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

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

[1] *School of Electrical Engineering and Computer Science, Queensland University of Technology, Brisbane, Australia*

[2] *Mathematical Sciences School, 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

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

The authors are grateful to Professor Tatsuaki Okamoto for valuable discussions on leakage-resilient cryptography, while Janaka Alawatugoda is visiting NTT Secure Platform Laboratories, NTT Corporation in Musashino-shi, Tokyo 180-8585, Japan for an internship program.

Contents

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 [5, 7, 17], electromagnetic radiation [14], and power consumption [20]. 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, 3, 6, 12, 13, 16, 15, 19]. 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 [4], Canetti–Krawczyk [8], and extended Canetti–Krawczyk (eCK) [18] 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 [3, 10, 21, 23, 2]. Among them the generic security model proposed by Alawatugoda et al. [2] in 2014 facilitates more granular leakage.

Alawatugoda et al. [2] 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. [2] 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. Up to 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 a *continuous* and *after-the-fact* leakage-resilient key exchange protocol.

1.1 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.

1.2 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 [4, 8, 18], After-the-fact leakage is the leakage which happens after the test session is established.

1.3 Our Contribution.

Alawatugoda et al. [2] 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 [11] for this construction.

Table 1 compares the proposed protocol P2, with the NAXOS protocol [18], the Moriyama-Okamoto (MO) protocol [21] and the generic Alawatugoda et al. [2] 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 [18]	4 **Exp**	4 **Exp**	None	None	Random oracle
MO [21]	8 **Exp**	8 **Exp**	Bounded	No	Standard
Alawatugoda et al. [2]	12 **Exp**	12 **Exp**	Bounded	Yes	Standard
Protocol P2	6 **Exp**	6 **Exp**	Continuous	Yes	Random oracle

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

In the 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 Leakage-Resilient Storage

We review the definitions of leakage-resilient storage according to Dziembowski et al. [11]. 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 2.1 (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 $s \in \mathbb{Z}^{*m}_q$ where:

- $\text{Encode}^{n,m}_{\mathbb{Z}^*_q}(s) : s_L \xleftarrow{\$} \mathbb{Z}^{*n}_q \backslash \{(0^n)\}$, then $s_R \leftarrow \mathbb{Z}^{*n \times m}_q$ 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 2.2 (λ-limited adversary)**.** If the amount of leakage obtained by the adversary from each of s_L and s_R is limited to λ bits in total, the adversary is known as a λ*-limited adversary.*

Definition 2.3 ($(\lambda_\Lambda, \epsilon_1)$-secure leakage-resilient storage scheme)**.** We say $\Lambda = (\text{Encode}, \text{Decode})$ is $(\lambda_\Lambda, \epsilon_1)$-secure leakage-resilient, if for any $s_0, s_1 \xleftarrow{\$} \mathcal{M}$ and any λ_Λ-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 2.1 ([11])**.** *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* $(0.3 \cdot n \log q, negl(n))$*-secure for some negligible function negl.*

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 2.4 (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 $L \in \mathbb{Z}^{*n}_q$ and $R \in \mathbb{Z}^{*n \times m}_q$.
- 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. Sets $X := M \cdot B$ and $R' := R + X$.

- 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. $Y := \tilde{A} \cdot \tilde{M}$ and $L' := L + Y$.

- Output : (L', R')

Let $\Lambda = (\text{Encode}, \text{Decode})$ be a $(\lambda_\Lambda, \epsilon_1)$-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 λ_{Refresh}-limited adversary $\mathcal{B}$, $\ell \in \mathbb{N}$ and $s \xleftarrow{\$} \mathcal{M}$, 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 $i = 1$ to ℓ run $\mathcal{B}$ against the ith 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 $s \xleftarrow{\$} \{s_0, s_1\} \in \mathcal{M}$. Now we define leakage-resilient security of a refreshing protocol.

Definition 2.5 ($(\ell, \lambda_{\text{Refresh}}, \epsilon_2)$-secure Leakage-Resilient Refreshing Protocol)**.** For a $(\lambda_\Lambda, \epsilon_1)$-secure leakage-resilient storage scheme $\Lambda = (\text{Encode}, \text{Decode})$ with message space $\mathcal{M}$, Refresh is $(\ell, \lambda_{\text{Refresh}}, \epsilon_2)$-secure leakage-resilient, if for every λ_{Refresh}-limited adversary $\mathcal{B}$ and any two secrets $s_0, s_1 \in \mathcal{M}$, 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 2.2 ([11])**.** *Let $m/3 \le n, n \ge 16$ and $\ell \in \mathbb{N}$. Let n, m and ${\mathbb{Z}_q}^*$ be such that $\Lambda^{n,m}_{{\mathbb{Z}_q}^*}$ is (λ, ϵ)-secure leakage-resilient storage scheme (Definition 2.1 and Definition 2.3). Then the refreshing protocol $\text{Refresh}^{n,m}_{\mathbb{Z}_q^*}$ (Definition 2.4) is a $(\ell, \lambda/2, \epsilon')$-secure leakage-resilient refreshing protocol for $\Lambda^{n,m}_{{\mathbb{Z}_q}^*}$ (Definition 2.5) with $\epsilon' := 2\ell p(3p^m\epsilon + mp^{-n-1})$.*

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

In 2014 Alawatugoda et al. [2] proposed a new security model for key exchange protocols, namely the generic after-the-fact leakage eCK ($(\cdot)$AFL-eCK) model which, in addition to the adversarial capabilities of the eCK model [18], is equipped with an adversary-chosen leakage function $\mathbf{f}$, enabling the adversary to obtain the leakage of long-term secret keys of protocol principals. Therefore the $(\cdot)$AFL-eCK model captures all the attacks captured by the eCK model.

3.0.1 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.

3.0.2 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 3.1 (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

The most realistic 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 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 $\mathbf{f} = (f_1, f_2, \ldots, f_t)$ are introduced; 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 allowed to obtain leakage from each key part independently: the adversary cannot use the output of f_1 as an input parameter to the f_2 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 $\mathcal{A}$ is a probabilistic polynomial time algorithm that 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 $\mathcal{A}$ attempts 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 3.2 ($\boldsymbol{\lambda}$-CAFL-eCK-freshness)**.** Let $\boldsymbol{\lambda} = (\lambda_1, \dots, \lambda_t)$ 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 $\prod^s_{U,V}$ or its partner, $\prod^{s'}_{V,U}$ (if it exists) has not been asked a `SessionKeyReveal`.
2. If the partner $\prod^{s'}_{V,U}$ 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 $\prod^{s'}_{V,U}$ 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`$(\cdot, U, \cdot, \cdot, \mathbf{f})$ query, size of the output of $f_i(sk_{U\,i}) \leq \lambda_i$.
5. For each `Send`$(\cdot, V, \cdot, \cdot, \mathbf{f})$ queries, size of the output of $f_i(sk_{V\,i}) \leq \lambda_i$.

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 3.3 (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 probabilistic polynomial time 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:** $\mathcal{A}$ may continue 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 4 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 3.3.

Definition 3.4 (CAFL-eCK-security)**.** A protocol π is said to be CAFL-eCK-secure if there is no probabilistic polynomial time algorithm $\mathcal{A}$ that can win the CAFL-eCK security game with non-negligible advantage. The advantage of an adversary $\mathcal{A}$ is defined as $\mathrm{Adv}_{\pi}^{\text{CAFL-eCK}}(\mathcal{A}) = |2\Pr(\mathrm{Succ}_{\mathcal{A}}) - 1|$.

eCK-security definition is the same as the CAFL-eCK-security definition.

4 eCK-Secure Key Exchange: Protocol P1

The motivation of LaMacchia et al. [18] 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, [18], 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. [2] have proposed a generic construction of an after-the-fact leakage eCK $((\cdot)$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 Construction of the 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 [9] key agreement protocol. Let k be the security parameter, $\mathbb{G}$ 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, the 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 an one protocol principal, which allows a proof of the eCK security.

4.1.1 Leakage-Resilient Rethinking of the 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 2.1 and 2.4) 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 the protocol P1, and later in section 5 we will look at the leakage-resilient modification to the protocol P1 in detail.

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

4.2 Security Analysis of the Protocol P1

Theorem 4.1. *If* H *is modeled as a random oracle and* $\mathbb{G}$ *is a group of a prime order* q *and generator* g*, where gap Diffie-Hellman* (GDH) *assumption holds, then the protocol* P1 *is secure in the* eCK *model.*

Proof Sketch: Let k be the security parameter and the adversary $\mathcal{A}$ be polynomially bounded in k. Let $\mathbf{A}$ denotes the event that $\mathcal{A}$ wins the eCK challenge with probability $\frac{1}{2} + \mathrm{Adv}_{\mathrm{P1}}^{\mathrm{eCK}}$. Let $\mathbf{H}$ denote the event that $\mathcal{A}$ queries the random oracle H with $(\mathrm{CDH}(A, B), \mathrm{CDH}(B, X), \mathrm{CDH}(A, Y), \mathrm{CDH}(X, Y), \mathit{initiator}, X, \mathit{responder}, Y)$ as in the test session.

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

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

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

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

Note. 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 $(g^\alpha, g^\beta, g^{\alpha\beta})$. $\Omega : \mathbb{G} \times \mathbb{G} \to \mathbb{G}$ is a random function known only to $\mathcal{B}$ and $\mathcal{B}$ 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 find 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 complementary 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 oracles 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 or 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, but does not corrupt the partner 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 be no longer fresh. If event $\mathbf{A}^*$ happens with non-negligible probability at least one of the following event should happen with non-negligible probability.

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

Hence,

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

□

Complete security analysis of each event is described in Appendix A.

5 Leakage-Resilient Construction of the Protocol P1: Protocol P2

The protocol P1 is an eCK-secure key exchange protocol (Theorem 4.1). 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 the 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 the protocol P1, in 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, our aim is to add leakage resilience capability to the eCK-secure protocol P1 and construct protocol P2 such that it is leakage-resilient and eCK-secure.

5.0.1 Obtaining Leakage Resiliency by Encoding Secrets.

In this setting we encode a secret s using an Encode function of a leakage-resilient storage scheme $\Lambda = (\mathrm{Encode}, \mathrm{Decode})$. So the secret s is encoded as $(s_L, s_R) \leftarrow \mathrm{Encode}(s)$. As mentioned in the Definition 2.3 the leakage-resilient storage scheme randomly chooses s_L and then computes s_R such that $s_L \cdot s_R = s$. A λ-limited adversary $\mathcal{A}$ sends a leakage function $\mathbf{f} = (f_1, f_2)$ and obtains at most λ amount of leakage from each of the two encodings of the secret s: $f_1(s_L)$ and $f_2(s_R)$. The leakage $f_1(s_L)$ is random because s_L is random and since the restriction of independent leakage $f_2(s_R)$ is almost random. Thus for a λ-limited adversary, unless the λ bound is sufficient to learn the connection between s_L and s_R, it is hard to recover the secret s.

As mentioned in Definition 2.4, 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 λ-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 idea behind our construction is to split the compassion of exponentiations into two parts. Let $\mathbb{G}$ be a group of prime order q and 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 is computed as $Z \leftarrow E^s$. In the leakage-resilient setting, in the initial setup the secret key is encoded as $s_L, s_R \leftarrow \mathrm{Encode}_{\mathbb{Z}_q^*}^{n,1}(s)$. Then computation of E^s is performed as $T \leftarrow E^{s_L}, Z \leftarrow T^{s_R}$. Since $s_L \cdot s_R$ results in s, Z is the same value as E^s.

5.1 Construction of the Protocol P2

Using the above ideas, by encoding the secret by using a leakage-resilient storage scheme, and refreshing the encoded secret using a refreshing protocol, it is possible to hide the secret from a λ-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.

Assume that at the initial setup key generation and encoding take place in a leakage-free environment. 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 $(\ell, \lambda, \epsilon)$-secure leakage-resilient refreshing protocol of $\Lambda^{n,1}_{\mathbb{Z}_q{}^*}$.

Table 3 shows the 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 $\mathbf{f} = (f_1, f_2)$ directs to the each individual encoding). During the exponentiation computations and the refreshing operation collectively at most λ 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 λ-bounded leakage is allowed. Thus, continuous leakage is allowed.

Alice (Initiator)		Bob (Responder)
	Initial Setup	
$a \xleftarrow{\$} \mathbb{Z}^*_q, A \leftarrow g^a$		$b \xleftarrow{\$} \mathbb{Z}^*_q, B \leftarrow g^b$
$(a^0_L, a^0_R) \leftarrow \text{Encode}^{n,1}_{\mathbb{Z}^*_q}(a)$		$(b^0_L, b^0_R) \leftarrow \text{Encode}^{n,1}_{\mathbb{Z}^*_q}(b)$
Erase a		Erase b
	Protocol Execution	
$x \xleftarrow{\$} \mathbb{Z}^*_q, X \leftarrow g^x$	$\xrightarrow{Alice, X}$	$y \xleftarrow{\$} \mathbb{Z}^*_q, Y \leftarrow g^y$
	$\xleftarrow{Bob, Y}$	
$T_1 \leftarrow B^{a^i_L}, Z_1 \leftarrow T_1^{a^i_R}$		$T_3 \leftarrow A^{b^i_L}, Z'_1 \leftarrow T_3^{b^i_R}$
$Z_2 \leftarrow B^x$		$T_4 \leftarrow X^{b^i_L}, Z'_2 \leftarrow T_4^{b^i_R}$
$T_2 \leftarrow Y^{a^i_L}, Z_3 \leftarrow T_2^{a^i_R}$		$Z'_3 \leftarrow A^y$
$Z_4 \leftarrow Y^x$		$Z'_4 \leftarrow X^y$
$(a^{i+1}_L, a^{i+1}_R) \leftarrow \text{Refresh}^{n,1}_{\mathbb{Z}^*_q}(a^i_L, a^i_R)$		$(b^{i+1}_L, b^{i+1}_R) \leftarrow \text{Refresh}^{n,1}_{\mathbb{Z}^*_q}(b^i_L, b^i_R)$
$K \leftarrow \text{H}(Z_1, Z_2, Z_3, Z_4, Alice, X, Bob, Y)$		$K \leftarrow \text{H}(Z'_1, Z'_2, Z'_3, Z'_4, Alice, X, Bob, Y)$
	K is the session key	

Table 3: Protocol P2

5.2 Security Analysis of the Protocol P2

Theorem 5.1. *If the underlying refreshing protocol* $\text{Refresh}^{n,1}_{\mathbb{Z}^*_q}$ *is $(\ell, \lambda, \epsilon)$-secure leakage-resilient refreshing protocol of the leakage-resilient storage scheme* $\Lambda^{n,1}_{\mathbb{Z}_q{}^*}$ *and the underlying key exchange protocol* P1 *is* eCK*-secure key exchange protocol, then the protocol* P2 *is* CAFL-eCK*-secure for ℓ-rounds.*

Proof. Let $\text{Adv}^{\text{CAFL-eCK}}_{\text{P2}}$ be the advantage that the adversary $\mathcal{A}$ wins the CAFL-eCK challenge of the protocol P2.

In order to formally prove the CAFL-eCK security of the protocol P2, we use the game hopping technique [22]: define a sequence of games and relate the adversary's advantage of distinguishing each game from the previous game to the advantage of breaking the underlying cryptographic primitive.

Sequence of Games.

Game 1: This is the original game.

Game 2: Same as the 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 an oracle which involves the principal U^* ($\Pi^{\cdot}_{U^*,\cdot}$ or $\Pi^{\cdot}_{\cdot,U^*}$). If not the challenger aborts the game.

Game 3: Same as the 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 $\mathbf{f} = (f_1, f_2)$.

Differences between games.

In this section the adversary's advantage of distinguishing each game from the previous game is investigated. Let $\mathrm{Adv}_{\mathrm{Game}\ x}(\mathcal{A})$ denotes the advantage of the adversary $\mathcal{A}$ winning the Game x.

Game 1 is the original game. Hence,

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

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,

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

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

The $(\ell, \lambda, \epsilon)$-$\mathrm{Refresh}_{\mathbb{Z}_q^*}^{n,1}$ 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 λ be the leakage bound and the refreshing protocol challenger continuously refresh the 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 the 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 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,

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

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 $\mathcal{A}$ have against eCK challenger of the protocol P1. Because both P1 and P2 are effectively doing the same computation, regardless of the leakage-resilient exponentiation computation in the protocol P2, and with no useful leakage the CAFL-eCK model is same as the eCK model. Hence,

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

Using equations 1–4 we find,

$$\mathrm{Adv}_{\mathrm{P2}}^{\text{CAFL-eCK}}(\mathcal{A}) \leq N_P\Big(\mathrm{Adv}_{\mathrm{P1}}^{\mathrm{eCK}}(\mathcal{A}) + \epsilon\Big) \ . \tag{5}$$

□

5.3 Leakage Tolerance of the Protocol P2

The order of the group $\mathbb{G}$ is $q-1$. Let $m = 1$ in the leakage-resilient storage scheme $\Lambda_{\mathbb{Z}_q^*}^{n,1}$. According to the Lemma 2.1, if $m < n/20$, then the leakage parameter for the leakage-resilient storage scheme is $\lambda_\Lambda = 0.3n\log(q-1)$. Let $n = 21$, then $\lambda_\Lambda = 6.3\log(q-1)$ bits. According to Theorem 2.2, if $m/3 \leq n$ and $n \geq 16$, the refreshing protocol $\mathrm{Refresh}_{\mathbb{Z}_q^*}^{n,1}$ of the leakage-resilient storage scheme $\Lambda_{\mathbb{Z}_q^*}^{n,1}$ is tolerance to (continuous) leakage upto $\lambda_{\mathrm{Refresh}} = \lambda_\Lambda/2 = 3.15\log(q-1)$ bits, per occurrence.

When a secret key s (of size $\log(q-1)$ bits) of the protocol P2 is encoded into two parts, the left part s_L will be $n \cdot \log(q-1) = 21\log(q-1)$ bits and the right part s_R will be $n \cdot m \cdot \log(q-1) = 21\log(q-1)$

bits. For a tuple leakage function $\mathbf{f} = (f_1, f_2)$ (each leakage function $f_{(\cdot)}$ for each of the two parts s_L and s_R), there exists a tuple leakage bound $\boldsymbol{\lambda} = (\lambda_1, \lambda_2)$ for each leakage function $f_{(\cdot)}$, such that $\lambda_1 = \lambda_2 = 3.15 \log(q-1)$ bits, per occurrence, which is $\frac{3.15 \log(q-1)}{21 \log(q-1)} \times 100\% = 15\%$ of the size of a part. The overall leakage amount in 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.

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, D. Stebila, and C. Boyd. Modelling after-the-fact leakage for key exchange. In *ASIACCS*, 2014.

[3] J. Alwen, Y. Dodis, and D. Wichs. Leakage-resilient public-key cryptography in the bounded-retrieval model. In *CRYPTO*, pages 36–54, 2009.

[4] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *CRYPTO*, pages 232–249, 1993.

[5] D. J. Bernstein. Cache-timing attacks on AES. Technical report, 2005. http://cr.yp.to/antiforgery/cachetiming-20050414.pdf.

[6] 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.

[7] D. Brumley and D. Boneh. Remote timing attacks are practical. In *USENIX Security Symposium*, pages 1–14, 2003.

[8] R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *EUROCRYPT*, pages 453–474, 2001.

[9] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, pages 644 – 654, 1976.

[10] 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.

[11] S. Dziembowski and S. Faust. Leakage-resilient cryptography from the inner-product extractor. In *ASIACRYPT*, pages 702–721, 2011.

[12] S. Dziembowski and K. Pietrzak. Leakage-resilient cryptography. In *IEEE Symposium on Foundations of Computer Science*, pages 293–302, 2008.

[13] S. Faust, E. Kiltz, K. Pietrzak, and G. N. Rothblum. Leakage-resilient signatures. *IACR Cryptology ePrint Archive*, Report 2009/282, 2009.

[14] M. Hutter, S. Mangard, and M. Feldhofer. Power and EM attacks on passive 13.56MHz RFID devices. In *CHES*, pages 320–333, 2007.

[15] J. Katz and V. Vaikuntanathan. Signature schemes with bounded leakage resilience. In *ASIACRYPT*, pages 703–720, 2009.

[16] E. Kiltz and K. Pietrzak. Leakage resilient elgamal encryption. In *ASIACRYPT*, pages 595–612, 2010.

[17] P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *CRYPTO*, pages 104–113, 1996.

[18] B. LaMacchia, K. Lauter, and A. Mityagin. Stronger security of authenticated key exchange. In *ProvSec*, pages 1–16, 2007.

[19] 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.

[20] 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.

[21] D. Moriyama and T. Okamoto. Leakage resilient eCK-secure key exchange protocol without random oracles. In *ASIACCS*, pages 441–447, 2011.

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

[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.

A Analysis of the Events of the Proof of Theorem 4.1

A.1 Event $(\mathbf{E_{1a}} \wedge \mathbf{A}^*)$

A.1.1 Event $(\mathbf{E_{1a}} \wedge \mathbf{A}^*)$: setup.

$\mathcal{B}$ establishes N_P number of honest parties which $\mathcal{B}$ assigns long-term secret/public key pairs. For each honest party $\mathcal{B}$ maintains at most N_S number of sessions. $\mathcal{B}$ chooses two distinct random principals $U^*, V^* \xleftarrow{\$} \{U_1, \ldots, U_{N_P}\}$ and two random numbers $s^*, t^* \xleftarrow{\$} \{1, \ldots, N_S\}$. $\mathcal{B}$ guesses the the oracle $\Pi^{s^*}_{U^*,V^*}$ as the target session and the oracle $\Pi^{t^*}_{V^*,U^*}$ as the partner to the target session. For the rest of this event consider A as the long-term public key or U^* and B as the long-term public key of V^*.

A.1.2 Event $(\mathbf{E_{1a}} \wedge \mathbf{A}^*)$: simulation.

- `Send`: On behalf of honest protocol principals $\mathcal{B}$ selects ephemeral secret/public key pairs according to the protocol specification. $\mathcal{B}$ uses $X = L = g^{\ell}$ as the ephemeral public key of $\Pi^{s^*}_{U^*,V^*}$ and $Y = W = g^w$ as the ephemeral public key of $\Pi^{t^*}_{V^*,U^*}$. Note that $\mathcal{B}$ does not possess the ephemeral secret keys of oracles $\Pi^{s^*}_{U^*,V^*}$ and $\Pi^{t^*}_{V^*,U^*}$.

- `Corrupt`: $\mathcal{B}$ answers all `Corrupt` queries faithfully.

- `EphemeralKeyReveal`: If `EphemeralKeyReveal` query to the oracle $\Pi^{s^*}_{U^*,V^*}$ or $\Pi^{t^*}_{V^*,U^*}$ is asked, $\mathcal{B}$ aborts the simulation. Otherwise answers all `EphemeralKeyReveal` queries faithfully.

- `SessionKeyReveal`: $\mathcal{B}$ answers all `SessionKeyReveal` queries faithfully.

- $\mathrm{H}(pos_1, pos_2, pos_3, pos_4, initiator, I, responder, J)$: $\mathcal{B}$ simulates the random oracle H in the usual way. If $\mathcal{A}$ asks a H query such that $pos_4 = \mathrm{CDH}(L, W)$, $\mathcal{B}$ aborts the game and answers the GDH challenge. ($\mathcal{B}$ can find whether $pos_4 = \mathrm{CDH}(L, W)$ or not by using DDH oracle)

- `Test`: If the `Test` query is not asked to $\Pi^{s^*}_{U^*,V^*}$ and partner to $\Pi^{t^*}_{V^*,U^*}$, $\mathcal{B}$ aborts the simulation. Otherwise, $\mathcal{B}$ obtains $K \leftarrow \mathrm{H}(\mathrm{CDH}(A, B), \mathrm{CDH}(B, L), \mathrm{CDH}(A, W), \Omega(L, W), U^*, L, V^*, W)$ (considering U^* is the initiator, otherwise exchange the positions of ($\mathrm{CDH}(B, L)$, $\mathrm{CDH}(A, W)$)), (U^*, V^*) and (L, W) respectively), and uses K as the real session key.

A.1.3 Event $(\mathbf{E_{1a}} \wedge \mathbf{A}^*)$: analysis.

The simulation of the view of eCK challenger to the adversary $\mathcal{A}$ is perfect except with negligible probability. The probability that $\mathcal{A}$ selects the oracles $\Pi^{s^*}_{U^*,V^*}$ and $\Pi^{t^*}_{V^*,U^*}$ as the test session and its partner is at least $\frac{1}{N_P^2 N_S^2}$, and the probability of event $(\mathbf{E_{1a}} \wedge \mathbf{A}^*)$ is non-negligible. According to the event $\mathbf{A}^*$, $\mathcal{A}$ queries the random oracle H with $(\mathrm{CDH}(A,B), \mathrm{CDH}(B,L), \mathrm{CDH}(A,W), \mathrm{CDH}(L,W), U^*, L, V^*, W)$. Hence, $\mathcal{B}$ can answer the GDH challenge with the probability,

$$\Pr^{\mathrm{GDH}}_{g,q}(\mathcal{B}) \geq \frac{\Pr\left(\mathbf{E_{1a}} \wedge \mathbf{A}^*\right)}{N_P^2 N_S^2} \ .$$

Thus,

$$\Pr\left(\mathbf{E_{1a}} \wedge \mathbf{A}^*\right) \leq N_P^2 N_S^2 \Big(\Pr^{\mathrm{GDH}}_{g,q}(\mathcal{B})\Big) \ . \tag{6}$$

A.2 Event $(\mathbf{E_{1b}} \wedge \mathbf{A}^*)$

A.2.1 Event $(\mathbf{E_{1b}} \wedge \mathbf{A}^*)$: setup.

$\mathcal{B}$ establishes N_P number of honest parties. For each honest party $\mathcal{B}$ maintains at most N_S number of sessions. $\mathcal{B}$ chooses two distinct random principals $U^*, V^* \xleftarrow{\$} \{U_1, \ldots, U_{N_P}\}$. $\mathcal{B}$ sets $A = L = g^\ell$ as the long-term public key of U^* and $B = W = g^w$ as the long-term public key of V^*. For the rest of the parties $\mathcal{B}$ sets long-term secret/public key pairs according to the protocol specification. Note that $\mathcal{B}$ does not possess the long-term secret keys of U^* and V^*.

A.2.2 Event $(\mathbf{E_{1b}} \wedge \mathbf{A}^*)$: simulation.

- `Send`: On behalf of honest protocol principals $\mathcal{B}$ selects ephemeral secret/public key pairs according to the protocol specification.
- `Corrupt`: If `Corrupt` query to U^* or V^* is asked, $\mathcal{B}$ aborts the simulation. Otherwise $\mathcal{B}$ answers all `Corrupt` queries faithfully.
- `EphemeralKeyReveal`: $\mathcal{B}$ answers all `EphemeralKeyReveal` queries faithfully.
- `SessionKeyReveal`: $\mathcal{B}$ answers all `SessionKeyReveal` queries as follows:
  1. If $\mathcal{A}$ asks a `SessionKeyReveal` query to an oracle where both U^* and V^* are involved ($\Pi^{\cdot}_{U^*,V^*}$ or $\Pi^{\cdot}_{V^*,U^*}$), $\mathcal{B}$ uses the function Ω to compute a value to replace $\mathrm{CDH}(L,W)$ as $\Omega(L,W)$.
  2. If $\mathcal{A}$ asks a `SessionKeyReveal` query to an oracle which is owned by principal U^* (or V^*) and does not have a partner (where $\mathcal{A}$ sends E having come from the partner and $\mathcal{B}$ does not posses the secret key corresponds to E), $\mathcal{B}$ uses the function Ω to compute a value to replace $\mathrm{CDH}(L,E)$ (or $\mathrm{CDH}(W,E)$) as $\Omega(L,E)$ (or $\Omega(W,E)$).
  3. For any other `SessionKeyReveal` query $\mathcal{B}$ can easily compute corresponding four Diffie-Hellman exponentiations, because $\mathcal{B}$ knows the long-term secret keys of other principals and ephemeral secret keys of partnered oracles.

  Then $\mathcal{B}$ queries the random oracle H with the corresponding values and answers the `SessionKeyReveal` query. Since $\mathcal{B}$ uses the function Ω to compute values to replace Diffie-Hellman exponentiations, in places where it does not possess both long-term and ephemeral secret keys, $\mathcal{B}$ can maintain consistency when querying the random oracle H.
- $\mathrm{H}(pos_1, pos_2, pos_3, pos_4, initiator, I, responder, J)$:
  1. If *initiator, responder* $\notin \{U^*, V^*\}$, $\mathcal{B}$ simulates the random oracle H in the usual way.
  2. If $initiator = U^*$, $\mathcal{B}$ checks the random oracle for a previously asked query, matching with the current one. If a match found, answers with the corresponding random-oracle value. Otherwise:
     - If $\mathcal{B}$ found all the positions except position 3 of a previously asked random oracle query respectively matching with the positions of the current random oracle query, $\mathcal{B}$ queries the DDH oracle with (L, J, pos_3), if the output is 1 and position 3 of the previously asked random oracle query equals to $\Omega(L, J)$, answers with the corresponding random-oracle value in the table. Else answers with a random value and stores the query and the answer in the random oracle table.

– Else, answers with a random value and stores the query and the answer in the random oracle table.

Similarly when $initiator = V^*$.

3. If $responder = V^*$, $\mathcal{B}$ checks the random oracle for a previously asked query, matching with the current one. If a match found, answers with the corresponding random-oracle value. Otherwise:
   – If $\mathcal{B}$ found all the positions except position 2 of a previously asked random oracle query respectively matching with the positions of the current random oracle query, $\mathcal{B}$ queries the DDH oracle with (W, I, pos_2), if the output is 1 and position 2 of the previously asked random oracle query equals to $\Omega(W, I)$, answers with the corresponding random-oracle value in the table. Else answers with a random value and stores the query and the answer in the random oracle table.
   – Else, answers with a random value and stores the query and the answer in the random oracle table.

   Similarly when $responder = U^*$.

4. If $initiator \in \{U^*, V^*\}$ and $responder \in \{U^*, V^*\}$ and $\mathcal{A}$ asks a H query such that $pos_1 = \mathrm{CDH}(L, W)$, $\mathcal{B}$ aborts the game and answers the GDH challenge. ($\mathcal{B}$ can find whether $pos_1 = \mathrm{CDH}(L, W)$ or not by using DDH oracle)

- `Test`: If the `Test` query is not asked to an oracle where both U^* and V^* involve ($\Pi^{\cdot}_{U^*,V^*}$ or $\Pi^{\cdot}_{V^*,U^*}$), $\mathcal{B}$ aborts the simulation. Otherwise, $\mathcal{B}$ obtains $K \leftarrow \mathrm{H}(\Omega(L, W), \mathrm{CDH}(W, X), \mathrm{CDH}(L, Y), \mathrm{CDH}(X, Y), U^*, X, V^*, Y)$ (considering U^* is the initiator, otherwise exchange the positions of $(\mathrm{CDH}(W, X)$, $\mathrm{CDH}(L, Y))$, (U^*, V^*) and (X, Y) respectively), and uses K as the real session key.

A.2.3 Event $(\mathbf{E_{1b}} \wedge \mathbf{A}^*)$: analysis.

The simulation of the view of eCK challenger to the adversary $\mathcal{A}$ is perfect except with negligible probability. The probability that $\mathcal{A}$ selects an oracle where both U^* and V^* involve ($\Pi^{\cdot}_{U^*,V^*}$ or $\Pi^{\cdot}_{V^*,U^*}$) as the test session is at least $\frac{1}{N_P^2}$, and the probability of event $(\mathbf{E_{1b}} \wedge \mathbf{A}^*)$ is non-negligible. According to the event $\mathbf{A}^*$, $\mathcal{A}$ queries the random oracle H with $(\mathrm{CDH}(L, W), \mathrm{CDH}(W, X), \mathrm{CDH}(L, Y), \mathrm{CDH}(X, Y), U^*, X, V^*, Y)$. Hence, $\mathcal{B}$ can answer the GDH challenge with the probability,

$$\mathrm{Pr}_{g,q}^{\mathrm{GDH}}(\mathcal{B}) \geq \frac{\Pr(\mathbf{E_{1b}} \wedge \mathbf{A}^*)}{N_P^2} \ .$$

Thus,

$$\Pr(\mathbf{E_{1b}} \wedge \mathbf{A}^*) \leq N_P^2 \left(\mathrm{Pr}_{g,q}^{\mathrm{GDH}}(\mathcal{B})\right) \ . \tag{7}$$

A.3 Event $(\mathbf{E_{1c}} \wedge \mathbf{A}^*)$

A.3.1 Event $(\mathbf{E_{1c}} \wedge \mathbf{A}^*)$: setup.

$\mathcal{B}$ establishes N_P number of honest parties. For each honest party $\mathcal{B}$ maintains at most N_S number of sessions. $\mathcal{B}$ chooses two distinct random principals $U^*, V^* \xleftarrow{\$} \{U_1, \ldots, U_{N_P}\}$. $\mathcal{B}$ sets $B = W = g^w$ as the long-term public key of V^*. For the rest of the parties $\mathcal{B}$ sets long-term secret/public key pairs according to the protocol specification. Note that $\mathcal{B}$ does not possess the long-term secret keys of V^*. $\mathcal{B}$ chooses a random number $s^* \xleftarrow{\$} \{1, \ldots, N_S\}$ and chooses the oracle $\Pi^{s^*}_{U^*,V^*}$ as the target session.

A.3.2 Event $(\mathbf{E_{1c}} \wedge \mathbf{A}^*)$: simulation.

Note. In this simulation we assume that U^* is the initiator of the target session. It is possible to easily simulate the case where U^* is the responder as follows: In a place we consider the random oracle query $\mathrm{H}(\mathrm{CDH}(A, B), \Omega(W, L), \mathrm{CDH}(A, Y), \mathrm{CDH}(X, Y), U^*, L, V^*, Y)$, exchange the positions of $(\Omega(W, L)$, $\mathrm{CDH}(A, Y))$, (U^*, V^*) and (X, Y) respectively. Particularly, in the simulation of `Test` query and in the simulation of the point 4 of H query checks for $pos_3 = \mathrm{CDH}(W, L)$.

- `Send`: On behalf of honest protocol principals $\mathcal{B}$ selects ephemeral secret/public key pairs according to the protocol specification. $\mathcal{B}$ uses $X = L = g^\ell$ as the ephemeral public key of $\Pi^{s^*}_{U^*,V^*}$. Note that $\mathcal{B}$ does not posses the ephemeral secret key of oracle $\Pi^{s^*}_{U^*,V^*}$.

- `Corrupt`: If `Corrupt` query to V^* is asked, $\mathcal{B}$ aborts the simulation. Otherwise $\mathcal{B}$ answers all `Corrupt` queries faithfully.
- `EphemeralKeyReveal`: If `EphemeralKeyReveal` query to $\Pi^{s^*}_{U^*,V^*}$ is asked, $\mathcal{B}$ aborts the simulation. Otherwise $\mathcal{B}$ answers all `EphemeralKeyReveal` queries faithfully.
- `SessionKeyReveal`: $\mathcal{B}$ answers all `SessionKeyReveal` queries as follows:

  1. If $\mathcal{A}$ asks a `SessionKeyReveal` query to an oracle which is owned by principal V^* and does not have a partner (where $\mathcal{A}$ sends E having come from the partner and $\mathcal{B}$ does not posses the secret key corresponds to E), $\mathcal{B}$ uses the function Ω to compute a value to replace $\mathrm{CDH}(W, E)$ as $\Omega(W, E)$.
  2. For any other `SessionKeyReveal` query $\mathcal{B}$ can easily compute corresponding four Diffie-Hellman exponentiations, because $\mathcal{B}$ knows the long-term secret keys of other principals and ephemeral secret keys of partnered oracles.

  Then $\mathcal{B}$ queries the random oracle H with the corresponding values and answers the `SessionKeyReveal` query. Since $\mathcal{B}$ uses the function Ω to compute values to replace Diffie-Hellman exponentiations, in places where it does not possess both long-term and ephemeral secret keys, $\mathcal{B}$ can maintain consistency when querying the random oracle H.

- $\mathrm{H}(pos_1, pos_2, pos_3, pos_4, initiator, I, responder, J)$:

  1. If *initiator, responder* $\notin \{V^*\}$, $\mathcal{B}$ simulates the random oracle H in the usual way.
  2. If $initiator = V^*$, $\mathcal{B}$ checks the random oracle for a previously asked query, matching with the current one. If a match found, answers with the corresponding random-oracle value. Otherwise:
     - If $\mathcal{B}$ found all the positions except position 3 of a previously asked random oracle query respectively matching with the positions of the current random oracle query, $\mathcal{B}$ queries the DDH oracle with (W, J, pos_3), if the output is 1 and position 3 of the previously asked random oracle query equals to $\Omega(W, J)$, answers with the corresponding random-oracle value in the table. Else answers with a random value and stores the query and the answer in the random oracle table.
     - Else, answers with a random value and stores the query and the answer in the random oracle table.
  3. If $responder = V^*$, $\mathcal{B}$ checks the random oracle for a previously asked query, matching with the current one. If a match found, answers with the corresponding random-oracle value. Otherwise:
     - If $\mathcal{B}$ found all the positions except position 2 of a previously asked random oracle query respectively matching with the positions of the current random oracle query, $\mathcal{B}$ queries the DDH oracle with (W, I, pos_2), if the output is 1 and position 2 of the previously asked random oracle query equals to $\Omega(W, I)$, answers with the corresponding random-oracle value in the table. Else answers with a random value and stores the query and the answer in the random oracle table.
     - Else, answers with a random value and stores the query and the answer in the random oracle table.
  4. If $\mathcal{A}$ asks a H query such that $pos_2 = \mathrm{CDH}(W, L)$, $\mathcal{B}$ aborts the game and answers the GDH challenge. ($\mathcal{B}$ can find whether $pos_2 = \mathrm{CDH}(W, L)$ or not by using DDH oracle)

- `Test`: If the `Test` query is not asked to $\Pi^{s^*}_{U^*,V^*}$, $\mathcal{B}$ aborts the simulation. Otherwise, $\mathcal{B}$ obtains $K \leftarrow \mathrm{H}(\mathrm{CDH}(A, W), \Omega(W, L), \mathrm{CDH}(L, Y), \mathrm{CDH}(X, Y), U^*, L, V^*, Y)$, and uses K as the real session key.

A.3.3 Event $(\mathbf{E_{1c}} \wedge \mathbf{A}^*)$: analysis.

The simulation of the view of eCK challenger to the adversary $\mathcal{A}$ is perfect except with negligible probability. The probability that $\mathcal{A}$ selects the oracle $\Pi^{s^*}_{U^*,V^*}$ as the test session is at least $\frac{1}{N_P^2 N_S}$, and the probability of event $(\mathbf{E_{1c}} \wedge \mathbf{A}^*)$ is non-negligible. According to the event $\mathbf{A}^*$, $\mathcal{A}$ queries the random oracle H with $(\mathrm{CDH}(A, W), \mathrm{CDH}(W, L), \mathrm{CDH}(A, Y), \mathrm{CDH}(L, Y), U^*, L, V^*, Y)$. Hence, $\mathcal{B}$ can answer the GDH challenge with the probability,

$$\Pr^{\mathrm{GDH}}_{g,q}(\mathcal{B}) \geq \frac{\Pr\left(\mathbf{E_{1c}} \wedge \mathbf{A}^*\right)}{N_P^2 N_S} \ .$$

Thus,

$$\Pr\left(\mathbf{E_{1c}} \wedge \mathbf{A}^*\right) \leq N_P^2 N_S \Big(\Pr^{\mathrm{GDH}}_{g,q}(\mathcal{B})\Big) \ . \qquad (8)$$

A.4 Event $(\mathbf{E_{1d}} \wedge \mathbf{A}^*)$

A.4.1 Event $(\mathbf{E_{1d}} \wedge \mathbf{A}^*)$: setup.

$\mathcal{B}$ establishes N_P number of honest parties. For each honest party $\mathcal{B}$ maintains at most N_S number of sessions. $\mathcal{B}$ chooses two distinct random principals $U^*, V^* \xleftarrow{\$} \{U_1, \ldots, U_{N_P}\}$. $\mathcal{B}$ sets $A = W = g^w$ as the long-term public key of U^*. For the rest of the parties $\mathcal{B}$ sets long-term secret/public key pairs according to the protocol specification. Note that $\mathcal{B}$ does not possess the long-term secret keys of U^*. $\mathcal{B}$ chooses a random number $t^* \xleftarrow{\$} \{1, \ldots, N_S\}$ and chooses the oracle $\Pi^{t^*}_{V^*,U^*}$ as the partner oracle to the target session.

A.4.2 Event $(\mathbf{E_{1d}} \wedge \mathbf{A}^*)$: simulation.

Note. In this simulation we assume that U^* is the initiator of the target session. It is possible to easily simulate the case where U^* is the responder as follows: In a place we consider the random oracle query $\mathrm{H}(\mathrm{CDH}(W,B), \mathrm{CDH}(B,X), \Omega(W,L), \mathrm{CDH}(X,L), U^*, X, V^*, L)$, exchange the positions of $(\mathrm{CDH}(B,X)$, $\Omega(A,Y))$, (U^*, V^*) and (X, Y) respectively. Particularly, in the simulation of `Test` query and in the simulation of the point 4 of H query checks for $pos_2 = \mathrm{CDH}(W, L)$.

- `Send`: On behalf of honest protocol principals $\mathcal{B}$ selects ephemeral secret/public key pairs according to the protocol specification. $\mathcal{B}$ uses $Y = L = g^\ell$ as the ephemeral public key of $\Pi^{t^*}_{V^*,U^*}$. Note that $\mathcal{B}$ does not posses the ephemeral secret key of oracle $\Pi^{t^*}_{V^*,U^*}$.
- `Corrupt`: If `Corrupt` query to U^* is asked, $\mathcal{B}$ aborts the simulation. Otherwise $\mathcal{B}$ answers all `Corrupt` queries faithfully.
- `EphemeralKeyReveal`: If `EphemeralKeyReveal` query to partner to $\Pi^{t^*}_{V^*,U^*}$ is asked, $\mathcal{B}$ aborts the simulation. Otherwise $\mathcal{B}$ answers all `EphemeralKeyReveal` queries faithfully.
- `SessionKeyReveal`: $\mathcal{B}$ answers all `SessionKeyReveal` queries as follows:
  1. If $\mathcal{A}$ asks a `SessionKeyReveal` query to an oracle which is owned by principal U^* and does not have a partner (where $\mathcal{A}$ sends E having come from the partner and $\mathcal{B}$ does not posses the secret key corresponds to E), $\mathcal{B}$ uses the function Ω to compute a value to replace $\mathrm{CDH}(W, E)$ as $\Omega(W, E)$.
  2. For any other `SessionKeyReveal` query $\mathcal{B}$ can easily compute corresponding four Diffie-Hellman exponentiations, because $\mathcal{B}$ knows the long-term secret keys of other principals and ephemeral secret keys of partnered oracles.

  Then $\mathcal{B}$ queries the random oracle H with the corresponding values and answers the `SessionKeyReveal` query. Since $\mathcal{B}$ uses the function Ω to compute values to replace Diffie-Hellman exponentiations, in places where it does not possess both long-term and ephemeral secret keys, $\mathcal{B}$ can maintain consistency when querying the random oracle H.
- $\mathrm{H}(pos_1, pos_2, pos_3, pos_4, initiator, I, responder, J)$:
  1. If *initiator, responder* $\notin \{U^*\}$, $\mathcal{B}$ simulates the random oracle H in the usual way.
  2. If *initiator* $= U^*$, $\mathcal{B}$ checks the random oracle for a previously asked query, matching with the current one. If a match found, answers with the corresponding random-oracle value. Otherwise:
     - If $\mathcal{B}$ found all the positions except position 3 of a previously asked random oracle query respectively matching with the positions of the current random oracle query, $\mathcal{B}$ queries the DDH oracle with (W, J, pos_3), if the output is 1 and position 3 of the previously asked random oracle query equals to $\Omega(W, J)$, answers with the corresponding random-oracle value in the table. Else answers with a random value and stores the query and the answer in the random oracle table.

– Else, answers with a random value and stores the query and the answer in the random oracle table.

3. If *responder* $= U^*$, $\mathcal{B}$ checks the random oracle for a previously asked query, matching with the current one. If a match found, answers with the corresponding random-oracle value. Otherwise:
   – If $\mathcal{B}$ found all the positions except position 2 of a previously asked random oracle query respectively matching with the positions of the current random oracle query, $\mathcal{B}$ queries the DDH oracle with (W, I, pos_2), if the output is 1 and position 2 of the previously asked random oracle query equals to $\Omega(W, I)$, answers with the corresponding random-oracle value in the table. Else answers with a random value and stores the query and the answer in the random oracle table.
   – Else, answers with a random value and stores the query and the answer in the random oracle table.
4. If $\mathcal{A}$ asks a H query such that $pos_3 = \text{CDH}(W, L)$, $\mathcal{B}$ aborts the game and answers the GDH challenge. ($\mathcal{B}$ can find whether $pos_3 = \text{CDH}(W, L)$ or not by using DDH oracle)

- `Test`: If the `Test` query is not asked to the partner of $\Pi^{t^*}_{V^*,U^*}$, $\mathcal{B}$ aborts the simulation. Otherwise, $\mathcal{B}$ obtains $K \leftarrow \text{H}(\text{CDH}(W, B), \text{CDH}(B, X), \Omega(W, L), \text{CDH}(X, L), U^*, X, V^*, L)$, and uses K as the real session key.

A.4.3 Event $(\mathbf{E_{1d}} \wedge \mathbf{A}^*)$: analysis.

The simulation of the view of eCK challenger to the adversary $\mathcal{A}$ is perfect except with negligible probability. The probability that $\mathcal{A}$ selects the partner to the oracle $\Pi^{t^*}_{V^*,U^*}$ as the test session is at least $\frac{1}{N_P^2 N_S}$, and the probability of event $(\mathbf{E_{1d}} \wedge \mathbf{A}^*)$ is non-negligible. According to the event $\mathbf{A}^*$, $\mathcal{A}$ queries the random oracle H with $(\text{CDH}(W, B), \text{CDH}(B, X), \text{CDH}(W, L), \text{CDH}(X, Y), U^*, X, V^*, L)$. Hence, $\mathcal{B}$ can answer the GDH challenge with the probability,

$$\Pr^{\text{GDH}}_{g,q}(\mathcal{B}) \geq \frac{\Pr(\mathbf{E_{1d}} \wedge \mathbf{A}^*)}{N_P^2 N_S} \ .$$

Thus,

$$\Pr(\mathbf{E_{1d}} \wedge \mathbf{A}^*) \leq N_P^2 N_S \Big(\Pr^{\text{GDH}}_{g,q}(\mathcal{B})\Big) \ . \tag{9}$$

A.5 Event $(\mathbf{E_{2a}} \wedge \mathbf{A}^*)$

A.5.1 Event $(\mathbf{E_{2a}} \wedge \mathbf{A}^*)$: setup.

$\mathcal{B}$ establishes N_P number of honest parties. For each honest party $\mathcal{B}$ maintains at most N_S number of sessions. $\mathcal{B}$ chooses two distinct random principals $U^*, V^* \xleftarrow{\$} \{U_1, \ldots, U_{N_P}\}$. $\mathcal{B}$ sets $B = W = g^w$ as the long-term public key of V^*. For the rest of the parties $\mathcal{B}$ sets long-term secret/public key pairs according to the protocol specification. Note that $\mathcal{B}$ does not possess the long-term secret keys of V^*. $\mathcal{B}$ chooses a random number $s^* \xleftarrow{\$} \{1, \ldots, N_S\}$ and chooses the oracle $\Pi^{s^*}_{U^*,V^*}$ as the target session.

A.5.2 Event $(\mathbf{E_{2a}} \wedge \mathbf{A}^*)$: simulation.

$\mathcal{B}$ uses $X = L = g^\ell$ as the ephemeral public key of $\Pi^{s^*}_{U^*,V^*}$. Note that $\mathcal{B}$ does not posses the ephemeral secret key of oracle $\Pi^{s^*}_{U^*,V^*}$. This simulation is same as the simulation of Event $\mathbf{E_{1c}} \wedge \mathbf{A}^*$.

A.5.3 Event $(\mathbf{E_{2a}} \wedge \mathbf{A}^*)$: analysis.

The simulation of the view of eCK challenger to the adversary $\mathcal{A}$ is perfect except with negligible probability. The probability that $\mathcal{A}$ selects the oracle $\Pi^{s^*}_{U^*,V^*}$ as the test session is at least $\frac{1}{N_P^2 N_S}$, and the probability of event $(\mathbf{E_{2a}} \wedge \mathbf{A}^*)$ is non-negligible. According to the event $\mathbf{A}^*$, $\mathcal{A}$ queries the random oracle H with $(\text{CDH}(A, W), \text{CDH}(W, L), \text{CDH}(A, Y), \text{CDH}(L, Y), U^*, L, V^*, Y)$. Hence, $\mathcal{B}$ can answer the GDH challenge with the probability,

$$\Pr^{\text{GDH}}_{g,q}(\mathcal{B}) \geq \frac{\Pr(\mathbf{E_{2a}} \wedge \mathbf{A}^*)}{N_P^2 N_S} \ .$$

Thus,

$$\Pr(\mathbf{E_{2a}} \wedge \mathbf{A}^*) \leq N_P^2 N_S \Big(\Pr^{\text{GDH}}_{g,q}(\mathcal{B})\Big) \ . \tag{10}$$

A.6 Event $(\mathbf{E_{2b}} \wedge \mathbf{A}^*)$

A.6.1 Event $(\mathbf{E_{2b}} \wedge \mathbf{A}^*)$: setup.

$\mathcal{B}$ establishes N_P number of honest parties. For each honest party $\mathcal{B}$ maintains at most N_S number of sessions. $\mathcal{B}$ chooses two distinct random principals $U^*, V^* \xleftarrow{\$} \{U_1, \ldots, U_{N_P}\}$. $\mathcal{B}$ sets $A = L = g^\ell$ as the long-term public key of U^* and $B = W = g^w$ as the long-term public key of V^*. For the rest of the parties $\mathcal{B}$ sets long-term secret/public key pairs according to the protocol specification. Note that $\mathcal{B}$ does not possess the long-term secret keys of U^* and V^*.

A.6.2 Event $(\mathbf{E_{2b}} \wedge \mathbf{A}^*)$: simulation.

This simulation is same as the simulation of Event $\mathbf{E_{2b}} \wedge \mathbf{A}^*$.

A.6.3 Event $(\mathbf{E_{2b}} \wedge \mathbf{A}^*)$: analysis.

The simulation of the view of eCK challenger to the adversary $\mathcal{A}$ is perfect except with negligible probability. The probability that $\mathcal{A}$ selects an oracle where both U^* and V^* involve ($\Pi^{\cdot}_{U^*,V^*}$ or $\Pi^{\cdot}_{V^*,U^*}$) as the test session is at least $\frac{1}{N_P^2}$, and the probability of event $(\mathbf{E_{1b}} \wedge \mathbf{A}^*)$ is non-negligible. According to the event $\mathbf{A}^*$, $\mathcal{A}$ queries the random oracle H with $(\mathrm{CDH}(L, W), \mathrm{CDH}(W, X), \mathrm{CDH}(L, Y), \mathrm{CDH}(X, Y), U^*, X, V^*, Y)$. Hence, $\mathcal{B}$ can answer the GDH challenge with the probability,

$$\mathrm{Pr}_{g,q}^{\mathrm{GDH}}(\mathcal{B}) \geq \frac{\mathrm{Pr}\left(\mathbf{E_{2b}} \wedge \mathbf{A}^*\right)}{N_P^2} \ .$$

Thus,

$$\mathrm{Pr}\left(\mathbf{E_{2b}} \wedge \mathbf{A}^*\right) \leq N_P^2 \Big(\mathrm{Pr}_{g,q}^{\mathrm{GDH}}(\mathcal{B})\Big) \ . \tag{11}$$

We know that,

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

Therefore, using equations 6–11 we get that,

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

□