On the Security of SL-DNSSEC

Aditya Singh Rawat
Ashoka University
aditya.rawat_phd21@ashoka.edu.in

Mahabir Prasad Jhanwar
Ashoka University
mahavir.jhawar@ashoka.edu.in

Abstract

The proposed SL-DNSSEC protocol (AsiaCCS'25) uses a quantum-safe KEM and a MAC to perform *signature-less* (SL) DNSSEC validations in a single UDP query / response style. In this paper, we give a formal analysis of SL-DNSSEC in the reductionist model.

1 Overview

Our approach is based on the reductionist security model, originally given for the TLS 1.3 handshake [4, 5], and then later adapted to KEMTLS(-PDK) [10, 11, 13]. In particular, the reductionist method amends the traditional Bellare-Rogaway [2] Authenticated Key Exchange (AKE) model to accommodate *multiple* stages of session keys — following the multi-stage AKE model given by Fischlin and Günther [6]. In this work, we model SL-DNSSEC [9] as a *two*-stage *unilaterally*-authenticated key exchange protocol.

1.1 Security Goal

The primary security aim is that the keys established in each stage of SL-DNSSEC should be *indistinguishable* from random, in the presence of an adversary who observes and controls all communication. We draw distinction between *implicit* authentication (wherein a key could only be known by the legitimate peer), and *explicit* authentication (which assures that the intended peer actually participated).

1.2 Quantum Adversaries

The proof proceeds in the standard model (without random oracles), and does not use methods such as the forking lemma or rewinding. Thus, techniques like the *lifting lemma* [12] can be applied to show that SL-DNSSEC is secure against quantum attackers, provided that each of the underlying primitives is also quantum-safe.

2 Background

2.1 SL-DNSSEC Protocol

SL-DNSSEC [9] is a backward-compatible protocol for DNSSEC that validates DNS resource records without signatures. Pursuant to this objective, it uses as its primary building blocks: 1) A post-quantum KEM to first establish a shared secret between a resolver and a nameserver, 2) A KDF to derive a symmetric MAC key of an appropriate length from the shared secret, and 3) A MAC to compute authentication tags on DNS records.

On a high-level, a nameserver generates a KEM keypair and adds the public key to its DNSKEY RRset, re-signing the latter with its $\mathsf{KSK}_{\mathsf{sk}}$. This RRset is fetched and DNSSEC-validated by a resolver using the covering RRSIG and a signed DS record from the zone's parent. For any subsequent interaction, the resolver and the nameserver compute a symmetric MAC key using KEM and KDF operations. The nameserver then sends MAC tags instead of signatures on DNS RRsets — as sketched in Fig. 1.

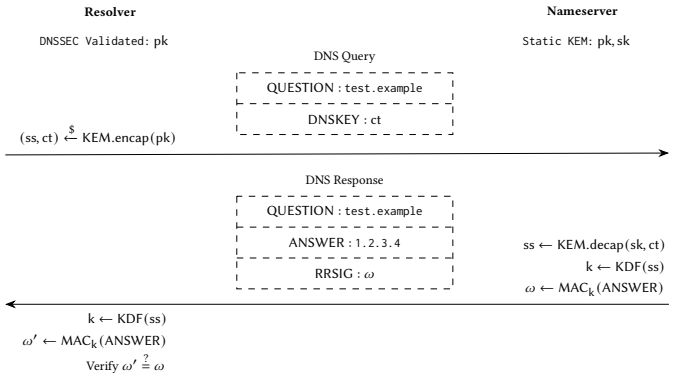


Figure 1: SL-DNSSEC Protocol. The resolver holds a DNSSEC-validated KEM public key of the server. Using a series of KEM and KDF operations, both parties derive a symmetric MAC key. The server sends a MAC tag instead of a signature on the answer record.

2.2 Reductionist Security Model

We first give a primer on the notations used, the adversary and the capabilities thereof, and the experiments for Match and Multi-Stage security. To put it succinctly, Match-security ensures correct session matching while Multi-Stage security models key indistinguishability and server authentication.

Model Syntax. Set $\mathcal{U}$ denotes the identities of honest participants. An identity $S \in \mathcal{U}$ that acts as a server is associated with a certified[1] long-term (static) KEM public key pk_S and a corresponding private key sk_S.

Each participant can run multiple instances of the protocol, each of which is called a *session*. A session $\pi = (U, n)$ denotes the $n^{th} \in \mathbb{N}$ local session of an identity $U \in \mathcal{U}$. In a multi-stage protocol, each session consists of multiple stages, run sequentially with a shared state. Each stage aims to establish a key. The total number of stages in a session is denoted by $\mathsf{M} \in \mathbb{N}$. Each party maintains the following information[2] *per session*.

- $\mathsf{id} \in \mathcal{U}$: the identity of the party that owns the session.
- $\mathsf{pid} \in \mathcal{U} \cup \{*\}$: the identity of the intended partner. '$*$' denotes an unknown party.
- $\mathsf{role} \in \{\mathsf{initiator}, \mathsf{responder}\}$
- $\mathsf{status} \in \{\perp, \mathsf{running}, \mathsf{accepted}, \mathsf{rejected}\}^{\mathsf{M}}$: the status of each stage key.
  - Set $\mathsf{status} \leftarrow (\mathsf{running}, \perp^{\times(\mathsf{M}-1)})$, initially.
  - Set $\mathsf{status}_i \leftarrow \mathsf{accepted}$, when stage i accepts a key.
  - Set $\mathsf{status}_i \leftarrow \mathsf{rejected}$, when stage i rejects a key. The protocol then aborts.

[1] A DNSSEC *chain of trust* can be constructed to the root.
[2] Many variables are vectors of length M, with values for each stage.

- stage $\in \{0, 1, \ldots, \mathsf{M}\}$: the last accepted stage. Initially set to 0. Set stage $\leftarrow i$, when $\text{status}_i \leftarrow$ accepted.
- sid $\in (\{0,1\}^* \cup \{\perp\})^{\mathsf{M}}$: the session identifier bitstring used for session pairing in stage i. Initially set to $\perp$. Updated to the specified value for stage i when $\text{status}_i \leftarrow$ accepted.
- cid $\in (\{0,1\}^* \cup \{\perp\})^{\mathsf{M}}$: the contributive session identifier bitstring used for *early* session pairing in stage i. Initially set to $\perp$. Updated to the specified value for stage i until $\text{status}_i \leftarrow$ accepted.
- key $\in (\mathcal{K} \cup \{\perp\})^{\mathsf{M}}$: the key established in stage i, when $\text{status}_i \leftarrow$ accepted. Initially set to $\perp$.
- revealed $\in \{\text{true}, \text{false}\}^{\mathsf{M}}$: records if key_i has been revealed by the adversary. Initially set to $(\text{false})^{\times \mathsf{M}}$.
- tested $\in \{\text{true}, \text{false}\}^{\mathsf{M}}$: records if key_i has been tested by the adversary. Initially set to $(\text{false})^{\times \mathsf{M}}$.
- auth $\in \{0, 1, \ldots, \mathsf{M}, \infty\}^{\mathsf{M}}$: indicates the stage at which key_i is explicitly authenticated.
  - $(\text{auth}_i = i) \implies \text{key}_i$ is immediately authenticated.
  - $(\text{auth}_i > i) \implies \text{key}_i$ is retroactively authenticated.
  - $(\text{auth}_i = \infty) \implies \text{key}_i$ is never authenticated.
- replay $\in \{\text{replayable}, \text{nonreplayable}\}^{\mathsf{M}}$: indicates whether a stage is unique against replay attacks or not. The adversary should not be able to distinguish a replayed accepted key and a random one.

$\pi.X$ denotes an element X of a session π. The *partner* of π at stage i is any π' such that $\pi.\text{sid}_i = \pi'.\text{sid}_i \neq \perp$ and $\pi.\text{role} \neq \pi'.\text{role}$. Likewise, a *contributive partner* is defined using cid(s). In an honest joint execution of the protocol, the foregoing predicate holds true for all stages on acceptance.

Adversary. The adversary $\mathcal{A}$ is a probabilistic algorithm that controls the communication between all parties and thus can intercept, inject or drop any message. In this adversarial model, even two honest participants require $\mathcal{A}$ to establish a new session.

$\mathcal{A}$ interacts with the parties via several queries.

- Queries modelling a benign protocol run:
  - $\text{NewSession}(U, V, role)$: Create a new session π. Set $\pi.\text{id} \leftarrow U$, $\pi.\text{pid} \leftarrow V$ and $\pi.\text{role} \leftarrow role$. V may be left unspecified ($V = *$).
  - $\text{Send}(\pi, m)$: Send message m to session π. If π has not been created using NewSession, return $\perp$. Otherwise, run the protocol on behalf of $\pi.\text{id}$ based on its current state. Subsequently, update the state, and return both the response message and $\pi.\text{status}_{\pi.\text{stage}}$.
- Queries modelling $\mathcal{A}$'s ability to compromise participants and learn some secret information:
  - $\text{Reveal}(\pi, i)$: Give $\pi.\text{key}_i$ to $\mathcal{A}$. Set $\pi.\text{revealed}_i \leftarrow \text{true}$. If π does not exist or $\text{status}_i \neq$ accepted, return $\perp$.
  - $\text{Corrupt}(S)$: Give $\mathcal{A}$ the long-term secret key sk_S of server $S \in \mathcal{U}$. Mark S as corrupted.
- Query modelling the challenge to $\mathcal{A}$ of breaking a stage key established by honest partners π and π':
  - $\text{Test}(\pi, i)$: Challenge $\mathcal{A}$ on the indistinguishability of $\pi.\text{key}_i$. If $(\pi.\text{status}_i \neq \text{accepted}) \vee (\pi.\text{tested}_i = \text{true}) \vee (\pi'.\text{tested}_i = \text{true})$, return $\perp$. The Test oracle has a hidden bit b fixed throughout the game. If $b = 0$, sample a uniformly random key $K \leftarrow\$ \mathcal{K}$. If $b = 1$, set $K \leftarrow \pi.\text{key}_i$. Finally, set $\pi.\text{tested}_i \leftarrow \text{true}$ and $\pi'.\text{tested}_i \leftarrow \text{true}$, and return K to $\mathcal{A}$.

Match Security. Match security models the soundness of session identifiers *i.e.* $\pi.\text{sid}_i = \pi'.\text{sid}_i$, where π, π' are partnered.

Definition 1. *(Match-security). Let* KE *be an* M*-stage key exchange protocol. Adversary $\mathcal{A}$ interacts with* KE *via several queries and tries to win the following game $G^{\text{Match}}_{\text{KE},\mathcal{A}}$:*

- *Setup.* The challenger generates long-term keypairs $(\text{pk}_S, \text{sk}_S)$ for each server $S \in \mathcal{U}$. All keys are given to $\mathcal{A}$.
- *Query.* $\mathcal{A}$ has access to the queries defined in §2.2.
- *Stop.* Eventually, $\mathcal{A}$ terminates with no output.

Let π and π' be distinct, partnered sessions such that $\pi.\text{sid}_i = \pi'.\text{sid}_i \neq \perp$ at stage $i \in \{0, 1, \ldots, \mathsf{M}\}$. $\mathcal{A}$ wins the Match-security game if it can falsify any of the following conditions:

(1) $\forall j \leq i$, $\pi.\text{key}_j = \pi'.\text{key}_j$: both sessions agree on the same key at every stage up to and including stage i.
(2) $\pi.\text{role} \neq \pi'.\text{role}$: both sessions have opposite roles.
(3) $\pi.\text{cid}_i = \pi'.\text{cid}_i$: both sessions have the same cid(s).
(4) $\forall j$, $((\pi.\text{status}_j = \text{accepted}) \wedge (\pi.\text{stage} \geq \pi.\text{auth}_j)) \implies (\pi.\text{pid} = \pi'.\text{id})$: sessions are partnered with the intended (explicitly authenticated) participant.
(5) $(\pi.\text{sid}_i = \pi'.\text{sid}_j) \implies (i = j)$: session identifiers are different in each stage.
(6) For any sessions π, π', π'', $((\pi.\text{replay}_i = \text{nonreplayable}) \wedge (\pi.\text{sid}_i = \pi'.\text{sid}_i = \pi''.\text{sid}_i \neq \perp)) \implies (((\pi = \pi') \oplus (\pi = \pi'') \oplus (\pi' = \pi'')) = 1)$: at most two sessions share the same session label.

KE is Match-secure if $\forall \mathcal{A}$ that are polynomially bounded:

$$\text{Adv}^{\text{Match}}_{\text{KE},\mathcal{A}} = \left| \Pr\left[G^{\text{Match}}_{\text{KE},\mathcal{A}} \implies 1 \right] - \frac{1}{2} \right|$$

is negligible in the security parameter.

Multi-Stage Security. In the Multi-Stage experiment, secrecy of each stage key is defined as being indistinguishable from a random key, following Bellare-Rogaway [2]. This formulation also includes explicit authentication.

We first define *freshness* and *malicious acceptance.*

Definition 2. *(Freshness.) Stage i of a session π is fresh if all the following conditions hold:*

(1) $\pi.\text{revealed}_i = \text{false}$: stage i key has not been revealed.
(2) $\forall i, \pi'$ $(\pi.\text{sid}_i = \pi'.\text{sid}_i) \implies (\pi'.\text{revealed}_i = \text{false})$: the stage key of the partner session (if it exists) at stage i has not been revealed.
(3) $(\pi.\text{replay}_i = \text{replayable}) \implies \text{Corrupt}(\pi.\text{pid})$ was never called: If the stage i is replayable, the partner has never been corrupted.

Definition 3. *(Malicious acceptance.) Stage i of session π has maliciously accepted if all the below conditions hold:*

(1) $\pi.\text{status}_{\pi.\text{auth}_i} = \text{accepted}$: stage i key has already been explicitly authenticated.
(2) $(\pi.\text{replay}_i = \text{nonreplayable}) \implies \nexists$ a unique partner of π at stage i.

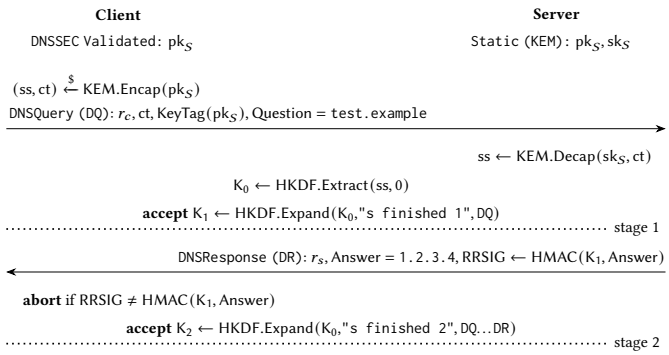


Figure 2: SL-DNSSEC protocol with unilateral (server-only) authentication using a pre-distributed DNSKEY.

(3) Corrupt(π.pid) was not called before stage $\pi.\mathsf{auth}_i$ accepted.

We now define the Multi-Stage experiment.

Definition 4. *(*Multi-Stage *security). Let* KE *be an* M*-stage key exchange protocol. Adversary $\mathcal{A}$ interacts with* KE *via several queries and tries to win the following game* $G^{\text{Multi-Stage}}_{\mathsf{KE},\mathcal{A}}$*:*

- *Setup.* The challenger generates long-term keypairs $(\mathsf{pk_S}, \mathsf{sk_S})$ for each server $S \in \mathcal{U}$ and picks the random test bit b (used in the Test queries). The public keys $\mathsf{pk_S}$ are given to $\mathcal{A}$.
- *Query.* $\mathcal{A}$ has access to the queries defined in §2.2.
- *Stop.* In due course, $\mathcal{A}$ stops and outputs its guess b' of b.
- *Finalize.* $\mathcal{A}$ wins the Multi-Stage security game if *either* of the following conditions hold:
  (1) all tested stages are *fresh*, and $b' = b$;
  (2) $\exists$ a stage that has *maliciously accepted.*

The Multi-Stage advantage of $\mathcal{A}$ is defined as:

$$\mathsf{Adv}^{\text{Multi-Stage}}_{\mathsf{KE},\mathcal{A}} = \left|\Pr\left[G^{\text{Multi-Stage}}_{\mathsf{KE},\mathcal{A}} \implies 1\right] - \frac{1}{2}\right|$$

3 Security Proof

We now prove the security of SL-DNSSEC under the reductionist model elucidated in the previous section.

3.1 SL-DNSSEC Particulars

The terms *client* and *server* refer to resolver and authoritative nameserver, respectively. The number of stages in our model is $\mathsf{M} = 2$, as sketched in Fig. 2.

Whenever stage i is accepted, its session identifier sid_i is set to comprise a label, the server DNSKEY (even though it is not transmitted), and all the DNS messages up to that point. A party terminates if it receives any unexpected message. Concretely,

$$\mathsf{sid}_1 = (\text{“}\mathsf{K}_1\text{”}, \mathsf{DNSKEY}, \mathtt{DNSQuery})$$
$$\mathsf{sid}_2 = (\text{“}\mathsf{K}_2\text{”}, \mathsf{DNSKEY}, \mathtt{DNSQuery} \ldots \mathtt{DNSResponse})$$

The contributive identifiers cid_i for each stage match up participant sessions before sid_i has been accepted. Initially, we set

$$\mathsf{cid}_1 = (\text{“}\mathsf{K}_1\text{”}, \mathsf{DNSKEY}, \emptyset)$$

to ensure that client sessions can be tested, even if the adversary $\mathcal{A}$ drops client's DNSQuery. When the client sends or the server receives DNSQuery, we update $\mathsf{cid}_1 = \mathsf{sid}_1$ in the respective sessions.

In case A of the Multi-Stage proof, we need to identify the unique pair of honest contributive server and client sessions, even if the adversary $\mathcal{A}$ drops server's DNSResponse. Hence, we initially set

$$\mathsf{cid}_2 = (\text{“}\mathsf{K}_2\text{”}, \mathsf{DNSKEY}, \mathtt{DNSQuery})$$

when sending or receiving DNSQuery. When the client receives or the server sends DNSResponse, we set $\mathsf{cid}_2 = \mathsf{sid}_2$.

Every client and server session in SL-DNSSEC uses the following property for the replayability of stages:

$$\mathsf{replay} = (\mathsf{replayable}, \mathsf{nonreplayable})$$

All SL-DNSSEC clients authenticate the server. Succinctly, all client sessions use:

$$\mathsf{auth} = (2^{\times 2})$$

The server is explicitly authenticated when the client accepts stage 2 after successfully verifying the MAC tag in DNSResponse. The security properties of each stage key in a client session are:

- Stage 1: The key has *implicit* authentication on acceptance. It is indistinguishable against any adversary that has never corrupted the server. Once stage 2 accepts, the key is *retroactively explicitly* authenticated. Stage 1 is replayable: a particular DNSQuery can be replayed multiple times to induce the same stage 1 key at server instances.
- Stage 2: The key has *explicit* authentication on acceptance. It is indistinguishable against any adversary that has never corrupted the server. This stage is nonreplayable.

Since clients are not authenticated in SL-DNSSEC, stage keys in server sessions never obtain explicit authentication. Thus, all server sessions use:

$$\mathsf{auth} = (\infty^{\times 2})$$

3.2 Match Security

We show that our model has sound behaviour of session matching. $\pi.\mathsf{sid}$ and $\pi'.\mathsf{sid}$ match, where π, π' are honest partnered sessions.

Theorem 3.1. SL-DNSSEC *is* Match*-secure. Any efficient adversary $\mathcal{A}$ has advantage*

$$\mathsf{Adv}^{\mathsf{Match}}_{\text{SL-DNSSEC},\mathcal{A}} \leq n_s \,.\, \delta + \frac{n_s^2}{2^{|\mathsf{nonce}|}}$$

where n_s is the number of sessions, $|\mathsf{nonce}|$ is the length of the nonces r_c and r_s in bits, and δ is the correctness of the static KEM.

Proof. We show that the properties of Match-security hold.

(1) By definition, the session identifiers sid(s) contain all DNS messages up to that point. The KEM shared secret ss and the hashes of the DNS messages are the only inputs to the key schedule. At stage 1, the input to the key schedule is ss and the DNSQuery. At stage 2, the input to the key schedule is ss and the messages up to and including DNSResponse. These inputs are all included[3] in the sid(s), with the consequence that both parties use the same inputs for key computations.

[3] ss is indirectly included via the corresponding ct present in DNSQuery.

The only way they could arrive at different keys is when the KEM fails. In each of the n_s sessions, the KEM may fail with a probability δ.

(2) All messages are exclusively sent or received by either role. Thus, no initiator or responder will accept an incoming message intended for the other role.
(3) By definition, cid_i is equal to sid_i whenever stage i accepts.
(4) Partnered sessions must agree once they reach (explicitly authenticated) stage 2. For Match security, we are only concerned with honest client and server sessions. The client is already cognizant of the server's identity through the DNSSEC-validated DNSKEY, which is included in sid(s).
(5) Each stage's sid has a unique label.
(6) sid_2 contains nonces r_c and r_s embedded[4] in `DNSQuery` and `DNSResponse`, respectively. To get a collision between sessions of honest parties, some session would need to pick the same nonce as another session. The probability for such a collision is given by the birthday bound $n_s^2 \cdot 2^{-|\text{nonce}|}$.

□

3.3 Multi-Stage Security

We prove the security of SL-DNSSEC via the Multi-Stage security games. The adversary $\mathcal{A}$ wins Bellare-Rogaway style if it a) correctly distinguishes the stage keys from random, or b) gets the protocol to maliciously accept (*i.e.* the client accepts an invalid `DNSResponse` containing forged MACs).

We bound the security of SL-DNSSEC on the assumption that the hash function H is collision-resistant, HKDF is a pseudorandom function in either its "salt" or "input keying material" arguments, HMAC is EUF-CMA secure and KEM is IND-CCA secure.

Theorem 3.2. *Let adversary $\mathcal{A}$ be a probabilistic polynomial-time algorithm. n_s is the number of sessions and n_u is the number of identities. There exist algorithms $\mathcal{B}_1, \ldots, \mathcal{B}_9$, such that*

$$\text{Adv}^{\text{Multi-Stage}}_{\text{SL-DNSSEC},\mathcal{A}} \leq \frac{n_s^2}{2^{|\text{nonce}|}} + \text{Adv}^{\text{COLL}}_{\text{H},\mathcal{B}_1}$$

$$+2n_s \left(n_s \left(\begin{array}{ll} \text{Adv}^{\text{IND-CCA}}_{\text{KEM},\mathcal{B}_2} & +\text{Adv}^{\text{PRF-SEC}}_{\text{HKDF.Extract},\mathcal{B}_3} \\ +\text{Adv}^{\text{PRF-SEC}}_{\text{HKDF.Expand},\mathcal{B}_4} & +\text{Adv}^{\text{EUF-CMA}}_{\text{HMAC},\mathcal{B}_5} \end{array} \right) \right.$$
$$\left. n_u \left(\begin{array}{ll} \text{Adv}^{\text{IND-CCA}}_{\text{KEM},\mathcal{B}_6} & +\text{Adv}^{\text{PRF-SEC}}_{\text{HKDF.Extract},\mathcal{B}_7} \\ +\text{Adv}^{\text{PRF-SEC}}_{\text{HKDF.Expand},\mathcal{B}_8} & +\text{Adv}^{\text{EUF-CMA}}_{\text{HMAC},\mathcal{B}_9} \end{array} \right) \right)$$

Proof. The proof proceeds by a sequence of games in which we keep reducing the advantage of the adversary until it has no chance of winning anymore. The security bound is obtained by adding up all the reductions in advantage. We further assume all tested sessions remain fresh throughout the experiment.

Game 0: Multi-Stage Game. Let G_0 be the original Multi-Stage game:

$$\text{Adv}^{\text{Multi-Stage}}_{\text{SL-DNSSEC},\mathcal{A}} = \text{Adv}^{G_0}_{\mathcal{A}}$$

[4] r_c includes 16-bit TxID and 64-bit client DNS cookie; r_s is 256-bit server DNS cookie.

Game 1: Nonce Collisions. If any honest session uses the same nonce r_c or r_s as any other session, the challenger aborts. Given that there are n_s sessions each using uniformly random nonces, the chance of a repeat is given by a birthday bound:

$$\text{Adv}^{G_0}_{\mathcal{A}} \leq \text{Adv}^{G_1}_{\mathcal{A}} + \frac{n_s^2}{2^{|\text{nonce}|}}$$

We can now rule out nonce collisions in future games.

Game 2: Hash Collisions. If any two honest sessions compute the same hash for different inputs of hash function H, the challenger aborts. If this event occurs, we obtain a reduction $\mathcal{B}_1$ that can break the collision resistance of H. $\mathcal{B}_1$ outputs the two distinct preimages when a collision occurs. We obtain:

$$\text{Adv}^{G_1}_{\mathcal{A}} \leq \text{Adv}^{G_2}_{\mathcal{A}} + \text{Adv}^{\text{COLL}}_{\text{H},\mathcal{B}_1}$$

Game 3: Single Test Query. By invoking a hybrid argument by [7], we restrict $\mathcal{A}$ to only make a single Test-query. This reduces the advantage at most by $(1/2) \cdot n_s$ for the two stages of SL-DNSSEC. Any single-query adversary $\mathcal{A}_1$ can emulate the original multi-query adversary $\mathcal{A}$ by guessing the to-be-tested session in advance. Any other Test(s) that $\mathcal{A}$ may submit, $\mathcal{A}_1$ simulates by carefully selected Reveal queries. We get the following advantage by letting $\mathcal{A}_1$ guess the right session and stage:

$$\text{Adv}^{G_2}_{\mathcal{A}} \leq 2n_s \cdot \text{Adv}^{G_3}_{\mathcal{A}_1}$$

This restriction of $\mathcal{A}$ to $\mathcal{A}_1$ means we can now refer to the session π at stage i that is tested.

Case Distinction. We consider two separate cases of Game 3.

A. In these games, denoted G_A, the tested session π has a unique contributive partner in stage 2. This means there exists a session $\pi' \neq \pi$ such that $\pi'.\text{cid}_2 = \pi.\text{cid}_2$.
B. In these games, denoted G_B, the tested session π *does not* have a unique contributive partner in stage 2. Moreover, Corrupt(π.pid) was not called before stage i of π accepted.

As by rejecting malicious acceptance, no cases exist that call Corrupt(π.pid) after stage i accepted, these cases are exhaustive.

The advantage of $\mathcal{A}$ can be considered separately for these cases:

$$\begin{aligned} \text{Adv}^{G_3}_{\mathcal{A}_1} &\leq \max\left\{\text{Adv}^{G_A}_{\mathcal{A}_1}, \text{Adv}^{G_B}_{\mathcal{A}_1}\right\} \\ &\leq \left\{\text{Adv}^{G_A}_{\mathcal{A}_1} + \text{Adv}^{G_B}_{\mathcal{A}_1}\right\} \end{aligned}$$

Case A: ∃ A Unique Stage 2 Contributive Partner

In this case, we assume that π has a π' such that $\pi.\text{cid}_2 = \pi'.\text{cid}_2$. If the tested session π is a client session, then $\pi.\text{cid}_2 = \pi.\text{sid}_2$ and a partner session at π' also exists. sid_2 includes the client and server nonces, and by game 1 no honest sessions repeat nonces. This means that the contributive partner at stage 2 is unique.

However, if π is a server session, then it may have received a replayed `DNSQuery` message. This would mean a contributive partner session exists at stage 2, but there is no partnered session. However, there exists only one honest client session that is a contributive partner: cid_2 includes the client nonce (unique by game 1) and contributive partnering includes roles.

Thus, we can speak of a particular tested session π and its unique contributive stage-2 partner π'. Let π_c and π_s denote client (role = initiator) and server (role = responder) session, respectively

Game A1: Guess contributive partner session. The challenger tries to guess the $\pi' \neq \pi$ that is the honest contributive partner to π at stage 2. As the challenger guesses correctly with probability $1/n_s$, this reduces the advantage of $\mathcal{A}_1$ as:

$$\mathsf{Adv}^{G_A}_{\mathcal{A}_1} \leq n_s \,.\, \mathsf{Adv}^{G_{A1}}_{\mathcal{A}_1}$$

Henceforth, in case A, we will keep replacing keys in π and π'.

Game A2: Static KEM. In this game, we replace the shared secret ss encapsulated to pk_S by a uniformly random $\widetilde{\mathsf{ss}}$. We make this replacement in π_c and π_s, and, as this stage is replayable, in any other sessions π'' of the server S that received ct.

Any adversary $\mathcal{A}_1$ that can detect this replacement can be used to construct an adversary $\mathcal{B}_2$ that breaks the IND-CCA security of KEM. $\mathcal{B}_2$ obtains the IND-CCA challenge $\mathsf{pk}^*, \mathsf{ct}^*$ and challenge shared secret ss^* and uses pk^* as the long-term key pk_S of the server S. In π_c , $\mathcal{B}_2$ sends ct^* in the DNSQuery. $\mathcal{B}_2$ uses ss^* for ss in both π_c and π_s. If $\mathcal{A}_1$ delivers ct^* to some other session π'' of S, $\mathcal{B}_2$ uses ss^* as value for ss in π''. If $\mathcal{A}_1$ delivers a different $\mathsf{ct}' \neq \mathsf{ct}^*$ to π'', $\mathcal{B}_2$ queries its IND-CCA decapsulation oracle with ct' to obtain the required shared secret.

Stage 1 cannot maliciously accept since it is replayable. By the definition of freshness, we do not need to answer Corrupt queries.

$\mathcal{A}_1$ terminates and outputs its guess of the uniform bit b. If ss^* was the real shared secret, $\mathcal{B}_2$ has exactly simulated G_{A1} to $\mathcal{A}_1$. If it was random, $\mathcal{B}_2$ has exactly simulated G_{A2} to $\mathcal{A}_1$. We obtain:

$$\mathsf{Adv}^{G_{A1}}_{\mathcal{A}_1} \leq \mathsf{Adv}^{G_{A2}}_{\mathcal{A}_1} + \mathsf{Adv}^{\text{IND-CCA}}_{\mathsf{KEM},\mathcal{B}_2}$$

Game A3: Replacing K_0. In this game, we replace K_0 by a uniformly random $\widetilde{\mathsf{K}_0}$ in π_c and π_s.

Any adversary $\mathcal{A}_1$ that can detect this change can be used to construct an adversary $\mathcal{B}_3$ that breaks the PRF security of HKDF.Extract in its second argument. When $\mathcal{B}_3$ needs to compute K_0 in π_c or π_s, it queries its HKDF.Extract challenge oracle (keyed with ss) on 0 and uses the response as K_0 . If the response was the real shared secret, $\mathcal{B}_3$ has exactly simulated G_{A2} to $\mathcal{A}_1$. If it was a random value, $\mathcal{B}_3$ has exactly simulated G_{A3} to $\mathcal{A}_1$. We obtain:

$$\mathsf{Adv}^{G_{A2}}_{\mathcal{A}_1} \leq \mathsf{Adv}^{G_{A3}}_{\mathcal{A}_1} + \mathsf{Adv}^{\text{PRF-SEC}}_{\mathsf{HKDF.Extract},\mathcal{B}_3}$$

Game A4: Replacing K_1 ***and*** K_2. In this game, we replace K_1 by random values in π_c and π_s. We also replace K_2 by a random value in π_s. If π_c received the same DNSResponse that π_s sent (and π_s had received the same ct), we make the same replacements in π_c.

Any adversary $\mathcal{A}_1$ that can detect this change can be used to construct an adversary $\mathcal{B}_4$ that breaks the PRF security of HKDF.Expand. When $\mathcal{B}_4$ needs to compute K_1 or K_2 in π_c or π_s, it queries its HKDF.Expand challenge oracle (keyed with K_0) with the corresponding labels and transcripts. If the response was the real output, $\mathcal{B}_4$ has exactly simulated G_{A3} to $\mathcal{A}_1$. If it was a random value, $\mathcal{B}_4$ has exactly simulated G_{A4} to $\mathcal{A}_1$.

Note that if π_c shares ct with π_s , but parts of DNSResponse were changed such that π_s and π_c are no longer partnered at stage 2, the adversary may issue Reveal query for stage 2. But since any changes to DNSResponse make the transcript in π_s and π_c different, the label input to HKDF.Expand is now different for K_2. This means the simulation in $\mathcal{B}_4$ remains good. Hence:

$$\mathsf{Adv}^{G_{A3}}_{\mathcal{A}_1} \leq \mathsf{Adv}^{G_{A4}}_{\mathcal{A}_1} + \mathsf{Adv}^{\text{PRF-SEC}}_{\mathsf{HKDF.Expand},\mathcal{B}_4}$$

The stage keys K_1 and K_2 are now a uniformly random strings independent of anything else in the game.

Malicious Acceptance. Let bad denote the event that G_{A4} maliciously accepts in stage 2 in the (fresh) tested client session without a session partner. Since server sessions that do not authenticate the client, we do not have explicit authentication.

Game A5: Identical-until-bad. This game is identical to game G_{A4}, except that we abort the game if the event bad occurs. Games G_{A4} and G_{A5} are identical-until-bad [3]. Thus,

$$\left|\Pr[G_{A4} \implies 1] - \Pr[G_{A5} \implies 1]\right| \leq \Pr[G_{A5} \text{ reaches bad}]$$

In game G_{A4}, all stage keys in the tested session are uniformly random and independent of all messages in the game. The adversary cannot distinguish stage keys anymore. By this game, it can no longer reach bad. Thus:

$$\mathsf{Adv}^{G_{A5}}_{\mathcal{A}_1} = 0$$

It remains to bound $\Pr[G_{A5} \text{ reaches bad}]$.

Game A6: HMAC Forgery. In this game, if π_c does not have a session partner in stage 2, it rejects the DNSResponse.

Any adversary that behaves differently in G_{A6} compared to G_{A5} can be used to construct an HMAC forger $\mathcal{B}_5$. The only way that G_{A6} and G_{A5} behave differently is when G_{A6} rejects a MAC that should have been accepted as valid. When rejecting DNSResponse, if no partner to π_c at stage 2 exists, no honest π_s exists with the same session identifier and thus transcript. This means no honest π_s ever created MAC tags for the DNS RRsets that the client verified, and thus it must be a forgery. Therefore,

$$\Pr[G_{A5} \text{ reaches bad}] \leq \Pr[G_{A6} \text{ reaches bad}] + \mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{HMAC},\mathcal{B}_5}$$

By the above, the event bad is never reached.

Analysis of Game A6. By G_{A4}, all stage keys are uniformly random and independent of all messages in the game. Hence, the hidden bit b is now independent of all the information sent to the adversary. By G_{A6}, all events bad are rejected. Thus:

$$\Pr[G_{A6} \text{ reaches bad}] = 0$$

This concludes case A, yielding:

$$\mathsf{Adv}^{G_A}_{\mathcal{A}_1} \leq n_s \begin{pmatrix} \mathsf{Adv}^{\text{IND-CCA}}_{\mathsf{KEM},\mathcal{B}_2} & +\mathsf{Adv}^{\text{PRF-SEC}}_{\mathsf{HKDF.Extract},\mathcal{B}_3} \\ +\mathsf{Adv}^{\text{PRF-SEC}}_{\mathsf{HKDF.Expand},\mathcal{B}_4} & +\mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{HMAC},\mathcal{B}_5} \end{pmatrix}$$

Case B: No contributive partner in stage 2 exists, and peer is not corrupted before stage i accepted

In this case, the tested client session π_c does not have a contributive partner in stage 2. This means that the adversary $\mathcal{A}$ is actively impersonating the server to π_c.

Case B allows $\mathcal{A}$ to corrupt the intended server's static key after the tested session accepts in stage 2. However, we show by assumption on the EUF-CMA security of HMAC that no client can be made to maliciously accept at stage 2. If a client accepts in stage 2, then it has a partner at stage 2 and all prior stages.

Thus, once stage 2 accepts, all client stages are retroactively authenticated.

Game B1: Guessing the intended peer. In this game, we guess the server identity V with which the tested π_c attempts to connect. In case of a wrong guess, *i.e.* $V \neq \pi_c$.pid, we abort. This reduces the advantage of $\mathcal{A}_1$ by a factor of the number of users n_u:

$$\mathsf{Adv}^{G_B}_{\mathcal{A}_1} \leq n_u \,.\, \mathsf{Adv}^{G_{B1}}_{\mathcal{A}_1}$$

Game B2: Static KEM. In this game, we replace the shared secret ss in π_c with a uniformly random $\widetilde{\mathsf{ss}}$. In any (server) sessions π_s of V that received the same ct as was sent by π_c in the DNSQuery, we make the same replacement. Thus, all values derived from ss in π_c and π_s use the new value $\widetilde{\mathsf{ss}}$.

Any adversary $\mathcal{A}_1$ that can detect this replacement can be used to construct an adversary $\mathcal{B}_6$ that breaks the IND-CCA security of KEM. $\mathcal{B}_6$ obtains the IND-CCA challenge $\mathsf{pk}^*, \mathsf{ct}^*$ and challenge shared secret ss^* and gives pk^* to $\mathcal{A}_1$. In π_c, $\mathcal{B}_7$ sends ct^* in DNSQuery and uses ss^*. If $\mathcal{A}_1$ delivers ct^* to any π_s of V, $\mathcal{B}_6$ uses ss^* in π_s. If $\mathcal{A}_1$ delivers some other $\mathsf{ct}' \neq \mathsf{ct}^*$, $\mathcal{B}_6$ queries its IND-CCA decapsulation oracle with ct' to obtain the required shared secret. By the definition of case B, we will never need to answer any $\mathsf{Corrupt}(V)$ queries.

In the end, $\mathcal{A}_1$ terminates and outputs its guess of the uniform bit b. If ss^* was the real shared secret, $\mathcal{B}_6$ has exactly simulated G_{B1} to $\mathcal{A}_1$. If it was a random value, $\mathcal{B}_6$ has exactly simulated G_{B2} to $\mathcal{A}_1$. We obtain:

$$\mathsf{Adv}^{G_{B1}}_{\mathcal{A}_1} \leq \mathsf{Adv}^{G_{B2}}_{\mathcal{A}_1} + \mathsf{Adv}^{\text{IND-CCA}}_{\mathsf{KEM},\mathcal{B}_6}$$

Game B3: Replacing K_0. In this game, we replace K_0 by a random value $\widetilde{\mathsf{K}_0}$. Additionally, in any sessions π_s of V which received the same ct that was sent in π_c, we make the same replacement.

Any $\mathcal{A}_1$ that can detect this change can be used to construct an adversary $\mathcal{B}_7$ that breaks the PRF security of HKDF.Extract in its second argument as follows. When $\mathcal{B}_7$ needs to compute K_0 in π_c or any π_s of V, $\mathcal{B}_8$ uses its HKDF.Extract challenge oracle (keyed with ss) on 0. It uses the response as K_0. If the response was the real value, $\mathcal{B}_7$ has exactly simulated G_{B2} to $\mathcal{A}_1$. If it was a random value,$\mathcal{B}_7$ has exactly simulated G_{B3} to $\mathcal{A}_1$. We obtain:

$$\mathsf{Adv}^{G_{B2}}_{\mathcal{A}_1} \leq \mathsf{Adv}^{G_{B3}}_{\mathcal{A}_1} + \mathsf{Adv}^{\text{PRF-SEC}}_{\mathsf{HKDF.Extract},\mathcal{B}_7}$$

Game B4: Replacing K_1 ***and*** K_2. In this game, we replace the values K_1 and K_2 by uniformly random values in π_c. Additionally, in any sessions π_s of V which received the same ct that was sent in π_c, we make the same replacements.

Any $\mathcal{A}_1$ that can detect this change can be used to construct an adversary $\mathcal{B}_8$ that breaks the PRF security of HKDF.Expand. When $\mathcal{B}_8$ needs to compute K_1 or K_2 in π_c or π_s, it queries its HKDF.Expand challenge oracle (keyed with K_0) with the corresponding labels and transcripts. If the response was the real shared secret, $\mathcal{B}_8$ has exactly simulated G_{B3} to $\mathcal{A}_1$. If it was a random value, $\mathcal{B}_8$ has exactly simulated G_{B4} to $\mathcal{A}_1$. We obtain:

$$\mathsf{Adv}^{G_{B3}}_{\mathcal{A}_1} \leq \mathsf{Adv}^{G_{B4}}_{\mathcal{A}_1} + \mathsf{Adv}^{\text{PRF-SEC}}_{\mathsf{HKDF.Expand},\mathcal{B}_8}$$

The stage keys K_1 and K_1 are now uniformly random strings independent of anything else in the game.

Malicious Acceptance. Let bad denote the event that G_{B4} maliciously accepts in stage 2 in the (fresh) tested client session without a session partner in stage 2. Since server sessions that do not authenticate the client, we do not have explicit authentication.

Game B5: Identical-until-bad. This game is identical to game G_{B4}, except that we abort the game if the event bad occurs. Games G_{B4} and G_{B5} are identical-until-bad [3]. Thus,

$$\left|\Pr[G_{B4} \implies 1] - \Pr[G_{B5} \implies 1]\right| \leq \Pr[G_{B5} \text{ reaches bad}]$$

In game G_{B4}, all stage keys in the tested session are uniformly random and independent of all messages in the game. The adversary cannot distinguish stage keys anymore. By this game, it can no longer reach bad. Thus:

$$\mathsf{Adv}^{G_{B5}}_{\mathcal{A}_1} = 0$$

It remains to bound $\Pr[G_{B5} \text{ reaches bad}]$.

Game B6: HMAC Forgery. In this game, π_c rejects upon receiving the DNSResponse message. Any adversary that behaves differently in G_{B6} compared to G_{B5} can be used to construct an HMAC forger $\mathcal{B}_9$. The only way that G_{B6} and G_{B5} behave differently is if G_{B6} rejects a MAC that should have been accepted as valid. When rejecting DNSResponse, if no partner to π_c at stage 2 exists, no honest π_s exists with the same session identifier and thus transcript. This means no honest π_s ever created MAC tags for the DNS RRSets that the client verified, and thus it must be a forgery. Therefore,

$$\Pr[G_{B5} \text{ reaches bad}] \leq \Pr[G_{B6} \text{ reaches bad}] + \mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{HMAC},\mathcal{B}_9}$$

Since this game rejects all DNSResponse messages, the event bad is never reached in client sessions.

Analysis of Game B6. By G_{B4}, all stage keys are uniformly random and independent of all messages in the game. Hence, the hidden bit b is now independent of all the information sent to the adversary. By G_{B6}, all events bad are rejected. Thus:

$$\Pr[G_{B6} \text{ reaches bad}] = 0$$

This concludes case B, yielding:

$$\mathsf{Adv}^{G_B}_{\mathcal{A}_1} \leq n_u \begin{pmatrix} \mathsf{Adv}^{\text{IND-CCA}}_{\mathsf{KEM},\mathcal{B}_6} & +\mathsf{Adv}^{\text{PRF-SEC}}_{\mathsf{HKDF.Extract},\mathcal{B}_7} \\ +\mathsf{Adv}^{\text{PRF-SEC}}_{\mathsf{HKDF.Expand},\mathcal{B}_8} & +\mathsf{Adv}^{\text{EUF-CMA}}_{\mathsf{HMAC},\mathcal{B}_9} \end{pmatrix}$$

Combining the bounds in cases A and B yields the theorem. □

References

[1] Mihir Bellare. 2006. New Proofs for NMAC and HMAC: Security Without Collision-Resistance. In *CRYPTO*.
[2] Mihir Bellare and Phillip Rogaway. 1994. Entity Authentication and Key Distribution. In *CRYPTO*.
[3] Mihir Bellare and Phillip Rogaway. 2006. The security of triple encryption and a framework for code-based game-playing proofs. In *EUROCRYPT*.
[4] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. 2021. A Cryptographic Analysis of the TLS 1.3 Handshake Protocol. *IACR JOC* (2021).
[5] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. 2015. A Cryptographic Analysis of the TLS 1.3 Handshake Protocol Candidates. In *ACM CCS*.
[6] Marc Fischlin and Felix Günther. 2014. Multi-Stage Key Exchange and the Case of Google's QUIC Protocol. In *ACM CCS*.
[7] Felix Günther. 2018. *Modeling Advanced Security Aspects of Key Exchange and Secure Channel Protocols.".* Ph. D. Dissertation. TU Darmstadt.
[8] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. 2017. A Modular Analysis of the Fujisaki-Okamoto Transformation. In *TCC*.
[9] Aditya Singh Rawat and Mahabir Prasad Jhanwar. 2025. Quantum-safe Signatureless DNSSEC. In *ACM AsiaCCS*.
[10] Peter Schwabe, Douglas Stebila, and Thom Wiggers. 2020. Post-Quantum TLS Without Handshake Signatures. In *ACM CCS*.
[11] Peter Schwabe, Douglas Stebila, and Thom Wiggers. 2021. More Efficient Post-quantum KEMTLS with Pre-distributed Public Keys. In *ESORICS*.
[12] Fang Song. 2014. A Note on Quantum Security for Post-Quantum Cryptography. In *PQCrypto*.
[13] Thom Wiggers. 2024. *Post-Quantum TLS.* Ph. D. Dissertation. Radboud University.

A Cryptography

Notation. $\mathbb{N}$ denotes the set of natural numbers $\{1, 2, \ldots\}$. $a \stackrel{?}{=} b$ is a boolean expression for equality. $x \leftarrow c$ indicates an assignment of the value c to a variable x. $x \xleftarrow{\$} X$ designates sampling an element uniformly at random from a set X and storing it in x. If A is a *deterministic* algorithm, then $y \leftarrow A(x)$ symbolizes running A with input x and storing the output in y. If A is a *probabilistic* algorithm, then $y \xleftarrow{\$} A(x)$ denotes running A with input x and uniformly random coins, and storing the output in y.

Adversary's Advantage

In order to compare suitable instances of cryptographic primitives, the security level of an algorithm is expressed in *bits*.

Definition 5. *Given a security level of n bits for a cryptographic primitive against a particular attack, a probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ has at most 2^{-n} chance of performing the attack successfully, i.e.*

$$\Pr[\mathcal{A} \text{ is successful}] \leq \frac{1}{2^n}$$

Alternatively, $\mathcal{A}$ requires at least 2^n operations to likely succeed.

In formal proofs, we model the security properties of a particular system as a *game* wherein an adversary wins if it can output a certain value[5]. Moreover, to win the game, the adversary may do better than simply guessing. This is called the adversary's *advantage*.

Definition 6. *The advantage* Adv *of a PPT adversary $\mathcal{A}$ in a security game G for a primitive P is:*

$$\mathsf{Adv}^{G}_{P,\mathcal{A}}$$

For a system to be secure, this advantage should be *negligible*.

[5] Usually, a bit that was set at the start of the game.

A.1 Symmetric Cryptography

The salient property of symmetric primitives is the absence of a so-called *public* operation. In this setting, both parties share a secret key (hence, the *symmetry*) that is usually *a priori* agreed upon by running a key exchange protocol.

Hash Functions

Hash functions map arbitrary inputs to fixed-length outputs.

Definition 7. *A hash function*

$$\mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda$$

maps an arbitrary-length message $m \in \{0,1\}^*$ to a hash value $\mathsf{H}(m) \in \{0,1\}^\lambda$ of a fixed length $\lambda \in \mathbb{N}$.

Cryptographic hash functions have the following properties:

- *Pre-image Resistance.* It must be infeasible to recover the input m from any output $\mathsf{H}(m)$.
- *Second Pre-image Resistance.* It must be intractable to find any other input m', such that $m' \neq m$, which produces the given output $\mathsf{H}(m)$.
- *Collision Resistance.* It must be hard to find two arbitrary inputs m, m' that produce the same output $\mathsf{H}(m) = \mathsf{H}(m')$.

In our proofs, we rely on the collision resistance of hashes.

Definition 8. *The collision resistance of a hash function* H *measures the ability of a PPT adversary $\mathcal{A}$ to find two distinct messages that hash to the same output:*

$$\mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{A}} = \Pr\left[(m, m') \xleftarrow{\$} \mathcal{A} : (m \neq m') \wedge (\mathsf{H}(m) = \mathsf{H}(m'))\right].$$

Many mechanisms, such as MAC and KDF, internally use hashes.

Message Authentication Code (MAC)

A MAC allows computing an *authentication tag* from a shared secret key and a message. The recipient of a tagged message can compute the tag autonomously and compare it to the received tag, thereby gaining the assurance that the message is from the legitimate party (authenticity) and was not tampered with during transit (integrity).

Definition 9. *A Message Authentication Code (MAC)*

$$\mathsf{MAC} : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^\lambda$$

maps a key $k \in \mathcal{K}$ and a message $m \in \{0,1\}^*$ to an authentication tag $\in \{0,1\}^\lambda$ of a fixed length $\lambda \in \mathbb{N}$.

It must be infeasible for anyone other than the legitimate participants to create valid tags for messages. To this end, the EUF-CMA security experiment for MACs specifies that even a polynomial-time adversary, which can ask for tags on arbitrary messages, cannot create a valid tag for a new message without querying the oracle.

Definition 10. *The Existential Unforgeability under Chosen Message Attack (*EUF-CMA*) for a MAC measures the ability of a PPT adversary $\mathcal{A}$ to forge an authentication tag on a new message, given access to a tag-generation oracle, as portrayed in Fig. 3:*

$$\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{MAC},\mathcal{A}} = \Pr\left[G^{\mathsf{EUF\text{-}CMA}}_{\mathsf{MAC},\mathcal{A}} \implies 1\right]$$

Game $G^{\text{EUF-CMA}}_{\text{MAC},\mathcal{A}}$

1 : $k \xleftarrow{\$} \mathcal{K}$
2 : $M \leftarrow \emptyset$
3 : $(m, \omega) \leftarrow \mathcal{A}^{O}$
4 : **return** $\left[(\omega \stackrel{?}{=} \text{MAC}(k, m)) \ \wedge \ (m \notin M)\right]$

Oracle $O(x)$

1 : $M \leftarrow M \cup \{x\}$
2 : **return** $\text{MAC}(k, x)$

Figure 3: EUF-CMA Game for a MAC

Key Derivation Function (KDF)

A KDF computes appropriate-length keys for symmetric primitives (*e.g.,* MACs) from arbitrary-length inputs. HKDF, a popular KDF, is a set of two algorithms that follow an *extract-then-expand* paradigm:

(1) The HKDF.Extract algorithm takes as input — a salt[6] and an input keying material — and outputs a pseudorandom key PRK.
(2) The HKDF.Expand algorithm takes as input — PRK, context information, and length — and generates an output of the desired length.

The context information is set to a label and a hash of the transcript. The labels separate the different arenas in which the key is used, and the transcript ensures that the key is bound to a specific DNS session. We also omit specifying the output length, and assume that a requisite-length for the output key is chosen *implicitly*.

We assume HKDF to be a pseudorandom function (PRF), and rely on the (dual) PRF-security assumption.

Definition 11. *A pseudorandom function*

$$\text{PRF} : \mathcal{K} \times \mathcal{L} \rightarrow \{0,1\}^{\lambda}$$

maps a key $k \in \mathcal{K}$ and a label $\ell \in \mathcal{L}$ to an output $\in \{0,1\}^{\lambda}$ of a fixed length $\lambda \in \mathbb{N}$.

Definition 12. *The PRF-security of a pseudorandom function* PRF *measures the ability of a PPT adversary $\mathcal{A}$ to distinguish the output of* PRF *from random:*

$$\text{Adv}^{\text{PRF-SEC}}_{\text{PRF},\mathcal{A}} = \left|\Pr\left[k \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\text{PRF}(k,\cdot)} \implies 1\right] - \Pr\left[\mathcal{A}^{R(\cdot)} \implies 1\right]\right|$$

where $R : \mathcal{L} \rightarrow \{0,1\}^{\lambda}$ is a true random function.

A pseudorandom function PRF satisfies *dual*-PRF security [1] if it is a pseudorandom function with respect to *either* of its inputs k or ℓ being the key, *i.e.* both $\text{PRF}(\ell, k)$ and $\text{PRF}(k, \ell)$ have PRF-security.

A.2 Asymmetric Cryptography

Asymmetric or public-key algorithms define a *public* and a *private* operation. The public key can be provided to anyone, including adversaries. Without the secret key however, they cannot perform the apposite private operation. Besides digital signatures, the other prominent asymmetric primitive is a key exchange mechanism.

[6] An optional, non-secret random value.

Key Encapsulation Mechanism (KEM)

Definition 13. *A Key Encapsulation Mechanism (KEM) is an asymmetric primitive that allows two parties to establish a shared secret in a key space $\mathcal{K}$.*

A KEM instance defines three probabilistic operations:

- *Key Generation.* KEM.keygen generates a public and private keypair (pk, sk).
- *Encapsulation.* KEM.encap(pk) generates a shared secret ss in a key space $\mathcal{K}$ and a ciphertext ct against pk.
- *Decapsulation.* KEM.decap(sk, ct) takes as input sk and ct, and decapsulates the shared secret $\text{ss}' \in \mathcal{K}$. In a δ-correct scheme (Def. 14), $\Pr(\text{ss} = \text{ss}') \geq 1 - \delta$.

Some KEMs have a minuscule chance of failure, wherein the sender and the receiver may derive *different* shared secrets from the *same* ciphertext. This property is captured with the notion of *delta*-correctness of KEMs.

Definition 14. *A* KEM *instance is δ-correct [8] if*

$$\Pr\left[\text{ss}' \neq \text{ss} \ \middle| \ (\text{ss}, \text{ct}) \xleftarrow{\$} \text{KEM.encap}(\text{pk}), \ \text{ss}' \xleftarrow{\$} \text{KEM.decap}(\text{sk}, \text{ct})\right] \leq \delta$$

taking the probability over $(\text{pk}, \text{sk}) \xleftarrow{\$} \text{KEM.keygen}(\,)$ *and the random coins that are used in* KEM.encap(pk).

Security requires that the shared secret ss should be indistinguishable from random (IND), given access to a decapsulation oracle — Chosen Ciphertext Attack (CCA). The indistinguishability experiment IND-CCA for a KEM is delineated in Fig. 4.

Game $G^{\text{IND-CCA}}_{\text{KEM},\mathcal{A}}$

1 : $(\text{pk}^{\star}, \text{sk}^{\star}) \xleftarrow{\$} \text{KEM.keygen}(\,)$
2 : $b \xleftarrow{\$} \{0,1\}$
3 : $(\text{ss}^{\star}_0, \text{ct}^{\star}) \xleftarrow{\$} \text{KEM.encap}(\text{pk}^{\star})$
4 : $\text{ss}^{\star}_1 \xleftarrow{\$} \mathcal{K}$
5 : $b' \xleftarrow{\$} \mathcal{A}^{O}(\text{pk}^{\star}, \text{ct}^{\star}, \text{ss}^{\star}_b)$
6 : **return** $[b' \stackrel{?}{=} b]$

Oracle $O(\text{ct})$ for IND-CCA

1 : **if** $\text{ct} \neq \text{ct}^{\star}$ **then**
2 : **return** $\text{KEM.decap}(\text{sk}^{\star}, \text{ct})$
3 : **else**
4 : **return** $\perp$
5 : **end if**

Figure 4: IND-CCA Game for a KEM

Definition 15. *The advantage of $\mathcal{A}$ in breaking the* IND-CCA *security of* KEM *is:*

$$\text{Adv}^{\text{IND-CCA}}_{\text{KEM},\mathcal{A}} = \left| \Pr\left[G^{\text{IND-CCA}}_{\text{KEM},\mathcal{A}} \implies 1\right] - \frac{1}{2} \right|$$