.

1. $\mathsf{Send}(U, V, s, m, \mathbf{f})$- We consider the following sub-cases:
   - $U = U^*$, $V = V^*$, $s = s^*$:
     - If U^* is the initiator, $\mathcal{S}$ sends the ciphertext C to $\mathcal{A}$as the first message of the test session. Upon receiving the second protocol message, it computes the session key $SK \leftarrow F(\mathsf{CDH}(U, V), U^*||C_A||V^*||C_B) \oplus F(\mathsf{CDH}(epk_{A_1}, epk_B), U^*||C_A||V^*||C_B)$.
     - If U^* is the responder, upon receiving the first protocol message sends C to $\mathcal{A}$, and computes the session key $SK \leftarrow F(\mathsf{CDH}(U, V), V^*||C_A||U^*||C_B) \oplus F(\mathsf{CDH}(epk_{A_1}, epk_B), V^*||C_A||U^*||C_B)$.
   - $U = U^*$, $V = V^*$, $s \neq s^*$: Executes the protocol normally.
   - $U = U^*$, $V \neq V^*$: Executes the protocol normally.
   - $U = V^*$
     - If U is the initiator and it is the first message, then executes the protocol normally.
     - If this is the initiator and the second protocol message, or the responder, and if the incoming message is same as C, then use epk_{A_1} as its decryption (,i.e. the ephemeral public key). Else it uses the decryption oracle to decrypt incoming messages.
   - U, $V \neq U^*$ or V^*: Executes the protocol normally.

   The leakage queries of $\mathcal{A}$ for party V^* are answered using the leakage oracle of the AFL-LR-CCA-2 secure PKE scheme $\mathcal{E}$ and the leakage oracle of the LRS scheme $\Lambda^{n,1}_{\mathbb{Z}^*_p}$. For party U^*, $\mathcal{S}$ already knows the its long-term secret keys, and so it can respond to the leakage queries.
2. $\mathsf{SessionKeyReveal}(U, V, s)$- $\mathsf{SessionKeyReveal}$ query is not allowed to the target session or the partner of the target session as per the freshness conditions. $\mathcal{S}$ can compute all the session keys by executing the protocol.
   - For sessions involving V^*, if the incoming message is same as C, then use epk_{A_1} as its decryption and also while computing the session key.
   - For sessions involving V^*, if the incoming message is different from C, use the decryption oracle to decrypt the incoming messages.
   - Otherwise, $\mathcal{S}$ can decrypt all the other incoming messages to protocol principals by its own.
3. $\mathsf{EphemeralKeyReveal}(U, V, s)$: $\mathcal{S}$ can answer to this query since it has all the ephemeral secret keys. Note that, in this case, $\mathcal{A}$ is not allowed to ask for $\mathsf{EphemeralKeyReveal}(U^*, V^*, s)$ query by the freshness condition.
4. $\mathsf{Corrupt}(U)$- In this case the adversary cannot ask for $\mathsf{Corrupt}(V^*)$ query, as per the freshness condition. Apart from that, $\mathcal{S}$ can answer all the $\mathsf{Corrupt}$ queries since it has all the secret keys.
5. $\mathsf{Test}(U, V, s)$- When $U = U^*$, $V = V^*$, $s = s^*$, $\mathcal{S}$ answers with SK as computed above in the Send query. Otherwise it aborts the game.

If the value C is the encryption of the value epk_{A_1}, the simulation constructed by $\mathcal{S}$ is identical to the Game 1, otherwise it is identical to Game 2. Hence,

$$|\mathsf{Adv}_{Game_2}(\mathcal{A}) - \mathsf{Adv}_{Game_1}(\mathcal{A})| \leq \mathsf{Adv}^{\mathsf{AFL\text{-}CCA\text{-}2}}_{\mathcal{S}}(\kappa). \tag{8}$$

Game 3. Same as Game 2 with the following exception: the challenger randomly chooses $SK \xleftarrow{\$} \mathcal{SK}$ and sends it to the adversary $\mathcal{A}$ as the answer to the $\mathsf{Test}(U^*, V^*, s^*)$ query.

If $\mathcal{A}$ can distinguish the difference between Game 2 and Game 3, then $\mathcal{A}$ can be used as a subroutine of an algorithm $\mathcal{D}$, which is used to distinguish whether the session key value SK is computed using the real PRF with a hidden key, or using a random function. $\mathcal{D}$ simulates the execution environment to $\mathcal{A}$ as follows:

1. $\mathsf{Send}(U, V, s, m, \mathbf{f})$- We consider the following sub-cases:
   - $U = U^*$, $V = V^*$, $s = s^*$:
     - If U^* is the initiator, upon receiving the second protocol message, it computes the session key $SK \leftarrow F(\mathsf{CDH}(U, V), U^*||C_A||V^*||C_B) \oplus \text{Oracle}^{\mathsf{PRF}}(U^*||C_A||V^*||C_B)$.
     - If U^* is the responder, upon receiving the first protocol message computes the session key $SK \leftarrow F(\mathsf{CDH}(U, V), U^*||C_A||V^*||C_B) \oplus \text{Oracle}^{\mathsf{PRF}}(U^*||C_A||V^*||C_B)$.
   - $U = U^*$, $V = V^*$, $s \neq s^*$: Executes the protocol normally.
   - $U = U^*$, $V \neq V^*$: Executes the protocol normally.
   - $U = V^*$
     - If U is the initiator and it is the first message, then executes the protocol normally.
     - If this is the initiator and the second protocol message, or the responder, and if the incoming message is same as C, then use $\text{Oracle}^{\mathsf{PRF}}$. Else, it executes the protocol normally.
   - U, $V \neq U^*$ or V^*: Executes the protocol normally.

   If U^* is the initiator, upon receiving the second protocol message C_B from V^* to U^*, it computes the session key as $SK \leftarrow \text{Oracle}^{\mathsf{PRF}}(U^*||C_A||V^*||C_B) \oplus F(\mathsf{CDH}(epk_A, epk_B), U^*||C_A||V^*||C_B)$.

   If U^* is the responder, upon receiving the first protocol message C_A from V^* to U^*, it computes the session key as $SK \leftarrow \text{Oracle}^{\mathsf{PRF}}(U^*||C_A||V^*||C_B) \oplus F(\mathsf{CDH}(epk_A, epk_B), U^*||C_A||V^*||C_B)$. For all the other cases of Send queries, $\mathcal{D}$ can execute the protocol normally. The leakage queries are also handled in a similar manner as defined above.
2. $\mathsf{SessionKeyReveal}(U, V, s)$- $\mathsf{SessionKeyReveal}$ query is not allowed to the target session or the partner of the target session as per the freshness conditions. $\mathcal{D}$ can compute all the other session keys by executing the protocol normally.
3. $\mathsf{EphemeralKeyReveal}(U, V, s)$- $\mathcal{B}$ can answer all the $\mathsf{EphemeralKeyReveal}$ queries, since it has all the the ephemeral keys.
4. $\mathsf{Corrupt}(U)$- The adversary can corrupt queries on all the protocol principles excluding the principles U^* and V^*. $\mathcal{B}$ can answer to all other $\mathsf{Corrupt}$ queries since it has the long-term of all other principles.
5. $\mathsf{Test}(U, V, s)$- When $U = U^*$, $V = V^*$, $s = s^*$, $\mathcal{B}$ answers with SK as computed above in the Send query. Otherwise it aborts the game.

For $\mathcal{A}$, the simulation by $\mathcal{D}$ is same as the Game 2 if the oracle is the PRF with hidden key. Otherwise, the simulation by $\mathcal{D}$ is same as Game 3. Thus we have,

$$|\mathsf{Adv}_{Game_2}(\mathcal{A}) - \mathsf{Adv}_{Game_1}(\mathcal{A})| \leq \epsilon_{\mathsf{PRF}}. \tag{9}$$

In this game, i.e., in Game 3, the session key in the Test session is *perfectly randomized*. Thus, $\mathcal{A}$ cannot obtain any advantage from Test query. So we have, $\mathsf{Adv}_{Game_3}(\mathcal{A}) = 0$.

Using the above equations in this case, we get:

$$\mathsf{Adv}_{\mathsf{P}}^{BAFL\text{-}eCK}(\mathcal{A}) \leq n^2\ell^2 \max(\mathsf{Adv}_{\mathcal{S}}^{\mathsf{AFL\text{-}LR\text{-}CCA\text{-}2}}(\kappa) + \epsilon_{\mathsf{prf}}).$$

Analysis of Case B.1.4.

This case is *symmetric* to the above case, i.e., **Case B.1.3**. Only the roles of $\mathsf{Corrupt}$ and $\mathsf{EphemeralKeyReveal}$ gets swapped from the above case. So we get,

$$\mathsf{Adv}_{\mathsf{P}}^{BAFL\text{-}eCK}(\mathcal{A}) \leq n^2\ell^2 \max(\mathsf{Adv}_{\mathcal{S}}^{\mathsf{AFL\text{-}LR\text{-}CCA\text{-}2}}(\kappa) + \epsilon_{\mathsf{prf}}).$$

6.5 Partner to the test session does not exist

In this case we consider that the partner to the test session *does not exist*, i.e., $\Pi^{s}_{U^*,V^*}$ do not have a matching session. Note that when the matching session to the test session does not exist, the adversary cannot obtain the long-term secret key of the peer to the test session. We consider the following sub-cases under this case:

(1) The adversary $\mathcal{A}$ issues $\mathsf{Corrupt}(U)$ query.

(2) $\mathcal{A}$ issues $\mathsf{EphemeralKeyReveal}(U, V, s)$ query.

Analysis of Case B.2.1. Here the adversary gets the long-term secret key of the owner of the test session by issuing the query $\mathsf{Corrupt}(U)$. The proof in this case is essentially the same as Case B.1.3. The situation that the ephemeral secret key of $\Pi^{s'}_{V^*,U^*}$ is given to $\mathcal{A}$ is the same as $\Pi^{s}_{U^*,V^*}$ has no matching session because $\mathcal{A}$ can decide arbitrary ephemeral key. Thus the analysis follows from Case B.1.3.

Analysis of Case B.2.2. Here the adversary gets the ephemeral secret key of the session $\Pi^{s}_{U^*,V^*}$. The proof in this case is essentially the same as Case B.1.2. The situation that the ephemeral secret key of $\Pi^{s'}_{V^*,U^*}$ is given to $\mathcal{A}$ is the same as $\Pi^{s}_{U^*,V^*}$ has no matching session because $\mathcal{A}$ can decide arbitrary ephemeral key. Thus the analysis follows from Case B.1.2.

Thus combining all the cases we get:

$$\mathsf{Adv}^{BAFL\text{-}eCK}_{\mathsf{P}}(\mathcal{A}) \leq n^2 l^2 \max\Big((2\epsilon_1 + \mathsf{Adv}^{\mathsf{DDH}}_{\mathcal{B}}(\kappa) + \epsilon_{\mathsf{prf}}), (\mathsf{Adv}^{\mathsf{AFL\text{-}LR\text{-}CCA\text{-}2}}_{\mathcal{S}}(\kappa) + \epsilon_{\mathsf{prf}})\Big).$$

This proves the statement of Theorem 2. □

7 Conclusion

In this paper, we proposed two generic compilers for after-the-fact leakage. One is a generic transformation from a leakage-resilient CPA-secure PKE to a leakage-resilient CCA-2-secure PKE in split-state bounded memory leakage model. The salient feature of our transformation is that the leakage tolerance of the transformed CCA-2 secure PKE is exactly same as the leakage tolerance of the underlying CPA-secure PKE and is also efficient. Our second compiler transforms any after-the-fact leakage-resilient CCA-2 secure PKE to a leakage-resilient AKE protocol in the BAFL-eCK model. An interesting open problem would be to design a generic compiler for transforming a CPA-secure PKE to a CCA-2 secure PKE in the presence of after-the-fact leakage, but in non-split state model. We also leave open the problem of constructing a generic compiler for leakage-resilient CPA-secure PKE to a leakage resilient CCA-2 secure PKE in the presence of continuous after-the-fact leakage.

References

[ABS14] Janaka Alawatugoda, Colin Boyd, and Douglas Stebila. Continuous after-the-fact leakage-resilient key exchange. In *ACISP*, pages 258–273. Springer, 2014.

[ADW09] Joël Alwen, Yevgeniy Dodis, and Daniel Wichs. Leakage-resilient public-key cryptography in the bounded-retrieval model. In *Advances in Cryptology-CRYPTO 2009*, pages 36–54. Springer, 2009.

[AGV09] Adi Akavia, Shafi Goldwasser, and Vinod Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In *Theory of Cryptography Conference*, pages 474–495. Springer, 2009.

[Ala15a] Janaka Alawatugoda. Generic construction of an\ mathrm {eCK}-secure key exchange protocol in the standard model. *International Journal of Information Security*, pages 1–17, 2015.

[Ala15b] Janaka Alawatugoda. Generic transformation of a cca2-secure public-key encryption scheme to an eck-secure key exchange protocol in the standard model. Cryptology ePrint Archive, Report 2015/1248, 2015. http://eprint.iacr.org/2015/1248.

[ASB14] Janaka Alawatugoda, Douglas Stebila, and Colin Boyd. Modelling after-the-fact leakage for key exchange. In *Proceedings of the 9th ACM symposium on Information, computer and communications security*, pages 207–216. ACM, 2014.

[ASB15] Janaka Alawatugoda, Douglas Stebila, and Colin Boyd. Continuous after-the-fact leakage-resilient eck-secure key exchange. In *IMA International Conference on Cryptography and Coding*, pages 277–294. Springer, 2015.

[BKKV] Zvika Brakerski, Yael Tauman Kalai, Jonathan Katz, and Vinod Vaikuntanathan. Cryptography resilient to continual memory leakage, 2010.

[BR94] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In *Advances in Cryptology – CRYPTO93*, pages 232–249. Springer, 1994.

[CK01] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *Advances in Cryptology - EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 453–474. Springer, 2001.

[CMY+16] Rongmao Chen, Yi Mu, Guomin Yang, Willy Susilo, and Fuchun Guo. Strongly leakage-resilient authenticated key exchange. In *Cryptographers Track at the RSA Conference*, pages 19–36. Springer, 2016.

[Cre11] Cas Cremers. Examining indistinguishability-based security models for key exchange protocols: the case of ck, ck-hmqv, and eck. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, pages 80–91. ACM, 2011.

[DF11] Stefan Dziembowski and Sebastian Faust. Leakage-resilient cryptography from the inner-product extractor. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 702–721. Springer, 2011.

[DHLAW10a] Yevgeniy Dodis, Kristiyan Haralambiev, Adriana López-Alt, and Daniel Wichs. Cryptography against continuous memory attacks. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, pages 511–520. IEEE, 2010.

[DHLAW10b] Yevgeniy Dodis, Kristiyan Haralambiev, Adriana López-Alt, and Daniel Wichs. Efficient public-key cryptography in the presence of key leakage. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 613–631. Springer, 2010.

[DORS08] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM journal on computing*, 38(1):97–139, 2008.

[FKN+15] Eiichiro Fujisaki, Akinori Kawachi, Ryo Nishimaki, Keisuke Tanaka, and Kenji Yasunaga. Post-challenge leakage resilient public-key cryptosystem in split state model. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 98(3):853–862, 2015.

[Gro06] Jens Groth. Simulation-sound nizk proofs for a practical language and constant size group signatures. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 444–459. Springer, 2006.

[HL11] Shai Halevi and Huijia Lin. After-the-fact leakage in public-key encryption. In *Theory of Cryptography Conference*, pages 107–124. Springer, 2011.

[HSH+09] J Alex Halderman, Seth D Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A Calandrino, Ariel J Feldman, Jacob Appelbaum, and Edward W Felten. Lest we remember: cold-boot attacks on encryption keys. *Communications of the ACM*, 52(5):91–98, 2009.

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in CryptologyCRYPTO99*, pages 388–397. Springer, 1999.

[Koc96] Paul C Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Advances in CryptologyCRYPTO96*, pages 104–113. Springer, 1996.

[Kra05] Hugo Krawczyk. Hmqv: A high-performance secure diffie-hellman protocol. In *Advances in Cryptology – CRYPTO 2005*, pages 546–566. Springer, 2005.

[LLM07] Brian LaMacchia, Kristin Lauter, and Anton Mityagin. Stronger security of authenticated key exchange. In *Provable Security*, pages 1–16. Springer, 2007.

[MO11] Daisuke Moriyama and Tatsuaki Okamoto. Leakage resilient eck-secure key exchange protocol without random oracles. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, pages 441–447. ACM, 2011.

[MR04] Silvio Micali and Leonid Reyzin. Physically observable cryptography. In *Theory of Cryptography Conference*, pages 278–296. Springer, 2004.

[MU08] Alfred Menezes and Berkant Ustaoglu. Comparing the pre-and post-specified peer models for key agreement. In *Information Security and Privacy*, pages 53–68. Springer, 2008.

[NS09] Moni Naor and Gil Segev. Public-key cryptosystems resilient to key leakage. In *Advances in Cryptology-CRYPTO 2009*, pages 18–35. Springer, 2009.

[QL13] Baodong Qin and Shengli Liu. Leakage-resilient chosen-ciphertext secure public-key encryption from hash proof system and one-time lossy filter. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 381–400. Springer, 2013.

[QL14] Baodong Qin and Shengli Liu. Leakage-flexible cca-secure public-key encryption: simple construction and free of pairing. In *International Workshop on Public Key Cryptography*, pages 19–36. Springer, 2014.

[SEVB10] Augustin P Sarr, Philippe Elbaz-Vincent, and Jean-Claude Bajard. A new security model for authenticated key agreement. In *Security and Cryptography for Networks*, pages 219–234. Springer, 2010.

[Sho99] Victor Shoup. *On formal models for secure key exchange*. Citeseer, 1999.

[Sho04] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. *IACR Cryptology ePrint Archive*, 2004:332, 2004.

[Too15] Mohsen Toorani. On continuous after-the-fact leakage-resilient key exchange. In *Proceedings of the Second Workshop on Cryptography and Security in Computing Systems*, page 31. ACM, 2015.

[YL16] Zheng Yang and Shuangqing Li. On security analysis of an after-the-fact leakage resilient key exchange protocol. *Information Processing Letters*, 116(1):33–40, 2016.

[ZCC15] Zongyang Zhang, Sherman SM Chow, and Zhenfu Cao. Post-challenge leakage in public-key encryption. *Theoretical Computer Science*, 572:25–49, 2015.