Analysing the HPKE Standard [⋆]

Joël Alwen[1], Bruno Blanchet[2], Eduard Hauck[3], Eike Kiltz[3], Benjamin Lipp[2], and Doreen Riepel[3]

[1] Wickr
jalwen@wickr.com
[2] Inria Paris
{bruno.blanchet,benjamin.lipp}@inria.fr
[3] Ruhr-Universität Bochum
{eduard.hauck,eike.kiltz,doreen.riepel}@rub.de

Abstract. The *Hybrid Public Key Encryption* (HPKE) scheme is an emerging standard currently under consideration by the Crypto Forum Research Group (CFRG) of the IETF as a candidate for formal approval. Of the four modes of HPKE, we analyse the authenticated mode $\mathsf{HPKE}_{\mathsf{Auth}}$ in its single-shot encryption form as it contains what is, arguably, the most novel part of HPKE.
$\mathsf{HPKE}_{\mathsf{Auth}}$'s intended application domain is captured by a new primitive which we call Authenticated Public Key Encryption (APKE). We provide syntax and security definitions for APKE schemes, as well as for the related Authenticated Key Encapsulation Mechanisms (AKEMs). We prove security of the AKEM scheme $\mathsf{DH\text{-}AKEM}$ underlying $\mathsf{HPKE}_{\mathsf{Auth}}$ based on the Gap Diffie-Hellman assumption and provide general AKEM/DEM composition theorems with which to argue about $\mathsf{HPKE}_{\mathsf{Auth}}$'s security. To this end, we also formally analyse $\mathsf{HPKE}_{\mathsf{Auth}}$'s key schedule and key derivation functions. To increase confidence in our results we use the automatic theorem proving tool CryptoVerif. All our bounds are quantitative and we discuss their practical implications for $\mathsf{HPKE}_{\mathsf{Auth}}$.
As an independent contribution we propose the new framework of *nominal groups* that allows us to capture abstract syntactical and security properties of practical elliptic curves, including the Curve25519 and Curve448 based groups (which do not constitute cyclic groups).

Keywords. public-key encryption, authentication, signcryption, key encapsulation mechanisms

1 Introduction

An effort is currently underway by the Crypto Forum Research Group (CFRG) to agree upon a new open standard for public key encryption [6]. The standard will be called *Hybrid Public Key Encryption* (HPKE) and it is, in particular, expected to be used as a building block by the Internet Engineering Task Force (IETF) in at least two further upcoming standardized security protocols [5,31]. The primary source for HPKE is an RFC [6] (on draft 8 at the time this analysis was done) which lays out the details of the construction and provides some rough intuition for its security properties.

At first glance the HPKE standard might be thought of as a "public key encryption" scheme in the spirit of the KEM/DEM paradigm [16]. That is, it combines a Key Encapsulation Mechanism (KEM) and an Authenticated Encryption with Associated Data

[⋆] This paper is an extended version of a paper published at Eurocrypt 2021 [3].

(AEAD) acting as a Data Encapsulation Mechanism (DEM) according to the KEM/DEM paradigm. However, upon closer inspection HPKE turns out to be more complex than this perfunctory description implies.

First, HPKE actually consists of 2 different KEM/DEM constructions. Moreover, each construction can be instantiated with a pre-shared key (PSK) known to both sender and receiver, which is used in the key schedule to derive the DEM key. In total this gives rise to 4 different *modes* for HPKE. The *basic* mode $\mathsf{HPKE}_{\mathsf{Base}}$ makes use of a standard (say IND-CCA-secure) KEM to obtain a "message privacy and integrity" only mode. This mode can be extended to $\mathsf{HPKE}_{\mathsf{PSK}}$ to support authentication of the sender via a PSK.

The remaining 2 HPKE modes make use of a different KEM/DEM construction built from a rather non-standard KEM variant which we call an *Authenticated KEM* (AKEM). Roughly speaking, an AKEM can be thought of the KEM analogue of signcryption [32]. In particular, sender and receiver both have their own public/private keys. Each party requires their own private and the other party's public key to perform en/decryption. The HPKE RFC constructs an AKEM based on a generic Diffie-Hellman group. It goes on to fix concrete instantiations of such groups using either the P-256, P-384, or P-521 NIST curves [29] or the Curve25519 or Curve448 curves [26]. The AKEM-based HPKE modes also intend to authenticate the sender to the receiver. Just as in the KEM-based case, the AKEM/DEM construction can be instantiated in modes either with or without a PSK. We refer to the AKEM/DEM-based mode without a PSK as the *authenticated mode* and, for reasons described below, it is the main focus of this work. The corresponding HPKE scheme is called $\mathsf{HPKE}_{\mathsf{Auth}}$.

Orthogonal to the choice of mode in use, HPKE also provides a so called single-shot and a multi-shot API. The single-shot API can be thought of as pairing a single instance of the DEM with a KEM ciphertext while the multi-shot API establishes a key schedule allowing a single KEM shared secret to be used to derive keys for an entire sequence of DEMs. Finally, HPKE also supports exporting keys from the key schedule for use by arbitrary higher-level applications.

Applications. As an open standard of the IRTF, we believe HPKE to be an interesting topic of study in its own right. Indeed, HPKE is already slated for use in at least two upcoming protocols; the Messaging Layer Security (MLS) [5] secure group messaging protocol and the Encrypted Server Name Indication (ESNI) extension for TLS 1.3 [31]. Both look to be well-served by the single-shot API as they require a single DEM to be produced (at the same time as the KEM) and the combined KEM/DEM ciphertext to be sent as one packet.

More interestingly, at least for MLS, authenticating the sender of an HPKE ciphertext (based on their public keys) is clearly also a useful property. (For the ESNI application things are less clear.[4])

In a bit more detail, MLS is already equipped with a notion of a PKI involving public keys bound to long-term identities of parties (as described in [30]). To invite a new member to an existing MLS protocol session the inviter must send an HPKE ciphertext to the new

[4] The ESNI RFC calls for a client initiating a TLS connection to send an HPKE ciphertext to the server. Although not as common, TLS can also be used in settings with bi-directional authentication. In particular, clients can use certificates binding their identities to their public key to authenticate themselves to the server. Unfortunately, it is unclear how the server would know, a priori, which public key to use for the client when attempting to decrypt the HPKE ciphertext.

member. In line with MLS's strong authentication goals, the new member is expected to be able to cryptographically validate the (supposed) identity of the sender of such ciphertexts.

Currently, MLS calls for the HPKE ciphertext to be produced using HPKE's basic mode $\mathsf{HPKE}_{\mathsf{Base}}$ and the resulting ciphertext to be signed by the inviter using a digital signature scheme (either ECDSA or EdDSA). However, an alternative approach to achieve the same ends could be to directly use HPKE in its authenticated mode $\mathsf{HPKE}_{\mathsf{Auth}}$. This would save on at least 2 modular exponentiations as well as result in packets containing 2 fewer group elements. Reducing computational and communication complexity has been a central focus of the MLS design process as such costs are considered the main hurdles to achieving the MLS's stated goal of supporting extremely large groups. Unfortunately, in our analysis, we discovered that $\mathsf{HPKE}_{\mathsf{Auth}}$ does not authenticate the sender when the receiver's secret key leaked, a key compromise impersonation (KCI) attack (Section 5.4). MLS aims to provide strong security in the face of state leakage (which includes KCI attacks), so switching from $\mathsf{HPKE}_{\mathsf{Base}}$ and signatures to $\mathsf{HPKE}_{\mathsf{Auth}}$ would result in a significant security downgrade.

$\mathsf{HPKE}_{\mathsf{Auth}}$ could also be a replacement for the public-key authenticated encryption originally implemented by the NaCl cryptographic library. $\mathsf{HPKE}_{\mathsf{Auth}}$ is safer than the NaCl implementation because, in $\mathsf{HPKE}_{\mathsf{Auth}}$, the shared secret is bound to the intended sender and recipient public keys.

1.1 Our Contributions

So far, there has been no formal analysis of the HPKE standard. Unfortunately, due to its many modes, options and features a complete analysis of HPKE from scratch seems rather too ambitious for a single work such as this one. Thus, we are forced to choose our scope more carefully. The basic mode $\mathsf{HPKE}_{\mathsf{Base}}$ (especially using the single-shot API) seems to be a quite standard construction. Therefore, and in light of the above discussion around MLS, we have opted to focus on the more novel authenticated mode in its single-shot API form $\mathsf{HPKE}_{\mathsf{Auth}}$. To this end we make the following contributions.

Authenticated KEM and PKE. We begin, in Section 5, by introducing *Authenticated Key Encapsulation Mechanisms* (AKEM) and *Authenticated Public Key Encryption* (APKE) schemes, where the syntax of APKE matches that of the single-shot authenticated mode of $\mathsf{HPKE}_{\mathsf{Auth}}$. In terms of security, we define (multi-user) security notions capturing both authenticity and (2 types of) privacy for an AKEM and an APKE. In a bit more detail, both for authenticity and for privacy we consider so called weaker *outsider* and stronger *insider* variants. Intuitively, outsider notions model settings where the adversary is an outside observer. Conversely, insider notions model settings where the adversary is somehow directly involved; in particular, even selecting some of the secrets used to produce target ciphertexts. A bit more formally, we call an honestly generated key pair *secure* if the secret key was not (explicitly) leaked to the adversary and *leaked* if it was. A key pair is called *bad* if it was sampled arbitrarily by the adversary. A scheme is outsider-secure if target ciphertexts are secure when produced using secure key pairs. Meanwhile, insider security holds even if one secure *and one bad key pair* are used. For example, insider privacy (Insider-CCA) for AKEM requires that an encapsulated key remains indistinguishable from random despite the encapsulating ciphertext being produced using bad sender keys (but secure receiver keys). Similarly, insider authenticity (Insider-Auth) requires that an

Table 1: Security properties needed to prove Outsider-Auth, Outsider-CCA, and Insider-CCA security of APKE obtained by the AKEM/DEM construction.

	AKEM			AEAD	
	Outsider-Auth	Outsider-CCA	Insider-CCA	INT-CTXT	IND-CPA
Outsider-Auth$_{\mathsf{APKE}}$	X	X		X	
Outsider-CCA$_{\mathsf{APKE}}$		X		X	X
Insider-CCA$_{\mathsf{APKE}}$			X	X	X

adversary cannot produce a valid ciphertext for bad receiver keys as long as the sender keys are secure. In particular, insider authenticity implies (but is strictly stronger than) Key Compromise Impersonation (KCI) security as KCI security only requires authenticity for leaked (but not bad) receiver keys.

Moreover, as an independent contribution we show that for each security notion of an AKEM a (significantly simpler) single-user and single-challenge-query version already implies security for its (more complex but practically relevant) multi-user version. In particular, this provides an easier target for future work on AKEMs, e.g. when building a post-quantum variant of HPKE$_{\mathsf{Auth}}$.

AKEM/DEM: FROM AKEM TO APKE. Next we turn to the AKEM/DEM construction used in the HPKE standard. We prove a set of composition results each showing a different type of security for the single-shot AKEM/DEM construction depending on which properties the underlying AKEM guarantees. Each of these results also assumes standard security properties for the AEAD (namely IND-CPA and INT-CTXT) and for the key schedule KS (namely pseudo-randomness). In particular, these results are proven in the standard model. Somewhat to our surprise, it turns out that the APKE obtained by the AKEM/DEM construction does not provide insider authenticity (and so, nor does HPKE$_{\mathsf{Auth}}$ itself). Indeed, we give an attack in Section 5.4.

Table 1 summarises the AKEM and AEAD properties we use to prove each of the remaining 3 types of security for the AKEM/DEM APKE construction.

THE HPKE$_{\mathsf{Auth}}$ SCHEME. In Section 6 we analyse the generic HPKE$_{\mathsf{Auth}}$ scheme proposed in the RFC. HPKE$_{\mathsf{Auth}}$ is an instantiation of the AKEM/DEM paradigm discussed above.

Thus, we first analyse DH-AKEM, the particular AKEM underlying HPKE$_{\mathsf{Auth}}$. The RFC builds DH-AKEM from a key-derivation function KDF and an underlying generic Diffie-Hellman group. As one of our main results we show that DH-AKEM provides authenticity and privacy based on the Gap Diffie-Hellman assumption over the underlying group. To show this we model KDF as a random oracle.

Next we consider HPKE$_{\mathsf{Auth}}$'s key schedule and prove it to be pseudo-random based on pseudo-randomness of its building blocks, the functions Extract and Expand. Similarly, we argue why DH-AKEM's key derivation function KDF can be modelled as a random oracle. Finally, by applying our results about the AKEM/DEM paradigm from the previous sections, we obtain security proofs capturing the privacy and authenticity of HPKE$_{\mathsf{Auth}}$ as an APKE. Our presentation ends with concrete bounds of HPKE$_{\mathsf{Auth}}$'s security and their interpretation.

Practice-Oriented Cryptography. Due to the very applied nature of HPKE we have taken care to maximise the practical relevance of our results. All security properties we analyse for $\mathsf{HPKE}_{\mathsf{Auth}}$ are defined directly for a multi-user setting. Further, to help practitioners set sound parameters for their HPKE applications, our results are stated in terms of very fine-grained exact (as opposed to asymptotic) terms. That is, the security loss for each result is bounded as an explicit function of various parameters such as the numbers of key pairs, queries, etc.

Finally, instead of relying on a generic prime-order group to state our underlying security assumptions, we ultimately reduce security to assumptions on each of the concrete elliptic-curve-based instantiations. For the P-256, P-384, and P-521 curves, this is relatively straightforward. However, for Curve25519 and Curve448, this is a less than trivial step as those groups (and their associated Diffie-Hellman functions X25519 and X448) depart significantly from the standard generic group abstraction. To this end we introduce the new abstraction of *nominal groups* which allows us to argue about correctness and security of our schemes over all above-mentioned elliptic curve groups, including Curve25519 and Curve448. (We believe this abstraction has applications well beyond its use in this work.) Ultimately, this approach results in both an additional security loss and the explicit consideration of (potential) new attacks not present for generic groups. In particular, both Curve25519 and Curve448 exhibit similar (but different) idiosyncrasies such as having non-equal but functionally equivalent curve points as well as self-reducibility with non-zero error probability, all of which we take into account in our reductions to the respective underlying assumption.

1.2 Proof Techniques

The results in this work have been demonstrated using a combination of traditional "pen-and-paper" techniques and the automated theorem proving tool CryptoVerif [14], which was already used to verify important practical protocols such as TLS 1.3 [13], Signal [23], and WireGuard [28]. CryptoVerif produces game-based proofs: it starts from an initial game provided by the user, which represents the protocol or scheme to prove; it transforms this game step by step using a predefined set of game transformations, until it reaches a game on which the desired security properties can easily be proved from the form of the game. The game transformations are guaranteed to produce computationally indistinguishable games, and either rely on a proof by reduction to a computational assumption or are syntactic transformations (e.g. replace a variable with its value). Using CryptoVerif to prove statements can result in greater confidence in their correctness, especially when the proofs require deriving (otherwise quite tedious) exact bounds on the security loss and/or reasoning about relatively complicated, e.g. multi-instance, security games.

However, CryptoVerif also has its limitations. Fortunately, these can be readily overcome using traditional techniques. The language used to define security statements in CryptoVerif is rather unconventional in the context of cryptography, not to mention (necessarily) very formal and detailed. Together this can make it quite challenging to build an intuitive understanding for a given notion (e.g. to verify that it captures the desired setting). To circumvent this, we present each of our security definitions using the more well-known language of game-based security. Next we map these to corresponding CryptoVerif definitions. Thus, the intuition can be built upon a game-based notion and it remains only to verify the *functional equivalence* of the CryptoVerif instantiation.

CryptoVerif was designed with multi-instance security in mind and so relies on more unconventional multi-instance number theoretic assumptions. However, the simpler a definition (say, for a KEM) the easier it is to demonstrate for a given construction. Similarly, in cryptography we tend to prefer simpler, static, not to mention well-known, number theoretic assumptions so as to build more confidence in them. Consequently, we have augmented the automated proofs with further pen-and-paper proofs reducing multi-instance security notions and assumptions to simpler (and more conventional) single-instance versions.

1.3 Related Work

Hybrid cryptography (of which the AKEM/DEM construction in this work is an example) is a widely used technique for constructing practically efficient asymmetric primitives. In particular, there exist several hybrid PKE-based concrete standards predating HPKE, mostly based on the DHIES scheme of [1] defined over a generic (discrete log) group. When the group is instantiated using elliptic curves the result is often referred to as ECIES (much like the Diffie-Hellman scheme over an elliptic curve group is referred to as ECDH). A description and comparison of the most important such standards can be found in [21]. However, per the HPKE RFC, “All these existing schemes have problems, e.g., because they rely on outdated primitives, lack proofs of IND-CCA2 security, or fail to provide test vectors.” Moreover, to the best of our knowledge, none of these standards provide a means for authenticating senders.

The APKE primitive we analyse in this paper can be viewed as a flavour of signcryption [32]; a family of primitives intended to efficiently combine signatures and public key encryption. Signcryption literature is substantial and we refer to the textbook [19] for an extensive exposition thereof. We highlight some chapters of particular relevance. Chapters 2 and 3 cover 2-party and multi-party security notions, respectively; both for insider and outsider variants. Chapter 4 of [19] contains several (Gap)-Diffie-Hellman-based signcryption constructions. Finally, Chapter 7 covers some AKEM security notions and constructions (aka. “signcryption KEM”) as well as hybrid signcryption constructions such as the outsider-secure one of [18] and insider-secure one of [17]. In contrast to our work, almost all security notions in [19] forbid honest parties from reusing the same key pair for both sending and receiving (even if sender and receiver keys have identical distribution). [5] Nor is it clear that a scheme satisfying a “key-separated” security notion could be converted into an equally efficient scheme supporting key reuse. The naïve transformation (embedding a sender and receiver key pair into a single reusable key pair) would double key sizes. However, an HPKE public key consists of a *single* group element which can be used simultaneously as a sender and receiver public key.

Recently, Bellare and Stepanovs analysed the signcryption scheme underlying the iMessage secure messaging protocol [10]. Although their security notions allow for key reuse as in our work, they fall outside the outsider/insider taxonomy common in signcryption literature. Instead, they capture an intermediary variant more akin to KCI security.

[5] The only exception we are aware of are the security notions used to analyse 2 bilinear-pairing-based schemes in Sections 5.5 and 5.6 of [19].

A detailed model of Curve25519 [26] in CryptoVerif was already presented in [28]; such a model was needed for the proof of the WireGuard protocol. In this paper, we present a more generic model that allows us to deal not only with Curve25519 but also with prime order groups such as NIST curves [29] in a single model. Moreover, we handle rerandomisation of curve elements, which was not taken into account in [28].

A very preliminary version of this work analyses HPKE as a single protocol, not in a modular KEM/DEM setting [27]. The proven theorems are less strong than the ones in this work, e.g. the adversary cannot choose secret keys but only compromise them. However, the analysis covers the single-shot encryption form of all four modes including the secret export API.

2 Preliminaries

Sets and Algorithms. We write $h \xleftarrow{\$} \mathcal{S}$ to denote that the variable h is uniformly sampled from the finite set $\mathcal{S}$. For integers $N, M \in \mathbb{N}$, we define $[N, M] := \{N, N+1, \ldots, M\}$ (which is the empty set for $M < N$), $[N] := [1, N]$ and $[N]_0 := [0, N]$. The statistical distance between two random variables U and V having a common domain $\mathcal{U}$ is defined as $\Delta[U, V] = \sum_{u \in \mathcal{U}} |\Pr[U = u] - \Pr[V = u]|$. The notation $[\![B]\!]$, where B is a boolean statement, evaluates to 1 if the statement is true and 0 otherwise.

We use uppercase letters $\mathcal{A}, \mathcal{B}$ to denote algorithms. Unless otherwise stated, algorithms are probabilistic, and we write $(y_1, \ldots) \xleftarrow{\$} \mathcal{A}(x_1, \ldots)$ to denote that $\mathcal{A}$ returns $(y_1, \ldots)$ when run on input $(x_1, \ldots)$. We write $\mathcal{A}^{\mathcal{B}}$ to denote that $\mathcal{A}$ has oracle access to $\mathcal{B}$ during its execution. For a randomised algorithm $\mathcal{A}$, we use the notation $y \in \mathcal{A}(x)$ to denote that y is a possible output of $\mathcal{A}$ on input x. We denote the running time of an algorithm $\mathcal{A}$ by $t_{\mathcal{A}}$.

Security Games. We use standard code-based security games [9]. A *game* $\mathbf{G}$ is a probability experiment in which an adversary $\mathcal{A}$ interacts with an implicit challenger that answers oracle queries issued by $\mathcal{A}$. The game $\mathbf{G}$ has one *main procedure* and an arbitrary amount of additional *oracle procedures* which describe how these oracle queries are answered. We denote the (binary) output b of game $\mathbf{G}$ between a challenger and an adversary $\mathcal{A}$ as $\mathbf{G}^{\mathcal{A}} \Rightarrow b$. $\mathcal{A}$ is said to *win* $\mathbf{G}$ if $\mathbf{G}^{\mathcal{A}} \Rightarrow 1$. Unless otherwise stated, the randomness in the probability term $\Pr[\mathbf{G}^{\mathcal{A}} \Rightarrow 1]$ is over all the random coins in game $\mathbf{G}$.

3 Standard Cryptographic Definitions

A keyed function F with a finite key space $\mathcal{K}$ and a finite output range $\mathcal{R}$ is a function $F : \mathcal{K} \times \{0,1\}^* \to \mathcal{R}$.

Definition 1 (Multi-Key Pseudorandom Function). *The* (n_k, q_{PRF})-PRF *advantage of an adversary $\mathcal{A}$ against a keyed function F with finite key space $\mathcal{K}$ and finite range $\mathcal{R}$ is defined as*

$$\mathsf{Adv}_{F,\mathcal{A}}^{(n_k, q_{\mathsf{PRF}})\text{-}\mathsf{PRF}} := \left| \Pr[\mathcal{A}^{f_1(\cdot), \ldots, f_{n_k}(\cdot)}] - \Pr_{k_1, \ldots, k_{n_k} \xleftarrow{\$} \mathcal{K}} [\mathcal{A}^{F(k_1, \cdot), \ldots, F(k_{n_k}, \cdot)}] \right| ,$$

where $f_i : \{0,1\}^* \to \mathcal{R}$ for $i \in [n_k]$ are perfect random functions and $\mathcal{A}$ makes at most q_{PRF} queries in total to the oracles f_i, resp. $F(k_i, \cdot)$.

Definition 2 (Collision Resistance). *Let $\mathcal{H}$ be a family of hash functions from $\{0,1\}^*$ to the finite range $\mathcal{R}$. We define the advantage of an adversary $\mathcal{A}$ against collision resistance of $\mathcal{H}$ as*

$$\mathsf{Adv}^{\mathsf{CR}}_{\mathcal{H},\mathcal{A}} := \Pr[\mathsf{H} \xleftarrow{\$} \mathcal{H}; x_1, x_2 \xleftarrow{\$} \mathcal{A}^{\mathsf{H}} : \mathsf{H}(x_1) = \mathsf{H}(x_2) \wedge x_1 \neq x_2] \ .$$

We now define (nonce-based) Authenticated Encryption with Associated Data.

Definition 3 (AEAD). *A nonce-based authenticated encryption scheme with associated data and key space $\mathcal{K}$ consists of the following two algorithms:*

– *Deterministic algorithm* AEAD.Enc *takes as input a key $k \in \mathcal{K}$, a message m, associated data aad and a nonce $nonce$ and outputs a ciphertext c.*
– *Deterministic algorithm* AEAD.Dec *takes as input a key $k \in \mathcal{K}$, a ciphertext c, associated data aad and a nonce $nonce$ and outputs a message m or the failure symbol $\perp$.*

We require that for all $aad \in \{0,1\}^*, m \in \{0,1\}^*, nonce \in \{0,1\}^{N_n}$

$$\Pr_{k \xleftarrow{\$} \mathcal{K}} [\mathsf{AEAD.Dec}(k, \mathsf{AEAD.Enc}(k, m, aad, nonce), aad, nonce) \neq \perp] = 1 \ ,$$

where N_n is the length of the nonce in bits.

We define the multi-key security games n_k-IND-CPA$_\ell$ and n_k-IND-CPA$_r$ in Listing 1 and (n_k, q_d)-INT-CTXT$_\ell$ and (n_k, q_d)-INT-CTXT$_r$ in Listing 2. The advantage of an adversary $\mathcal{A}$ is

$$\begin{aligned}\mathsf{Adv}^{n_k\text{-}\mathsf{IND\text{-}CPA}}_{\mathcal{A},\mathsf{AEAD}} &:= \big|\Pr[n_k\text{-}\mathsf{IND\text{-}CPA}_\ell(\mathcal{A}) \Rightarrow 1] \\ &\quad - \Pr[n_k\text{-}\mathsf{IND\text{-}CPA}_r(\mathcal{A}) \Rightarrow 1]\big| \ , \\ \mathsf{Adv}^{(n_k,q_d)\text{-}\mathsf{INT\text{-}CTXT}}_{\mathcal{A},\mathsf{AEAD}} &:= \big|\Pr[(n_k, q_d)\text{-}\mathsf{INT\text{-}CTXT}_\ell(\mathcal{A}) \Rightarrow 1] \\ &\quad - \Pr[(n_k, q_d)\text{-}\mathsf{INT\text{-}CTXT}_r(\mathcal{A}) \Rightarrow 1]\big| \ .\end{aligned}$$

Here, we define the IND-CPA and INT-CTXT notions as indistinguishability properties between a left game G_ℓ and a right game G_r, since this is required by CryptoVerif. In order to use such assumptions, CryptoVerif automatically recognizes when a game corresponds to an adversary interacting with G_ℓ, and it replaces G_ℓ with G_r in that game. Moreover, CryptoVerif requires the games G_ℓ and G_r to be formulated in a multi-key setting. That allows CryptoVerif to apply the assumption directly in case the scheme is used with several keys, without having to do a hybrid argument itself. (CryptoVerif infers the multi-key assumption automatically from a single-key assumption only in very simple cases.) Also, we allow only one query to the Enc oracle, where the experiment chooses nonces randomly. It is easy to see that these notions are implied by multi-key definitions of IND-CPA and INT-CTXT where the adversary can choose (non-repeating) nonces.

4 Elliptic Curves

In this section we introduce the elliptic curves relevant for the HPKE standard, P-256, P-384, P-521 [29], Curve25519 and Curve448 [26], together with relevant security assumptions.

Listing 1: Games n_k-$\mathsf{IND\text{-}CPA}_\ell$ and n_k-$\mathsf{IND\text{-}CPA}_r$ for AEAD. Adversary $\mathcal{A}$ makes at most one query per key to $\textsc{Enc}$.

```
$n_k$-$\mathsf{IND\text{-}CPA}_\ell$ and $n_k$-$\mathsf{IND\text{-}CPA}_r$
for $i \in [n_k]$
    $k_i \xleftarrow{\$} \mathcal{K}$
    $nonce_i \xleftarrow{\$} \{0,1\}^{N_n}$
$b \xleftarrow{\$} \mathcal{A}^{\textsc{Enc}}$
return $b$

Oracle $\textsc{Enc}(i, m, aad)$
$c \leftarrow \mathsf{AEAD.Enc}(k_i, m, aad, nonce_i)$
$c \leftarrow \mathsf{AEAD.Enc}(k_i, 0^{|m|}, aad, nonce_i)$
return $(c, nonce_i)$
```

Listing 2: Games (n_k, q_d)-$\mathsf{INT\text{-}CTXT}_\ell$ and (n_k, q_d)-$\mathsf{INT\text{-}CTXT}_r$ for AEAD. Adversary $\mathcal{A}$ makes at most one query per key to $\textsc{Enc}$ and at most q_d queries in total to $\textsc{Dec}$.

```
$(n_k, q_d)$-$\mathsf{INT\text{-}CTXT}_\ell$ and
$(n_k, q_d)$-$\mathsf{INT\text{-}CTXT}_r$
for $i \in [n_k]$
    $k_i \xleftarrow{\$} \mathcal{K}$
    $nonce_i \xleftarrow{\$} \{0,1\}^{N_n}$
$\mathcal{E} \leftarrow \emptyset$
$b \xleftarrow{\$} \mathcal{A}^{\textsc{Enc},\textsc{Dec}}$
return $b$

Oracle $\textsc{Enc}(i, m, aad)$
$c \leftarrow \mathsf{AEAD.Enc}(k_i, m, aad, nonce_i)$
$\mathcal{E} \leftarrow \mathcal{E} \cup \{i, m, c, aad\}$
return $(c, nonce_i)$

Oracle $\textsc{Dec}(i, c, aad)$
$m \leftarrow \mathsf{AEAD.Dec}(k_i, c, aad, nonce_i)$
if $\exists m' : (i, m', c, aad) \in \mathcal{E}$
    $m \leftarrow m'$
else
    $m \leftarrow \perp$
return $m$
```

4.1 Nominal Groups

We first define *nominal groups*, a general abstract model of elliptic curves, and then show how we instantiate it for each of the above-mentioned curves.

Definition 4. *A nominal group $\mathcal{N} = (\mathcal{G}, g, p, \mathcal{E}_H, \mathcal{E}_U, \mathsf{exp})$ consists of an efficiently recognizable finite set of elements $\mathcal{G}$ (also called "group elements"), a base element $g \in \mathcal{G}$, a prime p, a finite set of honest exponents $\mathcal{E}_H \subset \mathbb{Z}$, a finite set of exponents $\mathcal{E}_U \subset \mathbb{Z} \setminus p\mathbb{Z}$, and an efficiently computable exponentiation function $\mathsf{exp} : \mathcal{G} \times \mathbb{Z} \to \mathcal{G}$, where we write X^y for $\mathsf{exp}(X, y)$. The exponentiation function is required to have the following properties:*

1. *$(X^y)^z = X^{yz}$ for all $X \in \mathcal{G}$, $y, z \in \mathbb{Z}$;*
2. *the function ϕ defined by $\phi(x) = g^x$ is a bijection from $\mathcal{E}_U$ to $\{g^x \mid x \in [1, p-1]\}$.*

A nominal group is said to be rerandomisable *when:*

3. *$g^{x+py} = g^x$ for all $x, y \in \mathbb{Z}$;*
4. *for all $y \in \mathcal{E}_U$, the function ϕ_y defined by $\phi_y(x) = g^{xy}$ is a bijection from $\mathcal{E}_U$ to $\{g^x \mid x \in [1, p-1]\}$.*

We remark that even though $\mathcal{G}$ is called the set of (group) elements, it is not required to form a group. Property 2 guarantees that the discrete logarithm is unique in the set $\mathcal{E}_U$. (The index U in $\mathcal{E}_U$ stands for unique.) It is needed to define the DH oracle in Definitions 5 and 6.

For a nominal group $\mathcal{N} = (\mathcal{G}, g, p, \mathcal{E}_H, \mathcal{E}_U, \mathsf{exp})$, we let D_H be the distribution of honestly generated exponents, that is, the uniform distribution on $\mathcal{E}_H$. Let D_U be the uniform distribution on $\mathcal{E}_U$. Depending on the choice of $\mathcal{E}_H$ and $\mathcal{E}_U$, these distributions may differ. We define the two statistical parameters

$$\Delta_{\mathcal{N}} := \Delta[\mathrm{D}_H, \mathrm{D}_U], \quad \text{and} \quad P_{\mathcal{N}} = \max_{Y \in \mathcal{G}} \Pr_{x \xleftarrow{\$} \mathcal{E}_H} [Y = g^x] .$$

We summarise the expected security level and the concrete upper bounds for $\Delta_{\mathcal{N}}$ and $P_{\mathcal{N}}$ in Table 2 of Section 6.3 and compute them below.

Prime-Order Groups. The simplest example of a rerandomisable nominal group is when $\mathcal{G} = \mathbb{G}$ is a group of prime order p with generator g, exp is defined via the usual scalar multiplication on $\mathbb{G}$, and $\mathcal{E}_H = \mathcal{E}_U = [1, p-1]$.

Since $\mathcal{E}_U = [1, p-1]$, the image of ϕ is obviously $\{g^x \mid x \in [1, p-1]\}$. Furthermore, $\phi(x) = \phi(x')$ if and only if $x \equiv x' \mod p$, so ϕ is injective on $\mathcal{E}_U$.

For all $x, y \in \mathcal{E}_U$, there exists $x' \in [1, p-1]$ such that $xy \equiv x' \mod p$ (since x and y, and so xy, are prime to p). Since we work in a group of order p, $\phi_y(x) = g^{xy} = g^{x'}$, so the image of ϕ_y is included in $\{g^{x'} \mid x' \in [1, p-1]\}$. Moreover, for all $y \in \mathcal{E}_U$ and $x' \in [1, p-1]$, there exists $x \in \mathcal{E}_U$ such that $xy \equiv x' \mod p$, so $\phi_y(x) = g^{x'}$, so the image of ϕ_y is $\{g^{x'} \mid x' \in [1, p-1]\}$. Finally, for $y \in \mathcal{E}_U$, $\phi_y(x) = \phi_y(x')$ if and only if $xy \equiv x'y \mod p$, if and only if $x \equiv x' \mod p$ so ϕ_y is injective on $\mathcal{E}_U$.

The two distributions D_H and D_U are identical, so $\Delta_{\mathcal{N}} = 0$. Since all elements have the same probability, we have $P_{\mathcal{N}} = 1/(p-1)$. The NIST curves P-256, P-384, and P-521 [29] are examples of prime-order groups.

Curve25519 and Curve448. We now show that Curve25519 and Curve448 [26] can also be seen as rerandomisable nominal groups. They are elliptic curves defined by equations of the form $Y^2 = X^3 + AX^2 + X$ in the field $\mathbb{F}_q$ for a large prime q. The curve points are represented only by their X coordinate. When $X^3 + AX^2 + X$ is a square Y^2, X represents the curve point (X, Y) or $(X, -Y)$. When $X^3 + AX^2 + X$ is not a square, X does not represent a point on the curve, but on its quadratic twist. The curve is a group of cardinal kp and the twist is a group of cardinal $k'p'$, where p and p' are large primes and k and k' are small integers. For Curve25519, $q = 2^{255} - 19$, $k = 8$, $k' = 4$, $p = 2^{252} + \delta$, $p' = 2^{253} - 9 - 2\delta$ with $0 < \delta < 2^{125}$. For Curve448, $q = 2^{448} - 2^{224} - 1$, $k = k' = 4$, $p = 2^{446} - 2^{223} - \delta$, $p' = 2^{446} + \delta$ with $0 < \delta < 2^{220}$. The base point Q_0 is an element of the curve, of order p, which generates a subgroup $\mathbb{G}_s$ of the curve. The set of elements $\mathcal{G}$ is the set of bitstrings of 32 bytes for Curve25519, of 56 bytes for Curve448.

The exponentiation function is specified as follows, using [12, Theorem 2.1]: We consider the elliptic curve $E(\mathbb{F}_{q^2})$ defined by the equation $Y^2 = X^3 + AX^2 + X$ in a quadratic extension $\mathbb{F}_{q^2}$ of $\mathbb{F}_q$. We define $X_0 : E(\mathbb{F}_{q^2}) \to \mathbb{F}_{q^2}$ by $X_0(\infty) = 0$ and $X_0(X, Y) = X$. For $X \in \mathbb{F}_q$ and y an integer, we define $y \cdot X \in \mathbb{F}_q$ as $y \cdot X = X_0(yQ_X)$, where $Q_X \in E(\mathbb{F}_{q^2})$ is any of the two elements satisfying $X_0(Q_X) = X$. (It is not hard to verify that this mapping is well-defined.) Elements in $\mathcal{G}$ are mapped to elements of $\mathbb{F}_q$ by the function $\mathsf{decode\_pk} : \mathcal{G} \to \mathbb{F}_q$ and conversely, elements of $\mathbb{F}_q$ are mapped to the group elements by the function $\mathsf{encode\_pk} : \mathbb{F}_q \to \mathcal{G}$, such that $\mathsf{decode\_pk} \circ \mathsf{encode\_pk}$ is the identity. (For Curve25519 we have $\mathsf{decode\_pk}(X) = (X \bmod 2^{255}) \bmod q$, for Curve448 $\mathsf{decode\_pk}(X) = X \bmod q$, and $\mathsf{encode\_pk}(X)$ is the representation of X as an element of $\{0, \ldots, q-1\}$.) Finally, $X^y = \mathsf{encode\_pk}(y \cdot \mathsf{decode\_pk}(X))$.

As required by Definition 4, we have $(X^y)^z = X^{yz}$. Indeed,

$$\begin{aligned}(X^y)^z &= \mathsf{encode\_pk}(z \cdot \mathsf{decode\_pk}(\mathsf{encode\_pk}(y \cdot \mathsf{decode\_pk}(X)))) \\ &= \mathsf{encode\_pk}(z \cdot y \cdot \mathsf{decode\_pk}(X)) \\ &= \mathsf{encode\_pk}(yz \cdot \mathsf{decode\_pk}(X)) = X^{yz}\,.\end{aligned}$$

The base element is $g = \mathsf{encode\_pk}(X_0(Q_0))$. It is easy to check that $g^{x+py} = g^x$, since Q_0 is an element of order p. The honest exponents are chosen uniformly in the set $\mathcal{E}_H = \{kn \mid n \in [M, N]\}$. For Curve25519, $M = 2^{251}$, $N = 2^{252} - 1$. For Curve448, $M = 2^{445}$, $N = 2^{446} - 1$.

Our exponentiation function is closely related to the function $\mathsf{X25519}$ (resp. $\mathsf{X448}$ for Curve448) as defined in [26], namely $\mathsf{X25519}(y, X) = X^{\mathsf{clamp}(y)}$, where $\mathsf{clamp}(y)$ sets and resets some bits in the bitstring y to make sure that $\mathsf{clamp}(y) \in \mathcal{E}_H$. Instead of clamping secret keys together with exponentiation, we clamp them when we generate them, hence we generate honest secret keys in $\mathcal{E}_H$.

We let $\mathcal{E}_U = \{kn \mid n \in [(p+1)/2, p-1]\}$. We have $\phi(x) = g^x = \mathsf{encode\_pk}(X_0(xQ_0))$, so $\phi(x) = \phi(x')$ if and only if $\mathsf{encode\_pk}(X_0(xQ_0)) = \mathsf{encode\_pk}(X_0(x'Q_0))$, if and only if $X_0(xQ_0) = X_0(x'Q_0)$ since $\mathsf{encode\_pk}$ is injective, if and only if $xQ_0 = x'Q_0$ or $xQ_0 = -x'Q_0$, if and only if $x \equiv x' \mod p$ or $x \equiv -x' \mod p$ since Q_0 is an element of order p.

Let us show that ϕ is injective on $\mathcal{E}_U$. Suppose $x = kn$ and $x' = kn'$ are in $\mathcal{E}_U$. We have $\phi(x) = \phi(x')$ if and only if $kn \equiv kn' \mod p$ or $kn \equiv -kn' \mod p$, if and only if $n \equiv n' \mod p$ or $n \equiv -n' \mod p$ since k is prime to p, so invertible modulo p. Since n and n' are in $[(p+1)/2, p-1]$, that implies $n = n'$, so $x = x'$, which shows the injectivity of ϕ.

Let us now show that $\phi(\mathcal{E}_U) = \{g^x \mid x \in [1, p-1]\}$. If $x \in \mathcal{E}_U$, then there exists x' such that $x \equiv x' \mod p$ and $x' \in [1, p-1]$ (because x is prime to p), so $\phi(x) = \phi(x') \in \{g^x \mid x \in [1, p-1]\}$. Conversely, let $X' = g^{x'} = \phi(x')$ with $x' \in [1, p-1]$. Computing modulo p, we can find $n_0 \in [1, p-1]$ such that $kn_0 \equiv x' \mod p$ (by inverting k modulo p). If $n_0 \in [(p+1)/2, p-1]$, we let $n = n_0$. Otherwise, $n_0 \in [1, (p-1)/2]$, and we let $n = p - n_0$. In all cases, $n \in [(p+1)/2, p-1]$ and $kn \equiv x' \mod p$ or $kn \equiv -x' \mod p$, so $\phi(kn) = \phi(x') = X'$. Therefore, ϕ is a bijection from $\mathcal{E}_U$ to $\{g^x \mid x \in [1, p-1]\}$.

Similarly, for $y \in \mathcal{E}_U$, $\phi_y(x) = \phi_y(x')$ if and only if $xy \equiv x'y \mod p$ or $xy \equiv -x'y \mod p$ if and only if $x \equiv x' \mod p$ or $x \equiv -x' \mod p$ since y is prime to p. Like for ϕ, that shows the injectivity of ϕ_y on $\mathcal{E}_U$. Let us now show that $\phi_y(\mathcal{E}_U) = \{g^x \mid x \in [1, p-1]\}$. If $x \in \mathcal{E}_U$, then there exists x' such that $xy \equiv x' \mod p$ and $x' \in [1, p-1]$ (because x and y are prime to p), so $\phi_y(x) = g^{x'} \in \{g^x \mid x \in [1, p-1]\}$. Conversely, let $X' = g^{x'}$ with $x' \in [1, p-1]$. Computing modulo p, we can find $n_0 \in [1, p-1]$ such that $ykn_0 \equiv x' \mod p$ (by inverting k and y modulo p). If $n_0 \in [(p+1)/2, p-1]$, we let $n = n_0$. Otherwise, $n_0 \in [1, (p-1)/2]$, and we let $n = p - n_0$. In all cases, $n \in [(p+1)/2, p-1]$ and $ykn \equiv x' \mod p$ or $ykn \equiv -x' \mod p$, so $\phi_y(kn) = g^{x'} = X'$. Therefore, ϕ_y is a bijection from $\mathcal{E}_U$ to $\{g^x \mid x \in [1, p-1]\}$.

Lemma 1. *For Curve25519, $\Delta_{\mathcal{N}} < 2^{-126}$ and $P_{\mathcal{N}} = 2^{-250}$, and for Curve448, $\Delta_{\mathcal{N}} < 2^{-221}$ and $P_{\mathcal{N}} = 2^{-444}$.*

Proof. We have

$$\Delta_{\mathcal{N}} = \Delta[\mathrm{D}_H, \mathrm{D}_U] = \frac{1}{2} \sum_{x \in \mathbb{Z}} |\Pr_{\mathrm{D}_U}(x) - \Pr_{\mathrm{D}_H}(x)|\,.$$

For Curve25519, we have $M < (p+1)/2 < N < p-1$. The exponents kn for $n \in [M, (p-1)/2]$ each have probability 0 in D_U and $1/2^{251}$ in D_H. The exponents kn for $n \in [(p+1)/2, N]$ each have probability $2/(p-1)$ in D_U and $1/2^{251}$ in D_H. The exponents $kn \in [N+1, p-1]$ each have probability $2/(p-1)$ in D_U and 0 in D_H. Then

$$\begin{aligned}2\Delta_{\mathcal{N}} = {} & \left(\frac{p-1}{2} - M + 1\right) \times \frac{1}{2^{251}} + \left(N - \frac{p+1}{2} + 1\right) \times \left|\frac{2}{p-1} - \frac{1}{2^{251}}\right| \\ & + (p-1-(N+1)+1) \times \frac{2}{p-1}\,.\end{aligned}$$

Since $1/(p-1) < 2^{-252}$, a straightforward computation shows that

$$\Delta_{\mathcal{N}} < 2^{-126}\,.$$

For Curve448, we have $(p+1)/2 < M < p-1 < N$. The exponents kn for $n \in [(p+1)/2, M-1]$ each have probability $2/(p-1)$ in D_U and 0 in D_H. The exponents kn for $n \in [M, p-1]$ each have probability $2/(p-1)$ in D_U and $1/2^{445}$ in D_H. The exponents kn for $n \in [p, N]$ have probability 0 in D_U and $1/2^{445}$ in D_H. Then

$$\begin{aligned}2\Delta_{\mathcal{N}} = {} & \left(M - 1 - \frac{p+1}{2} + 1\right) \times \frac{2}{p-1} + (p-1-M+1) \times \left|\frac{2}{p-1} - \frac{1}{2^{445}}\right| \\ & + (N-p+1) \times \frac{1}{2^{445}}\,.\end{aligned}$$

Since $2/(p-1) > 2^{-445}$, a straightforward computation shows that

$$\Delta_{\mathcal{N}} < 2^{-221}\,.$$

Some elements g^x are generated from 2 exponents $x = kn$ for $n \in [M, N]$ (for Curve25519, the exponents kn for $n \in [M, (p+1)/2 - 1]$ yield the same elements as those for $n \in [(p+1)/2, p-M]$; for Curve448, the exponents kn for $n \in [p+1, N]$ yield the same elements as those for $n \in [2p-N, p-1]$), so they have probability $\frac{2}{N-M+1}$, and all other elements are generated from one exponent kn with $n \in [M, N]$, so they have probability $\frac{1}{N-M+1}$. Therefore, $P_{\mathcal{N}} = \frac{2}{N-M+1}$, which yields $P_{\mathcal{N}} = 2^{-250}$ for Curve25519 and $P_{\mathcal{N}} = 2^{-444}$ for Curve448. □

4.2 Diffie-Hellman Assumptions

Let us first recall the Gap Diffie-Hellman and Square Gap Diffie-Hellman assumptions. We adapt them to the setting of a nominal group $\mathcal{N} = (\mathcal{G}, g, p, \mathcal{E}_H, \mathcal{E}_U, \mathsf{exp})$ of the previous section, by allowing elements in $\mathcal{G}$ as arguments of the Diffie-Hellman decision oracle. Moreover, we choose secret keys in $\mathcal{E}_U$, not in $\mathcal{E}_H$, as it guarantees that the secret key p, or equivalently 0, is never chosen, which helps in the following theorems.

Definition 5 (Gap Diffie-Hellman (GDH) Problem). *We define the advantage function of an adversary $\mathcal{A}$ against the Gap Diffie-Hellman problem over nominal group $\mathcal{N}$ as*

$$\mathsf{Adv}^{\mathsf{GDH}}_{\mathcal{A},\mathcal{N}} := \Pr_{x,y \xleftarrow{\$} \mathcal{E}_U}[Z = g^{xy} \mid Z \xleftarrow{\$} \mathcal{A}^{\mathrm{DH}}(g^x, g^y)]$$

where DH *is a decision oracle that on input $(g^{\hat{x}}, Y, Z)$ with $\hat{x} \in \mathcal{E}_U$ and $Y, Z \in \mathcal{G}$, returns* 1 *iff $Y^{\hat{x}} = Z$ and* 0 *otherwise.*

Since ϕ is injective on $\mathcal{E}_U$ by Property 2 of Definition 4, the value of $\hat{x} \in \mathcal{E}_U$ is unambiguously defined by $g^{\hat{x}}$.

Definition 6 (Square Gap Diffie-Hellman (sqGDH) Problem). *We define the advantage function of an adversary $\mathcal{A}$ against the Square Gap Diffie-Hellman problem over nominal group $\mathcal{N}$ as*

$$\mathsf{Adv}^{\mathsf{sqGDH}}_{\mathcal{A},\mathcal{N}} := \Pr_{x \xleftarrow{\$} \mathcal{E}_U}\left[Z = g^{x^2} \mid Z \xleftarrow{\$} \mathcal{A}^{\mathrm{DH}}(g^x)\right]$$

where DH *is a decision oracle that on input $(g^{\hat{x}}, Y, Z)$, with $\hat{x} \in \mathcal{E}_U$ and $Y, Z \in \mathcal{G}$, returns* 1 *iff $Y^{\hat{x}} = Z$ and* 0 *otherwise.*

CryptoVerif cannot use cryptographic assumptions directly in this form: as explained in Section 3, it requires assumptions to be formulated as computational indistinguishability axioms between a left game G_ℓ and a right game G_r, formulated in a multi-key setting. Therefore, we reformulate the Gap Diffie-Hellman assumption to satisfy these requirements, and prove that our formulation is implied by the standard assumption.

We also take into account at this point that secret keys are actually chosen in $\mathcal{E}_H$ rather than in $\mathcal{E}_U$.

Definition 7 (Left-or-Right (n,m)-Gap Diffie-Hellman Problem). *We define the advantage function of an adversary $\mathcal{A}$ against the left-or-right (n,m)-Gap Diffie-Hellman problem over nominal group $\mathcal{N}$ as*

$$\mathsf{Adv}^{\mathsf{LoR}\text{-}(n,m)\text{-}\mathsf{GDH}}_{\mathcal{A},\mathcal{N}} := \Bigg| \Pr_{\substack{\forall i \in [n]:\, x_i \xleftarrow{\$} \mathcal{E}_H \\ \forall j \in [m]:\, y_j \xleftarrow{\$} \mathcal{E}_H}}\left[\mathcal{A}^{\mathrm{DH}_\ell, \mathrm{DH}_x, \mathrm{DH}_y}(g^{x_1}, \dots, g^{x_n}, g^{y_1}, \dots, g^{y_m}) \Rightarrow 1\right] - \Pr_{\substack{\forall i \in [n]:\, x_i \xleftarrow{\$} \mathcal{E}_H \\ \forall j \in [m]:\, y_j \xleftarrow{\$} \mathcal{E}_H}}\left[\mathcal{A}^{\mathrm{DH}_r, \mathrm{DH}_x, \mathrm{DH}_y}(g^{x_1}, \dots, g^{x_n}, g^{y_1}, \dots, g^{y_m}) \Rightarrow 1\right] \Bigg|,$$

where DH_x *is a decision oracle that on input (i, Y, Z) for $i \in [n]$ returns* 1 *iff $Y^{x_i} = Z$ and* 0 *otherwise;* DH_y *is a decision oracle that on input (j, Y, Z) for $j \in [m]$ returns* 1 *iff $Y^{y_j} = Z$ and* 0 *otherwise;* DH_ℓ *is a decision oracle that on input (i, j, Z) for $i \in [n], j \in [m]$ returns* 1 *iff $Z = g^{x_i y_j}$ and* 0 *otherwise; and* DH_r *is an oracle that on input (i, j, Z) for $i \in [n], j \in [m]$ always returns* 0.

Definition 8 (Left-or-Right n-Square Gap Diffie-Hellman Problem). *We define the advantage function of an adversary $\mathcal{A}$ against the left-or-right n-Square Gap Diffie-Hellman problem over nominal group $\mathcal{N}$ as*

$$\mathsf{Adv}^{\mathsf{LoR}\text{-}n\text{-}\mathsf{sqGDH}}_{\mathcal{A},\mathcal{N}} := \Bigg| \Pr_{\forall i \in [n]:\, x_i \xleftarrow{\$} \mathcal{E}_H}\left[\mathcal{A}^{\mathrm{DH}_\ell, \mathrm{DH}_x}(g^{x_1} \dots, g^{x_n}) \Rightarrow 1\right] - \Pr_{\forall i \in [n]:\, x_i \xleftarrow{\$} \mathcal{E}_H}\left[\mathcal{A}^{\mathrm{DH}_r, \mathrm{DH}_x}(g^{x_1}, \dots, g^{x_n}) \Rightarrow 1\right] \Bigg|,$$

where DH_x *is a decision oracle that on input* (i, Y, Z) *for* $i \in [n]$ *returns* 1 *iff* $Y^{x_i} = Z$ *and* 0 *otherwise;* DH_ℓ *is a decision oracle that on input* (i, j, Z) *for* $i, j \in [n]$ *returns* 1 *iff* $Z = g^{x_i x_j}$ *and* 0 *otherwise; and* DH_r *is an oracle that on input* (i, j, Z) *for* $i, j \in [n]$ *always returns* 0*.*

Theorem 1 ($\mathsf{GDH} \Rightarrow \mathsf{LoR}\text{-}(n,m)\text{-}\mathsf{GDH}$)**.** *Let* $\mathcal{N}$ *be a rerandomisable nominal group. For any adversary* $\mathcal{A}$ *against* $\mathsf{LoR}\text{-}(n,m)\text{-}\mathsf{GDH}$*, there exists an adversary* $\mathcal{B}$ *against* GDH *such that*

$$\mathsf{Adv}_{\mathcal{A},\mathcal{N}}^{\mathsf{LoR}\text{-}(n,m)\text{-}\mathsf{GDH}} \leq \mathsf{Adv}_{\mathcal{B},\mathcal{N}}^{\mathsf{GDH}} + (n+m)\Delta_{\mathcal{N}} \ ,$$

$\mathcal{B}$ *queries the* DH *oracle as many times as* $\mathcal{A}$ *queries* DH_x*,* DH_y*,* DH_ℓ*, or* DH_r*, and* $t_\mathcal{B} \approx t_\mathcal{A}$*.*

Proof. Let $\mathcal{A}$ be an adversary against $\mathsf{LoR}\text{-}(n,m)\text{-}\mathsf{GDH}$. We consider a game $G_1 = \forall i \in [n]\colon x_i \xleftarrow{\$} \mathcal{E}_H; \forall j \in [m]\colon y_j \xleftarrow{\$} \mathcal{E}_H; \mathcal{A}^{\mathrm{DH}_b, \mathrm{DH}_x, \mathrm{DH}_y}(g^{x_1}, \ldots, g^{x_n}, g^{y_1}, \ldots, g^{y_m})$ where DH_b is an oracle that on input $(i, j, \hat{Z})$ for $i \in [n], j \in [m]$ raises event BAD iff $\hat{Z} = g^{x_i y_j}$, and always returns 0. We have $\mathsf{Adv}_{\mathcal{A},\mathcal{N}}^{\mathsf{LoR}\text{-}(n,m)\text{-}\mathsf{GDH}} \leq \Pr[G_1 : \mathsf{BAD}]$.

We define a game G_2 that runs as G_1 except that the exponents x_i and y_j are chosen in $\mathcal{E}_U$ instead of $\mathcal{E}_H$. The probability of distinguishing one exponent in $\mathcal{E}_U$ from one in $\mathcal{E}_H$ is at most $\Delta_\mathcal{N}$ and there are $(n+m)$ such exponents, so the probability of distinguishing G_1 from G_2 is at most $(n+m)\Delta_\mathcal{N}$ and $\mathsf{Adv}_{\mathcal{A},\mathcal{N}}^{\mathsf{LoR}\text{-}(n,m)\text{-}\mathsf{GDH}} \leq \Pr[G_2 : \mathsf{BAD}] + (n+m)\Delta_\mathcal{N}$.

We define a game G_3 that runs as G_2 except that it defines $X_i = g^{x_i}$ and $Y_j = g^{y_j}$ and runs adversary $\mathcal{A}$ on input $(X_1, \ldots, X_n, Y_1, \ldots, Y_m)$; $\mathrm{DH}_x(i, \hat{Y}, \hat{Z})$ returns $\mathrm{DH}(X_i, \hat{Y}, \hat{Z})$, $\mathrm{DH}_y(j, \hat{Y}, \hat{Z})$ returns $\mathrm{DH}(Y_j, \hat{Y}, \hat{Z})$, $\mathrm{DH}_b(i, j, \hat{Z})$ raises event BAD when $\mathrm{DH}(X_i, Y_j, \hat{Z}) = 1$, and always returns 0, where DH is the oracle of Definition 5. In the call to DH inside DH_x, we have $\hat{x} = x_i$ since $g^{x_i} = X_i = g^{\hat{x}}$, x_i and $\hat{x}$ are in $\mathcal{E}_U$, and ϕ is injective on $\mathcal{E}_U$ by Property 2 of Definition 4. The situation is similar in the other oracles, so G_3 is perfectly indistinguishable from G_2.

We show how to construct an adversary $\mathcal{B}$ against GDH using random self-reducibility.

Adversary $\mathcal{B}$ inputs (X, Y) and samples $r_i \xleftarrow{\$} \mathcal{E}_U$ for $i \in [n]$ and $s_j \xleftarrow{\$} \mathcal{E}_U$ for $j \in [m]$. It computes $X_i = X^{r_i}$ and $Y_j = Y^{s_j}$ and runs adversary $\mathcal{A}$ on input $(X_1, \ldots, X_n, Y_1, \ldots, Y_m)$. On query $\mathrm{DH}_x(i, \hat{Y}, \hat{Z})$, $\mathcal{B}$ queries $\mathrm{DH}(X_i, \hat{Y}, \hat{Z})$ and forwards the answer to $\mathcal{A}$ (like G_3). On query $\mathrm{DH}_y(j, \hat{Y}, \hat{Z})$, $\mathcal{B}$ queries $\mathrm{DH}(Y_j, \hat{Y}, \hat{Z})$ and forwards the answer to $\mathcal{A}$ (like G_3). On query $\mathrm{DH}_b(i, j, \hat{Z})$, $\mathcal{B}$ queries $\mathrm{DH}(X_i, Y_j, \hat{Z})$. If the output is 0, $\mathcal{B}$ answers 0. If the output is 1, $\mathcal{B}$ immediately aborts the simulation. We can write $X = g^x$ and $Y = g^y$ with $x, y \in \mathcal{E}_U$. Then $X_i = X^{r_i} = g^{x r_i}$ and $Y_j = Y^{s_j} = g^{y s_j}$. Since DH outputs 1, there exists $\hat{x} \in \mathcal{E}_U$ such that $X_i = g^{\hat{x}}$ and $\hat{Z} = Y_j^{\hat{x}} = g^{\hat{x} y s_j} = X_i^{y s_j} = g^{x y r_i s_j}$. $\mathcal{B}$ computes the inverse modulo p of $r_i s_j$, which is an integer t such that $r_i s_j t \equiv 1 \mod p$, and $Z = \hat{Z}^t = g^{x y r_i s_j t} = g^{xy}$ since $g^{a+pb} = g^a$ for all $a, b \in \mathbb{Z}$ (Property 3 of Definition 4). $\mathcal{B}$ terminates with output Z.

Let us show that X_i in this simulation follows the same distribution as X_i in G_3. In G_3, since x_i is uniformly distributed in $\mathcal{E}_U$ and ϕ is a bijection from $\mathcal{E}_U$ to $\{g^x \mid x \in [1, p-1]\}$ (Property 2 of Definition 4), X_i is uniformly distributed in $\{g^x \mid x \in [1, p-1]\}$. In the simulation, we have $X_i = X^{r_i} = g^{x r_i}$. Since r_i is uniformly distributed in $\mathcal{E}_U$ and ϕ_x is a bijection from $\mathcal{E}_U$ to $\{g^x \mid x \in [1, p-1]\}$ (Property 4 of Definition 4), X_i is also uniformly distributed in $\{g^x \mid x \in [1, p-1]\}$. The situation is similar for the keys Y_j. Therefore, the simulation perfectly simulates G_3, and G_3 raises event BAD if and only if the simulation successfully computes Z, so $\Pr[G_3 : \mathsf{BAD}] = \mathsf{Adv}^{\mathsf{GDH}}_{\mathcal{B},\mathcal{N}}$, and $\mathsf{Adv}^{\mathsf{LoR}\text{-}(n,m)\text{-}\mathsf{GDH}}_{\mathcal{A},\mathcal{N}} \leq \mathsf{Adv}^{\mathsf{GDH}}_{\mathcal{B},\mathcal{N}} + (n+m)\Delta_{\mathcal{N}}$.

The rerandomisation performed here generalizes the one defined in [4] for Curve25519 and Curve448 to a rerandomisable nominal group. ◻

Theorem 2 ($\mathsf{sqGDH} \Rightarrow \mathsf{LoR}\text{-}n\text{-}\mathsf{sqGDH}$)**.** *Let $\mathcal{N}$ be a rerandomisable nominal group. For any adversary $\mathcal{A}$ against* $\mathsf{LoR}\text{-}n\text{-}\mathsf{sqGDH}$*, there exists an adversary $\mathcal{B}$ against* sqGDH *such that*

$$\mathsf{Adv}^{\mathsf{LoR}\text{-}n\text{-}\mathsf{sqGDH}}_{\mathcal{A},\mathcal{N}} \leq \mathsf{Adv}^{\mathsf{sqGDH}}_{\mathcal{B},\mathcal{N}} + n\Delta_{\mathcal{N}} \ ,$$

$\mathcal{B}$ queries the DH *oracle as many times as $\mathcal{A}$ queries* DH_x*,* DH_ℓ*, or* DH_r*, and $t_{\mathcal{B}} \approx t_{\mathcal{A}}$.*

Proof. The proof is similar to the proof of Theorem 1. Let $\mathcal{A}$ be an adversary against LoR-n-sqGDH. We consider a game $G_1 = \forall i \in [n] \colon x_i \xleftarrow{\$} \mathcal{E}_H; \mathcal{A}^{\mathrm{DH}_b,\mathrm{DH}_x}(g^{x_1},\dots,g^{x_n})$ where DH_b is an oracle that on input $(i,j,\hat{Z})$ for $i,j \in [n]$ raises event BAD iff $\hat{Z} = g^{x_i x_j}$, and always returns 0. We have $\mathsf{Adv}^{\mathsf{LoR}\text{-}n\text{-}\mathsf{sqGDH}}_{\mathcal{A},\mathcal{N}} \leq \Pr[G_1 : \mathsf{BAD}]$.

We define a game G_2 that runs as G_1 except that the exponents x_i are chosen in $\mathcal{E}_U$ instead of $\mathcal{E}_H$. The probability of distinguishing one exponent in $\mathcal{E}_U$ from one in $\mathcal{E}_H$ is at most $\Delta_{\mathcal{N}}$ and there are n such exponents, so the probability of distinguishing G_1 from G_2 is at most $n\Delta_{\mathcal{N}}$ and $\mathsf{Adv}^{\mathsf{LoR}\text{-}n\text{-}\mathsf{sqGDH}}_{\mathcal{A},\mathcal{N}} \leq \Pr[G_2 : \mathsf{BAD}] + n\Delta_{\mathcal{N}}$.

We define a game G_3 that runs as G_2 except that it defines $X_i = g^{x_i}$ and runs adversary $\mathcal{A}$ on input $(X_1,\dots,X_n)$; $\mathrm{DH}_x(i,\hat{Y},\hat{Z})$ returns $\mathrm{DH}(X_i,\hat{Y},\hat{Z})$, $\mathrm{DH}_b(i,j,\hat{Z})$ raises event BAD when $\mathrm{DH}(X_i,X_j,\hat{Z}) = 1$, and always returns 0, where DH is the oracle of Definition 6. Like in Theorem 1, G_3 is perfectly indistinguishable from G_2.

We show how to construct an adversary $\mathcal{B}$ against sqGDH.

Adversary $\mathcal{B}$ inputs X and samples $r_i \xleftarrow{\$} \mathcal{E}_U$ for $i \in [n]$. It computes $X_i = X^{r_i}$ and runs adversary $\mathcal{A}$ on input $(X_1,\dots,X_n)$. On query $\mathrm{DH}_x(i,\hat{Y},\hat{Z})$, $\mathcal{B}$ queries $\mathrm{DH}(X_i,\hat{Y},\hat{Z})$ and forwards the answer to $\mathcal{A}$ (like G_3). On query $\mathrm{DH}_b(i,j,\hat{Z})$, $\mathcal{B}$ queries $\mathrm{DH}(X_i,X_j,\hat{Z})$. If the output is 0, $\mathcal{B}$ answers 0. If the output is 1, $\mathcal{B}$ immediately aborts the simulation. We can write $X = g^x$ with $x \in \mathcal{E}_U$. Then $X_i = X^{r_i} = g^{xr_i}$ and $X_j = g^{xr_j}$. Since DH outputs 1, there exists $\hat{x} \in \mathcal{E}_U$ such that $X_i = g^{\hat{x}}$ and $\hat{Z} = X_j^{\hat{x}} = g^{\hat{x}xr_j} = X_i^{xr_j} = g^{x^2 r_i r_j}$. $\mathcal{B}$ computes the inverse modulo p of $r_i r_j$, which is an integer t such that $r_i r_j t \equiv 1 \mod p$, and $Z = \hat{Z}^t = g^{x^2 r_i r_j t} = g^{x^2}$ since $g^{a+pb} = g^a$ for all $a,b \in \mathbb{Z}$ (Property 3 of Definition 4). $\mathcal{B}$ terminates with output Z.

As in the proof of Theorem 1, X_i in this simulation follows the same distribution as X_i in G_3. Therefore, the simulation perfectly simulates G_3, and G_3 raises event BAD if and only if the simulation successfully computes Z, so $\Pr[G_3 : \mathsf{BAD}] = \mathsf{Adv}^{\mathsf{sqGDH}}_{\mathcal{B},\mathcal{N}}$, and $\mathsf{Adv}^{\mathsf{LoR}\text{-}n\text{-}\mathsf{sqGDH}}_{\mathcal{A},\mathcal{N}} \leq \mathsf{Adv}^{\mathsf{sqGDH}}_{\mathcal{B},\mathcal{N}} + n\Delta_{\mathcal{N}}$. ◻

Implementation in CryptoVerif. Definitions in this style for many cryptographic primitives are included in a standard library of cryptographic assumptions in CryptoVerif. As a matter of fact, this library includes a more general variant of the Gap Diffie-Hellman assumption, with corruption oracles and with a decision oracle $\mathrm{DH}(g,X,Y,Z)$, which allows the adversary to choose g. In this paper, we use the definition above as it is sufficient for our proofs.

5 Authenticated Key Encapsulation and Public Key Encryption

In Section 5.1, we introduce notation and security notions for an authenticated key encapsulation mechanism (AKEM), namely $\mathsf{Outsider\text{-}CCA}$, $\mathsf{Insider\text{-}CCA}$ and $\mathsf{Outsider\text{-}Auth}$. In Section 5.2, we introduce notation and security notions for authenticated public key encryption (APKE) which follow the ideas of the notions defined for AKEM. Additionally, we define $\mathsf{Insider\text{-}Auth}$ security.

In Section 5.3, we show how to construct an APKE scheme which achieves $\mathsf{Outsider\text{-}CCA}$, $\mathsf{Insider\text{-}CCA}$ and $\mathsf{Outsider\text{-}Auth}$, from an AKEM, a PRF, and an AEAD scheme. For $\mathsf{Insider\text{-}Auth}$, we give a concrete attack in Section 5.4.

5.1 Authenticated Key Encapsulation Mechanism

Definition 9 (AKEM)**.** *An authenticated key encapsulation mechanism* AKEM *consists of three algorithms:*

- Gen *outputs a key pair* (sk, pk)*, where* pk *defines a key space* $\mathcal{K}$*.*
- $\mathsf{AuthEncap}$ *takes as input a (sender) secret key* sk *and a (receiver) public key* pk*, and outputs an encapsulation* c *and a shared secret* $K \in \mathcal{K}$*.*
- *Deterministic* $\mathsf{AuthDecap}$ *takes as input a (receiver) secret key* sk*, a (sender) public key* pk*, and an encapsulation* c*, and outputs a shared key* $K \in \mathcal{K}$*.*

We require that for all $(sk_1, pk_1) \in \mathsf{Gen}, (sk_2, pk_2) \in \mathsf{Gen}$,

$$\Pr_{(c,K) \xleftarrow{\$} \mathsf{AuthEncap}(sk_1, pk_2)} [\mathsf{AuthDecap}(sk_2, pk_1, c) = K] = 1 \ .$$

The two sets of secret and public keys, $\mathcal{SK}$ and $\mathcal{PK}$, are defined via the support of the Gen algorithm as $\mathcal{SK} := \{sk \mid (sk, pk) \in \mathsf{Gen}\}$ and $\mathcal{PK} := \{pk \mid (sk, pk) \in \mathsf{Gen}\}$. We assume that there exists a projection function $\mu : \mathcal{SK} \to \mathcal{PK}$, such that for all $(sk, pk) \in \mathsf{Gen}$ it holds that $\mu(sk) = pk$. Note that such a function exists without loss of generality by defining sk to be the randomness rnd used in the key generation.

Finally, the key collision probability P_{AKEM} of AKEM is defined as

$$P_{\mathsf{AKEM}} := \max_{pk \in \mathcal{PK}} \Pr_{(sk', pk') \xleftarrow{\$} \mathsf{Gen}} [pk = pk'] \ .$$

Privacy. The following security notions for privacy aim to capture that KEM ciphertexts issued to an honest and uncompromised receiver are secure, i. e., do not leak any information about the KEM shared key. In $\mathsf{Outsider\text{-}CCA}$, the adversary is an outsider, meaning it has no knowledge about the secret keys of honest senders or honest receivers. In $\mathsf{Insider\text{-}CCA}$, the adversary is an insider, meaning it gets to choose the secret key of the sender. In both variants, the adversary gets access to encapsulation and decapsulation oracles that it can call multiple times, and in any order.

We define the games (n, q_e, q_d)-$\mathsf{Outsider\text{-}CCA}_\ell$ and (n, q_e, q_d)-$\mathsf{Outsider\text{-}CCA}_r$ in Listing 3 and the games (n, q_e, q_d, q_c)-$\mathsf{Insider\text{-}CCA}_\ell$ and (n, q_e, q_d, q_c)-$\mathsf{Insider\text{-}CCA}_r$ in Listing 4. The games follow the left-or-right style, as CryptoVerif requires this for assumptions, and we use these notions as assumptions in the composition theorems. In Appendix B, we compare the code-based game syntax with the CryptoVerif syntax for $\mathsf{Outsider\text{-}CCA}$.

In all games, the adversary has access to a key generation oracle Gen, which allows to create key pairs for up to n users. The adversary will get the public key and an index as identifier for use in other oracles. In the $\mathsf{Outsider\text{-}CCA}$ games, the adversary has additional access to oracles AEncap and ADecap. AEncap takes as input an index specifying the sender, as well as an arbitrary public key specifying the receiver, and returns a ciphertext and a KEM key. In the left game $\mathsf{Outsider\text{-}CCA}_\ell$, AEncap always returns the real KEM key. In the right game $\mathsf{Outsider\text{-}CCA}_r$, it outputs a uniformly random key if the receiver public key was generated by the experiment. Queries to ADecap, where the adversary specifies an index for a receiver public key, an arbitrary sender public key and a ciphertext, output a KEM key. In the $\mathsf{Outsider\text{-}CCA}_r$ game, we use a multiset $\mathcal{E}$ to store queries to AEncap ($\uplus$ denotes multiset union), which allows us to keep the output of ADecap consistent. We use a multiset rather than a set here since it may happen that $\mathcal{E}$ contains more than one entry (pk, pk', c, K), for the same or for different K, in case ciphertexts chosen by AEncap collide. The instruction **try get** K s.t. $(pk, pk_j, c, K) \in \mathcal{E}$ **then** computes the multiset $\{K \mid (pk, pk', c, K) \in \mathcal{E}\}$. If that multiset is not empty, it chooses one element of it uniformly at random and stores it in K. Then, it executes the code in the **then** branch. If that multiset is empty, the code in the **then** branch is not executed.

In the $\mathsf{Insider\text{-}CCA}$ games, there is an additional challenge oracle Chall. The adversary provides an index specifying the receiver and the secret key of the sender, thus taking the role of an insider. Chall will then output the real KEM key in the $\mathsf{Insider\text{-}CCA}_\ell$ game, and a uniformly random key in the $\mathsf{Insider\text{-}CCA}_r$ game. Thus, even if the target ciphertext was produced with a bad sender secret key (and honest receiver public key), the KEM key should be indistinguishable from a random key. AEncap will always output the real key and the output of ADecap is kept consistent with challenges using the same notation for the multiset $\mathcal{E}$ as described above.

In all games, the adversary makes at most n queries to Gen, at most q_e queries to oracle AEncap and at most q_d queries to oracle ADecap. In the $\mathsf{Insider\text{-}CCA}$ experiment, it can additionally make at most q_c queries to oracle Chall. We define the advantage of an adversary $\mathcal{A}$ as

$$\begin{aligned}
\mathsf{Adv}_{\mathcal{A},\mathsf{AKEM}}^{(n,q_e,q_d)\text{-}\mathsf{Outsider\text{-}CCA}} &:= \big|\Pr[(n,q_e,q_d)\text{-}\mathsf{Outsider\text{-}CCA}_\ell(\mathcal{A}) \Rightarrow 1] \\
&\quad - \Pr[(n,q_e,q_d)\text{-}\mathsf{Outsider\text{-}CCA}_r(\mathcal{A}) \Rightarrow 1]\big| \ , \\
\mathsf{Adv}_{\mathcal{A},\mathsf{AKEM}}^{(n,q_e,q_d,q_c)\text{-}\mathsf{Insider\text{-}CCA}} &:= \big|\Pr[(n,q_e,q_d,q_c)\text{-}\mathsf{Insider\text{-}CCA}_\ell(\mathcal{A}) \Rightarrow 1] \\
&\quad - \Pr[(n,q_e,q_d,q_c)\text{-}\mathsf{Insider\text{-}CCA}_r(\mathcal{A}) \Rightarrow 1]\big| \ .
\end{aligned}$$

Authenticity. The following $\mathsf{Outsider\text{-}Auth}$ security notion aims to capture that the adversary is unable to forge a KEM ciphertext to an honest receiver, pretending to come from an honest sender. We do not define an insider security notion because $\mathsf{Insider\text{-}Auth}$ security is infeasible both for the APKE construction used in HPKE (see Section 5.4), and the instantiation of AKEM proposed in HPKE (see end of Section 6.1).

We define the games (n, q_e, q_d)-$\mathsf{Outsider\text{-}Auth}_\ell$ and (n, q_e, q_d)-$\mathsf{Outsider\text{-}Auth}_r$ in Listing 5. The adversary has access to oracles Gen, AEncap and ADecap, where Gen is defined as in the CCA games. AEncap will always output the real KEM key. ADecap will output the real key in game $\mathsf{Outsider\text{-}Auth}_\ell$. In the $\mathsf{Outsider\text{-}Auth}_r$ game, the adversary (acting as an outsider) will receive a uniformly random key if the receiver public key was generated by the experiment. Thus, the adversary should not be able to distinguish the real KEM key

Listing 3: Games (n, q_e, q_d)-$\mathsf{Outsider\text{-}CCA}_\ell$ and (n, q_e, q_d)-$\mathsf{Outsider\text{-}CCA}_r$ for AKEM. Adversary $\mathcal{A}$ makes at most n queries to Gen, at most q_e queries to AEncap and at most q_d queries to ADecap.

```
$(n, q_e, q_d)$-$\mathsf{Outsider\text{-}CCA}_\ell$ and $\boxed{(n, q_e, q_d)\text{-}\mathsf{Outsider\text{-}CCA}_r}$
$\ell \leftarrow 0$
$\mathcal{E} \leftarrow \emptyset$
$b \xleftarrow{\$} \mathcal{A}^{\text{Gen},\text{AEncap},\text{ADecap}}$
return $b$

Oracle Gen
$\ell \leftarrow \ell + 1$
$(sk_\ell, pk_\ell) \xleftarrow{\$} \mathsf{Gen}$
return $(pk_\ell, \ell)$

Oracle AEncap$(i \in [\ell], pk)$
$(c, K) \xleftarrow{\$} \mathsf{AuthEncap}(sk_i, pk)$
if $pk \in \{pk_1, \ldots, pk_\ell\}$
    $K \xleftarrow{\$} \mathcal{K}$
    $\mathcal{E} \leftarrow \mathcal{E} \uplus \{(pk_i, pk, c, K)\}$
return $(c, K)$

Oracle ADecap$(j \in [\ell], pk, c)$
try get $K$ s.t. $(pk, pk_j, c, K) \in \mathcal{E}$
    then return $K$
$K \leftarrow \mathsf{AuthDecap}(sk_j, pk, c)$
return $K$
```

Listing 4: Games (n, q_e, q_d, q_c)-$\mathsf{Insider\text{-}CCA}_\ell$ and (n, q_e, q_d, q_c)-$\mathsf{Insider\text{-}CCA}_r$ for AKEM. Adversary $\mathcal{A}$ makes at most n queries to Gen, at most q_e queries to AEncap, at most q_d queries to ADecap and at most q_c queries to Chall.

```
$(n, q_e, q_d, q_c)$-$\mathsf{Insider\text{-}CCA}_\ell$ and $\boxed{(n, q_e, q_d, q_c)\text{-}\mathsf{Insider\text{-}CCA}_r}$
$\ell \leftarrow 0$
$\mathcal{E} \leftarrow \emptyset$
$b \xleftarrow{\$} \mathcal{A}^{\text{Gen},\text{AEncap},\text{ADecap},\text{Chall}}$
return $b$

Oracle Gen
$\ell \leftarrow \ell + 1$
$(sk_\ell, pk_\ell) \xleftarrow{\$} \mathsf{Gen}$
return $(pk_\ell, \ell)$

Oracle AEncap$(i \in [\ell], pk)$
$(c, K) \xleftarrow{\$} \mathsf{AuthEncap}(sk_i, pk)$
return $(c, K)$

Oracle ADecap$(j \in [\ell], pk, c)$
try get $K$ s.t. $(pk, pk_j, c, K) \in \mathcal{E}$
    then return $K$
$K \leftarrow \mathsf{AuthDecap}(sk_j, pk, c)$
return $K$

Oracle Chall$(j \in [\ell], sk)$
$(c, K) \xleftarrow{\$} \mathsf{AuthEncap}(sk, pk_j)$
$K \xleftarrow{\$} \mathcal{K}$
$\mathcal{E} \leftarrow \mathcal{E} \uplus \{(\mu(sk), pk_j, c, K)\}$
return $(c, K)$
```

Listing 5: Games (n, q_e, q_d)-$\mathsf{Outsider\text{-}Auth}_\ell$ and (n, q_e, q_d)-$\mathsf{Outsider\text{-}Auth}_r$ for AKEM. Adversary $\mathcal{A}$ makes at most n queries to Gen, at most q_e queries to AEncap and at most q_d queries to ADecap.

```
$(n, q_e, q_d)$-$\mathsf{Outsider\text{-}Auth}_\ell$ and $\boxed{(n, q_e, q_d)\text{-}\mathsf{Outsider\text{-}Auth}_r}$
$\ell \leftarrow 0$
$\mathcal{E} \leftarrow \emptyset$
$b \xleftarrow{\$} \mathcal{A}^{\text{Gen},\text{AEncap},\text{ADecap}}$
return $b$

Oracle Gen
$\ell \leftarrow \ell + 1$
$(sk_\ell, pk_\ell) \xleftarrow{\$} \mathsf{Gen}$
return $(pk_\ell, \ell)$

Oracle AEncap$(i \in [\ell], pk)$
$(c, K) \xleftarrow{\$} \mathsf{AuthEncap}(sk_i, pk)$
$\mathcal{E} \leftarrow \mathcal{E} \uplus \{(pk_i, pk, c, K)\}$
return $(c, K)$

Oracle ADecap$(j \in [\ell], pk, c)$
try get $K$ s.t. $(pk, pk_j, c, K) \in \mathcal{E}$
    then return $K$
$K \leftarrow \mathsf{AuthDecap}(sk_j, pk, c)$
if $pk \in \{pk_1, \ldots, pk_\ell\}$ and $K \neq \bot$
    $K \xleftarrow{\$} \mathcal{K}$
    $\mathcal{E} \leftarrow \mathcal{E} \uplus \{(pk, pk_j, c, K)\}$
return $K$
```

from a random key for two honest users, even if it can come up with the target ciphertext. Again we use the multiset $\mathcal{E}$ to ensure consistency.

The adversary makes at most n queries to Gen, at most q_e queries to oracle AEncap and at most q_d queries to oracle ADecap. We define the advantage of an adversary $\mathcal{A}$ as

$$\begin{aligned}\mathsf{Adv}^{(n,q_e,q_d)\text{-}\mathsf{Outsider\text{-}Auth}}_{\mathcal{A},\mathsf{AKEM}} := \big| &\Pr[(n,q_e,q_d)\text{-}\mathsf{Outsider\text{-}Auth}_\ell(\mathcal{A}) \Rightarrow 1] \\ &- \Pr[(n,q_e,q_d)\text{-}\mathsf{Outsider\text{-}Auth}_r(\mathcal{A}) \Rightarrow 1]\big| \ .\end{aligned}$$

In Appendix A, we provide simpler single-user or 2-user versions of these properties, and show that they non-tightly imply the definitions above. These results could be useful to simplify the proof for new AKEMs that could be added to HPKE, such as post-quantum AKEMs. However, because the reduction is not tight, a direct proof of multi-user security may yield better probability bounds. This is the case for our proof of DH-AKEM in Section 6.1.

5.2 Authenticated Public Key Encryption

Definition 10 (APKE). *An authenticated public key encryption scheme* APKE *consists of the following three algorithms:*

- Gen *outputs a key pair (sk, pk).*
- AuthEnc *takes as input a (sender) secret key sk, a (receiver) public key pk, a message m, associated data aad, a bitstring $info$, and outputs a ciphertext c.*
- *Deterministic* AuthDec *takes as input a (receiver) secret key sk, a (sender) public key pk, a ciphertext c, associated data aad and a bitstring $info$, and outputs a message m.*

We require that for all messages $m \in \{0,1\}^*$, $aad \in \{0,1\}^*$, $info \in \{0,1\}^*$,

$$\Pr_{\substack{(sk_S,pk_S)\xleftarrow{\$}\mathsf{Gen}\\(sk_R,pk_R)\xleftarrow{\$}\mathsf{Gen}}}\left[\begin{matrix}c \leftarrow \mathsf{AuthEnc}(sk_S, pk_R, m, aad, info),\\ \mathsf{AuthDec}(sk_R, pk_S, c, aad, info) = m\end{matrix}\right] = 1 \ .$$

Privacy. We define the games (n,q_e,q_d,q_c)-Outsider-CCA and (n,q_e,q_d,q_c)-Insider-CCA in Listing 6, which follow ideas similar to the games for outsider and insider-secure AKEM. The security notions for APKE use the common style where challenge queries are with respect to a random bit b. In particular, the additional challenge oracle Chall will encrypt either message m_0 or m_1 provided by the adversary, depending on b. Oracles AEnc and ADec will always encrypt and decrypt honestly (except for challenge ciphertexts). In these games, we use a set $\mathcal{E}$ to store queries to Chall and the adversary is not allowed to trivially decrypt a challenge using ADec.

In these games, the adversary $\mathcal{A}$ makes at most n queries to Gen, at most q_e queries to oracle AEnc, at most q_d queries to oracle ADec, and at most q_c queries to oracle Chall. The advantage of $\mathcal{A}$ is

$$\mathsf{Adv}^{(n,q_e,q_d,q_c)\text{-}\mathsf{Outsider\text{-}CCA}}_{\mathcal{A},\mathsf{APKE}} := \left|\Pr[(n,q_e,q_d,q_c)\text{-}\mathsf{Outsider\text{-}CCA}(\mathcal{A}) \Rightarrow 1] - \frac{1}{2}\right| \ ,$$

$$\mathsf{Adv}^{(n,q_e,q_d,q_c)\text{-}\mathsf{Insider\text{-}CCA}}_{\mathcal{A},\mathsf{APKE}} := \left|\Pr[(n,q_e,q_d,q_c)\text{-}\mathsf{Insider\text{-}CCA}(\mathcal{A}) \Rightarrow 1] - \frac{1}{2}\right| \ .$$

Listing 6: Games (n, q_e, q_d, q_c)-Outsider-CCA and (n, q_e, q_d, q_c)-Insider-CCA for APKE, where (n, q_e, q_d, q_c)-Outsider-CCA uses oracle CHALL in the dashed box and (n, q_e, q_d, q_c)-Insider-CCA uses oracle CHALL in the solid box. Adversary $\mathcal{A}$ makes at most n queries to GEN, at most q_e queries to AENC, at most q_d queries to ADEC and at most q_c queries to CHALL.

```
$(n, q_e, q_d, q_c)$-Outsider-CCA and $\boxed{(n, q_e, q_d, q_c)\text{-Insider-CCA}}$
01 $\ell \leftarrow 0$
02 $\mathcal{E} \leftarrow \emptyset$
03 $b \xleftarrow{\$} \{0,1\}$
04 $b' \xleftarrow{\$} \mathcal{A}^{\text{GEN,AENC,ADEC,CHALL}}$
05 return $[\![b = b']\!]$

Oracle GEN
06 $\ell \leftarrow \ell + 1$
07 $(sk_\ell, pk_\ell) \xleftarrow{\$} \mathsf{Gen}$
08 return $(pk_\ell, \ell)$

Oracle ADEC$(j \in [\ell], pk, c, aad, info)$
09 if $(pk, pk_j, c, aad, info) \in \mathcal{E}$
10     return $\perp$
11 $m \leftarrow \mathsf{AuthDec}(sk_j, pk, c, aad, info)$
12 return $m$

Oracle AENC$(i \in [\ell], pk, m, aad, info)$
13 $c \xleftarrow{\$} \mathsf{AuthEnc}(sk_i, pk, m, aad, info)$
14 return $c$

Oracle CHALL$(i \in [\ell], j \in [\ell], m_0, m_1, aad, info)$
15 if $|m_0| \neq |m_1|$ return $\perp$
16 $c \xleftarrow{\$} \mathsf{AuthEnc}(sk_i, pk_j, m_b, aad, info)$
17 $\mathcal{E} \leftarrow \mathcal{E} \cup \{(pk_i, pk_j, c, aad, info)\}$
18 return $c$

Oracle CHALL$(j \in [\ell], sk, m_0, m_1, aad, info)$
19 if $|m_0| \neq |m_1|$ return $\perp$
20 $c \xleftarrow{\$} \mathsf{AuthEnc}(sk, pk_j, m_b, aad, info)$
21 $\mathcal{E} \leftarrow \mathcal{E} \cup \{(\mu(sk), pk_j, c, aad, info)\}$
22 return $c$
```

Listing 7: Games (n, q_e, q_d)-Outsider-Auth and (n, q_e, q_d)-Insider-Auth for APKE. Adversary $\mathcal{A}$ makes at most n queries to GEN, at most q_e queries to AENC and at most q_d queries to ADEC.

```
$(n, q_e, q_d)$-Outsider-Auth
01 $\ell \leftarrow 0$
02 $\mathcal{E} \leftarrow \emptyset$
03 $(i^*, j^*, c^*, aad^*, info^*) \xleftarrow{\$} \mathcal{A}^{\text{GEN,AENC,ADEC}}$
04 return $[\![(pk_{i^*}, pk_{j^*}, c^*, aad^*, info^*) \notin \mathcal{E}$
       and $\mathsf{AuthDec}(sk_{j^*}, pk_{i^*}, c^*, aad^*, info^*) \neq \perp]\!]$

$(n, q_e, q_d)$-Insider-Auth
05 $\ell \leftarrow 0$
06 $\mathcal{E} \leftarrow \emptyset$
07 $(i^*, sk, c^*, aad^*, info^*) \xleftarrow{\$} \mathcal{A}^{\text{GEN,AENC,ADEC}}$
08 return $[\![(pk_{i^*}, \mu(sk), c^*, aad^*, info^*) \notin \mathcal{E}$
       and $\mathsf{AuthDec}(sk, pk_{i^*}, c^*, aad^*, info^*) \neq \perp]\!]$

Oracle GEN
09 $\ell \leftarrow \ell + 1$
10 $(sk_\ell, pk_\ell) \xleftarrow{\$} \mathsf{Gen}$
11 return $(pk_\ell, \ell)$

Oracle AENC$(i \in [\ell], pk, m, aad, info)$
12 $c \xleftarrow{\$} \mathsf{AuthEnc}(sk_i, pk, m, aad, info)$
13 $\mathcal{E} \leftarrow \mathcal{E} \cup \{(pk_i, pk, c, aad, info)\}$
14 return $c$

Oracle ADEC$(j \in [\ell], pk, c, aad, info)$
15 $m \leftarrow \mathsf{AuthDec}(sk_j, pk, c, aad, info)$
16 return $m$
```

Listing 8: Authenticated PKE scheme APKE[AKEM, KS, AEAD] construction from AKEM, KS and AEAD, where APKE.Gen = AKEM.Gen.

```
$\mathsf{AuthEnc}(sk, pk, m, aad, info)$
01 $(c_1, K) \xleftarrow{\$} \mathsf{AuthEncap}(sk, pk)$
02 $(k, nonce) \leftarrow \mathsf{KS}(K, info)$
03 $c_2 \leftarrow \mathsf{AEAD.Enc}(k, m, aad, nonce)$
04 return $(c_1, c_2)$

$\mathsf{AuthDec}(sk, pk, (c_1, c_2), aad, info)$
05 $K \leftarrow \mathsf{AuthDecap}(sk, pk, c_1)$
06 $(k, nonce) \leftarrow \mathsf{KS}(K, info)$
07 $m \leftarrow \mathsf{AEAD.Dec}(k, c_2, aad, nonce)$
08 return $m$
```

Authenticity. Furthermore, we define the games (n, q_e, q_d)-Outsider-Auth and (n, q_e, q_d)-Insider-Auth in Listing 7. The adversary has access to an encryption and decryption oracle and has to come up with a new tuple of ciphertext, associated data and info for any honest receiver secret key (Outsider-Auth) or any (possibly leaked or bad) receiver secret key (Insider-Auth), provided that the sender public key is honest.

In these games, adversary $\mathcal{A}$ makes at most n queries to Gen, at most q_e queries to oracle AEnc and at most q_d queries to oracle ADec. The advantage of $\mathcal{A}$ is defined as

$$\mathsf{Adv}^{(n,q_e,q_d)\text{-Outsider-Auth}}_{\mathcal{A},\mathsf{APKE}} := \Pr[(n, q_e, q_d)\text{-Outsider-Auth}(\mathcal{A}) \Rightarrow 1] \ ,$$
$$\mathsf{Adv}^{(n,q_e,q_d)\text{-Insider-Auth}}_{\mathcal{A},\mathsf{APKE}} := \Pr[(n, q_e, q_d)\text{-Insider-Auth}(\mathcal{A}) \Rightarrow 1] \ .$$

5.3 From AKEM to APKE

In this section we define and analyse a general transformation that models HPKE's way of constructing APKE from an AKEM (c.f. Definition 9) and an AEAD (c.f. Definition 3 on Page 8). It also uses a so-called *key schedule* KS which we model as a keyed function $\mathsf{KS} : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^*$, where $\mathcal{K}$ matches the AKEM's key space. KS outputs an AEAD key k and an initialisation vector *nonce* (called *base nonce* in the RFC) from which the AEAD's nonces are computed. (The key schedule defined in the HPKE standard also outputs an additional key called *exporter secret* that can be used to derive keys for use by arbitrary higher-level applications. This export API is not part of the single-shot encryption API that we are analysing, and thus we omit it in our definitions.) Listing 8 gives the formal specification of APKE built from AKEM, KS and AEAD.

We observe that in the single-shot encryption API, every AEAD key k is used to produce exactly one ciphertext, and thus is only used with one nonce. In HPKE, messages are counted with a sequence number s starting at 0 and the nonce for a message is computed by $nonce \oplus s$. For the single-shot encryption API this means that the nonce is equal to the initialisation vector *nonce*. At the same time, this means that *nonce* is by definition unique.

We now give theorems stating the (n, q_e, q_d, q_c)-Outsider-CCA, (n, q_e, q_d)-Outsider-Auth and (n, q_e, q_d, q_c)-Insider-CCA security of APKE[AKEM, KS, AEAD] defined in Listing 8. Theorems 3 to 5 are proven using CryptoVerif version 2.05. This version includes an improvement in the computation of probability bounds that allows us to express these bounds as functions of the total numbers of queries to the AEnc, ADec, and Chall oracles instead of the number of users and the numbers of queries per user. The CryptoVerif input files are given in hpke.auth.outsider-cca.ocv, hpke.auth.insider-cca.ocv, and hpke.auth.outsider-auth.ocv [2]. These proofs are fairly straightforward. As an example, we prefer explaining the proof of Theorem 7 later, which is more interesting. In Section 5.4, we show that APKE[AKEM, KS, AEAD] cannot achieve Insider-Auth security.

As detailed in Section 3, we define a multi-key PRF security experiment (n_k, q_{PRF})-PRF with n_k keys, in which the adversary makes at most q_{PRF} queries for each key. We also define multi-key IND-CPA and INT-CTXT security experiments for the AEAD: n_k-IND-CPA and (n_k, q_d)-INT-CTXT, with n_k keys, in which the adversary makes at most one encryption query for each key and, for the INT-CTXT experiment, at most q_d decryption queries in total. In these experiments, the nonces of the AEAD are chosen randomly.

Theorem 3 (AKEM Outsider-CCA + KS PRF + AEAD IND-CPA + AEAD INT-CTXT $\Rightarrow$ APKE Outsider-CCA)**.** *For any* (n, q_e, q_d, q_c)-Outsider-CCA *adversary* $\mathcal{A}$ *against* APKE[

$\mathsf{AKEM}, \mathsf{KS}, \mathsf{AEAD}]$, *there exist an* $(n, q_e + q_c, q_d)$-$\mathsf{Outsider\text{-}CCA}$ *adversary* $\mathcal{B}$ *against* AKEM, *an* $(q_c, q_c + q_d)$-PRF *adversary* $\mathcal{C}$ *against* KS, *an* q_c-$\mathsf{IND\text{-}CPA}$ *adversary* $\mathcal{D}_1$ *against* AEAD *and an* (q_c, q_d)-$\mathsf{INT\text{-}CTXT}$ *adversary* $\mathcal{D}_2$ *against* AEAD *such that* $t_\mathcal{B} \approx t_\mathcal{A}$, $t_\mathcal{C} \approx t_\mathcal{A}$, $t_{\mathcal{D}_1} \approx t_\mathcal{A}$, $t_{\mathcal{D}_2} \approx t_\mathcal{A}$, *and*

$$\begin{aligned}\mathsf{Adv}^{(n,q_e,q_d,q_c)\text{-}\mathsf{Outsider\text{-}CCA}}_{\mathcal{A},\mathsf{APKE}[\mathsf{AKEM},\mathsf{KS},\mathsf{AEAD}]} \leq{}& 2 \cdot \mathsf{Adv}^{(n,q_e+q_c,q_d)\text{-}\mathsf{Outsider\text{-}CCA}}_{\mathcal{B},\mathsf{AKEM}} + 2 \cdot \mathsf{Adv}^{(q_c,q_c+q_d)\text{-}\mathsf{PRF}}_{\mathcal{C},\mathsf{KS}} \\ &+ 2 \cdot \mathsf{Adv}^{q_c\text{-}\mathsf{IND\text{-}CPA}}_{\mathcal{D}_1,\mathsf{AEAD}} + 2 \cdot \mathsf{Adv}^{(q_c,q_d)\text{-}\mathsf{INT\text{-}CTXT}}_{\mathcal{D}_2,\mathsf{AEAD}} \\ &+ 6n^2 \cdot P_{\mathsf{AKEM}} \ .\end{aligned}$$

Theorem 4 (AKEM $\mathsf{Insider\text{-}CCA}$ + KS PRF + AEAD $\mathsf{IND\text{-}CPA}$ + AEAD $\mathsf{INT\text{-}CTXT}$ $\Rightarrow$ APKE $\mathsf{Insider\text{-}CCA}$)**.** *For any* (n, q_e, q_d, q_c)-$\mathsf{Insider\text{-}CCA}$ *adversary* $\mathcal{A}$ *against* $\mathsf{APKE}[\mathsf{AKEM}, \mathsf{KS}, \mathsf{AEAD}]$, *there exist an* (n, q_e, q_d, q_c)-$\mathsf{Insider\text{-}CCA}$ *adversary* $\mathcal{B}$ *against* AKEM, *an* $(q_c, q_c + q_d)$-PRF *adversary* $\mathcal{C}$ *against* KS, *an* q_c-$\mathsf{IND\text{-}CPA}$ *adversary* $\mathcal{D}_1$ *against* AEAD *and an* (q_c, q_d)-$\mathsf{INT\text{-}CTXT}$ *adversary* $\mathcal{D}_2$ *against* AEAD *such that* $t_\mathcal{B} \approx t_\mathcal{A}$, $t_\mathcal{C} \approx t_\mathcal{A}$, $t_{\mathcal{D}_1} \approx t_\mathcal{A}$, $t_{\mathcal{D}_2} \approx t_\mathcal{A}$, *and*

$$\begin{aligned}\mathsf{Adv}^{(n,q_e,q_d,q_c)\text{-}\mathsf{Insider\text{-}CCA}}_{\mathcal{A},\mathsf{APKE}[\mathsf{AKEM},\mathsf{KS},\mathsf{AEAD}]} \leq{}& 2 \cdot \mathsf{Adv}^{(n,q_e,q_d,q_c)\text{-}\mathsf{Insider\text{-}CCA}}_{\mathcal{B},\mathsf{AKEM}} + 2 \cdot \mathsf{Adv}^{(q_c,q_c+q_d)\text{-}\mathsf{PRF}}_{\mathcal{C},\mathsf{KS}} \\ &+ 2 \cdot \mathsf{Adv}^{q_c\text{-}\mathsf{IND\text{-}CPA}}_{\mathcal{D}_1,\mathsf{AEAD}} + 2 \cdot \mathsf{Adv}^{(q_c,q_d)\text{-}\mathsf{INT\text{-}CTXT}}_{\mathcal{D}_2,\mathsf{AEAD}} \\ &+ 6n^2 \cdot P_{\mathsf{AKEM}} \ .\end{aligned}$$

Theorem 5 (AKEM $\mathsf{Outsider\text{-}CCA}$+$\mathsf{AKEM}$ $\mathsf{Outsider\text{-}Auth}$+$\mathsf{KS}$ PRF+AEAD $\mathsf{INT\text{-}CTXT}$$\Rightarrow$ APKE $\mathsf{Outsider\text{-}Auth}$)**.** *For any* (n, q_e, q_d)-$\mathsf{Outsider\text{-}Auth}$ *adversary* $\mathcal{A}$ *against* $\mathsf{APKE}[\mathsf{AKEM}, \mathsf{KS}, \mathsf{AEAD}]$, *there exist an* $(n, q_e, q_d + 1)$-$\mathsf{Outsider\text{-}CCA}$ *adversary* $\mathcal{B}_1$ *against* AKEM, *an* $(n, q_e, q_d + 1)$-$\mathsf{Outsider\text{-}Auth}$ *adversary* $\mathcal{B}_2$ *against* AKEM, *an* $(q_e + q_d + 1, q_e + 2q_d + 1)$-$\mathsf{PRF}$ *adversary* $\mathcal{C}$ *against* KS, *and an* $(q_e + q_d + 1, 2q_d + 1)$-$\mathsf{INT\text{-}CTXT}$ *adversary* $\mathcal{D}$ *against* AEAD *such that* $t_{\mathcal{B}_1} \approx t_\mathcal{A}$, $t_{\mathcal{B}_2} \approx t_\mathcal{A}$, $t_\mathcal{C} \approx t_\mathcal{A}$, $t_\mathcal{D} \approx t_\mathcal{A}$, *and*

$$\begin{aligned}\mathsf{Adv}^{(n,q_e,q_d)\text{-}\mathsf{Outsider\text{-}Auth}}_{\mathcal{A},\mathsf{APKE}[\mathsf{AKEM},\mathsf{KS},\mathsf{AEAD}]} \leq{}& \mathsf{Adv}^{(n,q_e,q_d+1)\text{-}\mathsf{Outsider\text{-}CCA}}_{\mathcal{B}_1,\mathsf{AKEM}} + \mathsf{Adv}^{(n,q_e,q_d+1)\text{-}\mathsf{Outsider\text{-}Auth}}_{\mathcal{B}_2,\mathsf{AKEM}} \\ &+ \mathsf{Adv}^{(q_e+q_d+1,q_e+2q_d+1)\text{-}\mathsf{PRF}}_{\mathcal{C},\mathsf{KS}} \\ &+ \mathsf{Adv}^{(q_e+q_d+1,2q_d+1)\text{-}\mathsf{INT\text{-}CTXT}}_{\mathcal{D},\mathsf{AEAD}} + n(q_e + 11n) \cdot P_{\mathsf{AKEM}} \ .\end{aligned}$$

5.4 Infeasibility of $\mathsf{Insider\text{-}Auth}$ security

For any AKEM, KS, and AEAD, the construction $\mathsf{APKE}[\mathsf{AKEM}, \mathsf{KS}, \mathsf{AEAD}]$ given in Listing 8 is not (n, q_e, q_d)-$\mathsf{Insider\text{-}Auth}$ secure. The inherent reason for this construction to be vulnerable against this attack is that the KEM ciphertext does not depend on the message. Thus, the KEM ciphertext can be reused and the DEM ciphertext can be exchanged by the encryption of any other message.

Theorem 6. *There exists an efficient adversary* $\mathcal{A}$ *against* $(1, 1, 0)$-$\mathsf{Insider\text{-}Auth}$ *security of* $\mathsf{APKE}[\mathsf{AKEM}, \mathsf{KS}, \mathsf{AEAD}]$ *such that*

$$\mathsf{Adv}^{(1,1,0)\text{-}\mathsf{Insider\text{-}Auth}}_{\mathcal{A},\mathsf{APKE}[\mathsf{AKEM},\mathsf{KS},\mathsf{AEAD}]} = 1 \ ,$$

which means that $\mathcal{A}$ *makes a single query to* Gen, *a single query to* AEnc *and no query to* ADec.

Proof. We construct adversary $\mathcal{A}$ in Listing 9. It has oracle access to Gen, AEnc and ADec. It first creates a key pair using the Gen oracle to receive a public key pk_1. It then generates a challenge key pair (sk^*, pk^*) and queries the AEnc oracle on any index 1, receiver public key pk^*, an arbitrary message m_1, as well as arbitrary associated data aad and string $info$.

Listing 9: Adversary $\mathcal{A}$ against $(1, 1, 0)$-Insider-Auth security (as defined in Listing 7) of APKE[AKEM, KS, AEAD].

```
Adversary $\mathcal{A}^{\textsc{Gen},\textsc{AEnc},\textsc{ADec}}$
$(pk_1, \ell = 1) \leftarrow \textsc{Gen}$
$(sk^*, pk^*) \leftarrow \mathsf{AKEM.Gen}$
$m_1 := aad := info := 1$
$(c_1, c_2) \leftarrow \textsc{AEnc}(1, pk^*, m_1, aad, info)$
$K \leftarrow \mathsf{AuthDecap}(sk^*, pk_1, c_1)$
$(k, nonce) \leftarrow \mathsf{KS}(K, info)$
$m_2 := 2$
$c_2' \leftarrow \mathsf{AEAD.Enc}(k, m_2, aad, nonce)$
return $(1, sk^*, (c_1, c_2'), aad, info)$
```

The challenger computes $(c_1, K) \xleftarrow{\$} \mathsf{AuthEncap}(sk_1, pk^*)$, $(k, nonce) \leftarrow \mathsf{KS}(K, info)$ and $c_2 \leftarrow \mathsf{AEAD.Enc}(k, m_1, aad, nonce)$, and returns (c_1, c_2) to $\mathcal{A}$.

Since $\mathcal{A}$ knows the secret key sk^*, it is able to compute the underlying KEM key K using AuthDecap. Next, it computes $(k, nonce)$ and thus retrieves the key k used in the AEAD scheme. Finally, $\mathcal{A}$ encrypts any other message m_2 to ciphertext c_2' and replaces the AEAD ciphertext c_2 with the new ciphertext. Since $(c_1, c_2) \neq (c_1, c_2')$, the latter constitutes a valid forgery in the $(1, 1, 0)$-Insider-Auth security experiment. ☐

6 The HPKE Standard

In Section 6.1, we show how to construct HPKE's abstract AKEM construction DH-AKEM from a nominal group $\mathcal{N}$ and a key derivation function KDF. In Section 6.2, we define and analyse HPKE's specific key schedule $\mathsf{KS}_{\mathsf{Auth}}$ and key derivation function HKDF_N. Finally, in Section 6.3 we put everything together and obtain the HPKE standard in Auth mode from all previous sections.

6.1 HPKE's AKEM Construction DH-AKEM

In this section we present the RFC's instantiation of the AKEM definition, and prove that it satisfies the security notions defined earlier. Listing 10 shows the formal definition of DH-AKEM[$\mathcal{N}$, KDF] relative to a nominal group $\mathcal{N}$ (c.f. Definition 4) and a key derivation function $\mathsf{KDF} : \{0, 1\}^* \rightarrow \mathcal{K}$, where $\mathcal{K}$ is the key space. (The RFC uses a key space $\mathcal{K}$, consisting of bitstrings of length N, which corresponds to `Nsecret` in the RFC.) The construction also depends on the fixed-size protocol constants `"HPKE-v1"` and $suite_{id}$, where $suite_{id}$ identifies the KEM in use: it is a string `"KEM"` plus a two-byte identifier of

Listing 10: DH-AKEM$[\mathcal{N}, \mathsf{KDF}] = (\mathsf{Gen}, \mathsf{AuthEncap}, \mathsf{AuthDecap})$ as defined in the RFC [6], constructed from a nominal group $\mathcal{N}$ and key derivation function $\mathsf{KDF} : \{0,1\}^* \to \mathcal{K}$, with $\mathcal{K} = \{0,1\}^N$.

```
Gen
  $sk \xleftarrow{\$} \mathcal{E}_H$
  $pk \leftarrow g^{sk}$
  return $(sk, pk)$

ExtractAndExpand$(dh, context)$
  $IKM \leftarrow$ "HPKE-v1" || $suite_{id}$ ||
          "eae_prk" || $dh$
  $info \leftarrow$ Encode$(N)$ || "HPKE-v1" ||
          $suite_{id}$ || "shared_secret" ||
          $context$
  return KDF("", $IKM, info$)

AuthEncap$(sk \in \mathcal{E}_H, pk \in \mathcal{G})$
  $(esk, epk) \xleftarrow{\$}$ Gen
  $context \leftarrow (epk, pk, g^{sk})$
  $dh \leftarrow (pk^{esk}, pk^{sk})$
  $K \leftarrow$ ExtractAndExpand$(dh, context)$
  return $(epk, K)$

AuthDecap$(sk \in \mathcal{E}_H, pk \in \mathcal{G}, epk \in \mathcal{G})$
  $context \leftarrow (epk, g^{sk}, pk)$
  $dh \leftarrow (epk^{sk}, pk^{sk})$
  return ExtractAndExpand$(dh, context)$
```

the KEM algorithm. The bitstring Encode(N) is the two-byte encoding of the length N expressed in bytes. Correctness follows by property 1 of Definition 4. We make the implicit convention that AuthEncap and AuthDecap return reject ($\perp$) if their inputs are not of the right data type as specified in Listing 10.

We continue with statements about the (n, q_e, q_d)-Outsider-CCA, (n, q_e, q_d, q_c)-Insider-CCA, and (n, q_e, q_d)-Outsider-Auth security of DH-AKEM$[\mathcal{N}, \mathsf{KDF}]$, modelling KDF as a random oracle. The proofs are written with CryptoVerif version 2.05; the input files are dhkem.auth.outsider-cca-lr.ocv, dhkem.auth.insider-cca-lr.ocv, and dhkem.auth.outsider-auth-lr.ocv [2]. We sketch the proof of one of the three theorems as an example, to help understand CryptoVerif's approach.

Our results hold for any rerandomisable nominal group, which covers the three NIST curves allowed by the RFC, as well as for the other two allowed curves, Curve25519 and Curve448. The bounds given in Theorems 7 to 9 depend on the probabilities $\Delta_{\mathcal{N}}$ and $P_{\mathcal{N}}$, which can be instantiated for these five different curves using the values indicated in Table 2 on Page 30.

The RFC mandates that implementations abort if the Diffie-Hellman shared secret is the point at infinity for P-256, P-384, and P-521, or the all-zero value for Curve25519 and Curve448. The security notions in this work do not contain this check, and the theorems are valid for both implementations that do or do not contain it (this is because the properties we prove are not concerned about contributiveness). More formally, the adversary could check for all public keys leading to problematic Diffie-Hellman shared secrets before calling the oracles in our security games.

At the end of this section, we sketch the attack against the Insider-Auth security.

Theorem 7 (Outsider-CCA **security of** DH-AKEM)**.** *Let $\mathcal{N}$ be a rerandomisable nominal group. Under the* GDH *assumption in $\mathcal{N}$ and modelling* KDF *as a random oracle,* DH-AKEM$[\mathcal{N}, \mathsf{KDF}]$ *is* Outsider-CCA *secure. In particular, for any adversary $\mathcal{A}$ against* (n, q_e, q_d)-Outsider-CCA *security of* DH-AKEM$[\mathcal{N}, \mathsf{KDF}]$ *that issues at most q_h queries to* *the random oracle* KDF*, there exists an adversary* $\mathcal{B}$ *against* GDH *such that*

$$\begin{aligned}\mathsf{Adv}^{(n,q_e,q_d)\text{-}\mathsf{Outsider\text{-}CCA}}_{\mathcal{A},\mathsf{DH\text{-}AKEM}[\mathcal{N},\mathsf{KDF}]} &\leq \mathsf{Adv}^{\mathsf{GDH}}_{\mathcal{B},\mathcal{N}} + (n+q_e)\cdot \varDelta_{\mathcal{N}}\\ &+ (q_e q_d + 2nq_e + 10q_e^2 + 11n^2)\cdot P_{\mathcal{N}}\end{aligned}$$

$\mathcal{B}$ *issues* $5q_h$ *queries to the* DH *oracle, and* $t_{\mathcal{B}} \approx t_{\mathcal{A}}$*.*

Proof. This proof is mechanized using the tool CryptoVerif. We give to the tool the assumptions that $\mathcal{N}$ is a nominal group that satisfies the GDH assumption, formalized by Definition 7 and Theorem 1, and that KDF is a random oracle. We also give the definition of $\mathsf{DH\text{-}AKEM}$, and ask it to show that the games (n, q_e, q_d)-$\mathsf{Outsider\text{-}CCA}_\ell$ and (n, q_e, q_d)-$\mathsf{Outsider\text{-}CCA}_r$ are computationally indistinguishable. In the particular case of $\mathsf{DH\text{-}AKEM}$, these two games include an additional oracle: the random oracle KDF. The theorem, the initial game definitions, and the proof indications are available in the file dhkem.auth.outsider-cca-lr.ocv [2].

The proof proceeds by transforming the game (n, q_e, q_d)-$\mathsf{Outsider\text{-}CCA}_\ell$ by several steps into a game G_{final} and the game (n, q_e, q_d)-$\mathsf{Outsider\text{-}CCA}_r$ into the same game G_{final}. Since all transformation steps performed by CryptoVerif are designed to preserve computational indistinguishability, we obtain that (n, q_e, q_d)-$\mathsf{Outsider\text{-}CCA}_\ell$ and (n, q_e, q_d)-$\mathsf{Outsider\text{-}CCA}_r$ are computationally indistinguishable. We guide the transformations with the following main steps.

Starting from (n, q_e, q_d)-$\mathsf{Outsider\text{-}CCA}_\ell$, in the oracle AEncap, we first distinguish whether the provided public key pk is honest, by testing whether $pk = pk_i$ for some i (a test that appears in (n, q_e, q_d)-$\mathsf{Outsider\text{-}CCA}_r$). We rename some variables to give them different names when $pk \in \{pk_1, \dots, pk_n\}$ and when $pk \notin \{pk_1, \dots, pk_n\}$, to facilitate future game transformations. In the oracle ADecap, we test whether $\exists K : (pk, pk_j, c, K) \in \mathcal{E}$, which corresponds to a test done in (n, q_e, q_d)-$\mathsf{Outsider\text{-}CCA}_r$. Furthermore, when this test succeeds, we replace the result normally returned by ADecap, $\mathsf{AuthDecap}(sk_j, pk, c)$ with the key K found in $\mathcal{E}$. CryptoVerif shows that this replacement does not modify the result, which corresponds to the correctness of $\mathsf{DH\text{-}AKEM}$. In the random oracle, we distinguish whether the argument received from the adversary has a format that matches the one used by $\mathsf{DH\text{-}AKEM}$ or not. Only when the format matches, this argument may coincide with a call to the hash oracle made from $\mathsf{DH\text{-}AKEM}$. Next, we apply the random oracle assumption. Each call to the random oracle is replaced with the following test: if the argument is equal to the argument of a previous call, we return the previous result; otherwise, we return a fresh random value. Finally, we apply the GDH assumption, which allows us to show that some comparisons between Diffie-Hellman values are false. In particular, CryptoVerif shows that the arguments of calls to the random oracle coming from AEncap with $pk \in \{pk_1, \dots, pk_n\}$ cannot coincide with arguments of other calls. Hence, they return a fresh random key, as in (n, q_e, q_d)-$\mathsf{Outsider\text{-}CCA}_r$.

Starting from (n, q_e, q_d)-$\mathsf{Outsider\text{-}CCA}_r$, in the random oracle, we distinguish whether the argument received from the adversary has a format that matches the one used by $\mathsf{DH\text{-}AKEM}$ or not. Next, we apply the random oracle assumption, as we did on the left-hand side.

The transformed games obtained respectively from (n, q_e, q_d)-$\mathsf{Outsider\text{-}CCA}_\ell$ and from (n, q_e, q_d)-$\mathsf{Outsider\text{-}CCA}_r$ are then equal, which concludes the proof.

CryptoVerif computes the bound on the probability of distinguishing the games (n, q_e, q_d)-$\mathsf{Outsider\text{-}CCA}_\ell$ and (n, q_e, q_d)-$\mathsf{Outsider\text{-}CCA}_r$ by adding bounds computed at each transformation step. During this proof, CryptoVerif automatically eliminates unlikely collisions, in particular between public Diffie-Hellman keys. By default, CryptoVerif eliminates these collisions aggressively, even when that is not required for the proof to succeed, which results in a large probability bound. To avoid that, we allow the tool to eliminate collisions of probability $P_{\mathcal{N}}$ times a power 4 in n, $q_e^{per\ user}$, and $q_d^{per\ user}$, where $q_e^{per\ user}$ and $q_d^{per\ user}$ are the number of AEncap and ADecap queries respectively, per user. But we do not allow eliminating collisions with more than a power 4 in n, $q_e^{per\ user}$, and $q_d^{per\ user}$, nor collisions that involve q_h. $\square$

Theorem 8 **(Insider-CCA security of DH-AKEM).** *Let $\mathcal{N}$ be a rerandomisable nominal group. Under the GDH assumption in $\mathcal{N}$ and modelling KDF as a random oracle, DH-AKEM$[\mathcal{N}, \mathsf{KDF}]$ is Insider-CCA secure. In particular, for any (n, q_e, q_d, q_c)-Insider-CCA adversary $\mathcal{A}$ against DH-AKEM$_{\mathcal{N}}$ that issues at most q_h queries to the random oracle, there exists an adversary $\mathcal{B}$ against GDH such that*

$$\begin{aligned}\mathsf{Adv}^{(n,q_e,q_d,q_c)\text{-Insider-CCA}}_{\mathcal{A},\mathsf{DH\text{-}AKEM}[\mathcal{N},\mathsf{KDF}]} \le\ & \mathsf{Adv}^{\mathsf{GDH}}_{\mathcal{B},\mathcal{N}} + (n + q_c) \cdot \Delta_{\mathcal{N}} \\ & + (q_c q_d + 4 q_c q_e + 2 n q_e + 10 q_e^2 + 3 q_c^2 + 9 n^2) \cdot P_{\mathcal{N}}\end{aligned}$$

$\mathcal{B}$ makes $5q_h$ queries to the DH *oracle, and $t_{\mathcal{B}} \approx t_{\mathcal{A}}$.*

Theorem 9 **(Outsider-Auth security of DH-AKEM).** *Let $\mathcal{N}$ be a rerandomisable nominal group. Under the sqGDH assumption in $\mathcal{N}$ and modelling KDF as a random oracle, DH-AKEM$[\mathcal{N}, \mathsf{KDF}]$ is Outsider-Auth secure. In particular, for any (n, q_e, q_d)-Outsider-Auth adversary $\mathcal{A}$ against DH-AKEM$_{\mathcal{N}}$ that issues at most q_h queries to the random oracle, there exists an adversary $\mathcal{B}$ against sqGDH such that*

$$\begin{aligned}\mathsf{Adv}^{(n,q_e,q_d)\text{-Outsider-Auth}}_{\mathcal{A},\mathsf{DH\text{-}AKEM}[\mathcal{N},\mathsf{KDF}]} \le\ & 2\mathsf{Adv}^{\mathsf{sqGDH}}_{\mathcal{B},\mathcal{N}} + 2(n + q_e) \cdot \Delta_{\mathcal{N}} \\ & + (2 q_e q_d + 4 n q_d + 16 q_e^2 + 4 n q_e + 12 n^2) \cdot P_{\mathcal{N}}\end{aligned}$$

$\mathcal{B}$ issues $7q_h$ queries to the DH *oracle, and $t_{\mathcal{B}} \approx t_{\mathcal{A}}$.*

Infeasibility of Insider-Auth security. As for APKE, we could define an Insider-Auth security notion for AKEM, which precludes forgeries even when the receiver key pair is dishonest, provided the sender key pair is honest. However, the DH-AKEM construction does not even achieve KCI security, a relaxation of Insider-Auth security only precluding forgeries for leaked, but still honestly generated, receiver key pairs. Indeed, in DH-AKEM, knowledge of an arbitrary receiver secret key is already sufficient to compute the Diffie-Hellman shared key for any sender public key. Thus, in a KCI attack, an adversary that learns a target receiver's keys can trivially produce a KEM ciphertext and corresponding encapsulated key for any target sender public key.

6.2 HPKE's Key Schedule and Key Derivation Function

HPKE's key schedule $\mathsf{KS}_{\mathsf{Auth}}$ and key derivation function HKDF_N are both instantiated via the functions Extract and Expand which are defined below. We proceed to prove a theorem that $\mathsf{KS}_{\mathsf{Auth}}$ is a PRF, as needed for the composition results presented in Theorems 3 to 5. Then, we argue why HKDF_N can be modelled as a random oracle, as assumed by

Theorems 7 to 9 on DH-AKEM. Finally, we indicate how the entire $\mathsf{HPKE}_{\mathsf{Auth}}$ scheme is assembled from the individual building blocks presented in the previous sections.

Extract and Expand. The RFC defines two functions Extract and Expand as follows.

- $\mathsf{Extract}(\mathit{salt}, \mathit{IKM})$ is a function keyed by a bitstring *salt*, with input keying material *IKM* as parameter, and returns a bitstring of fixed length N_h bits.
- $\mathsf{Expand}(\mathit{PRK}, \mathit{info}, L)$ is a function keyed by *PRK*, with an arbitrary bitstring *info* and a length L as parameters, and returns a bitstring of length L.

In Theorem 10, we assume that Extract and Expand are PRFs with the first parameter being the PRF key. HPKE instantiates Extract and Expand with HMAC-SHA-2, for which the PRF assumption is justified by [7,8]. (Generally, HPKE's instantiation of Expand uses HMAC iteratively to achieve the variable output length L. However, all values L used in HPKE are less or equal than the output length of one HMAC call.) We also assume that Extract is collision resistant, provided its keys are not larger than blocks of SHA-2, which is needed to avoid that the keys be hashed before computing HMAC, and true in HPKE. This property is immediate from the collision resistance of SHA-2, studied in [22].

Key Schedule. The key schedule $\mathsf{KS}_{\mathsf{Auth}}$ serves as a bridging step between the AKEM and the AEAD of APKE. The computations done by $\mathsf{KS}_{\mathsf{Auth}}$ are as indicated in Listing 11. The function KeySchedule used internally is the common key schedule function that the RFC defines for all modes. In $\mathsf{HPKE}_{\mathsf{Auth}}$, the *mode* parameter is set to the constant one-byte value `0x02` identifying the mode Auth. Similarly, mode Auth does not use a pre-shared key, so the *psk* parameter is always set to the empty string `""`, and the value *psk_id* that is identifying which pre-shared key is used, is equally set to `""`. The RFC defines LabeledExtract and LabeledExpand as wrappers around Extract and Expand, for domain separation and context binding. The value suite_{id} is a 10-byte string identifying the ciphersuite, composed as a concatenation of the string `"HPKE"`, and two-byte identifiers of the KEM, the KDF, and the AEAD algorithm in use. The bitstring $\mathsf{Encode}(L)$ is the two-byte encoding of the length L expressed in bytes. The values N_k and N_n indicate the length of the AEAD key and nonce.

The composition results established by Theorems 3 to 5 assume that $\mathsf{KS}_{\mathsf{Auth}}$ is a PRF. The following theorem proves this property for $\mathsf{HPKE}_{\mathsf{Auth}}$'s instantiation of $\mathsf{KS}_{\mathsf{Auth}}$.

Theorem 10 (Extract CR + Extract PRF + Expand PRF $\Rightarrow$ $\mathsf{KS}_{\mathsf{Auth}}$ PRF)**.** *Assuming that* Extract *is a collision-resistant hash function for calls with the labels* `"psk_id_hash"` *and* `"info_hash"`*, that* Extract *is a* PRF *for calls with the label* `"secret"`*, and that* Expand *is a* PRF*, it follows that* $\mathsf{KS}_{\mathsf{Auth}}$ *is a* PRF*.*

In particular, for any (n_k, q_{PRF})-PRF *adversary* $\mathcal{A}$ *against* $\mathsf{KS}_{\mathsf{Auth}}$*, there exist an adversary* $\mathcal{B}$ *against the collision resistance of* Extract*, a* (n_k, n_k)-PRF *adversary* $\mathcal{C}_1$ *against* Extract*, and a* $(n_k, 2q_{\mathsf{PRF}})$-PRF *adversary* $\mathcal{C}_2$ *against* Expand *such that* $t_{\mathcal{B}} \approx t_{\mathcal{A}}$*,* $t_{C_1} \approx t_{\mathcal{A}}$*,* $t_{C_2} \approx t_{\mathcal{A}}$*, and*

$$\mathsf{Adv}^{(n_k, q_{\mathsf{PRF}})\text{-}\mathsf{PRF}}_{\mathcal{A},\mathsf{KS}_{\mathsf{Auth}}} \leq \mathsf{Adv}^{\mathsf{CR}}_{\mathcal{B},\mathsf{Extract}} + \mathsf{Adv}^{(n_k, n_k)\text{-}\mathsf{PRF}}_{\mathcal{C}_1,\mathsf{Extract}} + \mathsf{Adv}^{(n_k, 2q_{\mathsf{PRF}})\text{-}\mathsf{PRF}}_{\mathcal{C}_2,\mathsf{Expand}} \; .$$

This theorem is proven by CryptoVerif in keyschedule.auth.prf.ocv [2].

The Key Derivation Function KDF in DH-AKEM. The AKEM instantiation DH-AKEM as we defined it in Listing 10 uses a function KDF to derive the KEM shared secret. In

Listing 11: The key schedule $\mathsf{KS}_{\mathsf{Auth}}$ used in $\mathsf{HPKE}_{\mathsf{Auth}}$ [6].

```
KS_Auth(k_PRF, info)
    return KeySchedule(k_PRF, 0x02, info, "", "")

KeySchedule(k_PRF, mode, info, psk, psk_id)
    context ← mode ||
                  LabeledExtract("", "psk_id_hash", psk_id) ||
                  LabeledExtract("", "info_hash"   , info)
    secret ← LabeledExtract(k_PRF, "secret", psk)
    k ← LabeledExpand(secret, "key", context, N_k)
    nonce ← LabeledExpand(secret, "base_nonce", context, N_n)
    return (k, nonce)

LabeledExtract(salt, label, IKM')
    return Extract(salt, "HPKE-v1" || suite_id || label || IKM')

LabeledExpand(PRK, label, context, L)
    return Expand(PRK, Encode(L) || "HPKE-v1" || suite_id || label || context, L)
```

$\mathsf{HPKE}_{\mathsf{Auth}}$, this function is instantiated by HKDF_N, as defined in Listing 12, using the above-defined Extract and Expand internally. The output length N corresponds to `Nsecret` in the RFC.

In the analysis of the key schedule presented above, we assume that Extract and Expand are pseudo-random functions. However, this assumption would not be sufficient to prove the security of DH-AKEM: the random oracle model is required. The simplest choice is to assume that the whole key derivation function $\mathsf{KDF} = \mathsf{HKDF}_N$ is a random oracle, as we do in Theorems 7 to 9. (Alternatively, we could probably rely on some variant of the PRF-ODH assumption [15]. While in principle the PRF-ODH assumption is weaker than the random oracle model, Brendel, Fischlin, Günther, and Janson [15] show that it is implausible to instantiate the PRF-ODH assumption without a random oracle, so that would not make a major difference.) The invocations of Extract and Expand in DH-AKEM and $\mathsf{KS}_{\mathsf{Auth}}$ use different labels for domain separation, so choosing different assumptions is sound. Next, we further justify the random oracle assumption for HKDF_N.

As mentioned at the beginning of Section 6, HPKE instantiates Extract and Expand with HMAC [24], which makes HKDF_N exactly the widely-used HKDF key derivation function [25]. HPKE specifies SHA-2 as the hash function underlying HMAC. Lemma 6 in [28] shows that HKDF is indifferentiable from a random oracle under the following assumptions[6]: (1) HMAC is indifferentiable from a random oracle. For HMAC-SHA-2, this is justified by Theorem 4.4 in [20] assuming the compression function underlying SHA-2 is a random oracle. The theorem's restriction on HMAC's key size is fulfilled, because DH-AKEM uses either the empty string, or a bitstring of hash output length as key. (2) Values of IKM do not collide with values of $info \,||\, \texttt{0x01}$. This is guaranteed by the prefix `"HPKE-v1"` of IKM, which is used as a prefix for $info$ as well, but shifted by two characters, because the two-byte encoding of the length N comes before it. The shared secret lengths `Nsecret` specified in the RFC correspond exactly to the output length of

[6] The exact probability bound is indicated in Lemma 8 of that paper's full version.

Listing 12: Function $\mathsf{HKDF}_N[\mathsf{Extract}, \mathsf{Expand}]$ as used in $\mathsf{HPKE}_{\mathsf{Auth}}$.

```
$\mathsf{HKDF}_N(salt, IKM, info)$
$PRK \leftarrow \mathsf{Extract}(salt, IKM)$
return $\mathsf{Expand}(PRK, info, N)$
```

the hash function; this means there is only one internal call to Expand, and thus we do not need to consider collisions of *IKM* with the input to later HMAC calls.

6.3 HPKE's APKE Scheme $\mathsf{HPKE}_{\mathsf{Auth}}$

Let $\mathsf{HPKE}_{\mathsf{Auth}} := \mathsf{APKE}[\mathsf{DH\text{-}AKEM}[\mathcal{N}, \mathsf{HKDF}_N], \mathsf{KS}_{\mathsf{Auth}}, \mathsf{AEAD}]$ be the APKE construction obtained by applying the black-box AKEM/DEM composition of Listing 8 to the $\mathsf{DH\text{-}AKEM}[\mathcal{N}, \mathsf{HKDF}_N]$ authenticated KEM (Listing 10), where $\mathcal{N}$ is a rerandomisable nominal group. For the key schedule of $\mathsf{HPKE}_{\mathsf{Auth}}$ we use $\mathsf{KS}_{\mathsf{Auth}}$ of Listing 11 and for the key derivation function we use HKDF_N of Listing 12. For both $\mathsf{KS}_{\mathsf{Auth}}$ and HKDF_N we implement the Extract and Expand functions using HMAC (as described in the HPKE specification). Finally, we instantiate HMAC using one of the SHA2 family of hash functions. (Which one depends on the target bit security of $\mathsf{HPKE}_{\mathsf{Auth}}$, as we discuss below.)

The AKEM/DEM composition Theorems 3 to 5, together with Theorem 10 on the key schedule $\mathsf{KS}_{\mathsf{Auth}}$, and Theorems 7 to 9 on DH-AKEM's security, and $P_{\mathsf{DH\text{-}AKEM}} = P_{\mathcal{N}}$ provide the following concrete security bounds for $\mathsf{HPKE}_{\mathsf{Auth}}$. For simplicity, we ignore all constants and set $q := q_e + q_d + q_c$.

$$\begin{aligned}
\mathsf{Adv}^{(n,q_e,q_d,q_c)\text{-}\mathsf{Outsider\text{-}CCA}}_{\mathcal{A},\mathsf{HPKE}_{\mathsf{Auth}}} &\le \mathsf{Adv}^{\mathsf{GDH}}_{\mathcal{B}_1,\mathcal{N}} + (n+q)^2 \cdot P_{\mathcal{N}} + (n+q) \cdot \Delta_{\mathcal{N}} \\
&\quad + \mathsf{Adv}^{(q,q)\text{-}\mathsf{PRF}}_{\mathcal{C},\mathsf{KS}_{\mathsf{Auth}}} + \mathsf{Adv}^{q\text{-}\mathsf{IND\text{-}CPA}}_{\mathcal{D}_1,\mathsf{AEAD}} + \mathsf{Adv}^{(q,q)\text{-}\mathsf{INT\text{-}CTXT}}_{\mathcal{D}_2,\mathsf{AEAD}} \\
\mathsf{Adv}^{(n,q_e,q_d)\text{-}\mathsf{Outsider\text{-}Auth}}_{\mathcal{A},\mathsf{HPKE}_{\mathsf{Auth}}} &\le \mathsf{Adv}^{\mathsf{GDH}}_{\mathcal{B}_1,\mathcal{N}} + \mathsf{Adv}^{\mathsf{sqGDH}}_{\mathcal{B}_2,\mathcal{N}} + (n+q)^2 \cdot P_{\mathcal{N}} + (n+q) \cdot \Delta_{\mathcal{N}} \\
&\quad + \mathsf{Adv}^{(q,q)\text{-}\mathsf{PRF}}_{\mathcal{C},\mathsf{KS}_{\mathsf{Auth}}} + \mathsf{Adv}^{(q,q)\text{-}\mathsf{INT\text{-}CTXT}}_{\mathcal{D}_1,\mathsf{AEAD}} .
\end{aligned}$$

The bound for Insider-CCA is the same as the one for Outsider-CCA. In all bounds, we have $\mathsf{Adv}^{(q,q)\text{-}\mathsf{PRF}}_{\mathcal{C},\mathsf{KS}_{\mathsf{Auth}}} \le \mathsf{Adv}^{\mathsf{CR}}_{\mathcal{C}_1,\mathsf{Extract}} + \mathsf{Adv}^{(q,q)\text{-}\mathsf{PRF}}_{\mathcal{C}_2,\mathsf{Extract}} + \mathsf{Adv}^{(q,q)\text{-}\mathsf{PRF}}_{\mathcal{C}_3,\mathsf{Expand}}$. Moreover, the adversaries $\mathcal{B}_1, \mathcal{B}_2, \mathcal{C}, \mathcal{D}_1, \mathcal{D}_2$ have (roughly) the same running time as $\mathcal{A}$.

Parameter Choices of $\mathsf{HPKE}_{\mathsf{Auth}}$. To obtain a concrete instance of $\mathsf{HPKE}_{\mathsf{Auth}}$, the HPKE standard allows different choices of rerandomisable nominal groups $\mathcal{N}$ that lead to different bounds on the statistical parameters $P_{\mathcal{N}}$ and $\Delta_{\mathcal{N}}$. The standard also fixes the length N of the KEM keyspace, c.f. Table 2. Even though lengths are expressed in bytes in the RFC and the implementation, we express them in bits in this section as this is more convenient to discuss the number of bits of security.

All concrete instances of $\mathsf{HPKE}_{\mathsf{Auth}}$ proposed by the HPKE standard build Extract and Expand from HMAC which, in turn, uses a hash function. HPKE proposes several concrete hash functions (all in the SHA2 family). For our security bounds, the relevant consequence of choosing a particular hash function is the resulting key length N_h of Expand when used as a PRF, c.f. Table 3.

Table 2: Parameters of $\mathsf{DH\text{-}AKEM}[\mathcal{N}, \mathsf{HKDF}_N]$ depending on the choice of the rerandomisable nominal group $\mathcal{N}$.

	P-256	**P-384**	**P-521**	**Curve25519**	**Curve448**
Security level $\kappa_{\mathcal{N}}$ (bits)	128	192	256	128	224
$P_{\mathcal{N}} \leq$	2^{-255}	2^{-383}	2^{-520}	2^{-250}	2^{-444}
$\Delta_{\mathcal{N}} \leq$	0	0	0	2^{-126}	2^{-221}
KEM keyspace N (bits)	256	384	512	256	512

Table 3: Choices of HMAC and the PRF key lengths of Expand, instantiated with HMAC.

	HMAC-SHA256	**HMAC-SHA384**	**HMAC-SHA512**
PRF key length N_h of Expand (bits)	256	384	512

Table 4: Choices of the AEAD scheme and their parameters.

	AES-128-GCM	**AES-256-GCM**	**ChaCha20-Poly1305**
AEAD key length N_k (bits)	128	256	256
AEAD nonces length N_n (bits)	96	96	96
AEAD tag length N_t (bits)	128	128	128

Finally, to instantiate $\mathsf{HPKE}_{\mathsf{Auth}}$, we must also specify the AEAD scheme. HPKE allows for several choices which affect the AEAD key length N_k, nonces length N_n, and tag length N_t, c.f. Table 4.

Discussion. We say that an instance of $\mathsf{HPKE}_{\mathsf{Auth}}$ achieves κ *bits of security* if the success ratio $\mathsf{Adv}_{\mathcal{A},\mathsf{HPKE}_{\mathsf{Auth}}}/t_{\mathcal{A}}$ is upper bounded by $2^{-\kappa}$ for any adversary $\mathcal{A}$ with runtime $t_{\mathcal{A}} \leq 2^{\kappa}$. In particular, we say that a term ε *has* κ *bits of security* if $\varepsilon/t_{\mathcal{A}} \leq 2^{-\kappa}$. We discuss the implications of our results for the bit security of the various instances of $\mathsf{HPKE}_{\mathsf{Auth}}$ proposed by the standard.

The runtime $t_{\mathcal{A}}$ of any adversary $\mathcal{A}$ in an APKE security game is lower-bounded by $n+q$, since the adversary needs n steps to parse the n public keys and additional q steps to make the oracle queries. We assume that $t_{\mathcal{A}} \leq 2^{\kappa}$, where κ is the target security level.

We now estimate the security level supported by each term in $\mathsf{Adv}_{\mathcal{A},\mathsf{HPKE}_{\mathsf{Auth}}}$.

- **Term** $\mathsf{Adv}^{\mathsf{GDH}}_{\mathcal{B}_1,\mathcal{N}}$**.** Rerandomisable nominal groups $\mathcal{N}$ proposed for use by the HPKE standard were designed to provide $\kappa_{\mathcal{N}}$ bits of security (c.f. Table 2). That is, we assume that $\mathsf{Adv}^{\mathsf{GDH}}_{\mathcal{B}_1,\mathcal{N}}/t_{\mathcal{B}_1} \leq 2^{-\kappa_{\mathcal{N}}}$. Since $t_{\mathcal{A}} \approx t_{\mathcal{B}_1}$, we conclude that this term has $\kappa_{\mathcal{N}}$ bits of security. The same arguments hold for $\mathsf{Adv}^{\mathsf{sqGDH}}_{\mathcal{B}_2,\mathcal{N}}$.
- **Term** $(n+q)^2 \cdot P_{\mathcal{N}}$**.** Let us show that this term also has $\kappa_{\mathcal{N}}$ bits of security. We have $n+q \leq t_{\mathcal{A}}$. Thus, it suffices to show that $(n+q) \cdot P_{\mathcal{N}} \leq 2^{-\kappa_{\mathcal{N}}}$. Since $t_{\mathcal{A}} \leq 2^{\kappa_{\mathcal{N}}}$, we get that $(n+q) \leq 2^{\kappa_{\mathcal{N}}}$. The statement now follows as, according to Table 2, $P_{\mathcal{N}} \lesssim 2^{-2\kappa_{\mathcal{N}}}$.

– **Term** $(n+q) \cdot \Delta_{\mathcal{N}}$**.** Let us show that this term also has $\kappa_{\mathcal{N}}$ bits of security. For all NIST curves, we have $\Delta_{\mathcal{N}} = 0$ trivially implying the statement. In contrast, for Curve25519 and Curve448, $\Delta_{\mathcal{N}} \lesssim 2^{-\kappa_{\mathcal{N}}}$, so $(n+q) \cdot \Delta_{\mathcal{N}} \approx (n+q) \cdot 2^{-\kappa_{\mathcal{N}}}$. As $n+q \le t_{\mathcal{A}}$, the statement also holds for these curves.
– **Term** $\mathsf{Adv}^{\mathsf{CR}}_{\mathcal{C}_1,\mathsf{Extract}}$**.** The output length N_h of the concrete hash functions are listed in Table 3. Since the generic bound on collision resistance is $t^2_{\mathcal{C}_1}/2^{N_h}$, this term has $N_h/2$ bits of security.
– **Term** $\mathsf{Adv}^{(q,q)\text{-}\mathsf{PRF}}_{\mathcal{C}_3,\mathsf{Expand}}$**.** The PRF key lengths N_h of Expand are specified in Table 3. Modelling the PRF as a random oracle, we have $\mathsf{Adv}^{(q,q)\text{-}\mathsf{PRF}}_{\mathcal{C}_3,\mathsf{Expand}} \le q^2/2^{N_h}$. So this term also has $N_h/2$ bits of security.
– **Term** $\mathsf{Adv}^{(q,q)\text{-}\mathsf{PRF}}_{\mathcal{C}_2,\mathsf{Extract}}$**.** The PRF key length N of $\mathsf{Extract}$ is specified in Table 2. By the same argument as for the previous term, this term has $N/2$ bits of security. Since $N/2 \ge \kappa_{\mathcal{N}}$ by Table 2, this term has $\kappa_{\mathcal{N}}$ bits of security.
– **Terms** $\mathsf{Adv}^{q\text{-}\mathsf{IND\text{-}CPA}}_{\mathcal{D}_1,\mathsf{AEAD}} + \mathsf{Adv}^{(q,q)\text{-}\mathsf{INT\text{-}CTXT}}_{\mathcal{D}_2,\mathsf{AEAD}}$**.** The terms refer to the multi-key security of the AEAD schemes (c.f. Section 3), studied for instance in [11]. However, the current results are not sufficient to guarantee the expected security level, such as 128 bits for AES-128-GCM. We recommend further research to study the exact bounds of the terms instantiated with the AEAD schemes from Table 4. In any case a simple key/nonce-collision attack has success probability $\mathsf{Adv}^{q\text{-}\mathsf{IND\text{-}CPA}}_{\mathcal{D}_1,\mathsf{AEAD}} = q^2/2^{N_k+N_n}$, where N_k is the AEAD key length and N_n is the nonce length. A simple computation shows that this term has at most N_k bits of security (assuming $q \le 2^{N_n}$). Moreover, a simple attack against $\mathsf{INT\text{-}CTXT}$ by guessing the authentication tag has success probability $\mathsf{Adv}^{(q,q)\text{-}\mathsf{INT\text{-}CTXT}}_{\mathcal{D}_2,\mathsf{AEAD}} = q/2^{N_t}$, where N_t is the length of the authentication tag. Hence, this term has at most N_t bits of security. Assuming these attacks also serve as an upper bound, these terms would have $\min(N_k, N_t)$ bits of security if $q \le 2^{N_n}$. Since for all AEAD schemes of Table 4, we have $N_t = 128$ bits, that limits the security level of HPKE to 128 bits.

To sum up, the analysis above suggests that HPKE has about $\kappa = \min(\kappa_{\mathcal{N}}, N_h/2, N_k, N_t)$ bits of security, under the assumption that $t_{\mathcal{A}} \le 2^{\kappa}$ and $q \le 2^{N_n}$. Since the tag length of the AEAD is $N_t = 128$ bits, we obtain $\kappa = 128$ bits; a greater security level could be obtained by using AEADs with longer tags. More research on the multi-key security of AEAD schemes is still needed to confirm this analysis.

Acknowledgements. The authors would like to thank the HPKE RFC co-authors Richard Barnes, Karthikeyan Bhargavan, and Christopher Wood for fruitful discussions during the preparation of this paper. They also thank Pierre Boutry and Christian Doczkal for very helpful feedback on a previous version of this paper.

Bruno Blanchet was supported by ANR TECAP (decision number ANR-17-CE39-0004-03). Eduard Hauck was supported by the DFG SPP 1736 Big Data. Eike Kiltz was supported by the BMBF iBlockchain project, the EU H2020 PROMETHEUS project 780701, the DFG SPP 1736 Big Data, and the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972. Benjamin Lipp was supported by ERC CIRCUS (grant agreement nº 683032) and ANR TECAP (decision number ANR-17-CE39-0004-03). Doreen Riepel was funded by the DFG under Germany's Excellence Strategy - EXC 2092 CASA - 390781972.

References

1. Abdalla, M., Bellare, M., Rogaway, P.: The oracle Diffie-Hellman assumptions and an analysis of DHIES. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 143–158. Springer, Heidelberg (Apr 2001). https://doi.org/10.1007/3-540-45353-9_12 6
2. Alwen, J., Blanchet, B., Hauck, E., Kiltz, E., Lipp, B., Riepel, D.: Analysing the HPKE standard – supplementary material. https://doi.org/10.5281/zenodo.4297811 21, 24, 25, 27, 45, 48
3. Alwen, J., Blanchet, B., Hauck, E., Kiltz, E., Lipp, B., Riepel, D.: Analysing the HPKE standard. In: Canteaut, A., Standaert, F.X. (eds.) Eurocrypt 2021. LNCS, vol. 12696, pp. 87–116. Springer, Heidelberg, Zagreb, Croatia (Oct 2021). https://doi.org/10.1007/978-3-030-77870-5_4 1
4. Barnes, R.L., Alwen, J., Corretti, S.: Homomorphic multiplication for X25519 and X448. Internet draft, IETF (Nov 2019), https://tools.ietf.org/html/draft-barnes-cfrg-mult-for-7748-00 15
5. Barnes, R.L., Beurdouche, B., Millican, J., Omara, E., Cohn-Gordon, K., Robert, R.: The Messaging Layer Security (MLS) Protocol. Internet-Draft draft-ietf-mls-protocol-09, IETF Secretariat (Mar 2020), https://tools.ietf.org/html/draft-ietf-mls-protocol-09 1, 2
6. Barnes, R.L., Bhargavan, K., Lipp, B., Wood, C.A.: Hybrid Public Key Encryption. Internet-Draft draft-irtf-cfrg-hpke-08, IETF Secretariat (Oct 2020), https://tools.ietf.org/html/draft-irtf-cfrg-hpke-08 1, 24, 28
7. Bellare, M.: New proofs for NMAC and HMAC: Security without collision resistance. Journal of Cryptology **28**(4), 844–878 (Oct 2015). https://doi.org/10.1007/s00145-014-9185-x 27
8. Bellare, M., Canetti, R., Krawczyk, H.: Keying hash functions for message authentication. In: Koblitz, N. (ed.) CRYPTO'96. LNCS, vol. 1109, pp. 1–15. Springer, Heidelberg (Aug 1996). https://doi.org/10.1007/3-540-68697-5_1 27
9. Bellare, M., Rogaway, P.: Code-based game-playing proofs and the security of triple encryption. Cryptology ePrint Archive, Report 2004/331 (2004), https://eprint.iacr.org/2004/331 7
10. Bellare, M., Stepanovs, I.: Security under message-derived keys: Signcryption in iMessage. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part III. LNCS, vol. 12107, pp. 507–537. Springer, Heidelberg (May 2020). https://doi.org/10.1007/978-3-030-45727-3_17 6
11. Bellare, M., Tackmann, B.: The multi-user security of authenticated encryption: AES-GCM in TLS 1.3. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part I. LNCS, vol. 9814, pp. 247–276. Springer, Heidelberg (Aug 2016). https://doi.org/10.1007/978-3-662-53018-4_10 31
12. Bernstein, D.J.: Curve25519: New Diffie-Hellman speed records. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. LNCS, vol. 3958, pp. 207–228. Springer, Heidelberg (Apr 2006). https://doi.org/10.1007/11745853_14 10
13. Bhargavan, K., Blanchet, B., Kobeissi, N.: Verified models and reference implementations for the TLS 1.3 standard candidate. In: 2017 IEEE Symposium on Security and Privacy. pp. 483–502. IEEE Computer Society Press (May 2017). https://doi.org/10.1109/SP.2017.26 5
14. Blanchet, B.: A computationally sound mechanized prover for security protocols. IEEE Transactions on Dependable and Secure Computing **5**(4), 193–207 (Oct–Dec 2008) 5
15. Brendel, J., Fischlin, M., Günther, F., Janson, C.: PRF-ODH: Relations, instantiations, and impossibility results. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part III. LNCS, vol. 10403, pp. 651–681. Springer, Heidelberg (Aug 2017). https://doi.org/10.1007/978-3-319-63697-9_22 28
16. Cramer, R., Shoup, V.: Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. SIAM Journal on Computing **33**(1), 167–226 (2003) 1

17. Dent, A.W.: Hybrid signcryption schemes with insider security. In: Boyd, C., Nieto, J.M.G. (eds.) ACISP 05. LNCS, vol. 3574, pp. 253–266. Springer, Heidelberg (Jul 2005) 6
18. Dent, A.W.: Hybrid signcryption schemes with outsider security. In: Zhou, J., Lopez, J., Deng, R.H., Bao, F. (eds.) ISC 2005. LNCS, vol. 3650, pp. 203–217. Springer, Heidelberg (Sep 2005) 6
19. Dent, A.W., Zheng, Y. (eds.): Practical Signcryption. Information Security and Cryptography, Springer (2010). https://doi.org/10.1007/978-3-540-89411-7 6
20. Dodis, Y., Ristenpart, T., Steinberger, J., Tessaro, S.: To hash or not to hash again? (In)differentiability results for H^2 and HMAC. Cryptology ePrint Archive, Report 2013/382 (2013), https://eprint.iacr.org/2013/382 28
21. Gayoso Martínez, V., Alvarez, F., Hernandez Encinas, L., Sánchez Ávila, C.: A comparison of the standardized versions of ECIES. 2010 6th International Conference on Information Assurance and Security, IAS 2010 (Aug 2010). https://doi.org/10.1109/ISIAS.2010.5604194 6
22. Gilbert, H., Handschuh, H.: Security analysis of SHA-256 and sisters. In: Matsui, M., Zuccherato, R.J. (eds.) SAC 2003. LNCS, vol. 3006, pp. 175–193. Springer, Heidelberg (Aug 2004). https://doi.org/10.1007/978-3-540-24654-1_13 27
23. Kobeissi, N., Bhargavan, K., Blanchet, B.: Automated verification for secure messaging protocols and their implementations: A symbolic and computational approach. In: 2nd IEEE European Symposium on Security and Privacy. pp. 435–450. IEEE (Apr 2017) 5
24. Krawczyk, H., Bellare, M., Canetti, R.: HMAC: Keyed-hashing for message authentication. RFC 2104, RFC Editor (Feb 1997), https://www.rfc-editor.org/rfc/rfc2104.html 28
25. Krawczyk, H., Eronen, P.: HMAC-based extract-and-expand key derivation function (HKDF). RFC 5869, RFC Editor (May 2010), https://www.rfc-editor.org/rfc/rfc5869.html 28
26. Langley, A., Hamburg, M., Turner, S.: Elliptic curves for security. RFC 7748, RFC Editor (Jan 2016), https://www.rfc-editor.org/rfc/rfc7748.html 2, 7, 8, 10, 11
27. Lipp, B.: An analysis of hybrid public key encryption. Cryptology ePrint Archive, Report 2020/243 (2020), https://eprint.iacr.org/2020/243 7
28. Lipp, B., Blanchet, B., Bhargavan, K.: A mechanised cryptographic proof of the WireGuard virtual private network protocol. In: 4th IEEE European Symposium on Security and Privacy. pp. 231–246. IEEE Computer Society, Stockholm, Sweden (Jun 2019), full version: https://hal.inria.fr/hal-02100345 5, 7, 28
29. National Institute of Standards and Technology: Digital Signature Standard (DSS). FIPS Publication 186-4 (Jul 2013), https://doi.org/10.6028/nist.fips.186-4 2, 7, 8, 10
30. Omara, E., Beurdouche, B., Rescorla, E., Inguva, S., Kwon, A., Duric, A.: The Messaging Layer Security (MLS) Architecture. Internet-Draft draft-ietf-mls-architecture-05, IETF Secretariat (Jul 2020), https://tools.ietf.org/html/draft-ietf-mls-architecture-05 2
31. Rescorla, E., Oku, K., Sullivan, N., Wood, C.A.: TLS Encrypted Client Hello. Internet-Draft draft-ietf-tls-esni-07, IETF Secretariat (Jun 2020), https://tools.ietf.org/html/draft-ietf-tls-esni-07 1, 2
32. Zheng, Y.: Digital signcryption or how to achieve cost(signature & encryption) $\ll$ cost(signature) + cost(encryption). In: Kaliski Jr., B.S. (ed.) CRYPTO'97. LNCS, vol. 1294, pp. 165–179. Springer, Heidelberg (Aug 1997). https://doi.org/10.1007/BFb0052234 2, 6

A Single- and Two-User Definitions for AKEM

In this section, we give simplified variants of AKEM security notions which only consider a single user (insider security) or two users (outsider security). We show that these notions non-tightly imply their corresponding n-user notion given in Section 5.1.

As in Section 5.1, we distinguish between Outsider-CCA, Insider-CCA and Outsider-Auth. In all notions, we have a dedicated challenge oracle CHALL, whereas AENCAP and

ADecap always reveal the real key (provided that ADecap is not queried on a challenge ciphertext).

We give games (q_e, q_d, q_c)-2-Outsider-CCA$_\ell$ and (q_e, q_d, q_c)-2-Outsider-CCA$_r$ in Listing 13 and games (q_e, q_d, q_c)-2-Outsider-Auth$_\ell$ and (q_e, q_d, q_c)-2-Outsider-Auth$_r$ in Listing 15. We also provide the (q_e, q_d, q_c)-1-Insider-CCA games in Listing 14, where compared to the (n, q_e, q_d, q_c)-Insider-CCA games we omit the key generation oracle Gen since we need to generate only one challenge key and w.l.o.g. we can do this at the beginning of the game. In all games, adversary $\mathcal{A}$ is allowed to make q_e queries to AEncap, q_d queries to ADecap and q_c queries to Chall. We define the advantage of $\mathcal{A}$ in each game as

$$\begin{aligned}
\mathsf{Adv}^{(q_e,q_d,q_c)\text{-2-Outsider-CCA}}_{\mathcal{A},\mathsf{AKEM}} &:= \Big|\Pr[(q_e, q_d, q_c)\text{-2-Outsider-CCA}_\ell(\mathcal{A}) \Rightarrow 1] \\
&\quad - \Pr[(q_e, q_d, q_c)\text{-2-Outsider-CCA}_r(\mathcal{A}) \Rightarrow 1]\Big| , \\
\mathsf{Adv}^{(q_e,q_d,q_c)\text{-1-Insider-CCA}}_{\mathcal{A},\mathsf{AKEM}} &:= \Big|\Pr[(q_e, q_d, q_c)\text{-1-Insider-CCA}_\ell(\mathcal{A}) \Rightarrow 1] \\
&\quad - \Pr[(q_e, q_d, q_c)\text{-1-Insider-CCA}_r(\mathcal{A}) \Rightarrow 1]\Big| , \\
\mathsf{Adv}^{(q_e,q_d,q_c)\text{-2-Outsider-Auth}}_{\mathcal{A},\mathsf{AKEM}} &:= \Big|\Pr[(q_e, q_d, q_c)\text{-2-Outsider-Auth}_\ell(\mathcal{A}) \Rightarrow 1] \\
&\quad - \Pr[(q_e, q_d, q_c)\text{-2-Outsider-Auth}_r(\mathcal{A}) \Rightarrow 1]\Big| .
\end{aligned}$$

Listing 13: Games (q_e, q_d, q_c)-2-Outsider-CCA$_\ell$ and (q_e, q_d, q_c)-2-Outsider-CCA$_r$ for AKEM. Adversary $\mathcal{A}$ makes at most q_e queries to AEncap, at most q_d queries to ADecap and at most q_c queries to Chall.

```
$(q_e, q_d, q_c)$-2-Outsider-CCA$_\ell$ and
[ $(q_e, q_d, q_c)$-2-Outsider-CCA$_r$ ]
01 $(sk_1, pk_1) \xleftarrow{\$}$ Gen
02 $(sk_2, pk_2) \xleftarrow{\$}$ Gen
03 $\mathcal{C} \leftarrow \emptyset$
04 $b \xleftarrow{\$} \mathcal{A}^{\text{AEncap,ADecap,Chall}}(pk_1, pk_2)$
05 return $b$

Oracle AEncap$(i \in [2], pk)$
06 $(c, K) \xleftarrow{\$}$ AuthEncap$(sk_i, pk)$
07 return $(c, K)$

Oracle Chall$(i \in [2], j \in [2])$
08 $(c, K) \xleftarrow{\$}$ AuthEncap$(sk_i, pk_j)$
[09 $K \xleftarrow{\$} \mathcal{K}$ ]
10 $\mathcal{C} \leftarrow \mathcal{C} \cup \{(pk_i, pk_j, c, K)\}$
11 return $(c, K)$

Oracle ADecap$(j \in [2], pk, c)$
12 if $\exists K : (pk, pk_j, c, K) \in \mathcal{C}$
13     return $\perp$
14 $K \leftarrow$ AuthDecap$(sk_j, pk, c)$
15 return $K$
```

For each security notion, we prove that the corresponding single- or two-user notion with one challenge query implies the n-user notion of Section 5.1 with multiple challenge queries. This is captured in Theorems 11 to 13. The proofs are given in Appendices A.1 to A.3.

Theorem 11 **($(q_e, q_d, 1)$-2-Outsider-CCA $\implies$ (n, q_e, q_d)-Outsider-CCA).** *For any adversary $\mathcal{A}$ against the (n, q_e, q_d)-Outsider-CCA security experiment, there exists an adversary $\mathcal{B}$ against the $(q_e, q_d, 1)$-2-Outsider-CCA security experiment such that $t_\mathcal{B} \approx t_\mathcal{A}$ and*

$$\mathsf{Adv}^{(n,q_e,q_d)\text{-Outsider-CCA}}_{\mathcal{A},\mathsf{AKEM}} \le n^2 q_e \cdot \mathsf{Adv}^{(q_e,q_d,1)\text{-2-Outsider-CCA}}_{\mathcal{B},\mathsf{AKEM}} + n^2 \cdot P_{\mathsf{AKEM}} .$$

Listing 14: Games (q_e, q_d, q_c)-1-Insider-CCA$_\ell$ and (q_e, q_d, q_c)-1-Insider-CCA$_r$ for AKEM. Adversary $\mathcal{A}$ makes at most q_e queries to AEncap, at most q_d queries to ADecap and at most q_c queries to Chall.

```
$(q_e, q_d, q_c)$-1-Insider-CCA$_\ell$ and $\boxed{(q_e, q_d, q_c)\text{-1-Insider-CCA}_r}$
$(sk^*, pk^*) \xleftarrow{\$} \mathsf{Gen}$
$\mathcal{C} \leftarrow \emptyset$
$b \xleftarrow{\$} \mathcal{A}^{\text{AEncap},\text{ADecap},\text{Chall}}(pk^*)$
return $b$

Oracle AEncap$(pk)$
$(c, K) \xleftarrow{\$} \mathsf{AuthEncap}(sk^*, pk)$
return $(c, K)$

Oracle Chall$(sk)$
$(c, K) \xleftarrow{\$} \mathsf{AuthEncap}(sk, pk^*)$
$\boxed{K \xleftarrow{\$} \mathcal{K}}$
$\mathcal{C} \leftarrow \mathcal{C} \cup \{(\mu(sk), c, K)\}$
return $(c, K)$

Oracle ADecap$(pk, c)$
if $\exists K : (pk, c, K) \in \mathcal{C}$
    return $\perp$
$K \leftarrow \mathsf{AuthDecap}(sk^*, pk, c)$
return $K$
```

Listing 15: Games (q_e, q_d, q_c)-2-Outsider-Auth$_\ell$ and (q_e, q_d, q_c)-2-Outsider-Auth$_r$ for AKEM. Adversary $\mathcal{A}$ makes at most q_e queries AEncap, at most q_d queries ADecap and at most q_c queries Chall.

```
$(q_e, q_d, q_c)$-2-Outsider-Auth$_\ell$ and $\boxed{(q_e, q_d, q_c)\text{-2-Outsider-Auth}_r}$
$(sk_1, pk_1) \xleftarrow{\$} \mathsf{Gen}$
$(sk_2, pk_2) \xleftarrow{\$} \mathsf{Gen}$
$\mathcal{E} \leftarrow \emptyset$
$b \xleftarrow{\$} \mathcal{A}^{\text{AEncap},\text{ADecap},\text{Chall}}(pk_1, pk_2)$
return $b$

Oracle AEncap$(i \in [2], pk)$
$(c, K) \xleftarrow{\$} \mathsf{AuthEncap}(sk_i, pk)$
$\mathcal{E} \leftarrow \mathcal{E} \cup \{(pk_i, pk, c, K)\}$
return $(c, K)$

Oracle ADecap$(j \in [2], pk, c)$
if $\exists K : (pk, pk_j, c, K) \in \mathcal{E}$
    return $\perp$
$K \leftarrow \mathsf{AuthDecap}(sk_j, pk, c)$
$\mathcal{E} \leftarrow \mathcal{E} \cup \{(pk, pk_j, c, K)\}$
return $K$

Oracle Chall$(i \in [2], j \in [2], c)$
if $\exists K : (pk_i, pk_j, c, K) \in \mathcal{E}$
    return $\perp$
$K \leftarrow \mathsf{AuthDecap}(sk_j, pk_i, c)$
if $K \neq \perp$
    $\boxed{K \xleftarrow{\$} \mathcal{K}}$
    $\mathcal{E} \leftarrow \mathcal{E} \cup \{(pk_i, pk_j, c, K)\}$
return $K$
```

Theorem 12 ($(q_e, q_d, 1)$-1-Insider-CCA $\implies$ (n, q_e, q_d, q_c)-Insider-CCA)**.** *For any adversary $\mathcal{A}$ against the (n, q_e, q_d, q_c)-Insider-CCA security experiment, there exists an adversary $\mathcal{B}$ against the $(q_e, q_d, 1)$-1-Insider-CCA security experiment such that $t_\mathcal{B} \approx t_\mathcal{A}$ and*

$$\mathsf{Adv}_{\mathcal{A},\mathsf{AKEM}}^{(n,q_e,q_d,q_c)\text{-Insider-CCA}} \leq nq_c \cdot \mathsf{Adv}_{\mathcal{B},\mathsf{AKEM}}^{(q_e,q_d,1)\text{-1-Insider-CCA}} .$$

Theorem 13 ($(q_e, q_d, 1)$-2-Outsider-Auth $\implies$ (n, q_e, q_d)-Outsider-Auth)**.** *For any adversary $\mathcal{A}$ against (n, q_e, q_d)-Outsider-Auth security of AKEM, there exists an adversary $\mathcal{B}$ against $(q_e, q_d, 1)$-2-Outsider-Auth security of AKEM such that $t_\mathcal{B} \approx t_\mathcal{A}$ and*

$$\mathsf{Adv}_{\mathcal{A},\mathsf{AKEM}}^{(n,q_e,q_d)\text{-Outsider-Auth}} \leq n^2 q_d \cdot \mathsf{Adv}_{\mathcal{B},\mathsf{AKEM}}^{(q_e,q_d,1)\text{-2-Outsider-Auth}} + n^2 \cdot P_{\mathsf{AKEM}} .$$

A.1 Proof of Theorem 11

Proof. Let $\mathcal{A}$ be an adversary against (n, q_e, q_d)-Outsider-CCA security of AKEM that makes at most n queries to Gen, at most q_e queries to AEncap and q_d queries to ADecap. Consider the games G_0-G_4 in Listing 16.

Listing 16: Games G_0-G_4 for the proof of Theorem 11.

```
$\mathsf{G}_0$, $\mathsf{G}_1$, $(\mathsf{G}_{2,u,v,q})_{u,v\in[n],q\in[q_e]_0}$, $\mathsf{G}_3$, $\mathsf{G}_4$
$ctr \leftarrow 0$                                        //$\mathsf{G}_{2,u,v,q}$
$\ell \leftarrow 0$
$\mathcal{E} \leftarrow \emptyset$
$\mathsf{COLL}_{pk} \leftarrow$ false
$b \xleftarrow{\$} \mathcal{A}^{\text{Gen,AEncap,ADecap}}$
return $b$

Oracle Gen
$\ell \leftarrow \ell + 1$
$(sk_\ell, pk_\ell) \xleftarrow{\$} \mathsf{Gen}$
if $pk_\ell \in \{pk_1, \dots pk_{\ell-1}\}$                  //$\mathsf{G}_1$-$\mathsf{G}_3$
    $\mathsf{COLL}_{pk} \leftarrow$ true; abort              //$\mathsf{G}_1$-$\mathsf{G}_3$
return $(pk_\ell, \ell)$

Oracle ADecap($j \in [\ell], pk, c$)
try get $K$ s.t. $(pk, pk_j, c, K) \in \mathcal{E}$
    then return $K$
$K \leftarrow \mathsf{AuthDecap}(sk_j, pk, c)$
return $K$

Oracle AEncap($i \in [\ell], pk$)
$(c, K) \xleftarrow{\$} \mathsf{AuthEncap}(sk_i, pk)$
if $pk \in \{pk_1, \dots, pk_\ell\}$
    find $j$ such that $pk = pk_j$                         //$\mathsf{G}_{2,u,v,q}$
    if $i < u$ or $(i = u \wedge j < v)$                   //$\mathsf{G}_{2,u,v,q}$
        $K \xleftarrow{\$} \mathcal{K}$                     //$\mathsf{G}_{2,u,v,q}$
    if $i = u$ and $j = v$                                 //$\mathsf{G}_{2,u,v,q}$
        $ctr \leftarrow ctr + 1$                           //$\mathsf{G}_{2,u,v,q}$
        if $ctr \leq q$                                    //$\mathsf{G}_{2,u,v,q}$
            $K \xleftarrow{\$} \mathcal{K}$                 //$\mathsf{G}_{2,u,v,q}$
    $K \xleftarrow{\$} \mathcal{K}$                         //$\mathsf{G}_3$-$\mathsf{G}_4$
    $\mathcal{E} \leftarrow \mathcal{E} \uplus \{(pk_i, pk, c, K)\}$
return $(c, K)$
```

Game G_0. Note that this game is the (n, q_e, q_d)-Outsider-CCA$_\ell$ game. We already initialise multiset $\mathcal{E}$ and boolean flag COLL_{pk}, which we will need in the following games to exclude collisions. In G_0, we model $\mathcal{E}$ as a multiset to store all challenge queries to AEncap and to already match the syntax of the (n, q_e, q_d)-Outsider-CCA$_r$ game. If a challenge is queried to the decapsulation oracle ADecap, then the challenge key stored in $\mathcal{E}$ is directly returned. Due to perfect correctness, this is only a conceptual change. We have

$$\Pr[\mathsf{G}_0 \Rightarrow 1] = \Pr[(n, q_e, q_d)\text{-Outsider-CCA}_\ell(\mathcal{A}) \Rightarrow 1] \ .$$

Game G_1. In this game, we raise flag COLL_{pk} and abort, whenever a query to Gen generates a public key that was already generated in a previous query to Gen. Due to the difference lemma,

$$|\Pr[\mathsf{G}_0 \Rightarrow 1] - \Pr[\mathsf{G}_1 \Rightarrow 1]| \leq \Pr[\mathsf{COLL}_{pk}] \ ,$$

which we can bound by

$$\Pr[\mathsf{COLL}_{pk}] \leq \binom{n}{2} \cdot P_{\mathsf{AKEM}} \leq \frac{n^2}{2} \cdot P_{\mathsf{AKEM}} \ .$$

In the following we want to iterate over all pairs of honest users using indices $u, v \in [n]$ and all their challenges using index $q \in [q_e]_0$ and replace the corresponding challenge key by a random key. We capture each iteration in an intermediate game $\mathsf{G}_{2,u,v,q}$, where the key of the q-th query to AEncap for sender public key pk_u and receiver public key pk_v is replaced by random.

At this point we want to remark that q_e is the number of queries for *all* users and regardless whether it is a challenge query or not. In particular, the actual number of challenge queries for a specific user pair might be lower. However, in our following analysis we cannot easily use this fact to achieve a better bound because we have to cover all cases, which explains the tightness loss of $n^2 \cdot q_e$.

Games $(\mathsf{G}_{2,u,v,q})_{u,v\in[n],q\in[q_e]_0}$. As each public key generated by the game is unique, we can now find a unique index j for queries to AEncap such that the public key pk provided by $\mathcal{A}$ matches exactly one pk_j. To identify the corresponding challenge query, we use a counter ctr. Each time, a challenge query for sender public key pk_u and receiver public key pk_v is issued, the counter is increased. Note that the first intermediate game $\mathsf{G}_{2,1,1,0}$ is exactly game G_1, thus

$$\Pr[\mathsf{G}_{2,1,1,0} \Rightarrow 1] = \Pr[\mathsf{G}_1 \Rightarrow 1] \ .$$

Also note that whenever q reaches q_e, there will be no change in the next step. In particular, for all $u \in [n]$, $v \in [n-1]$, G_{2,u,v,q_e} is the same as $\mathsf{G}_{2,u,v+1,0}$, as well as for all $u \in [n-1]$, G_{2,u,n,q_e} is the same as $\mathsf{G}_{2,u+1,1,0}$ and thus

$$\Pr[\mathsf{G}_{2,u,v,q_e} \Rightarrow 1] = \Pr[\mathsf{G}_{2,u,v+1,0} \Rightarrow 1] \ ,$$

$$\Pr[\mathsf{G}_{2,u,n,q_e} \Rightarrow 1] = \Pr[\mathsf{G}_{2,u+1,1,0} \Rightarrow 1] \ .$$

On a high level each game $\mathsf{G}_{2,u,v,q}$ separates the output of AEncap on a query (i, pk_j), where $pk_j \in \{pk_1, \dots, pk_\ell\}$ is one of the public keys generated by the game so far, into two sets. The game outputs 1) a random key if $i < u$ or if $i = u$ and $j < v$ as well as if $i = u$ and $j = v$ and $ctr \leq q$, and 2) the real key otherwise.

We now need to bound the difference between games $\mathsf{G}_{2,u,v,q-1}$ and $\mathsf{G}_{2,u,v,q}$ for all $u, v \in [n]$, $q \in [q_e]$. Therefore, we construct adversary $\mathcal{B}_{u,v,q}$ against $(q_e, q_d, 1)$-2-Outsider-CCA in Listing 17.

Adversary $\mathcal{B}_{u,v,q}$ has access to oracles `AEncap`, `ADecap` and `Chall`. It inputs two public keys which we denote by pk'_1 and pk'_2 in order to avoid confusion with public keys pk_1 and pk_2 which will be the first two public keys that $\mathcal{B}_{u,v,q}$ outputs to $\mathcal{A}$ when $\mathcal{A}$ queries Gen. On the u-th query to Gen, $\mathcal{B}_{u,v,q}$ will then output pk'_1 and on the v-the query it will output pk'_2. Note that when $u = v$, $\mathcal{B}_{u,v,q}$ will only use pk'_1. This captures queries to AEncap of the form (i, pk_i). All remaining key pairs will be honestly generated with the condition that $\mathcal{B}_{u,v,q}$ aborts whenever the same key is generated twice, as introduced in game G_1. In order to avoid convoluted case distinctions in Listing 17, we define a function $f : \{u, v\} \mapsto \{1, 2\}$ such that, when $u = v$, $f(u) = f(v) = 1$, and when $u \neq v$, $f(u) = 1$ and $f(v) = 2$. This ensures that oracles `AEncap`, `ADecap` and `Chall` are queried on the correct indices.

Queries to ADecap are simulated using the `ADecap` oracle whenever $\mathcal{A}$ makes a query on u or v. Otherwise, $\mathcal{B}_{u,v,q}$ can run the AuthDecap algorithm, since it knows the secret key sk_j.

Queries to AEncap are simulated as follows: We first want to run the AuthEncap algorithm for all queries, except for the q-th challenge query of sender pk_u and receiver pk_v.

Listing 17: Adversary $\mathcal{B}_{u,v,q}$ against $(q_e, q_d, 1)$-2-$\mathsf{Outsider\text{-}CCA}$, where $u, v \in [n]$, $q \in [q_e]$ and $f : \{u, v\} \mapsto \{1, 2\}$ is a function such that, when $u = v$, $f(u) = f(v) = 1$, and when $u \neq v$, $f(u) = 1$ and $f(v) = 2$.

```
$\mathcal{B}_{u,v,q}^{\mathtt{AEncap},\mathtt{ADecap},\mathtt{Chall}}(pk'_1, pk'_2)$
01 $ctr \leftarrow 0$
02 $\ell \leftarrow 0$
03 $\mathcal{E} \leftarrow \emptyset$
04 $b \xleftarrow{\$} \mathcal{A}^{\text{Gen},\text{AEncap},\text{ADecap}}$
05 return $b$

Oracle Gen
06 $\ell \leftarrow \ell + 1$
07 if $\ell = u$
08     $pk_u \leftarrow pk'_1$
09 else if $\ell = v$
10     $pk_v \leftarrow pk'_2$
11 else
12     $(sk_\ell, pk_\ell) \xleftarrow{\$} \mathsf{Gen}$
13 if $pk_\ell \in \{pk_1, \ldots pk_{\ell-1}\}$
14     abort
15 return $(pk_\ell, \ell)$

Oracle ADecap$(j \in [\ell], pk, c)$
16 try get $K$ s.t. $(pk, pk_j, c, K) \in \mathcal{E}$
17     then return $K$
18 if $j \in \{u, v\}$
19     $K \leftarrow \mathtt{ADecap}(f(j), pk, c)$
20 else
21     $K \leftarrow \mathsf{AuthDecap}(sk_j, pk, c)$
22 return $K$

Oracle AEncap$(i \in [\ell], pk)$
23 if $i \notin \{u, v\}$
24     $(c, K) \xleftarrow{\$} \mathsf{AuthEncap}(sk_i, pk)$
25 else if $i = u$ and $pk = pk_v$
26     $ctr \leftarrow ctr + 1$
27     if $ctr = q$
28         $(c, K) \leftarrow \mathtt{Chall}(f(i), f(v))$
29     else
30         $(c, K) \leftarrow \mathtt{AEncap}(f(i), pk)$
31 else
32     $(c, K) \leftarrow \mathtt{AEncap}(f(i), pk)$
33 if $pk \in \{pk_1, \ldots, pk_\ell\}$
34     find $j$ such that $pk = pk_j$
35     if $i < u$ or $(i = u \wedge j < v)$
       or $(i = u \wedge j = v \wedge ctr < q)$
36         $K \xleftarrow{\$} \mathcal{K}$
37     $\mathcal{E} \leftarrow \mathcal{E} \uplus \{(pk_i, pk, c, K)\}$
38 return $(c, K)$
```

Thus, if the index i supplied by $\mathcal{A}$ is not u or v, $\mathcal{B}_{u,v,q}$ can run the $\mathsf{AuthEncap}$ algorithm itself, since it knows the secret key sk_i. For all other queries, $\mathcal{B}_{u,v,q}$ needs to calls its $\mathtt{AEncap}$ oracle. However, if this is the q-th challenge query of sender pk_u and receiver pk_v, $\mathcal{B}_{u,v,q}$ calls the challenge oracle $\mathtt{Chall}$ instead, after incrementing ctr. At this point note that $\mathcal{B}_{u,v,q}$ makes at most q_e queries to the $\mathtt{AEncap}$ oracle and only a single query to $\mathtt{Chall}$.

We check for public key collisions and $\mathcal{B}_{u,v,q}$ aborts whenever $\mathsf{G}_{2,u,v,q}$ would abort. Now we can proceed to replacing all challenge keys which were considered in previous games. In particular, if $i < u$ or $i = u$ and $j < v$, where $pk = pk_j$ for some index $j \in [\ell]$, we choose a random key. We do the same in case $i = u$ and $j = v$ and $ctr < q$. Note that if $\mathcal{B}_{u,v,q}$ is in the $(q_e, q_d, 1)$-2-$\mathsf{Outsider\text{-}CCA}_\ell$ game, it perfectly simulates $\mathsf{G}_{2,u,v,q-1}$. Otherwise, if $\mathcal{B}_{u,v,q}$ is in the $(q_e, q_d, 1)$-2-$\mathsf{Outsider\text{-}CCA}_r$ game, it perfectly simulates $\mathsf{G}_{2,u,v,q}$. Thus for all $q \in [q_e]$,

$$|\Pr[\mathsf{G}_{2,u,v,q-1} \Rightarrow 1] - \Pr[\mathsf{G}_{2,u,v,q} \Rightarrow 1]| \leq \mathsf{Adv}_{\mathcal{B}_{u,v,q},\mathsf{AKEM}}^{(q_e,q_d,1)\text{-2-}\mathsf{Outsider\text{-}CCA}} .$$

Game G_3. In this game, all challenge keys output by AEncap are random keys. Note that this game is the same as G_{2,u,v,q_e}, hence

$$\Pr[\mathsf{G}_3 \Rightarrow 1] = \Pr[\mathsf{G}_{2,n,n,q_e} \Rightarrow 1] .$$

Game G_4. In game G_4 we undo the change introduced in game G_1 and do not abort if a public key collision happens. Thus, we get

$$|\Pr[\mathsf{G}_3 \Rightarrow 1] - \Pr[\mathsf{G}_4 \Rightarrow 1]| \leq \frac{n^2}{2} \cdot P_{\mathsf{AKEM}} \ .$$

Finally note that G_4 is exactly the same as the (n, q_e, q_d)-$\mathsf{Outsider\text{-}CCA}_r$ game and

$$\Pr[\mathsf{G}_4 \Rightarrow 1] = \Pr[(n, q_e, q_d)\text{-}\mathsf{Outsider\text{-}CCA}_r(\mathcal{A}) \Rightarrow 1] \ .$$

Folding adversaries $\mathcal{B}_{u,v,q}$ into a single adversary $\mathcal{B}$ and collecting the probabilities yields the bound claimed in Theorem 11. □

A.2 Proof of Theorem 12

Proof. Let $\mathcal{A}$ be an adversary in the (n, q_e, q_d, q_c)-$\mathsf{Insider\text{-}CCA}$ security experiment that makes at most n queries to Gen, at most q_e queries to AEncap, q_d queries to ADecap and q_c queries to Chall. Consider the sequence of games G_0-G_2 in Listing 18.

Listing 18: Games G_0-G_2 for the proof of Theorem 12.

```
$\mathsf{G}_0$, $(\mathsf{G}_{1,u,q})_{u\in[n],q\in[q_c]_0}$, $\mathsf{G}_2$
$ctr \leftarrow 0$
$\ell \leftarrow 0$
$\mathcal{C} \leftarrow \emptyset$
$b \xleftarrow{\$} \mathcal{A}^{\text{Gen},\text{AEncap},\text{ADecap},\text{Chall}}$
return $b$

Oracle Gen
$\ell \leftarrow \ell + 1$
$(sk_\ell, pk_\ell) \xleftarrow{\$} \mathsf{Gen}$
return $(pk_\ell, \ell)$

Oracle ADecap($j \in [\ell], pk, c$)
try get $K$ s.t. $(pk, pk_j, c, K) \in \mathcal{C}$
    then return $K$
$K \leftarrow \mathsf{AuthDecap}(sk_j, pk, c)$
return $K$

Oracle AEncap($i \in [\ell], pk$)
$(c, K) \leftarrow \mathsf{AuthEncap}(sk_i, pk)$
return $(c, K)$

Oracle Chall($j \in [\ell], sk$)
$(c, K) \leftarrow \mathsf{AuthEncap}(sk, pk_j)$
if $j < u$                                   // $\mathsf{G}_{1,u,q}$
    $K \xleftarrow{\$} \mathcal{K}$          // $\mathsf{G}_{1,u,q}$
if $j = u$                                   // $\mathsf{G}_{1,u,q}$
    $ctr \leftarrow ctr + 1$                 // $\mathsf{G}_{1,u,q}$
    if $ctr \leq q$                          // $\mathsf{G}_{1,u,q}$
        $K \xleftarrow{\$} \mathcal{K}$      // $\mathsf{G}_{1,u,q}$
$K \xleftarrow{\$} \mathcal{K}$              // $\mathsf{G}_2$
$\mathcal{C} \leftarrow \mathcal{C} \uplus \{(\mu(sk), pk_j, c, K)\}$
return $(c, K)$
```

Game G_0. This is the original (n, q_e, q_d, q_c)-$\mathsf{Insider\text{-}CCA}_\ell$ game. The set $\mathcal{C}$ is used to store all queries to the challenge oracle Chall. If a challenge is queried to the decapsulation oracle ADecap, then the challenge key is directly returned. Due to perfect correctness, this is only a conceptual change. Thus,

$$\Pr[\mathsf{G}_0 \Rightarrow 1] = \Pr[(n, q_e, q_d, q_c)\text{-}\mathsf{Insider\text{-}CCA}_\ell(\mathcal{A}) \Rightarrow 1] \ .$$

In the following, we want to replace all real challenge keys by random keys, iterating over all users (indicated by index u) and their challenges (indicated by index q). We capture each iteration in an intermediate game $\mathsf{G}_{1,u,q}$, where the key of the q-th query to Chall for receiver public key pk_u is replaced by random.

At this point we want to remark that q_c is the total number of challenge queries for *all* users. In particular, the actual number of challenge queries for a specific user might be lower. However, in our following analysis we cannot easily use this fact to achieve a better bound because we have to cover all cases, which explains the tightness loss of $n \cdot q_c$.

Games $(\mathsf{G}_{1,u,q})_{u\in[n],q\in[q_c]_0}$. To identify the corresponding challenge query, we use a counter ctr. Each time, a challenge query for receiver public key pk_u is issued, the counter is increased. Note that the first intermediate game $\mathsf{G}_{1,1,0}$ is the same as G_0 as well as G_{1,u,q_c} is the same as $\mathsf{G}_{1,u+1,0}$ for all $u \in [n-1]$. Hence,

$$\Pr[\mathsf{G}_{1,1,0} \Rightarrow 1] = \Pr[\mathsf{G}_0 \Rightarrow 1] \ ,$$

and

$$\Pr[\mathsf{G}_{1,u,q_c} \Rightarrow 1] = \Pr[\mathsf{G}_{1,u+1,0} \Rightarrow 1] \ .$$

In order to bound the difference between $\mathsf{G}_{1,u,q-1}$ and $\mathsf{G}_{1,u,q}$ for $u \in [n]$, $q \in [q_c]$, we construct an adversary $\mathcal{B}_{u,q}$ against $(q_e, q_d, 1)$-1-Insider-CCA in Listing 19.

Listing 19: Adversary $\mathcal{B}_{u,q}$ against $(q_e, q_d, 1)$-1-Insider-CCA, where $u \in [n]$ and $q \in [q_c]$.

```
$\mathcal{B}_{u,q}^{\texttt{AEncap},\texttt{ADecap},\texttt{Chall}}(pk^*)$
$ctr \leftarrow 0$
$\ell \leftarrow 0$
$\mathcal{C} \leftarrow \emptyset$
$b \xleftarrow{\$} \mathcal{A}^{\textsc{Gen},\textsc{AEncap},\textsc{ADecap},\textsc{Chall}}$
return $b$

Oracle Gen
$\ell \leftarrow \ell + 1$
if $\ell = u$
    $pk_u \leftarrow pk^*$
else
    $(sk_\ell, pk_\ell) \xleftarrow{\$} \mathsf{Gen}$
return $(pk_\ell, \ell)$

Oracle ADecap$(j \in [\ell], pk, c)$
try get $K$ s.t. $(pk, pk_j, c, K) \in \mathcal{C}$
  then return $K$
if $j = u$
    $K \leftarrow \texttt{ADecap}(pk, c)$
else
    $K \leftarrow \mathsf{AuthDecap}(sk_j, pk, c)$
return $K$

Oracle AEncap$(i \in [\ell], pk)$
if $i = u$
    $(c, K) \leftarrow \texttt{AEncap}(pk)$
else
    $(c, K) \leftarrow \mathsf{AuthEncap}(sk_i, pk)$
return $(c, K)$

Oracle Chall$(j \in [\ell], sk)$
$(c, K) \leftarrow \mathsf{AuthEncap}(sk, pk_j)$
if $j = u$
    $ctr \leftarrow ctr + 1$
    if $ctr = q$
        $(c, K) \leftarrow \texttt{Chall}(sk)$
if $j < u$ or $(j = u \wedge ctr < q)$
    $K \xleftarrow{\$} \mathcal{K}$
$\mathcal{C} \leftarrow \mathcal{C} \uplus \{(\mu(sk), pk_j, c, K)\}$
return $(c, K)$
```

$\mathcal{B}_{u,q}$ inputs a challenge public key pk^* and has access to oracles `AEncap`, `ADecap`, `Chall`. It initialises counters ctr for the number of challenge queries and ℓ for the number of users. On the u-th query to Gen, $\mathcal{B}_{u,q}$ outputs pk^* to $\mathcal{A}$. On all other queries, it samples a new key pair. If $\mathcal{A}$ queries AEncap on index u, $\mathcal{B}_{u,q}$ uses its `AEncap` oracle to answer the query. For all other indices, it knows the corresponding secret key and can run the AuthEncap algorithm on its own. The same applies for queries to the ADecap oracle.

If $\mathcal{A}$ issues a challenge query on an index j and a secret key sk, $\mathcal{B}_{u,q}$ first runs the $\mathsf{AuthEncap}$ algorithm itself. If $j = u$, the counter ctr is incremented and if this is the q-th query, $\mathcal{B}_{u,q}$ queries the challenge oracle `Chall`. It then checks whether the challenge key needs to be replaced by a random key, that means in case $j < u$ or in case $j = u$ and $ctr < q$.

If $\mathcal{B}_{u,q}$ is in the $(q_e, q_d, 1)$-$\mathsf{1\text{-}Insider\text{-}CCA}_\ell$ game, it perfectly simulates $\mathsf{G}_{1,u,q-1}$. Otherwise, if $\mathcal{B}_{u,q}$ is in the $(q_e, q_d, 1)$-$\mathsf{1\text{-}Insider\text{-}CCA}_r$ game, it perfectly simulates $\mathsf{G}_{1,u,q}$, hence

$$|\Pr[\mathsf{G}_{1,u,q-1} \Rightarrow 1] - \Pr[\mathsf{G}_{1,u,q} \Rightarrow 1]| \le \mathsf{Adv}^{(q_e,q_d,1)\text{-}\mathsf{1\text{-}Insider\text{-}CCA}}_{\mathcal{B}_{u,q},\mathsf{AKEM}} .$$

Game G_2. In this game, all keys output by the challenge oracle Chall are random keys. Note that this game is the same as G_{1,n,q_c}, hence

$$\Pr[\mathsf{G}_2 \Rightarrow 1] = \Pr[\mathsf{G}_{1,n,q_c} \Rightarrow 1] .$$

Finally note that G_2 is exactly the same as the (n, q_e, q_d, q_c)-$\mathsf{Insider\text{-}CCA}_r$ game and

$$\Pr[\mathsf{G}_2 \Rightarrow 1] = \Pr[(n, q_e, q_d, q_c)\text{-}\mathsf{Insider\text{-}CCA}_r(\mathcal{A}) \Rightarrow 1] .$$

Folding adversaries $\mathcal{B}_{u,q}$ into a single adversary $\mathcal{B}$ and collecting the probabilities yields the bound claimed in Theorem 12. □

A.3 Proof of Theorem 13

Proof. Let $\mathcal{A}$ be an adversary against (n, q_e, q_d)-$\mathsf{Outsider\text{-}Auth}$ security of AKEM that makes at most n queries to Gen, at most q_e queries to AEncap and q_d queries to ADecap. Consider the games G_0-G_4 in Listing 20. We proceed similarly to the proof of Theorem 11.

Game G_0. Note that this game is the (n, q_e, q_d)-$\mathsf{Outsider\text{-}Auth}_\ell$ game. We already initialise the multiset $\mathcal{E}$ and boolean flag COLL_{pk}, which we will need in the following games. In G_0, $\mathcal{E}$ is used to store all queries to the encapsulation oracle AEncap and all queries to the decapsulation oracle ADecap, which correspond to challenge queries, that is all queries where pk supplied by $\mathcal{A}$ is one of the public keys generated by the game. If a challenge or a previous AEncap query is queried to ADecap again, then the corresponding key is directly returned. Due to perfect correctness, this is only a conceptual change. Thus,

$$\Pr[\mathsf{G}_0 \Rightarrow 1] = \Pr[(n, q_e, q_d)\text{-}\mathsf{Outsider\text{-}Auth}_\ell(\mathcal{A}) \Rightarrow 1] .$$

Game G_1. In this game, we raise flag COLL_{pk} and abort, whenever a query to Gen generates a public key that was already generated in a previous query to Gen. Due to the difference lemma,

$$|\Pr[\mathsf{G}_0 \Rightarrow 1] - \Pr[\mathsf{G}_1 \Rightarrow 1]| \le \Pr[\mathsf{COLL}_{pk}] ,$$

which we can bound by

$$\Pr[\mathsf{COLL}_{pk}] \le \binom{n}{2} \cdot P_{\mathsf{AKEM}} \le \frac{n^2}{2} \cdot P_{\mathsf{AKEM}} .$$

In the following we want to iterate over all pairs of honest users using indices $u, v \in [n]$ and all their challenges using index $q \in [q_d]_0$ and replace the corresponding challenge key

Listing 20: Games G_0-G_4 for the proof of Theorem 13, where $u \in [n], v \in [n]_0$.

```
$\mathsf{G}_0$, $\mathsf{G}_1$, $(\mathsf{G}_{2,u,v})_{u\in[n],v\in[n]_0}$, $\mathsf{G}_3$, $\mathsf{G}_4$
$ctr \leftarrow 0$                                          $/\!/\mathsf{G}_{2,u,v,q}$
$\ell \leftarrow 0$
$\mathcal{E} \leftarrow \emptyset$
$\mathsf{COLL}_{pk} \leftarrow$ false
$b \xleftarrow{\$} \mathcal{A}^{\textsc{Gen},\textsc{AEncap},\textsc{ADecap}}$
return $b$

Oracle Gen
$\ell \leftarrow \ell + 1$
$(sk_\ell, pk_\ell) \xleftarrow{\$} \mathsf{Gen}$
if $pk_\ell \in \{pk_1, \dots pk_{\ell-1}\}$                  $/\!/\mathsf{G}_1$-$\mathsf{G}_3$
    $\mathsf{COLL}_{pk} \leftarrow$ true; abort                $/\!/\mathsf{G}_1$-$\mathsf{G}_3$
return $(pk_\ell, \ell)$

Oracle AEncap$(i \in [\ell], pk)$
$(c, K) \xleftarrow{\$} \mathsf{AuthEncap}(sk_i, pk)$
$\mathcal{E} \leftarrow \mathcal{E} \uplus \{(pk_i, pk, c, K)\}$
return $(c, K)$
```

```
Oracle ADecap$(j \in [\ell], pk, c)$
try get $K$ s.t. $(pk, pk_j, c, K) \in \mathcal{E}$
    then return $K$
$K \leftarrow \mathsf{AuthDecap}(sk_j, pk, c)$
if $pk \in \{pk_1, \dots, pk_\ell\}$ and $K \neq \perp$
    find $i$ such that $pk = pk_i$                      $/\!/\mathsf{G}_{2,u,v,q}$
    if $j < u$ or $(j = u \wedge i < v)$                 $/\!/\mathsf{G}_{2,u,v,q}$
        $K \xleftarrow{\$} \mathcal{K}$                                   $/\!/\mathsf{G}_{2,u,v,q}$
    if $j = u \wedge i = v$                              $/\!/\mathsf{G}_{2,u,v,q}$
        $ctr \leftarrow ctr + 1$                            $/\!/\mathsf{G}_{2,u,v,q}$
        if $ctr \leq q$                                  $/\!/\mathsf{G}_{2,u,v,q}$
            $K \xleftarrow{\$} \mathcal{K}$                               $/\!/\mathsf{G}_{2,u,v,q}$
    $K \xleftarrow{\$} \mathcal{K}$                                       $/\!/\mathsf{G}_3$-$\mathsf{G}_4$
    $\mathcal{E} \leftarrow \mathcal{E} \uplus \{(pk_i, pk_j, c, K)\}$
return $K$
```

by a random key. We capture each iteration in an intermediate game $\mathsf{G}_{2,u,v,q}$, where the key of the q-th query to ADecap for sender public key pk_u and receiver public key pk_v is replaced by random.

At this point we want to remark that q_d is the number of queries for *all* users and regardless whether it is a challenge query or not. In particular, the actual number of challenge queries for a specific user pair might be lower. However, in our following analysis we cannot easily use this fact to achieve a better bound because we have to cover all cases, which explains the tightness loss of $n^2 \cdot q_d$.

Games $(\mathsf{G}_{2,u,v,q})_{u,v\in[n],q\in[q_d]_0}$. To identify the corresponding challenge query, we use a counter ctr. Each time a challenge query for receiver public key pk_u and sender public key pk_v is issued, the counter is increased. Note that the first intermediate game $\mathsf{G}_{2,1,1,0}$ is exactly game G_1, thus

$$\Pr[\mathsf{G}_{2,1,1,0} \Rightarrow 1] = \Pr[\mathsf{G}_1 \Rightarrow 1] \ .$$

Also note that whenever q reaches q_d, there will be no change in the next step. In particular, for all $u \in [n]$, $v \in [n-1]$, G_{2,u,v,q_d} is the same as $\mathsf{G}_{2,u,v+1,0}$, as well as for all $u \in [n-1]$, G_{2,u,n,q_e} is the same as $\mathsf{G}_{2,u+1,1,0}$ and thus

$$\Pr[\mathsf{G}_{2,u,v,q_d} \Rightarrow 1] = \Pr[\mathsf{G}_{2,u,v+1,0} \Rightarrow 1] \ ,$$

$$\Pr[\mathsf{G}_{2,u,n,q_d} \Rightarrow 1] = \Pr[\mathsf{G}_{2,u+1,1,0} \Rightarrow 1] \ .$$

On a high level each game $\mathsf{G}_{2,u,v,q}$ separates the output of ADecap on a query (j, pk_i), where $pk_i \in \{pk_1, \dots, pk_\ell\}$ is one of the public keys generated by the game so far, into two sets. The game outputs 1) a random key if $j < u$ or if $j = u$ and $i < v$ as well as if $j = u$ and $i = v$ and $ctr \leq q$, and 2) the real key otherwise.

At this point note that for challenges the index i is uniquely defined as each public key generated by the game is unique. Also, each entry in the multiset $\mathcal{E}$ is unique.

We now need to bound the difference between games $\mathsf{G}_{2,u,v,q-1}$ and $\mathsf{G}_{2,u,v,q}$ for all $u, v \in [n]$, $q \in [q_d]$. Therefore, we construct adversary $\mathcal{B}_{u,v,q}$ against $(q_e, q_d, 1)$-2-$\mathsf{Outsider\text{-}Auth}$ in Listing 21.

Listing 21: Adversary $\mathcal{B}_{u,v,q}$ against $(q_e, q_d, 1)$-2-$\mathsf{Outsider\text{-}Auth}$, where $f : \{u, v\} \mapsto \{1, 2\}$ is a function such that, when $u = v$, $f(u) = f(v) = 1$, and when $u \neq v$, $f(u) = 1$ and $f(v) = 2$.

```
$\mathcal{B}_{u,v,q}^{\texttt{AEncap},\texttt{ADecap},\texttt{Chall}}(pk'_1, pk'_2)$
$ctr \leftarrow 0$
$\ell \leftarrow 0$
$\mathcal{E} \leftarrow \emptyset$
$b \xleftarrow{\$} \mathcal{A}^{\textsc{Gen},\textsc{AEncap},\textsc{ADecap}}$
return $b$

Oracle Gen
$\ell \leftarrow \ell + 1$
if $\ell = u$
    $pk_u \leftarrow pk'_1$
else if $\ell = v$
    $pk_v \leftarrow pk'_2$
else
    $(sk_\ell, pk_\ell) \xleftarrow{\$} \mathsf{Gen}$
if $pk_\ell \in \{pk_1, \dots pk_{\ell-1}\}$
    abort
return $(pk_\ell, \ell)$

Oracle AEncap$(i \in [\ell], pk)$
if $i \in \{u, v\}$
    $(c, K) \leftarrow \texttt{AEncap}(f(i), pk)$
else
    $(c, K) \xleftarrow{\$} \mathsf{AuthEncap}(sk_i, pk)$
$\mathcal{E} \leftarrow \mathcal{E} \uplus \{(pk_i, pk, c, K)\}$
return $(c, K)$

Oracle ADecap$(j \in [\ell], pk, c)$
try get $K$ s.t. $(pk, pk_j, c, K) \in \mathcal{E}$
    then return $K$
if $pk \notin \{pk_1, \dots, pk_\ell\}$
    if $j \in \{u, v\}$
        $K \leftarrow \texttt{ADecap}(f(j), pk, c)$
    else
        $K \leftarrow \mathsf{AuthDecap}(sk_j, pk, c)$
    return $K$
find $i$ such that $pk = pk_i$
if $j \notin \{u, v\}$
    $K \leftarrow \mathsf{AuthDecap}(sk_j, pk, c)$
else if $j = u$ and $i = v$
    $ctr \leftarrow ctr + 1$
    if $ctr = q$
        $K \leftarrow \texttt{Chall}(f(j), f(i), c)$
    else
        $K \leftarrow \texttt{ADecap}(f(j), pk_i, c)$
else
    $K \leftarrow \texttt{ADecap}(f(j), pk_i, c)$
if $j < u$ or $(j = u \wedge i < v)$
or $(j = u \wedge i = v \wedge ctr < q)$
    if $K \neq \perp$
        $K \xleftarrow{\$} \mathcal{K}$
if $K \neq \perp$
    $\mathcal{E} \leftarrow \mathcal{E} \uplus \{(pk, pk_j, c, K)\}$
return $K$
```

Adversary $\mathcal{B}_{u,v,q}$ has access to oracles $\texttt{AEncap}$, $\texttt{ADecap}$ and $\texttt{Chall}$. It inputs two public keys which we denote by pk'_1 and pk'_2 in order to avoid confusion with public keys pk_1 and pk_2 which will be the first two public keys that $\mathcal{B}_{u,v,q}$ outputs to $\mathcal{A}$ when $\mathcal{A}$ queries $\textsc{Gen}$. On the u-th query to $\textsc{Gen}$, $\mathcal{B}_{u,v,q}$ will then output pk'_1 and on the v-the query it will output pk'_2. Note that when $u = v$, $\mathcal{B}_{u,v,q}$ will only use pk'_1. This captures queries to $\textsc{ADecap}$ of the form (j, pk_j). All remaining key pairs will be honestly generated with the condition that $\mathcal{B}_{u,v,q}$ aborts whenever the same key is generated twice, as introduced in game G_1. As in the proof of Theorem 11, we define a function $f : \{u, v\} \mapsto \{1, 2\}$ such that, when $u = v$, $f(u) = f(v) = 1$, and when $u \neq v$, $f(u) = 1$ and $f(v) = 2$. This ensures that oracles $\texttt{AEncap}$, $\texttt{ADecap}$ and $\texttt{Chall}$ are queried on the correct indices.

Queries to $\textsc{AEncap}$ are simulated using the $\texttt{AEncap}$ oracle whenever $\mathcal{A}$ makes a query on u or v. Otherwise, $\mathcal{B}_{u,v,q}$ can run the $\mathsf{AuthEncap}$ algorithm, since it knows the secret key sk_i.

Queries to ADecap are simulated as follows: We first take care of non-challenges, that means where the public key supplied by $\mathcal{A}$ is not one of the public keys generated by $\mathcal{B}_{u,v,q}$ so far. If the index j supplied by $\mathcal{A}$ is either u or v, $\mathcal{B}_{u,v,q}$ queries its `ADecap` oracle. Otherwise, it knows the corresponding secret key and can run the AuthDecap algorithm itself. The result will then returned to $\mathcal{A}$. If the query was a challenge query, then pk must be pk_i for some unique $i \in [\ell]$. Before we can replace any challenge keys, we first need to compute the output of AuthDecap using the algorithm itself or the correct oracle. If $j \notin \{u, v\}$, $\mathcal{B}_{u,v,q}$ knows the secret key and can thus compute the output on its own. If $j = u$ and $i = v$, we need to increment the counter ctr and if this is the q-th query, $\mathcal{B}_{u,v,q}$ calls the challenge oracle `Chall`. In all other cases, it calls `ADecap`. At this point note that $\mathcal{B}_{u,v,q}$ makes at most q_d queries to the `ADecap` oracle and only a single query to `Chall`.

Now we can proceed to replacing all challenge keys which were considered in previous games. In particular, if $j < u$ or $j = u$ and $i < v$, we choose a random key. We do the same in case $j = u$ and $i = v$ and $ctr < q$. Note that if $\mathcal{B}_{u,v,q}$ is in the $(q_e, q_d, 1)$-2-Outsider-Auth$_\ell$ game, it perfectly simulates $\mathsf{G}_{2,u,v,q-1}$. Otherwise, if $\mathcal{B}_{u,v,q}$ is in the $(q_e, q_d, 1)$-2-Outsider-Auth$_r$ game, it perfectly simulates $\mathsf{G}_{2,u,v,q}$. Thus for all $q \in [q_d]$,

$$|\Pr[\mathsf{G}_{2,u,v,q-1} \Rightarrow 1] - \Pr[\mathsf{G}_{2,u,v,q} \Rightarrow 1]| \leq \mathsf{Adv}^{(q_e,q_d,1)\text{-2-Outsider-Auth}}_{\mathcal{B}_{u,v,q},\mathsf{AKEM}} .$$

Game G_3. In this game, all challenge keys output by the oracle ADecap are random keys. Note that this game is the same as G_{2,n,n,q_d}, hence

$$\Pr[\mathsf{G}_3 \Rightarrow 1] = \Pr[\mathsf{G}_{2,n,n,q_d} \Rightarrow 1] .$$

Game G_4. In game G_4 we undo the change introduced in game G_1 and do not abort if a public key collision happens. Thus, we get

$$|\Pr[\mathsf{G}_3 \Rightarrow 1] - \Pr[\mathsf{G}_4 \Rightarrow 1]| \leq \frac{n^2}{2} \cdot P_{\mathsf{AKEM}} .$$

Finally note that G_4 is exactly the same as the (n, q_e, q_d)-Outsider-Auth$_r$ game. We get

$$\Pr[\mathsf{G}_4 \Rightarrow 1] = \Pr[(n, q_e, q_d)\text{-Outsider-Auth}_r(\mathcal{A}) \Rightarrow 1] .$$

Folding adversaries $\mathcal{B}_{u,v,q}$ into a single adversary $\mathcal{B}$ and collecting the probabilities yields the bound claimed in Theorem 13. □

B Comparison of Pseudocode Definitions and their Implementation in CryptoVerif

The goal of this section is to convince the readers of this paper that the CryptoVerif models provided as supplementary material actually prove the theorems presented in the main body of this paper. Rather than explaining syntax and semantics of CryptoVerif on an abstract level, we take one of this paper's security notions and explain by example how the pseudocode presented in the main body relates to the code written in CryptoVerif. The security notion we use as example is (n, q_e, q_d)-Outsider-CCA of AKEM, presented in Listing 3. This notion appears in Theorem 7, where we show that DH-AKEM satisfies it, and it appears in the composition theorems 3 and 5, where we use it as an assumption. In Listing 22, we display the pseudocode definition of (n, q_e, q_d)-Outsider-CCA from Listing 3 in the left column next to its implementation as an assumption in CryptoVerif in the right column. The modelling of assumptions and proof goals in CryptoVerif has only some slight syntactical differences. Both columns contain line numbers, and we will use `Lxy` to refer to line number `xy` in the left column, and `Rxy` to refer to line number `xy` in the right column. As with all the listings throughout the paper so far, the game of the real version of the notion contains only the non-boxed lines, and the game of the ideal version additionally contains the boxed lines. The complete rolled-out version of CryptoVerif code for the (n, q_e, q_d)-Outsider-CCA notion as assumption can be found in lib.authkem.ocvl, from lines 102 to 143. The CryptoVerif files are available at [2].

In pseudocode, `L01` to `L04` constitute the *main procedure* of the game; they set up the experiment and call the adversary, giving it the oracles Gen, AEncap and ADecap. AEncap and ADecap explicitly take the indices i and j as parameter, by which the adversary chooses the private key upon which it wants to act. These indices are assigned by Gen.

In CryptoVerif code, the first line `R01` states that everything below will be executed in parallel `N` times. In CryptoVerif, this is called a *replication*. A replication index `i` is assigned to each of the parallel executions. Continuing to `R02`, in each of these parallel *processes*, a key pair will be generated. In our CryptoVerif model, we chose to represent key pairs in a general way by a key pair seed `s`, the private key computed from it by `skgen(s)` and the public key by `pkgen(s)`. This allows to cover a broad range of KEM constructions with one model. The type of key pair seed elements in our models is called `keypairseed`. The operator `<-R` denotes that the variable on the left is sampled randomly from the type given on the right. These keys can be dynamically generated, that is, line `R02` implicitly defines an oracle GenCV which generates a key pair seed `s`.

Line `R03` begins a block of oracle definitions, and it ends with line `R29`. In CryptoVerif, all oracles are implicitly accessible by the adversary. The three oracles that are defined are `Opk` in `R04`, `OAEncap` in `R09`, and `OADecap` in `R24`. Lines `R06` and `R22` define that these oracles are accessible by the adversary in parallel, in any order it chooses to call them.

The oracles GenCV, `Opk`, `OAEncap`, and `OADecap` are located *inside* the `N` times replication started in `R01`. This means that these oracles are available *once for each* $\mathtt{i} \in [1, \mathtt{N}]$, and they are defined with this index as implicit argument. Thus, the adversary has access to `N` oracles GenCV`[i]` for $1 \leq \mathtt{i} \leq \mathtt{N}$, and the variable defined in GenCV`[i]` is `s[i]`. All variables in CryptoVerif are implicitly stored in arrays indexed by the replication indices above their definition. Similarly, the adversary has access to `N` oracles `Opk[i]` for $1 \leq \mathtt{i} \leq \mathtt{N}$, and the variable accessed inside `Opk[i]` is `s[i]`. A call to GenCV`[i]` immediately followed by a call to `Opk[i]` is equivalent to a call to Gen, up to the renumbering of keys. (In pseudocode, the keys are indexed in increasing order. In CryptoVerif, the keys can be generated in any order. The renumbering just consists in applying a bijection known to the adversary to the indices of keys.) CryptoVerif separates generating keys by GenCV`[i]` from acquiring the public key by `Opk[i]`, while pseudocode groups these two actions in one oracle call Gen. However, a call to `Opk[i]` can be added immediately after GenCV`[i]` and all other calls to `Opk[i]` removed without changing the state of the system, so this separation leaves the security notion unchanged.

The main procedure in the pseudocode (lines `L01`–`L04`) is essentially absent in CryptoVerif. Line `L01` initializes the index for the last generated key ℓ. There is no such index

Listing 22: Games (n, q_e, q_d)-Outsider-CCA$_\ell$ and (n, q_e, q_d)-Outsider-CCA$_r$ for AKEM, with their modelling in CryptoVerif shown on the right side.

```
$(n, q_e, q_d)$-Outsider-CCA$_\ell$ and $(n, q_e, q_d)$-Outsider-CCA$_r$
$\ell \leftarrow 0$
$\mathcal{E} \leftarrow \emptyset$
$b \xleftarrow{\$} \mathcal{A}^{\textsc{Gen},\textsc{AEncap},\textsc{ADecap}}$
return $b$

Oracle Gen
$\ell \leftarrow \ell + 1$
$(sk_\ell, pk_\ell) \xleftarrow{\$} \mathsf{Gen}$
return $(pk_\ell, \ell)$

Oracle AEncap$(i \in [\ell], pk)$
$(c, K) \xleftarrow{\$} \mathsf{AuthEncap}(sk_i, pk)$
if $pk \in \{pk_1, \ldots, pk_\ell\}$
    $K \xleftarrow{\$} \mathcal{K}$
    $\mathcal{E} \leftarrow \mathcal{E} \uplus \{(pk_i, pk, c, K)\}$
return $(c, K)$

Oracle ADecap$(j \in [\ell], pk, c)$
try get $K$ s.t. $(pk, pk_j, c, K) \in \mathcal{E}$
    then return $K$
$K \leftarrow \mathsf{AuthDecap}(sk_j, pk, c)$
return $K$
```

```
foreach i <= N do
s <-R keypairseed;
(
Opk() :=
return(pkgen(s))

|

foreach ie <= Qeperuser do
ks <-R kemseed; (
OAEncap(pk_R) :=
find i2 <= N suchthat defined(s[i2])
    && pk_R = pkgen(s[i2]) then (
  sk <- skgen(s);
  let (k, ce) = AuthEncap_r(ks, pk_R, sk) in (
    k' <-R key;
    insert E(pkgen(s), pk_R, ce, k');
    return(AuthEncap_tuple(k', ce))
  ) else (
    return(AuthEncap_None)
  )
) else
  return(AuthEncap_r(ks, pk_R, skgen(s))))

|

foreach id <= Qdperuser do (
OADecap(pk_S, cd) :=
get E(=pk_S, =pkgen(s), =cd, k'') in (
  return(k'')
) else
  return(AuthDecap(cd, skgen(s), pk_S)))

)
```

in CryptoVerif, since the adversary provides the index of the key it generates at generation time, as argument to oracle GenCV. Line L02 initialises the multiset $\mathcal{E}$. In the CryptoVerif model, $\mathcal{E}$ is represented by a table `E`, which is implicitly empty at the beginning. Line L03 calls the adversary and line L04 returns its result. The adversary is not explicitly called in CryptoVerif: the adversary is implicit and is allowed to call all oracles provided by the CryptoVerif code.

Lines R07 and R23 define a replication, indicating that the adversary can call `OAEncap[i](pk_R)` for each key pair `Qeperuser` times, and `OADecap[i](pk_S, cd)` for each key pair `Qdperuser` times. The oracle `Opk` has no replication in front of it, which means the adversary can call it only once per key pair (which is sufficient to acquire the public key).

In the real version of the security notion, the oracle AEncap directly returns the result of the probabilistic AuthEncap function, see line L08. This corresponds to R21. In CryptoVerif, all functions are deterministic; to model a probabilistic function, one has to provide randomness to a function. For `AuthEncap_r` this is done with the `ks` parameter. It is chosen randomly by the oracle `OAEncap` independently for each call; the CryptoVerif syntax requires that this statement is located directly after the replication definition in R08, and thus before the oracle definition.

The public key given as parameter to `OAEncap` has the role of the recipient public key, which is why it is called `pk_R`. We proceed to explain how the ideal version of the security notion is formalised in CryptoVerif. The if statement in L09 corresponds to the `find` statement started in lines R10 and R11. This `find` implements a lookup in the set of honest public keys pk_i: it looks for an index `i2` $\in [1, \texttt{N}]$, and tests if the public key `pk_R` given as argument to the oracle is equal to the public key `pkgen(s[i2])` corresponding to the key pair seed `s[i2]`, that is, the value of `s` generated within the replicated copy indexed by `i2`. (The operator `&&` is the logical and.)

The syntax `let (...) = ...` in R13 denotes a pattern matching; it is used here to split up the tuple that the function `AuthEncap_r` returns. The actual pattern matching used in the CryptoVerif model is `let AuthEncap_tuple(k: key, ce: ciphertext) = ...`, but we abbreviated it to `let (k, ce) = ...` such that it fits into one line in the listing. In general, a pattern matching can fail in CryptoVerif, which is why we need to include an `else` case with lines R17 and R18; we return an error symbol `AuthEncap_None`, which corresponds to $\perp$. The actual AuthEncap does not fail; we model this with a simple game transformation `eliminate_failing` that expresses that `AuthEncap_r` always returns a pair and results in removing the `else` branch during the proofs. Lines R14 and R16 are executed if the pattern matching and thus the call to `AuthEncap_r` succeeded: A random KEM key `k'` is generated, a new element is added to the table `E`, which represents the multiset $\mathcal{E}$, and a new tuple is constructed from `k'` and the KEM ciphertext `ce`, and returned.

The difference remaining to be explained is that on the left side, there is only one call to AuthEncap, at the beginning of the oracle in line L08, but two calls to `AuthEncap_r` on the right side, in lines R13 and R21. These two calls are in disjoint branches of the code and thus both variants are equivalent. The reason we do not call `AuthEncap_r` once at the beginning of the oracle is purely technical: The proof is easier if we assign unique variable names per branch from the start (here: `(k, ce)` and no variable assignment in R21).

Oracle `OADecap` takes a sender public key `pk_S` and a KEM ciphertext `cd` as arguments. In the real version, the result of the actual AuthDecap function is returned, see lines L15 and R28.

The **try get** statement with the lookup in multiset $\mathcal{E}$ in L13 has its counterpart in the get statement in line R25, which looks up in table E. In E(=pk_S, =pkgen(s), =cd, k''), the equality tests =pk_S, =pkgen(s), and =cd indicate that we look for an element in E whose first 3 components are equal to the given values. The fourth component is a variable k'', which is bound to the fourth component of the found element of E, if such an element is found. If an element is found, OADecap returns the random key k'' found in the table E (R26). In case several elements match, the get statement chooses one at random, like the **try get** statement.

The CryptoVerif property expresses a bound on the probability that the adversary distinguishes the real from the ideal game; it leaves implicit the return of the bit b by which the adversary says with which of the two games it thinks it interacts.

If CryptoVerif concludes a proof, it displays the probability bound it could prove. In CryptoVerif's output, this bound is indicated in the line that starts with RESULT Proved. The proof output for this paper's theorems can be found in the files with filenames ending in .proof [2]. The probability bound depends on replication parameters (for example N, Qeperuser, Qdperuser), total number of calls to oracles, written #O for oracle O (for example #OAEncap, #OADecap), probability bounds of the assumptions used in the proof (for example Adv_PRF_KeySchedule and Adv_Outsider_CCA), and collision probabilities (for example P_pk_coll, which corresponds to P_{AKEM}, for AKEM public keys in our composition proofs, and P_hashcoll for the collision resistant hash function in the key schedule proof). Probability bounds like Adv_Outsider_CCA are expressed as functions with several arguments: the execution time of the adversary (indicated by time_*) and the numbers of queries. The execution time of the adversary is detailed in the lines below RESULT Proved.