Continuous After-the-fact Leakage-Resilient eCK-secure Key Exchange

Janaka Alawatugoda[1,4], Douglas Stebila[1,2], and Colin Boyd[3]

[1] School of Electrical Engineering and Computer Science,
[2] School of Mathematical Sciences,
Queensland University of Technology, Brisbane, Australia
janaka.alawatugoda@qut.edu.au, stebila@qut.edu.au
[3] Department of Telematics,
Norwegian University of Science and Technology, Trondheim, Norway
colin.boyd@item.ntnu.no
[4] Department of Computer Engineering,
University of Peradeniya, Peradeniya, Sri Lanka

Abstract. Security models for two-party authenticated key exchange (AKE) protocols have developed over time to capture the security of AKE protocols even when the adversary learns certain secret values. Increased granularity of security can be modelled by considering partial leakage of secrets in the manner of models for leakage-resilient cryptography, designed to capture side-channel attacks. In this work, we use the strongest known partial-leakage-based security model for key exchange protocols, namely continuous after-the-fact leakage eCK (CAFL-eCK) model. We resolve an open problem by constructing the first concrete two-pass leakage-resilient key exchange protocol that is secure in the CAFL-eCK model.

Keywords: key exchange protocols, side-channel attacks, security models, leakage-resilience, after-the-fact leakage

1 Introduction

During the past two decades side-channel attacks have become a familiar method of attacking cryptographic systems. Examples of information which may leak during executions of cryptographic systems, and so allow side-channel attacks, include timing information [6,8,18], electromagnetic radiation [15], and power consumption [21]. Leakage may reveal partial information about the secret parameters which have been used for computations in cryptographic systems. In order to abstractly model leakage attacks, cryptographers have proposed the notion of *leakage-resilient* cryptography [1,4,7,13,14,17,16,20]. In this notion the information that leaks is not fixed, but instead chosen adversarially, so as to model any possible physical leakage function. A variety of different cryptographic primitives have been developed in recent years. As one of the most widely used cryptographic primitives, it is important to analyze the leakage resilience of key exchange protocols.

Earlier key exchange security models, such as the Bellare–Rogaway [5], Canetti–Krawczyk [9], and extended Canetti–Krawczyk (eCK) [19] models, aim to capture security against an adversary who can fully compromise some, but not all, secret values. This is not a very granular form of leakage, and thus is not suitable for modelling side-channel attacks in key exchange protocols enabled by partial leakage of secret keys. This motivates the development of leakage-resilient key exchange security models [4,11,22,23,3]. Among them the generic security model proposed by Alawatugoda et al. [3] in 2014 facilitates more granular leakage.

Alawatugoda et al. [3] proposed a *generic leakage-security model* for key exchange protocols, which can be instantiated as either a *bounded* leakage variant or as a *continuous* leakage variant. In the bounded leakage variant, the total amount of leakage is bounded, whereas in the continuous leakage variant, each protocol execution may reveal a fixed amount of leakage. Further, the adversary is allowed to obtain the leakage even after the session key is established for the session under attack (after-the-fact leakage). In Section 3 we review the continuous leakage instantiation of the security model proposed by Alawatugoda et al.

Alawatugoda et al. [3] also provided a generic construction for a protocol which is proven secure in their generic leakage-security model. However, when it comes to a concrete construction, the proposed generic protocol can only be instantiated in a way that is secure in the *bounded* version of the security model. Until now there are no suitable cryptographic primitives which can be used to instantiate the generic protocol in the continuous leakage variant of the security model.

Our aim is to propose a concrete protocol construction which can be proven secure in the continuous leakage instantiation of the security model of Alawatugoda et al. Thus, we introduce the first concrete construction of *continuous* and *after-the-fact* leakage-resilient key exchange protocol.

Bounded Leakage and Continuous Leakage. Generally, in models assuming bounded leakage there is an upper bound on the amount of leakage information for the entire period of execution. The security guarantee only holds if the leakage amount is below the prescribed bound. Differently, in models allowing continuous leakage the adversary is allowed to obtain leakage over and over for a polynomial number of iterations during the period of execution. Naturally, there is a bound on the amount of leakage that the adversary can obtain in each single iteration, but the total amount of leakage that the adversary can obtain for the entire period of execution is unbounded.

After-the-fact Leakage. The concept of after-the-fact leakage has been applied previously to encryption primitives. Generally, leakage which happens after the challenge is given to the adversary is considered as after-the-fact leakage. In key exchange security models, the challenge to the adversary is to distinguish the session key of a chosen session, usually called the *test session*, from a random

session key [5,9,19], After-the-fact leakage is the leakage which happens after the test session is established.

Our Contribution. Alawatugoda et al. [3] left the construction of a continuous after-the-fact leakage-resilient eCK secure key exchange protocol as an open problem. In this paper, we construct such a protocol (protocol P2) using existing leakage-resilient cryptographic primitives. We use leakage-resilient storage schemes and their refreshing protocols [12] for this construction.

Table 1 compares the proposed protocol P2, with the NAXOS protocol [19], the Moriyama-Okamoto (MO) protocol [22] and the generic Alawatugoda et al. [3] protocol instantiation, by means of computation cost, security model and the proof model.

Protocol	Initiator cost	Responder cost	Leakage Feature	After-the-fact	Proof model
NAXOS [19]	4 **Exp**	4 **Exp**	None	None	Random oracle
MO [22]	8 **Exp**	8 **Exp**	Bounded	No	Standard
Alawatugoda et al. [3]	12 **Exp**	12 **Exp**	Bounded	Yes	Standard
Protocol P2 (this paper)	6 **Exp**	6 **Exp**	Continuous	Yes	Random oracle

Table 1. Security and efficiency comparison of leakage-resilient key exchange protocols

In protocol P2, the secret key is encoded into two equal-sized parts of some chosen size, and the leakage bound from each of the two parts is 15% of the size of a part, per occurrence. Since this is a continuous leakage model the total leakage amount is unbounded. More details of the leakage tolerance of protocol P2 may be found in Section 5.3.

2 Preliminaries

We discuss the preliminaries which we use for the protocol constructions.

2.1 Diffie–Hellman Problems

Let $\mathcal{G}$ be a group generation algorithm and $(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^k)$, where $\mathbb{G}$ is a cyclic group of prime order q and g is an arbitrary generator.

Definition 1 (Computational Diffie–Hellman (CDH**) Problem).** *Given an instance (g, g^a, g^b) for $a, b \xleftarrow{\$} \mathbb{Z}_q$, the* CDH *problem is to compute g^{ab}.*

Definition 2 (Decision Diffie–Hellman (DDH**) Problem).** *Given an instance (g, g^a, g^b, g^c) for $a, b \xleftarrow{\$} \mathbb{Z}_q$ and either $c \xleftarrow{\$} \mathbb{Z}_q$ or $c = ab$, the* DDH *problem is to distinguish whether $c = ab$ or not.*

Definition 3 (Gap Diffie–Hellman (GDH**) Problem).** *Given an instance (g, g^a, g^b) for $a, b \xleftarrow{\$} \mathbb{Z}_q$, the* GDH *problem is to find g^{ab} given access to an oracle $\mathcal{O}$ that solves the* DDH *problem.*

2.2 Leakage-Resilient Storage

We review the definitions of leakage-resilient storage according to Dziembowski et al. [12]. The idea behind their construction is to split the storage of elements into two parts using a randomized encoding function. As long as leakage is then limited from each of its two parts then no adversary can learn useful information about an encoded element. The key observation of Dziembowski et al. is then to show how such encodings can be *refreshed* in a leakage-resilient way so that the new parts can be re-used. To construct a continuous leakage-resilient primitive the relevant secrets are split, used separately, and then refreshed between any two usages.

Definition 4 (Dziembowski-Faust leakage-resilient storage scheme). *For any $m, n \in \mathbb{N}$, the storage scheme $\Lambda^{n,m}_{\mathbb{Z}^*_q} = (\text{Encode}^{n,m}_{\mathbb{Z}^*_q}, \text{Decode}^{n,m}_{\mathbb{Z}^*_q})$ efficiently stores elements $s \in (\mathbb{Z}^*_q)^m$ where:*

- $\text{Encode}^{n,m}_{\mathbb{Z}^*_q}(s) : s_L \xleftarrow{\$} (\mathbb{Z}^*_q)^n \backslash \{(0^n)\}$, *then* $s_R \leftarrow (\mathbb{Z}^*_q)^{n \times m}$ *such that* $s_L \cdot s_R = s$ *and outputs* (s_L, s_R).
- $\text{Decode}^{n,m}_{\mathbb{Z}^*_q}(s_L, s_R)$: *outputs* $s_L \cdot s_R$.

In the model we expect an adversary to see the results of a leakage function applied to s_L and s_R. This may happen each time computation occurs.

Definition 5 ($\boldsymbol{\lambda}$-limited adversary). *If the amount of leakage obtained by the adversary from each of s_L and s_R is limited to $\boldsymbol{\lambda} = (\lambda_1, \lambda_2)$ bits in total respectively, the adversary is known as a* $\boldsymbol{\lambda}$-limited adversary.

Definition 6 ($(\boldsymbol{\lambda}_\Lambda, \epsilon_1)$-secure leakage-resilient storage scheme). *We say $\Lambda = (\text{Encode}, \text{Decode})$ is $(\boldsymbol{\lambda}_\Lambda, \epsilon_1)$-secure leakage-resilient, if for any $s_0, s_1 \xleftarrow{\$} (\mathbb{Z}^*_q)^m$ and any $\boldsymbol{\lambda}_\Lambda$-limited adversary $\mathcal{C}$, the leakage from $\text{Encode}(s_0) = (s_{0L}, s_{0R})$ and $\text{Encode}(s_1) = (s_{1L}, s_{1R})$ are statistically ϵ_1-close. For an adversary-chosen leakage function $\mathbf{f} = (f_1, f_2)$, and a secret s such that $\text{Encode}(s) = (s_L, s_R)$, the leakage is denoted as $\big(f_1(s_L), f_2(s_R)\big)$.*

Lemma 1 ([12]). *Suppose that $m < n/20$. Then $\Lambda^{n,m}_{\mathbb{Z}^*_q} = (\text{Encode}^{n,m}_{\mathbb{Z}^*_q}, \text{Decode}^{n,m}_{\mathbb{Z}^*_q})$ is $(\boldsymbol{\lambda}, \epsilon)$-secure for some ϵ and $\boldsymbol{\lambda} = (0.3 \cdot n \log q, 0.3 \cdot n \log q)$.*

The encoding function can be used to design different leakage resilient schemes with bounded leakage. The next step is to define how to *refresh* the encoding so that a continuous leakage is also possible to defend against.

Definition 7 (Refreshing of Leakage-Resilient Storage). *Let $(L', R') \leftarrow \text{Refresh}^{n,m}_{\mathbb{Z}^*_q}(L, R)$ be a refreshing protocol that works as follows:*

- *Input* : (L, R) *such that* $L \in (\mathbb{Z}^*_q)^n$ *and* $R \in (\mathbb{Z}^*_q)^{n \times m}$.
- *Refreshing* R :
  1. $A \xleftarrow{\$} (\mathbb{Z}^*_q)^n \backslash \{(0^n)\}$ *and* $B \leftarrow$ *non singular* $(\mathbb{Z}^*_q)^{n \times m}$ *such that* $A \cdot B = (0^m)$.

2. $M \leftarrow$ *non-singular* $(\mathbb{Z}_q^*)^{n\times n}$ *such that* $L \cdot M = A$.
3. $X \leftarrow M \cdot B$ *and* $R' \leftarrow R + X$.

– *Refreshing* L :
1. $\tilde{A} \xleftarrow{\$} (\mathbb{Z}_q^*)^n \backslash \{(0^n)\}$ *and* $\tilde{B} \leftarrow$ *non singular* $(\mathbb{Z}_q^*)^{n\times m}$ *such that* $\tilde{A} \cdot \tilde{B} = (0^m)$.
2. $\tilde{M} \leftarrow$ *non-singular* $(\mathbb{Z}_q^*)^{n\times n}$ *such that* $\tilde{M} \cdot R' = \tilde{B}$.
3. $Y \leftarrow \tilde{A} \cdot \tilde{M}$ *and* $L' \leftarrow L + Y$.

– *Output* : (L', R')

Let $\Lambda = (\text{Encode}, \text{Decode})$ be a $(\boldsymbol{\lambda}_\Lambda, \epsilon_1)$-secure leakage-resilient storage scheme and Refresh be a refreshing protocol. We consider the following experiment Exp, which runs Refresh for ℓ rounds and lets the adversary obtain leakage in each round. For refreshing protocol Refresh, a $\boldsymbol{\lambda}_{\text{Refresh}}$-limited adversary $\mathcal{B}$, $\ell \in \mathbb{N}$ and $s \xleftarrow{\$} (\mathbb{Z}_q^*)^m$, we denote the following experiment by $\text{Exp}_{(\text{Refresh},\Lambda)}(\mathcal{B}, s, \ell)$:

1. For a secret s, the initial encoding is generated as $(s_L^0, s_R^0) \leftarrow \text{Encode}(s)$.
2. For $j = 1$ to ℓ run $\mathcal{B}$ against the j^{th} round of the refreshing protocol.
3. Return whatever $\mathcal{B}$ outputs.

We require that the adversary $\mathcal{B}$ outputs a single bit $b \in \{0, 1\}$ upon performing the experiment Exp using $s \xleftarrow{\$} \{s_0, s_1\} \in (\mathbb{Z}_q^*)^m$. Now we define leakage-resilient security of a refreshing protocol.

Definition 8 ($(\ell, \boldsymbol{\lambda}_{\text{Refresh}}, \epsilon_2)$-secure Leakage-Resilient Refreshing Protocol). *For a* $(\boldsymbol{\lambda}_\Lambda, \epsilon_1)$*-secure leakage-resilient storage scheme* $\Lambda = (\text{Encode}, \text{Decode})$ *with message space* $(\mathbb{Z}_q^*)^m$, Refresh *is* $(\ell, \boldsymbol{\lambda}_{\text{Refresh}}, \epsilon_2)$*-secure leakage-resilient, if for every* $\boldsymbol{\lambda}_{\text{Refresh}}$*-limited adversary* $\mathcal{B}$ *and any two secrets* $s_0, s_1 \in (\mathbb{Z}_q^*)^m$*, the statistical distance between* $\text{Exp}_{(\text{Refresh},\Lambda)}(\mathcal{B}, s_0, \ell)$ *and* $\text{Exp}_{(\text{Refresh},\Lambda)}(\mathcal{B}, s_1, \ell)$ *is bounded by* ϵ_2.

Theorem 1 ([12]). *Let* $m/3 \leq n, n \geq 16$ *and* $\ell \in \mathbb{N}$. *Let* n, m *and* $\mathbb{Z}_q^*$ *be such that* $\Lambda_{\mathbb{Z}_q^*}^{n,m}$ *is* $(\boldsymbol{\lambda}, \epsilon)$*-secure leakage-resilient storage scheme (Definition 4 and Definition 6). Then the refreshing protocol* $\text{Refresh}_{\mathbb{Z}_q^*}^{n,m}$ *(Definition 7) is a* $(\ell, \boldsymbol{\lambda}/2, \epsilon')$*-secure leakage-resilient refreshing protocol for* $\Lambda_{\mathbb{Z}_q^*}^{n,m}$ *(Definition 8) with* $\epsilon' = 2\ell q(3q^m\epsilon + mq^{-n-1})$.

3 Continuous After-the-Fact Leakage eCK Model and the eCK Model

In 2014 Alawatugoda et al. [3] proposed a new security model for key exchange protocols, namely the generic after-the-fact leakage eCK ((·)AFL-eCK) model which, in addition to the adversarial capabilities of the eCK model [19], is equipped with an adversary-chosen, efficiently computable, adaptive leakage function $\mathbf{f}$, enabling the adversary to obtain the leakage of long-term secret keys of protocol principals. Therefore the (·)AFL-eCK model captures all the attacks captured by the eCK model, and captures the partial leakage of long-term secret keys due to side-channel attacks.

The eCK Model. In the eCK model, in sessions where the adversary does not modify the communication between parties (passive sessions), the adversary is allowed to reveal both ephemeral secrets, long-term secrets, or one of each from two different parties, whereas in sessions where the adversary may forge the communication of one of the parties (active sessions), the adversary is allowed to reveal the long-term or ephemeral secret of the other party. The security challenge is to distinguish the real session key from a random session key, in an adversary-chosen protocol session.

Generic After-the-Fact Leakage eCK Model. The generic (·)AFL-eCK model can be instantiated in two different ways which leads to two security models. Namely, *bounded* after-the-fact leakage eCK (BAFL-eCK) model and *continuous* after-the-fact leakage eCK (CAFL-eCK) model. The BAFL-eCK model allows the adversary to obtain a bounded amount of leakage of the long-term secret keys of the protocol principals, as well as reveal session keys, long-term secret keys and ephemeral keys. Differently, the CAFL-eCK model allows the adversary to continuously obtain arbitrarily large amount of leakage of the long-term secret keys of the protocol principals, enforcing the restriction that the amount of leakage per observation is bounded.

Below we revisit the definitions of the CAFL-eCK model, and we also recall the definitions of the eCK model as a comparison to the CAFL-eCK definitions.

3.1 Partner Sessions in the CAFL-eCK Model

Definition 9 (Partner sessions in the CAFL-eCK **model).** *Two oracles* $\Pi^{s}_{U,V}$ *and* $\Pi^{s'}_{U',V'}$ *are said to be partners if all of the following hold:*

1. *both* $\Pi^{s}_{U,V}$ *and* $\Pi^{s'}_{U',V'}$ *have computed session keys;*
2. *messages sent from* $\Pi^{s}_{U,V}$ *and messages received by* $\Pi^{s'}_{U',V'}$ *are identical;*
3. *messages sent from* $\Pi^{s'}_{U',V'}$ *and messages received by* $\Pi^{s}_{U,V}$ *are identical;*
4. $U' = V$ *and* $V' = U$*;*
5. *Exactly one of* U *and* V *is the initiator and the other is the responder.*

The protocol is said to be correct if two partner oracles compute identical session keys.

The definition of partner sessions is the same in the eCK model.

3.2 Leakage in the CAFL-eCK Model

A realistic way in which side-channel attacks can be mounted against key exchange protocols seems to be to obtain the leakage information from the protocol computations which use the secret keys. Following the previously used premise in other leakage models that "only computation leaks information", leakage is modelled where any computation takes place using secret keys. In normal protocol

models, by issuing a `Send` query, the adversary will get a protocol message which is computed according to the normal protocol computations. Sending an adversary-chosen, efficiently computable adaptive leakage function with the `Send` query thus reflects the concept "only computation leaks information".

A tuple of t adaptively chosen efficiently computable leakage functions $\mathbf{f} = (f_{1j}, f_{2j}, \ldots, f_{tj})$ are introduced; j indicates the jth leakage occurrence and the size t of the tuple is *protocol-specific*. A key exchange protocol may use more than one cryptographic primitive where each primitive uses a distinct secret key. Hence, it is necessary to address the leakage of secret keys from each of those primitives. Otherwise, some cryptographic primitives which have been used to construct a key exchange protocol may be stateful and the secret key is encoded into number of parts. The execution of a stateful cryptographic primitive is split into a number of sequential stages and each of these stages uses one part of the secret key. Hence, it is necessary to address the leakage of each of these encoded parts of the secret key.

Note that the adversary is restricted to obtain leakage from each key part independently: the adversary cannot use the output of f_{1j} as an input parameter to the f_{2j} and so on. This prevents the adversary from observing a connection between each part.

3.3 Adversarial Powers of the CAFL-eCK Model

The adversary $\mathcal{A}$ controls the whole network. $\mathcal{A}$ interacts with a set of oracles which represent protocol instances. The following query allows the adversary to run the protocol.

- `Send`$(U, V, s, m, \mathbf{f})$ query: The oracle $\Pi^s_{U,V}$, computes the next protocol message according to the protocol specification and sends it to the adversary $\mathcal{A}$, along with the leakage $\mathbf{f}(sk_U)$. $\mathcal{A}$ can also use this query to activate a new protocol instance as an initiator with blank m.

In the eCK model `Send` query is same as the above except the leakage function $\mathbf{f}$.

The following set of queries allow the adversary $\mathcal{A}$ to compromise certain session specific ephemeral secrets and long-term secrets from the protocol principals.

- `SessionKeyReveal`(U, V, s) query: $\mathcal{A}$ is given the session key of the oracle $\Pi^s_{U,V}$.
- `EphemeralKeyReveal`(U, V, s) query: $\mathcal{A}$ is given the ephemeral keys (per-session randomness) of the oracle $\Pi^s_{U,V}$.
- `Corrupt`(U) query: $\mathcal{A}$ is given the long-term secrets of the principal U. This query does not reveal any session keys or ephemeral keys to $\mathcal{A}$.

`SessionKeyReveal`, `EphemeralKeyReveal` and `Corrupt` (Long-term key reveal) queries are the same in the eCK model.

Once the oracle $\Pi^s_{U,V}$ has accepted a session key, asking the following query the adversary $\mathcal{A}$ attempt to distinguish it from a random session key. The `Test` query is used to formalize the notion of the semantic security of a key exchange protocol.

– `Test`(U, s) query: When $\mathcal{A}$ asks the `Test` query, the challenger first chooses a random bit $b \xleftarrow{\$} \{0,1\}$ and if $b = 1$ then the actual session key is returned to $\mathcal{A}$, otherwise a random string chosen from the same session key space is returned to $\mathcal{A}$. This query is only allowed to be asked once across all sessions.

The `Test` query is the same in the eCK model.

3.4 Freshness Definition of the CAFL-eCK Model

Definition 10 ($\boldsymbol{\lambda}-$CAFL-eCK-freshness). *Let $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_t)$ be a vector of t elements (same size as $\mathbf{f}$ in* `Send` *query). An oracle $\Pi^{s}_{U,V}$ is said to be $\boldsymbol{\lambda}-$*CAFL-eCK*-fresh if and only if:*

1. *The oracle $\Pi^{s}_{U,V}$ or its partner, $\Pi^{s'}_{V,U}$ (if it exists) has not been asked a* `SessionKeyReveal`.
2. *If the partner $\Pi^{s'}_{V,U}$ exists, none of the following combinations have been asked:*
   (a) `Corrupt`(U) *and* `EphemeralKeyReveal`(U, V, s).
   (b) `Corrupt`(V) *and* `EphemeralKeyReveal`(V, U, s').
3. *If the partner $\Pi^{s'}_{V,U}$ does not exist, none of the following combinations have been asked:*
   (a) `Corrupt`(V).
   (b) `Corrupt`(U) *and* `EphemeralKeyReveal`(U, V, s).
4. *For each* `Send`$(U, \cdot, \cdot, \cdot, \mathbf{f})$ *query, size of the output of $|f_{ij}(sk_{Ui})| \leq \lambda_i$.*
5. *For each* `Send`$(V, \cdot, \cdot, \cdot, \mathbf{f})$ *queries, size of the output of $|f_{ij}(sk_{Vi})| \leq \lambda_i$.*

The eCK-freshness is slightly different from the $\boldsymbol{\lambda}-$CAFL-eCK-freshness by stripping off points 4 and 5.

3.5 Security Game and Security Definition of the CAFL-eCK Model

Definition 11 ($\boldsymbol{\lambda}-$CAFL-eCK security game). *Security of a key exchange protocol in the* CAFL-eCK *model is defined using the following security game, which is played by the adversary $\mathcal{A}$ against the protocol challenger.*

- ***Stage 1:*** *$\mathcal{A}$ may ask any of* `Send`, `SessionKeyReveal`, `EphemeralKeyReveal` *and* `Corrupt` *queries to any oracle at will.*
- ***Stage 2:*** *$\mathcal{A}$ chooses a $\boldsymbol{\lambda}-$*CAFL-eCK*-fresh oracle and asks a* `Test` *query. The challenger chooses a random bit $b \xleftarrow{\$} \{0,1\}$, and if $b = 1$ then the actual session key is returned to $\mathcal{A}$, otherwise a random string chosen from the same session key space is returned to $\mathcal{A}$.*
- ***Stage 3:*** *$\mathcal{A}$ continues asking* `Send`, `SessionKeyReveal`, `EphemeralKeyReveal` *and* `Corrupt` *queries. $\mathcal{A}$ may not ask a query that violates the $\boldsymbol{\lambda}-$*CAFL-eCK*-freshness of the test session.*
- ***Stage 4:*** *At some point $\mathcal{A}$ outputs the bit $b' \leftarrow \{0,1\}$ which is its guess of the value b on the test session. $\mathcal{A}$ wins if $b' = b$.*

The eCK security game is same as the above, except that in Stage 2 and Stage 3 eCK-fresh oracles are chosen instead of $\boldsymbol{\lambda}-$ CAFL-eCK-fresh oracles. $Succ_{\mathcal{A}}$ is the event that the adversary $\mathcal{A}$ wins the security game in Definition 11.

Definition 12 ($\boldsymbol{\lambda}-$ CAFL-eCK-security). *A protocol π is said to be $\boldsymbol{\lambda}-$ CAFL-eCK secure if there is no adversary $\mathcal{A}$ that can win the $\boldsymbol{\lambda}-$ CAFL-eCK security game with significant advantage. The advantage of an adversary $\mathcal{A}$ is defined as* $\mathrm{Adv}_{\pi}^{\boldsymbol{\lambda}-\text{CAFL-eCK}}(\mathcal{A}) = |2\Pr(Succ_{\mathcal{A}}) - 1|$.

3.6 Practical Interpretation of Security of CAFL-eCK Model

We review the relationship between the CAFL-eCK model and real world attack scenarios.

- **Active adversarial capabilities:** `Send` queries address the powers of an active adversary who can control the message flow over the network. In the previous security models, this property is addressed by introducing the *send* query.
- **Side-channel attacks:** Leakage functions are embedded with the `Send` query. Thus, assuming that the leakage happens when computations take place in principals, a wide variety of side-channel attacks such as timing attacks, EM emission based attacks, power analysis attacks, which are based on *continuous leakage of long-term secrets* are addressed. This property is not addressed in the earlier security models such as the BR models, the CK model, the eCK model and the Moriyama-Okamoto model.
- **Malware attacks:** `EphemeralKeyReveal` queries cover the malware attacks which steal stored ephemeral keys, given that the long-term keys may be securely stored separately from the ephemeral keys in places such as smart cards or hardware security modules. Separately, `Corrupt` queries address malware attacks which steal the long-term secret keys of protocol principals. In the previous security models, this property is addressed by introducing the *ephemeral-key reveal, session-state reveal* and *corrupt* queries.
- **Weak random number generators:** Due to weak random number generators, the adversary may correctly determine the produced random number. `EphemeralKeyReveal` query addresses situations where the adversary can get the ephemeral secrets. In the previous security models, this property is addressed by introducing the *ephemeral-key reveal query* or the *session-state reveal* query.

4 Protocol P1: Simple eCK-Secure Key Exchange

The motivation of LaMacchia et al. [19] in designing the eCK model was that an adversary should have to compromise both the long-term and ephemeral secret keys of a party in order to recover the session key. In this section we look at construction paradigms of eCK-secure key exchange protocols, because our aim

is to construct a CAFL-eCK-secure key exchange protocol using a eCK-secure key exchange protocol as the underlying primitive.

In the NAXOS protocol, [19], this is accomplished using what is now called the "NAXOS trick": a "pseudo" ephemeral key $\widetilde{esk}$ is computed as the hash of the long-term key lsk and the actual ephemeral key esk: $\widetilde{esk} \leftarrow H(esk, lsk)$. The value $\widetilde{esk}$ is never stored, and thus in the eCK model the adversary must learn both esk and lsk in order to be able to compute $\widetilde{esk}$. The initiator must compute $\widetilde{esk} = H(esk, lsk)$ twice: once when sending its Diffie–Hellman ephemeral public key $g^{\widetilde{esk}}$, and once when computing the Diffie–Hellman shared secrets from the received values. This is to avoid storing a single value that, when compromised, can be used to compute the session key.

Moving to the leakage-resilient setting requires rethinking the NAXOS trick. Alawatugoda et al. [3] have proposed a generic construction of an after-the-fact leakage eCK ((·)AFL-eCK)-secure key exchange protocol, which uses a leakage-resilient NAXOS trick. The leakage-resilient NAXOS trick is obtained using a decryption function of an after-the-fact leakage-resilient public key encryption scheme. A concrete construction of a BAFL-eCK-secure protocol is possible since there exists a bounded after-the-fact leakage-resilient public key encryption scheme which can be used to obtain the required leakage-resilient NAXOS trick, but it is not possible to construct a CAFL-eCK-secure protocol since there is no continuous after-the-fact leakage-resilient scheme available. Therefore, an attempt to construct a CAFL-eCK-secure key exchange protocol using the leakage-resilient NAXOS approach is not likely at this stage.

4.1 Description of Protocol P1

Our aim is to construct an eCK-secure key exchange protocol which does not use the NAXOS trick, but combines long-term secret keys and ephemeral secret keys to compute the session key, in a way that guarantees eCK security of the protocol. The protocol P1 shown in Table 2 is a Diffie–Hellman-type [10] key agreement protocol. Let $\mathbb{G}$ be a group of prime order q and generator g. After exchanging the public values both principals compute a Diffie–Hellman-type shared secret, and then compute the session key using a random oracle H. We use the random oracle because otherwise it is not possible to perfectly simulate the interaction between the adversary and the protocol, in a situation where the simulator does not know a long-term secret key of a protocol principal.

In order to compute the session key, protocol P1 combines four components ($Z_1 \leftarrow B^a$, $Z_3 \leftarrow Y^a$, $Z_4 \leftarrow Y^x$, $Z_2 \leftarrow B^x$) using the random oracle function H. These four components cannot be recovered by the attacker without both the ephemeral and long-term secret keys of at least one protocol principal, which allows a proof of eCK security.

Though the design of protocol P1 is quite straightforward, we could not find it given explicitly in the literature: most work on the design of eCK-secure protocols seeks to create more efficient protocols than this naive protocol, but the naive protocol is more appropriate for building into a leakage-resilient protocol.

Alice (Initiator)		Bob (Responder)
	Initial Setup	
$a \xleftarrow{\$} \mathbb{Z}_q^*, A \leftarrow g^a$		$b \xleftarrow{\$} \mathbb{Z}_q^*, B \leftarrow g^b$
	Protocol Execution	
$x \xleftarrow{\$} \mathbb{Z}_q^*, X \leftarrow g^x$	$\xrightarrow{Alice, X}$ $\xleftarrow{Bob, Y}$	$y \xleftarrow{\$} \mathbb{Z}_q^*, Y \leftarrow g^y$
$Z_1 \leftarrow B^a, Z_2 \leftarrow B^x$ $Z_3 \leftarrow Y^a, Z_4 \leftarrow Y^x$ $K \leftarrow \mathrm{H}(Z_1, Z_2, Z_3, Z_4, Alice, X, Bob, Y)$		$Z_1' \leftarrow A^b, Z_2' \leftarrow X^b$ $Z_3' \leftarrow A^y, Z_4' \leftarrow X^y$ $K \leftarrow \mathrm{H}(Z_1', Z_2', Z_3', Z_4', Alice, X, Bob, Y)$
	K is the session key	

Table 2. Protocol P1

Leakage-Resilient Rethinking of Protocol P1. Moving to the leakage-resilient setting requires rethinking the exponentiation computation in a leakage-resilient manner. Since there exist leakage-resilient encoding schemes and leakage-resilient refreshing protocols for them (Definition 4 and 7) our aim is computing the required exponentiations in a leakage-resilient manner using the available leakage-resilient storage and refreshing schemes. For now we look at the eCK security of protocol P1, and later in Section 5 we will look at the leakage-resilient modification to protocol P1 in detail.

4.2 Security Analysis of Protocol P1

Theorem 2. *If* H *is modeled as a random oracle and* $\mathbb{G}$ *is a group of prime order* q *and generator* g*, where the gap Diffie–Hellman* (GDH) *problem is hard, then protocol* P1 *is secure in the* eCK *model.*

Let $\mathcal{U} = \{U_1, \ldots, U_{N_P}\}$ *be a set of* N_P *parties. Each party* U_i *owns at most* N_S *number of protocol sessions. Let* $\mathcal{A}$ *be an adversary against protocol* P1*. Then,* $\mathcal{B}$ *is an algorithm which is constructed using the adversary* $\mathcal{A}$*, against the* GDH *problem such that the advantage of* $\mathcal{A}$ *against the* eCK*-security of protocol* P1*,* $\mathrm{Adv}_{\mathrm{P1}}^{\mathrm{eCK}}$ *is:*

$$\mathrm{Adv}_{\mathrm{P1}}^{\mathrm{eCK}}(\mathcal{A}) \leq \max\Big(N_P^2 N_S^2\big(\mathrm{Pr}_{g,q}^{\mathrm{GDH}}(\mathcal{B})\big), N_P^2\big(\mathrm{Pr}_{g,q}^{\mathrm{GDH}}(\mathcal{B})\big), N_P^2 N_S\big(\mathrm{Pr}_{g,q}^{\mathrm{GDH}}(\mathcal{B})\big), \\ N_P^2 N_S\big(\mathrm{Pr}_{g,q}^{\mathrm{GDH}}(\mathcal{B})\big), N_P^2 N_S\big(\mathrm{Pr}_{g,q}^{\mathrm{GDH}}(\mathcal{B})\big), N_P^2\big(\mathrm{Pr}_{g,q}^{\mathrm{GDH}}(\mathcal{B})\big)\Big) \ .$$

Proof Sketch: Let $\mathbf{A}$ denote the event that $\mathcal{A}$ wins the eCK challenge. Let $\mathbf{H}$ denote the event that $\mathcal{A}$ queries the random oracle H with $(\mathrm{CDH}(A^*, B^*), \mathrm{CDH}(B^*, X^*), \mathrm{CDH}(A^*, Y^*), \mathrm{CDH}(X^*, Y^*), \mathit{initiator}, X, \mathit{responder}, Y)$, where A^*, B^* are the long-term public-keys of the two partners to the test session, and X^*, Y^* are their ephemeral public keys for this session. Note that when $A = g^a, B = g^b, \mathrm{CDH}(A, B) = g^{ab}$; also *initiator* is the initiator of the session and *responder* is the responder of the session.

$$\Pr(\mathbf{A}) \leq \Pr(\mathbf{A} \wedge \mathbf{H}) + \Pr(\mathbf{A} \wedge \bar{\mathbf{H}}) \ .$$

Without the event $\mathbf{H}$ occurring, the session key given as the answer to the `Test` query is random-looking to the adversary, and therefore $\Pr(\mathbf{A}|\bar{\mathbf{H}}) = \frac{1}{2}$. $\Pr(\mathbf{A} \wedge \bar{\mathbf{H}}) = \Pr(\mathbf{A}|\bar{\mathbf{H}}) \Pr(\bar{\mathbf{H}})$, and therefore $\Pr(\mathbf{A} \wedge \bar{\mathbf{H}}) \leq \frac{1}{2}$. Hence,

$$\Pr(\mathbf{A}) \leq \frac{1}{2} + \Pr(\mathbf{A} \wedge \mathbf{H}),$$

that is $\Pr(\mathbf{A} \wedge \mathbf{H}) = \mathrm{Adv}_{\mathrm{P1}}^{\mathrm{eCK}}(\mathcal{A})$. Henceforth, the event $(\mathbf{A} \wedge \mathbf{H})$ is denoted as $\mathbf{A}^*$.

Note 1. Let $\mathcal{B}$ be an algorithm against a GDH challenger. $\mathcal{B}$ receives $L = g^{\ell}, W = g^{w}$ as the GDH challenge and $\mathcal{B}$ has access to a DDH oracle, which outputs 1 if the input is a tuple of $(g^{\alpha}, g^{\beta}, g^{\alpha\beta})$. $\Omega : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}$ is a random function known only to $\mathcal{B}$, such that $\Omega(\Phi, \Theta) = \Omega(\Theta, \Phi)$ for all $\Phi, \Theta \in \mathbb{G}$. $\mathcal{B}$ will use $\Omega(\Phi, \Theta)$ as $\mathrm{CDH}(\Phi, \Theta)$ in situations where $\mathcal{B}$ does not know $\log_g \Phi$ and $\log_g \Theta$. Except with negligible probability, $\mathcal{A}$ will not recognize that $\Omega(\Phi, \Theta)$ is being used as $\mathrm{CDH}(\Phi, \Theta)$.

We construct the algorithm $\mathcal{B}$ using $\mathcal{A}$ as a sub-routine. $\mathcal{B}$ receives $L = g^{\ell}, W = g^{w}$ as the GDH challenge. We consider the following mutually exclusive events, under two main cases:

1. A partner to the test session exists: the adversary is allowed to corrupt both principals or reveal ephemeral keys from both sessions of the test session.
   (a) Adversary corrupts both the owner and partner principals to the test session - Event $\mathbf{E_{1a}}$
   (b) Adversary corrupts neither owner nor partner principal to the test session - Event $\mathbf{E_{1b}}$
   (c) Adversary corrupts the owner to the test session, but does not corrupt the partner to the test session - Event $\mathbf{E_{1c}}$
   (d) Adversary corrupts the partner to the test session, but does not corrupt the owner to the test session - Event $\mathbf{E_{1d}}$
2. A partner to the test session does not exist: the adversary is not allowed to corrupt the intended partner principal to the test session.
   (a) Adversary corrupts the owner to the test session - Event $\mathbf{E_{2a}}$
   (b) Adversary does not corrupt the owner to the test session - Event $\mathbf{E_{2b}}$

In any other situation the test session is no longer fresh. If event $\mathbf{A}^*$ happens at least one of the following event should happen.

$$[(\mathbf{E_{1a}} \wedge \mathbf{A}^*), (\mathbf{E_{1b}} \wedge \mathbf{A}^*), (\mathbf{E_{1c}} \wedge \mathbf{A}^*), (\mathbf{E_{1d}} \wedge \mathbf{A}^*), (\mathbf{E_{2a}} \wedge \mathbf{A}^*), (\mathbf{E_{2b}} \wedge \mathbf{A}^*)]$$

Hence,

$$\mathrm{Adv}_{\mathrm{P1}}^{\mathrm{eCK}} \leq \max \Big(\Pr(\mathbf{E_{1a}} \wedge \mathbf{A}^*), \Pr(\mathbf{E_{1b}} \wedge \mathbf{A}^*), \Pr(\mathbf{E_{1c}} \wedge \mathbf{A}^*), \Pr(\mathbf{E_{1d}} \wedge \mathbf{A}^*), \Pr(\mathbf{E_{2a}} \wedge \mathbf{A}^*), \Pr(\mathbf{E_{2b}} \wedge \mathbf{A}^*) \Big) .$$

Complete security analysis of each event is available in the full version of this paper [2]. □

5 Protocol P2: A Leakage-Resilient Version of P1

Protocol P1 is an eCK-secure key exchange protocol (Theorem 2). The eCK model considers an environment where partial information leakage does not take place. Following the concept that only computation leaks information, we now assume that the leakage of long-term secret keys happens when computations are performed using them. Then, instead of the *non-leakage* eCK model which we used for the security proof of protocol P1, we consider the CAFL-eCK model which additionally allows the adversary to obtain continuous leakage of long-term secret keys.

Our idea is to perform the computations which use long-term secret keys (exponentiation operations) in such a way that the resulting leakage from the long-term secrets should not leak sufficient information to reveal them to the adversary. To overcome that challenge we use a leakage-resilient storage scheme and a leakage-resilient refreshing protocol, and modify the architecture of protocol P1, in such a way that the secret keys s are encoded into two portions s_L, s_R, Exponentiations are computed using two portions s_L, s_R instead of directly using s, and the two portions s_L, s_R are being refreshed continuously. Thus, we add leakage resiliency to the eCK-secure protocol P1 and construct protocol P2 such that it is leakage-resilient and eCK-secure.

Obtaining Leakage Resiliency by Encoding Secrets. In this setting we encode a secret s using an Encode function of a leakage-resilient storage scheme $\Lambda = (\text{Encode}, \text{Decode})$. So the secret s is encoded as $(s_L, s_R) \leftarrow \text{Encode}(s)$. As mentioned in the Definition 2.4.1 the leakage-resilient storage scheme randomly chooses s_L and then computes s_R such that $s_L \cdot s_R = s$. We define the tuple leakage parameter $\boldsymbol{\lambda} = (\lambda_1, \lambda_2)$ as follows: $\boldsymbol{\lambda}$-limited adversary $\mathcal{A}$ sends a leakage function $\mathbf{f} = (f_{1j}, f_{2j})$ and obtains at most λ_1, λ_2 amount of leakage from each of the two encodings of the secret s respectively: $f_{1j}(s_L)$ and $f_{2j}(s_R)$.

As mentioned in Definition 7, the leakage-resilient storage scheme can continuously refresh the encodings of the secret. Therefore, after executing the refreshing protocol it outputs new random-looking encodings of the same secret. So for the $\boldsymbol{\lambda}$-limited adversary again the situation is as before. Thus, refreshing the encodings will help to obtain leakage resilience over a number of protocol executions.

The computation of exponentiations is also split into two parts. Let $\mathbb{G}$ be a group of prime order q with generator g. Let $s \xleftarrow{\$} \mathbb{Z}_q^*$ be a long-term secret key and $E = g^e$ be a received ephemeral value. Then, the value Z needs to be computed as $Z \leftarrow E^s$. In the leakage-resilient setting, in the initial setup the secret key is encoded as $s_L, s_R \leftarrow \text{Encode}^{n,1}_{\mathbb{Z}_q^*}(s)$. So the vector $s_L = (s_{L1}, \cdots, s_{Ln})$ and the vector $s_R = (s_{R1}, \cdots, s_{Rn})$ are such that $s = s_{L1}s_{R1} + \cdots + s_{Ln}s_{Rn}$. Then the computation of E^s can be performed as two component-wise computations as follows: compute the intermediate vector $T \leftarrow E^{s_L} = (E^{s_{L1}}, \cdots, E^{s_{Ln}})$ and then compute the element $Z \leftarrow T^{s_R} = E^{s_{L1}s_{R1}}E^{s_{L2}s_{R2}} \cdots E^{s_{L1}s_{R1}} = E^{s_{L1}s_{R1} + \cdots + s_{Ln}s_{Rn}} = E^s$.

5.1 Description of Protocol P2

Using the above ideas, by encoding the secret using a leakage-resilient storage scheme, and refreshing the encoded secret using a refreshing protocol, it is possible to hide the secret from a $\boldsymbol{\lambda}$-limited adversary. Further, it is possible to successfully compute the exponentiation using the encoded secrets. We now use these primitives to construct a CAFL-eCK-secure key exchange protocol, using an eCK-secure key exchange protocol as an underlying primitive.

Let $\Lambda^{n,1}_{\mathbb{Z}^*_q} = (\text{Encode}^{n,1}_{\mathbb{Z}^*_q}, \text{Decode}^{n,1}_{\mathbb{Z}^*_q})$ be the leakage-resilient storage scheme which is used to encode secret keys and $\text{Refresh}^{n,1}_{\mathbb{Z}^*_q}$ be the $(\ell, \boldsymbol{\lambda}, \epsilon)$-secure leakage-resilient refreshing protocol of $\Lambda^{n,1}_{\mathbb{Z}^*_q}$.

As we can see, the obvious way of key generation (initial setup) in a protocol principal of this protocol is as follows: first pick $a \xleftarrow{\$} \mathbb{Z}^*_q$ as the long-term secret key, then encode the secret key as $(a^0_L, a^0_R) \leftarrow \text{Encode}^{n,1}_{\mathbb{Z}^*_q}(a)$, then compute the long-term public key $A = g^a$ using the two encodings (a^0_L, a^0_R), and finally erase a from the memory. The potential threat to that key generation mechanism is that even though the long-term secret key a is erased from the memory, it might not be properly erased and can be leaked to the adversary during the key generation. In order to avoid such a vulnerability, we randomly picks two values $a^0_L \xleftarrow{\$} (\mathbb{Z}^*_q)^n \backslash \{(0^n)\}$, $a^0_R \xleftarrow{\$} (\mathbb{Z}^*_q)^{n\times 1} \backslash \{(0^{n\times 1})\}$ and use them as the encodings of the long-term secret key a of a protocol principal. As explained earlier, we use a^0_L, a^0_R to compute the corresponding long-term public key A in two steps as $a' \leftarrow g^{a^0_L}$ and $A \leftarrow a'^{a^0_R}$. Thus, it is possible to avoid exposing the un-encoded secret key a at any point of time in the key generation and hence avoid leaking directly from a at the key generation step. Further, the random vector a^0_L is multiplied with the random vector a^0_R, such that $a = a^0_L \cdot a^0_R$, which will give a random integer a in the group $\mathbb{Z}^*_q$. Therefore, this approach is same as picking $a \xleftarrow{\$} \mathbb{Z}^*_q$ at first and then encode, but in the reverse order. During protocol execution both a^0_L, a^0_R are continuously refreshed and refreshed encodings a^j_L, a^j_R are used to exponentiation computations.

Table 3 shows protocol P2. In this setting leakage of a long-term secret key does not happen directly from the long-term secret key itself, but from the two encodings of the long-term secret key (the leakage function $\mathbf{f} = (f_{1j}, f_{2j})$ directs to the each individual encoding). During the exponentiation computations and the refreshing operation collectively at most $\boldsymbol{\lambda} = (\lambda_1, \lambda_2)$ leakage is allowed to the adversary from each of the two portions independently. Then, the two portions of the encoded long-term secret key are refreshed and in the next protocol session another $\boldsymbol{\lambda}$-bounded leakage is allowed. Thus, continuous leakage is allowed.

5.2 Security Analysis of Protocol P2

Theorem 3. *If the underlying refreshing protocol* $\text{Refresh}^{n,1}_{\mathbb{Z}^*_q}$ *is* $(\ell, \boldsymbol{\lambda}, \epsilon)$*-secure leakage-resilient refreshing protocol of the leakage-resilient storage scheme* $\Lambda^{n,1}_{\mathbb{Z}^*_q}$

Alice (Initiator)		Bob (Responder)
Initial Setup		
$a_L^0 \xleftarrow{\$} (\mathbb{Z}_q^*)^n \backslash \{(0^n)\}, a_R^0 \xleftarrow{\$} (\mathbb{Z}_q^*)^{n\times 1} \backslash \{(0^{n\times 1})\}$		$b_L^0 \xleftarrow{\$} (\mathbb{Z}_q^*)^n \backslash \{(0^n)\}, b_R^0 \xleftarrow{\$} (\mathbb{Z}_q^*)^{n\times 1} \backslash \{(0^{n\times 1})\}$
$a' \leftarrow g^{a_L^0}, A \leftarrow (a')^{a_R^0}$		$b' \leftarrow g^{b_L^0}, B \leftarrow (b')^{b_R^0}$
Protocol Execution		
$x \xleftarrow{\$} \mathbb{Z}_q^*, X \leftarrow g^x$	$\xrightarrow{Alice, X}$ $\xleftarrow{Bob, Y}$	$y \xleftarrow{\$} \mathbb{Z}_q^*, Y \leftarrow g^y$
$T_1 \leftarrow B^{a_L^j}, Z_1 \leftarrow T_1^{a_R^j}$		$T_3 \leftarrow A^{b_L^j}, Z_1' \leftarrow T_3^{b_R^j}$
$Z_2 \leftarrow B^x$		$T_4 \leftarrow X^{b_L^j}, Z_2' \leftarrow T_4^{b_R^j}$
$T_2 \leftarrow Y^{a_L^j}, Z_3 \leftarrow T_2^{a_R^j}$		$Z_3' \leftarrow A^y$
$Z_4 \leftarrow Y^x$		$Z_4' \leftarrow X^y$
$(a_L^{j+1}, a_R^{j+1}) \leftarrow \text{Refresh}_{\mathbb{Z}_q^*}^{n,1}(a_L^j, a_R^j)$		$(b_L^{j+1}, b_R^{j+1}) \leftarrow \text{Refresh}_{\mathbb{Z}_q^*}^{n,1}(b_L^j, b_R^j)$
$K \leftarrow \text{H}(Z_1, Z_2, Z_3, Z_4, Alice, X, Bob, Y)$		$K \leftarrow \text{H}(Z_1', Z_2', Z_3', Z_4', Alice, X, Bob, Y)$
K is the session key		

Table 3. Concrete construction of Protocol P2

and the underlying key exchange protocol P1 *is* eCK*-secure key exchange protocol, then protocol* P2 *is* $\boldsymbol{\lambda}$-CAFL-eCK*-secure.*

Let $\mathcal{A}$ *be an adversary against the key exchange protocol* P2*. Then the advantage of* $\mathcal{A}$ *against the* CAFL-eCK*-security of protocol* P2 *is:*

$$\text{Adv}_{\text{P2}}^{\boldsymbol{\lambda}-\text{CAFL-eCK}}(\mathcal{A}) \leq N_P\Big(\text{Adv}_{\text{P1}}^{\text{eCK}}(\mathcal{A}) + \epsilon\Big) \ .$$

Proof. The proof proceeds by a sequence of games.

- **Game 1.** This is the original game.
- **Game 2.** Same as Game 1 with the following exception: before $\mathcal{A}$ begins, an identity of a random principal $U^* \xleftarrow{\$} \{U_1, \ldots, U_{N_P}\}$ is chosen. Challenger expects that the adversary will issue the `Test` for a session which involves the principal U^* ($\Pi_{U^*,\cdot}^{\cdot}$ or $\Pi_{\cdot,U^*}^{\cdot}$). If not the challenger aborts the game.
- **Game 3.** Same as Game 2 with the following exception: challenger picks a random $s \xleftarrow{\$} \mathbb{Z}_q^*$ and uses encodings of s to simulate the adversarial leakage queries $\mathbf{f} = (f_{1j}, f_{2j})$ of the principal U^*.

We now analyze the adversary's advantage of distinguishing each game from the previous game. Let $\text{Adv}_{\text{Game } x}(\mathcal{A})$ denote the advantage of the adversary $\mathcal{A}$ winning Game x.

Game 1 is the original game. Hence,

$$\text{Adv}_{\text{Game } 1}(\mathcal{A}) = \text{Adv}_{\text{P2}}^{\boldsymbol{\lambda}-\text{CAFL-eCK}}(\mathcal{A}) \ . \tag{1}$$

Game 1 and Game 2. The probability of Game 2 to be halted due to incorrect choice of the test session is $1 - \frac{1}{N_P}$. Unless the incorrect choice happens, Game 2 is identical to Game 1. Hence,

$$\text{Adv}_{\text{Game } 2}(\mathcal{A}) = \frac{1}{N_P}\text{Adv}_{\text{Game } 1}(\mathcal{A}) \ . \tag{2}$$

Game 2 and Game 3. We construct an algorithm $\mathcal{B}$ against a leakage-resilient refreshing protocol challenger of $\text{Refresh}^{n,1}_{\mathbb{Z}^*_q}$, using the adversary $\mathcal{A}$ as a subroutine.

The $(\ell, \boldsymbol{\lambda}, \epsilon)$-$\text{Refresh}^{n,1}_{\mathbb{Z}^*_q}$ refreshing protocol challenger chooses $s_0, s_1 \xleftarrow{\$} \mathbb{Z}^*_q$ and sends them to the algorithm $\mathcal{B}$. Further, the refreshing protocol challenger randomly chooses $s \xleftarrow{\$} \{s_0, s_1\}$ and uses s as the secret to compute the leakage from encodings of s. Let $\boldsymbol{\lambda} = (\lambda_1, \lambda_2)$ be the leakage bound and the refreshing protocol challenger continuously refresh the two encodings of the secret s.

When the algorithm $\mathcal{B}$ gets the challenge of s_0, s_1 from the refreshing protocol challenger, $\mathcal{B}$ uses s_0 as the secret key of the protocol principal U^* and computes the corresponding public key. For all other protocol principals $\mathcal{B}$ sets secret/public key pairs by itself. Using the setup keys, $\mathcal{B}$ computes answers to all the queries from $\mathcal{A}$ and simulates the view of CAFL-eCK challenger of protocol P2. $\mathcal{B}$ computes the leakage of secret keys by computing the adversarial leakage function $\mathbf{f}$ on the corresponding secret key (encodings of secret key), except the secret key of the protocol principal U^*. In order to obtain the leakage of the secret key of U^*, algorithm $\mathcal{B}$ queries the refreshing protocol challenger with the adversarial leakage function $\mathbf{f}$, and passes that leakage to $\mathcal{A}$.

If the secret s chosen by the refreshing protocol challenger is s_0, the leakage of the secret key of U^* simulated by $\mathcal{B}$ (with the aid of the refreshing protocol challenger) is the real leakage. Then the simulation is identical to Game 2. Otherwise, the leakage of the secret key of U^* simulated by $\mathcal{B}$ is a leakage of a random value. Then the simulation is identical to Game 3. Hence,

$$|\text{Adv}_{\text{Game 2}}(\mathcal{A}) - \text{Adv}_{\text{Game 3}}(\mathcal{A})| \leq \epsilon \ . \tag{3}$$

Game 3. Since the leakage is computed using a random s value, the adversary $\mathcal{A}$ will not get any advantage due to the leakage. Therefore, the advantage $\mathcal{A}$ will get is same as the advantage that $\mathcal{A}$ has against eCK challenger of protocol P1. Because both P1 and P2 are effectively doing the same computation, regardless of the protocol P2, and with no useful leakage the CAFL-eCK model is same as the eCK model. Hence,

$$\text{Adv}_{\text{Game 3}}(\mathcal{A}) = \text{Adv}^{\text{eCK}}_{\text{P1}}(\mathcal{A}) \ . \tag{4}$$

We find,

$$\text{Adv}^{\boldsymbol{\lambda}-\text{CAFL-eCK}}_{\text{P2}}(\mathcal{A}) \leq N_P \Big(\text{Adv}^{\text{eCK}}_{\text{P1}}(\mathcal{A}) + \epsilon \Big) \ .$$

□

5.3 Leakage Tolerance of Protocol P2

The order of the group $\mathbb{G}$ is q. Let $m = 1$ in the leakage-resilient storage scheme $\Lambda^{n,1}_{\mathbb{Z}^*_q}$. According to the Lemma 1, if $m < n/20$, then the leakage parameter for the leakage-resilient storage scheme is $\boldsymbol{\lambda}_\Lambda = (0.3n \log q, 0.3n \log q)$. Let $n = 21$, then $\boldsymbol{\lambda}_\Lambda = (6.3 \log q, 6.3 \log q)$ bits. According to the Theorem 1, if $m/3 \leq n$ and $n \geq 16$, the refreshing protocol $\text{Refresh}^{n,1}_{\mathbb{Z}^*_q}$ of the leakage-resilient storage scheme $\Lambda^{n,1}_{\mathbb{Z}^*_q}$

is tolerant to (continuous) leakage up to $\boldsymbol{\lambda}_{\text{Refresh}} = \boldsymbol{\lambda}_{\Lambda}/2 = (3.15 \log q, 3.15 \log q)$ bits, per occurrence.

When a secret key s (of size $\log q$ bits) of protocol P2 is encoded into two parts, the left part s_L will be $n \cdot \log q = 21 \log q$ bits and the right part s_R will be $n \cdot 1 \cdot \log q = 21 \log q$ bits. For a tuple leakage function $\mathbf{f} = (f_{1j}, f_{2j})$ (each leakage function $f_{(\cdot)}$ for each of the two parts s_L and s_R), there exists a tuple leakage bound $\boldsymbol{\lambda} = (\lambda, \lambda)$ for each leakage function $f_{(\cdot)}$, such that $\lambda = 3.15 \log q$ bits, per occurrence, which is $\frac{3.15 \log q}{21 \log q} \times 100\% = 15\%$ of the size of a part. The overall leakage amount is unbounded since continuous leakage is allowed.

6 Conclusion

In this paper we answered that open problem of constructing a concrete CAFL-eCK secure key exchange protocol by using a leakage-resilient storage scheme and its refreshing protocol. The main technique used to achieve after-the-fact leakage resilience is encoding the secret key into two parts and only allowing the independent leakage from each part. As future work it is worthwhile to investigate whether there are other techniques to achieve after-the-fact leakage resilience, rather than encoding the secret into parts. Moving to the standard model is another possible research direction. Strengthening the security model, by not just restricting to the independent leakage from each part, would be a more challenging research direction.

Acknowledgements

This research was supported in part by Australian Research Council (ARC) Discovery Project grant DP130104304.

References

1. A. Akavia, S. Goldwasser, and V. Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In *Theory of Cryptology Conference*, pages 474–495, 2009.
2. J. Alawatugoda, C. Boyd, and D. Stebila. Continuous after-the-fact leakage-resilient eck-secure key exchange. *IACR Cryptology ePrint Archive*, 2015:335, 2015.
3. J. Alawatugoda, D. Stebila, and C. Boyd. Modelling after-the-fact leakage for key exchange. In *ASIACCS*, 2014.
4. J. Alwen, Y. Dodis, and D. Wichs. Leakage-resilient public-key cryptography in the bounded-retrieval model. In *CRYPTO*, pages 36–54, 2009.
5. M. Bellare and P. Rogaway. Entity authentication and key distribution. In *CRYPTO*, pages 232–249, 1993.
6. D. J. Bernstein. Cache-timing attacks on AES. Technical report, 2005. http://cr.yp.to/antiforgery/cachetiming-20050414.pdf.
7. Z. Brakerski, Y. T. Kalai, J. Katz, and V. Vaikuntanathan. Overcoming the hole in the bucket: Public-key cryptography resilient to continual memory leakage. *IACR Cryptology ePrint Archive*, Report 2010/278, 2010.

8. D. Brumley and D. Boneh. Remote timing attacks are practical. In *USENIX Security Symposium*, pages 1–14, 2003.
9. R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *EUROCRYPT*, pages 453–474, 2001.
10. W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, pages 644 – 654, 1976.
11. Y. Dodis, K. Haralambiev, A. López-Alt, and D. Wichs. Efficient public-key cryptography in the presence of key leakage. In *ASIACRYPT*, pages 613–631, 2010.
12. S. Dziembowski and S. Faust. Leakage-resilient cryptography from the inner-product extractor. In *ASIACRYPT*, pages 702–721, 2011.
13. S. Dziembowski and K. Pietrzak. Leakage-resilient cryptography. In *IEEE Symposium on Foundations of Computer Science*, pages 293–302, 2008.
14. S. Faust, E. Kiltz, K. Pietrzak, and G. N. Rothblum. Leakage-resilient signatures. *IACR Cryptology ePrint Archive*, Report 2009/282, 2009.
15. M. Hutter, S. Mangard, and M. Feldhofer. Power and EM attacks on passive 13.56MHz RFID devices. In *CHES*, pages 320–333, 2007.
16. J. Katz and V. Vaikuntanathan. Signature schemes with bounded leakage resilience. In *ASIACRYPT*, pages 703–720, 2009.
17. E. Kiltz and K. Pietrzak. Leakage resilient elgamal encryption. In *ASIACRYPT*, pages 595–612, 2010.
18. P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *CRYPTO*, pages 104–113, 1996.
19. B. LaMacchia, K. Lauter, and A. Mityagin. Stronger security of authenticated key exchange. In *ProvSec*, pages 1–16, 2007.
20. T. Malkin, I. Teranishi, Y. Vahlis, and M. Yung. Signatures resilient to continual leakage on memory and computation. In *Theory of Cryptology Conference*, pages 89–106, 2011.
21. T. Messerges, E. Dabbish, and R. Sloan. Examining smart-card security under the threat of power analysis attacks. *IEEE Transactions on Computers*, pages 541–552, 2002.
22. D. Moriyama and T. Okamoto. Leakage resilient eCK-secure key exchange protocol without random oracles. In *ASIACCS*, pages 441–447, 2011.
23. G. Yang, Y. Mu, W. Susilo, and D. S. Wong. Leakage resilient authenticated key exchange secure in the auxiliary input model. In *ISPEC*, pages 204–217, 2013.