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, New York, USA
jalwen@wickr.com
[2] Inria Paris, Paris, France
{bruno.blanchet,benjamin.lipp}@inria.fr
[3] Ruhr-Universität Bochum, Bochum, Germany
{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 [5]. 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 [4,30]. The primary source for HPKE is an RFC [5] (currently on draft 8) which lays out the details of the construction and provides some rough intuition for its security properties.

A. Canteaut and F.-X. Standaert (Eds.): EUROCRYPT 2021, LNCS 12696, pp. 87–116, 2021.
https://doi.org/10.1007/978-3-030-77870-5_4

At first glance the HPKE standard might be thought of as a "public key encryption" scheme in the spirit of the KEM/DEM paradigm [15]. That is, it combines a Key Encapsulation Mechanism (KEM) and an Authenticated Encryption with Associated Data (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 $\mathsf{IND\text{-}CCA}$-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 [31]. 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 [28] or the Curve25519 or Curve448 curves [25]. 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 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 IETF, 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) [4] secure group messaging protocol and the Encrypted Server Name Indication (ESNI) extension for TLS 1.3 [30]. 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.[1])

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 [29]). To invite a new member to an existing MLS protocol session the inviter must send an HPKE ciphertext to the new 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 (Sect. 4.4). 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 Sect. 4, by introducing *Authenticated Key Encapsulation Mechanisms* (AKEM) and *Authenticated Public Key*

[1] 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.

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
$\mathsf{Outsider\text{-}Auth}_{\mathsf{APKE}}$	X	X		X	
$\mathsf{Outsider\text{-}CCA}_{\mathsf{APKE}}$		X		X	X
$\mathsf{Insider\text{-}CCA}_{\mathsf{APKE}}$			X	X	X

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 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 $\mathsf{HPKE}_{\mathsf{Auth}}$.

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 $\mathsf{HPKE}_{\mathsf{Auth}}$ itself). Indeed, we give an attack in Sect. 4.4.

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 $\mathsf{HPKE}_{\mathsf{Auth}}$ Scheme. In Sect. 5 we analyse the generic $\mathsf{HPKE}_{\mathsf{Auth}}$ scheme proposed in the RFC. $\mathsf{HPKE}_{\mathsf{Auth}}$ is an instantiation of the AKEM/DEM paradigm discussed above.

Thus, we first analyse DH-AKEM, the particular AKEM underlying $\mathsf{HPKE}_{\mathsf{Auth}}$. 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 $\mathsf{HPKE}_{\mathsf{Auth}}$'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 $\mathsf{HPKE}_{\mathsf{Auth}}$ as an APKE. Our presentation ends with concrete bounds of $\mathsf{HPKE}_{\mathsf{Auth}}$'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 [13], which was already used to verify important practical protocols such as TLS 1.3 [12], Signal [22], and WireGuard [27]. 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 produced 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 [20]. 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 [31]; a family of primitives intended to efficiently combine signatures and public key encryption. Signcryption literature is substantial and we refer to the textbook [18] 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 [18] 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 [17] and insider-secure one of [16]. In contrast to our work, almost all security notions in [18] forbid honest parties from reusing the same key pair for both sending and receiving (even if sender and receiver keys have identical distribution).[2] 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 [9]. 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.

A detailed model of Curve25519 [25] in CryptoVerif was already presented in [27]; 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 [28] in a single model. Moreover, we handle rerandomisation of curve elements, which was not taken into account in [27].

A very preliminary version of this work analyses HPKE as a single protocol, not in a modular KEM/DEM setting [26]. 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 $h \overset{\$}{\leftarrow} \mathcal{S}$ 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.

[2] 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 [18].

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 [8]. 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 Elliptic Curves

In this section we introduce the elliptic curves relevant for the HPKE standard, P-256, P-384, P-521 [28], Curve25519 and Curve448 [25], together with relevant security assumptions.

3.1 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 1. *A nominal group* $\mathcal{N} = (\mathcal{G}, g, p, \mathcal{E}_H, \mathsf{exp})$ *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* $\mathcal{E}_H \subset \mathbb{Z}$*, 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) $g^{x+py} = g^x$ *for all* $x, y \in \mathbb{Z}$.

We remark that even though $\mathcal{G}$ is called the set of (group) elements, it is not required to form a group.

For a nominal group $\mathcal{N} = (\mathcal{G}, g, p, \mathcal{E}_H, \mathsf{exp})$ we let G_H be the distribution of honestly generated elements, that is, the distribution of g^x with $x \xleftarrow{\$} \mathcal{E}_H$. Let G_U be the distribution of g^x with $x \xleftarrow{\$} [1, p-1]$. Depending on the choice of $\mathcal{E}_H$, these distributions may differ. We define the two statistical parameters

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

We summarise the expected security level and the concrete upper bounds for $\Delta_{\mathcal{N}}$ and $P_{\mathcal{N}}$ in Table 2 of Sect. 5.3 and compute them below.

Prime-Order Groups. The simplest example of a nominal group is when $\mathcal{G} = \mathbb{G}$ is a prime-order group with generator g, exp is defined via the usual scalar multiplication on $\mathbb{G}$, and $\mathcal{E}_H = [1, p-1]$. The two distributions G_H and G_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 [28] are examples of prime-order groups.

Curve25519 and Curve448. We now show that Curve25519 and Curve448 [25] can also be seen as 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 [11, 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 $\{0, \dots, q-1\}$.) Finally, $X^y = \mathsf{encode\_pk}(y \cdot \mathsf{decode\_pk}(X))$.

As required by Definition 1, 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 [25], 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 $\mathcal{E}_H$.

The proof of the following Lemma 1 is in the long version [3].

Lemma 1. *For Curve25519, $\Delta_{\mathcal{N}} < 2^{-125}$ and $P_{\mathcal{N}} = 2^{-250}$, and for Curve448, $\Delta_{\mathcal{N}} < 2^{-220}$ and $P_{\mathcal{N}} = 2^{-444}$.*

3.2 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 $\mathcal{N} = (\mathcal{G}, g, p, \mathcal{E}_H, \mathsf{exp})$ of the previous section, by allowing elements in $\mathcal{G}$ as arguments of the Diffie-Hellman decision oracle. Moreover, we still choose secret keys in $[1, p-1]$, 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 2 (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*

$$\mathsf{Adv}^{\mathsf{GDH}}_{\mathcal{A},\mathcal{N}} := \Pr_{x,y \xleftarrow{\$} [1,p-1]} [Z = g^{xy} \mid Z \xleftarrow{\$} \mathcal{A}^{\mathrm{DH}}(g^x, g^y)]$$

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

Definition 3 (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*

$$\mathsf{Adv}^{\mathsf{sqGDH}}_{\mathcal{A},\mathcal{N}} := \Pr_{x \xleftarrow{\$} [1,p-1]} \left[Z = g^{x^2} \mid Z \xleftarrow{\$} \mathcal{A}^{\mathrm{DH}}(g^x)\right]$$

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

CryptoVerif cannot use cryptographic assumptions directly in this form: it requires assumptions to be formulated as computational indistinguishability axioms between a left game G_ℓ and a right game G_r. In order to use such assumptions, it 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.) 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 $[1, p-1]$.

Definition 4 (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*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{LoR}\text{-}(n,m)\text{-}\mathsf{GDH}}_{\mathcal{A},\mathcal{N}} := \Bigg| &\Pr_{\substack{\forall i\in[n]:\, x_i \xleftarrow{\$} \mathcal{E}_H \\ \forall j\in[m]:\, y_j \xleftarrow{\$} \mathcal{E}_H}} \left[\mathcal{A}^{\mathrm{DH}_\ell,\mathrm{DH}_0}(g^{x_1},\ldots,g^{x_n},g^{y_1},\ldots,g^{y_m}) \Rightarrow 1\right] \\ &- \Pr_{\substack{\forall i\in[n]:\, x_i \xleftarrow{\$} \mathcal{E}_H \\ \forall j\in[m]:\, y_j \xleftarrow{\$} \mathcal{E}_H}} \left[\mathcal{A}^{\mathrm{DH}_r,\mathrm{DH}_0}(g^{x_1},\ldots,g^{x_n},g^{y_1},\ldots,g^{y_m}) \Rightarrow 1\right]\Bigg| ,\end{aligned}$$

where DH_0 *is a decision oracle that on input* $(g^{\hat{x}}, Y, Z)$ *returns* 1 *iff* $Y^{\hat{x}} = Z$ *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 5 (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*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{LoR}\text{-}n\text{-}\mathsf{sqGDH}}_{\mathcal{A},\mathcal{N}} := \Bigg| &\Pr_{\forall i\in[n]:\, x_i \xleftarrow{\$} \mathcal{E}_H} \left[\mathcal{A}^{\mathrm{DH}_\ell,\mathrm{DH}_0}(g^{x_1}\ldots,g^{x_n}) \Rightarrow 1\right] \\ &- \Pr_{\forall i\in[n]:\, x_i \xleftarrow{\$} \mathcal{E}_H} \left[\mathcal{A}^{\mathrm{DH}_r,\mathrm{DH}_0}(g^{x_1},\ldots,g^{x_n}) \Rightarrow 1\right]\Bigg| ,\end{aligned}$$

where DH_0 *is a decision oracle that on input* $(g^{\hat{x}}, Y, Z)$ *returns* 1 *iff* $Y^{\hat{x}} = Z$ *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*.*

The proofs of Theorems 1 and 2 are in the long version [3].

Theorem 1 ($\mathsf{GDH} \Rightarrow \mathsf{LoR}$-$(n, m)$-$\mathsf{GDH}$)**.** *For any adversary $\mathcal{A}$ against* LoR-(n, m)-GDH*, there exists an adversary $\mathcal{B}$ against* GDH *such that*

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

$\mathcal{B}$ queries the DH *oracle as many times as $\mathcal{A}$ queries* DH_0, DH_ℓ*, or* DH_r*, and* $t_{\mathcal{B}} \approx t_{\mathcal{A}}$*.*

Theorem 2 ($\mathsf{sqGDH} \Rightarrow \mathsf{LoR}$-$n$-$\mathsf{sqGDH}$)**.** *For any adversary $\mathcal{A}$ against* LoR-n-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_0, DH_ℓ*, or* DH_r*, and* $t_{\mathcal{B}} \approx t_{\mathcal{A}}$*.*

In these theorems, the terms in $\Delta_{\mathcal{N}} = \Delta[\mathrm{G}_H, \mathrm{G}_U]$ come from the rerandomisation of keys, which yields keys distributed according to G_U, while the adversary expects keys distributed according to G_H. (Choosing secret keys in $\mathcal{E}_H$ in Definitions 2 and 3 would not avoid this term.)

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.

4 Authenticated Key Encapsulation and Public Key Encryption

In Sect. 4.1, we introduce notation and security notions for an authenticated key encapsulation mechanism (AKEM), namely Outsider-CCA, Insider-CCA and Outsider-Auth. In Sect. 4.2, 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 Insider-Auth security.

In Sect. 4.3, we show how to construct an APKE scheme which achieves Outsider-CCA, Insider-CCA and Outsider-Auth, from an AKEM, a pseudo-random function (PRF), and a nonce-based authenticated encryption with associated data (AEAD) scheme. For Insider-Auth, we give a concrete attack in Sect. 4.4.

4.1 Authenticated Key Encapsulation Mechanism

Definition 6 (AKEM). *An authenticated key encapsulation mechanism* AKEM *consists of three algorithms:*

- Gen *outputs a key pair (sk, pk), where pk defines a key space $\mathcal{K}$.*
- 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* 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. 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 1 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 2. 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 the long version [3, Appendix B], we compare the code-based game syntax with the CryptoVerif syntax for $\mathsf{Outsider\text{-}CCA}$.

In all games, we generate key pairs for n users and run the adversary on the public keys. In the $\mathsf{Outsider\text{-}CCA}$ games, the adversary has access to oracles AEncap and ADecap. AEncap takes as input an index specifying a sender, as well as an arbitrary public key specifying a 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. This models the adversary as an outsider and ensures that target ciphertexts from an honest sender to an honest receiver are secure, i. e. do not leak any information about the shared key. 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, the output is kept consistent with the output of AEncap.

In the $\mathsf{Insider\text{-}CCA}$ games, there is an additional challenge oracle Chall. The adversary gives 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.

In all games, the adversary makes 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. Furthermore, 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 3.

Listing 1: 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 q_e queries to AENCAP and at most q_d queries to ADECAP.

```
$(n, q_e, q_d)$-$\mathsf{Outsider\text{-}CCA}_\ell$ and
$(n, q_e, q_d)$-$\mathsf{Outsider\text{-}CCA}_r$
for $i \in [n]$
    $(sk_i, pk_i) \xleftarrow{\$} \mathsf{Gen}$
$\mathcal{E} \leftarrow \emptyset$
$b \xleftarrow{\$} \mathcal{A}^{\text{AEncap},\text{ADecap}}(pk_1, \dots, pk_n)$
return $b$

Oracle AEncap$(i \in [n], pk)$
$(c, K) \xleftarrow{\$} \mathsf{AuthEncap}(sk_i, pk)$
if $pk \in \{pk_1, \dots, pk_n\}$
    $K \xleftarrow{\$} \mathcal{K}$
    $\mathcal{E} \leftarrow \mathcal{E} \cup \{(pk_i, pk, c, K)\}$
return $(c, K)$

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

Listing 2: 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 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
$(n, q_e, q_d, q_c)$-$\mathsf{Insider\text{-}CCA}_r$
for $i \in [n]$
    $(sk_i, pk_i) \xleftarrow{\$} \mathsf{Gen}$
$\mathcal{E} \leftarrow \emptyset$
$b \xleftarrow{\$} \mathcal{A}^{\text{AEncap},\text{ADecap},\text{Chall}}(pk_1, \dots, pk_n)$
return $b$

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

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

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

Listing 3: 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 q_e queries to AENCAP and at most q_d queries to ADECAP.

```
$(n, q_e, q_d)$-$\mathsf{Outsider\text{-}Auth}_\ell$ and
$(n, q_e, q_d)$-$\mathsf{Outsider\text{-}Auth}_r$
for $i \in [n]$
    $(sk_i, pk_i) \xleftarrow{\$} \mathsf{Gen}$
$\mathcal{E} \leftarrow \emptyset$
$b \xleftarrow{\$} \mathcal{A}^{\text{AEncap},\text{ADecap}}(pk_1, \dots, pk_n)$
return $b$

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

Oracle ADecap$(j \in [n], pk, c)$
if $\exists K : (pk, pk_j, c, K) \in \mathcal{E}$
    return $K$
$K \leftarrow \mathsf{AuthDecap}(sk_j, pk, c)$
if $pk \in \{pk_1, \dots, pk_n\}$ and $K \neq \bot$
    $K \xleftarrow{\$} \mathcal{K}$
    $\mathcal{E} \leftarrow \mathcal{E} \cup \{(pk, pk_j, c, K)\}$
return $K$
```

The adversary has access to oracles AEncap and ADecap. 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 from a random key for two honest users, even if it can come up with the target ciphertext.

The adversary makes 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

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

In the long version [3, 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 Sect. 5.1.

4.2 Authenticated Public Key Encryption

Definition 7 (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)-$\mathsf{Outsider\text{-}CCA}$ and (n,q_e,q_d,q_c)-$\mathsf{Insider\text{-}CCA}$ in Listing 4, 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).

Listing 4: 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 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
$(n, q_e, q_d, q_c)$-Insider-CCA
for $i \in [n]$
    $(sk_i, pk_i) \xleftarrow{\$} \mathsf{Gen}$
$\mathcal{E} \leftarrow \emptyset$
$b \xleftarrow{\$} \{0,1\}$
$b' \xleftarrow{\$} \mathcal{A}^{\text{AEnc},\text{ADec},\text{Chall}}(pk_1, \ldots, pk_n)$
return $[\![b = b']\!]$

Oracle ADec$(j \in [n], pk, c, aad, info)$
if $(pk, pk_j, c, aad, info) \in \mathcal{E}$
    return $\bot$
$m \leftarrow \mathsf{AuthDec}(sk_j, pk, c, aad, info)$
return $m$
```

```
Oracle AEnc$(i \in [n], pk, m, aad, info)$
$c \xleftarrow{\$} \mathsf{AuthEnc}(sk_i, pk, m, aad, info)$
return $c$

Oracle Chall$(i \in [n], j \in [n], m_0, m_1, aad, info)$
if $|m_0| \neq |m_1|$ return $\bot$
$c \xleftarrow{\$} \mathsf{AuthEnc}(sk_i, pk_j, m_b, aad, info)$
$\mathcal{E} \leftarrow \mathcal{E} \cup \{(pk_i, pk_j, c, aad, info)\}$
return $c$

Oracle Chall$(j \in [n], sk, m_0, m_1, aad, info)$
if $|m_0| \neq |m_1|$ return $\bot$
$c \xleftarrow{\$} \mathsf{AuthEnc}(sk, pk_j, m_b, aad, info)$
$\mathcal{E} \leftarrow \mathcal{E} \cup \{(\mu(sk), pk_j, c, aad, info)\}$
return $c$
```

Listing 5: Games (n, q_e, q_d)-Outsider-Auth and (n, q_e, q_d)-Insider-Auth for APKE. Adversary $\mathcal{A}$ makes at most q_e queries to AEnc and at most q_d queries to ADec.

```
$(n, q_e, q_d)$-Outsider-Auth
for $i \in [n]$
    $(sk_i, pk_i) \xleftarrow{\$} \mathsf{Gen}$
$\mathcal{E} \leftarrow \emptyset$
$(i^*, j^*, c^*, aad^*, info^*) \xleftarrow{\$} \mathcal{A}^{\text{AEnc},\text{ADec}}(pk_1, \ldots, pk_n)$
return $[\![(pk_{i^*}, pk_{j^*}, c^*, aad^*, info^*) \notin \mathcal{E}$
    and $\mathsf{AuthDec}(sk_{j^*}, pk_{i^*}, c^*, aad^*, info^*) \neq \bot]\!]$

$(n, q_e, q_d)$-Insider-Auth
for $i \in [n]$
    $(sk_i, pk_i) \xleftarrow{\$} \mathsf{Gen}$
$\mathcal{E} \leftarrow \emptyset$
$(i^*, sk, c^*, aad^*, info^*) \xleftarrow{\$} \mathcal{A}^{\text{AEnc},\text{ADec}}(pk_1, ..., pk_n)$
return $[\![(pk_{i^*}, \mu(sk), c^*, aad^*, info^*) \notin \mathcal{E}$
    and $\mathsf{AuthDec}(sk, pk_{i^*}, c^*, aad^*, info^*) \neq \bot]\!]$
```

```
Oracle AEnc$(i \in [n], pk, m, aad, info)$
$c \xleftarrow{\$} \mathsf{AuthEnc}(sk_i, pk, m, aad, info)$
$\mathcal{E} \leftarrow \mathcal{E} \cup \{(pk_i, pk, c, aad, info)\}$
return $c$

Oracle ADec$(j \in [n], pk, c, aad, info)$
$m \leftarrow \mathsf{AuthDec}(sk_j, pk, c, aad, info)$
return $m$
```

Listing 6: Authenticated PKE scheme APKE[AKEM, KS, AEAD] construction from AKEM, KS and AEAD, where APKE.Gen = AKEM.Gen.

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

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

In these games, the adversary $\mathcal{A}$ makes 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

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

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

Authenticity. Furthermore, we define the games (n, q_e, q_d)-Outsider-Auth and (n, q_e, q_d)-Insider-Auth in Listing 5. 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 q_e queries to oracle AEnc and at most q_d queries to oracle ADec. The advantage of $\mathcal{A}$ is defined as

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

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

4.3 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 6) and an AEAD (c.f. [3, Section 3]). 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 6 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 6. Theorems 3 to 5 are proven using CryptoVerif version 2.04. 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 Sect. 4.4, we show that $\mathsf{APKE}[\mathsf{AKEM}, \mathsf{KS}, \mathsf{AEAD}]$ cannot achieve $\mathsf{Insider\text{-}Auth}$ security.

As detailed in the long version [3, 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 $\mathsf{IND\text{-}CPA}$ and $\mathsf{INT\text{-}CTXT}$ security experiments for the AEAD: n_k-$\mathsf{IND\text{-}CPA}$ and (n_k, q_d)-$\mathsf{INT\text{-}CTXT}$, with n_k keys, in which the adversary makes at most one encryption query for each key and, for the $\mathsf{INT\text{-}CTXT}$ 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 ⇒ APKE Outsider-CCA). *For any* (n, q_e, q_d, q_c)-$\mathsf{Outsider\text{-}CCA}$ *adversary* $\mathcal{A}$ *against* $\mathsf{APKE}[\mathsf{AKEM}, \mathsf{KS}, \mathsf{AEAD}]$*, 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 Insider-CCA + KS PRF + AEAD IND-CPA + AEAD INT-CTXT ⇒ APKE Insider-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 Outsider-CCA + AKEM Outsider-Auth + KS PRF + AEAD INT-CTXT ⇒ APKE Outsider-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* $(q_e + 3q_d + 3, 4q_d + 1)$-$\mathsf{INT\text{-}CTXT}$ *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*

$$\begin{aligned}\mathsf{Adv}^{(n,q_e,q_d)\text{-}\mathsf{Outsider\text{-}Auth}}_{\mathcal{A},\mathsf{APKE[AKEM,KS,AEAD]}} \leq\ & \mathsf{Adv}^{(n,q_e,q_d+1)\text{-}\mathsf{Outsider\text{-}CCA}}_{\mathcal{B}_1,\mathsf{AKEM}} + \mathsf{Adv}^{(n,q_e,q_d+1)\text{-}\mathsf{Outsider\text{-}Auth}}_{\mathcal{B}_2,\mathsf{AKEM}} \\ & + \mathsf{Adv}^{(q_e+q_d+1,q_e+2q_d+1)\text{-}\mathsf{PRF}}_{\mathcal{C},\mathsf{KS}} \\ & + \mathsf{Adv}^{(q_e+3q_d+3,4q_d+1)\text{-}\mathsf{INT\text{-}CTXT}}_{\mathcal{D},\mathsf{AEAD}} + n(q_e+13n)\cdot P_{\mathsf{AKEM}}\ .\end{aligned}$$

4.4 Infeasibility of Insider-Auth Security

For any AKEM, KS, and AEAD, the construction APKE[AKEM, KS, AEAD] given in Listing 6 is not (n, q_e, q_d)-Insider-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 (n, q_e, q_d)-*Insider-Auth *security of* APKE[AKEM, KS, AEAD] *such that*

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

Proof. We construct adversary $\mathcal{A}$ in Listing 7. It takes as input n public keys and has oracle access to AEnc and ADec. It first generates a key pair (sk^*, pk^*) and queries the AEnc oracle on any index i^*, receiver public key pk^*, an arbitrary message m_1, as well as arbitrary associated data *aad* and string *info*.

Listing 7: Adversary $\mathcal{A}$ against (n, q_e, q_d)-Insider-Auth as defined in Listing 5, of APKE[AKEM, KS, AEAD].

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

The challenger computes $(c_1, K) \xleftarrow{\$} \mathsf{AuthEncap}(sk_{i^*}, pk^*)$, $(k, nonce) \leftarrow \mathsf{KS}(K, info)$ 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 (n, q_e, q_d)-Insider-Auth security experiment. □

Listing 8: $\mathsf{DH\text{-}AKEM}[\mathcal{N}, \mathsf{KDF}] = (\mathsf{Gen}, \mathsf{AuthEncap}, \mathsf{AuthDecap})$ as defined in the RFC [5], 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$(dh, context)$
```

5 The HPKE Standard

In Sect. 5.1, 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 Sect. 5.2, we define and analyse HPKE's specific key schedule $\mathsf{KS}_{\mathsf{Auth}}$ and key derivation function HKDF_N. Finally, in Sect. 5.3 we put everything together and obtain the HPKE standard in Auth mode from all previous sections.

5.1 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 8 shows the formal definition of $\mathsf{DH\text{-}AKEM}[\mathcal{N}, \mathsf{KDF}]$ relative to a nominal group $\mathcal{N}$ (c.f. Definition 1) and a key derivation function $\mathsf{KDF} : \{0,1\}^* \to \mathcal{K}$, 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 the KEM algorithm. The bitstring $\mathsf{Encode}(N)$ is the two-byte encoding of the length N expressed in bytes. Correctness follows by property (1) of Definition 1. 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 8.

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 $\mathsf{DH\text{-}AKEM}[\mathcal{N}, \mathsf{KDF}]$, modelling KDF as a random oracle. The proofs are written with CryptoVerif version 2.04; 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 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 27.

At the end of this section, we sketch the attack against the $\mathsf{Insider\text{-}Auth}$ security.

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

$$\begin{aligned}\mathsf{Adv}^{(n,q_e,q_d)\text{-}\mathsf{Outsider\text{-}CCA}}_{\mathcal{A},\mathsf{DH\text{-}AKEM}[\mathcal{N},\mathsf{KDF}]} &\leq \mathsf{Adv}^{\mathsf{GDH}}_{\mathcal{B},\mathcal{N}} + (n + q_e) \cdot \Delta_{\mathcal{N}} \\ &\quad + (q_e q_d + 2nq_e + 7q_e^2 + 13n^2) \cdot P_{\mathcal{N}}\end{aligned}$$

$\mathcal{B}$ *issues* $nq_e + nq_d + 2q_d q_h + 3nq_h$ *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 4, 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 $pk \in \{pk_1, \ldots, pk_n\}$ and when $pk \notin \{pk_1, \ldots, pk_n\}$, 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 $pk \in \{pk_1, \ldots, pk_n\}$ cannot coincide with arguments of other calls. Hence, they return a fresh random key, as in (n, q_e, q_d)-Outsider-CCA$_r$.

Starting from (n, q_e, q_d)-Outsider-CCA$_r$, in the random oracle, we distinguish whether the argument received from the adversary has a format that matches the one used by DH-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)-Outsider-CCA$_\ell$ and from (n, q_e, q_d)-Outsider-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)-Outsider-CCA$_\ell$ and (n, q_e, q_d)-Outsider-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 guide the tool by giving estimates for n, $q_e^{per\ user}$, $q_d^{per\ user}$, q_h, $P_{\mathcal{N}}$, where $q_e^{per\ user}$ and $q_d^{per\ user}$ are the number of AEncap and ADecap queries respectively, per user. We also give a maximum probability for which we allow eliminating collisions. Our estimates are such that we allow eliminating collisions of probability $P_{\mathcal{N}}$ times a cubic factor in n, $q_e^{per\ user}$, and $q_d^{per\ user}$, but do not allow eliminating collisions with more than a cubic factor in n, $q_e^{per\ user}$, and $q_d^{per\ user}$, nor collisions that involve q_h. These estimates are used only to decide whether to eliminate collisions. The obtained probability formula is then valid even if the actual numbers do not match the given estimates.

The probability formula computed by CryptoVerif involves both the total numbers of queries q_e, q_d and the number of queries per user $q_e^{per\ user}$, $q_d^{per\ user}$. For simplicity, we upper bound $q_e^{per\ user}$ by q_e and $q_d^{per\ user}$ by q_d, yielding the formula given in the theorem. □

Theorem 8 (**Insider-CCA security of DH-AKEM**). *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*

$$\begin{aligned}\mathsf{Adv}^{(n,q_e,q_d,q_c)\text{-Insider-CCA}}_{\mathcal{A},\mathsf{DH\text{-}AKEM}[\mathcal{N},\mathsf{KDF}]} \leq\ & \mathsf{Adv}^{\mathsf{GDH}}_{\mathcal{B},\mathcal{N}} + (n + q_c) \cdot \Delta_{\mathcal{N}} \\ & + (2q_eq_d + q_cq_d + q_cq_e + 2nq_e + 7q_e^2 + 2q_c^2 + 17n^2) \cdot P_{\mathcal{N}}\end{aligned}$$

$\mathcal{B}$ *makes* $nq_e + 2q_cq_e + 2q_dq_h + 3nq_h$ *queries to the* DH *oracle, and* $t_{\mathcal{B}} \approx t_{\mathcal{A}}$.

Theorem 9 (**Outsider-Auth security of DH-AKEM**). *Under the* sqGDH *assumption in* $\mathcal{N}$ *and modelling* KDF *as a random oracle,* DH-AKEM$[\mathcal{N}, \mathsf{KDF}]$ *is*

$\mathsf{Outsider\text{-}Auth}$ *secure. In particular, for any* (n, q_e, q_d)-$\mathsf{Outsider\text{-}Auth}$ *adversary* $\mathcal{A}$ *against* $\mathsf{DH\text{-}AKEM}_{\mathcal{N}}$ *that issues at most* q_h *queries to the random oracle, there exists an adversary* $\mathcal{B}$ *against* sqGDH *such that*

$$\begin{aligned}\mathsf{Adv}^{(n,q_e,q_d)\text{-}\mathsf{Outsider\text{-}Auth}}_{\mathcal{A},\mathsf{DH\text{-}AKEM}[\mathcal{N},\mathsf{KDF}]} &\le 2\mathsf{Adv}^{\mathsf{sqGDH}}_{\mathcal{B},\mathcal{N}} + 2(n+q_e)\cdot \varDelta_{\mathcal{N}} \\ &+ (q_e q_d + 4nq_d + 12q_e^2 + 4nq_e + 20n^2)\cdot P_{\mathcal{N}}\end{aligned}$$

$\mathcal{B}$ *issues* $nq_e + nq_d + 4q_dq_h + 3nq_h$ *queries to the* DH *oracle, and* $t_{\mathcal{B}} \approx t_{\mathcal{A}}$.

Infeasibility of $\mathsf{Insider\text{-}Auth}$ 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 $\mathsf{DH\text{-}AKEM}$ 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 $\mathsf{DH\text{-}AKEM}$, 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.

5.2 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 $\mathsf{Extract}$ 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 $\mathsf{DH\text{-}AKEM}$. Finally, we indicate how the entire $\mathsf{HPKE}_{\mathsf{Auth}}$ scheme is assembled from the individual building blocks presented in the previous sections.

$\mathsf{Extract}$ and Expand. The RFC defines two functions $\mathsf{Extract}$ and Expand as follows.

- $\mathsf{Extract}(salt, IKM)$ 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.
- $\mathsf{Expand}(PRK, info, L)$ 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 $\mathsf{Extract}$ and Expand are PRFs with the first parameter being the PRF key. HPKE instantiates $\mathsf{Extract}$ and Expand with HMAC-SHA-2, for which the PRF assumption is justified by [6,7]. (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 $\mathsf{Extract}$ 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 [21].

Listing 9: The key schedule $\mathsf{KS}_{\mathsf{Auth}}$ used in $\mathsf{HPKE}_{\mathsf{Auth}}$ [5].

```
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)
```

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 9. The function $\mathsf{KeySchedule}$ 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 $\mathsf{LabeledExtract}$ and $\mathsf{LabeledExpand}$ as wrappers around $\mathsf{Extract}$ 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* $\mathsf{Extract}$ *is a collision-resistant hash function for calls with the labels* `"psk_id_hash"` *and* `"info_hash"`*, that* $\mathsf{Extract}$ *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* $\mathsf{Extract}$*, a* (n_k, n_k)-PRF *adversary* $\mathcal{C}_1$ *against* $\mathsf{Extract}$*, and a* $(n_k, 2q_{\mathsf{PRF}})$-PRF *adversary* $\mathcal{C}_2$ *against* Expand *such that* $t_{\mathcal{B}} \approx t_{\mathcal{A}}$*,* $t_{\mathcal{C}_1} \approx t_{\mathcal{A}}$*,* $t_{\mathcal{C}_2} \approx t_{\mathcal{A}}$*, 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 8 uses a function KDF to derive the KEM shared secret. In $\mathsf{HPKE}_{\mathsf{Auth}}$, this function is instantiated by HKDF_N, as defined in Listing 10, 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 [14]. While in principle the PRF-ODH assumption is weaker than the random oracle model, Brendel et al. [14] 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 Sect. 5, HPKE instantiates Extract and Expand with HMAC [23], which makes HKDF_N exactly the widely-used HKDF key derivation function [24]. HPKE specifies SHA-2 as the hash function underlying HMAC. Lemma 6 in [27] shows that HKDF is indifferentiable from a random oracle under the following assumptions[3]: (1) HMAC is indifferentiable from a random oracle. For HMAC-SHA-2, this is justified by Theorem 4.4 in [19] 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* || 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 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.

5.3 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 6 to the $\mathsf{DH\text{-}AKEM}[\mathcal{N}, \mathsf{HKDF}_N]$ authenticated KEM (Listing 8), where $\mathcal{N}$ is a nominal group. For the key schedule of $\mathsf{HPKE}_{\mathsf{Auth}}$ we use $\mathsf{KS}_{\mathsf{Auth}}$ of Listing 9 and for the key derivation function we use HKDF_N of Listing 10. For both

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

Listing 10: 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 $\mathsf{Expand}(PRK, info, N)$
```

$\mathsf{KS}_{\mathsf{Auth}}$ and HKDF_N we implement the $\mathsf{Extract}$ 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 $\mathsf{DH\text{-}AKEM}$'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 $q := q_e + q_d + q_c$.

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

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

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.

Table 2. Parameters of $\mathsf{DH\text{-}AKEM}[\mathcal{N}, \mathsf{HKDF}_N]$ depending on the choice of the 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^{-125}	2^{-220}
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

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** $\mathsf{Adv}^{\mathsf{GDH}}_{\mathcal{B}_1,\mathcal{N}}$. 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 $(n + q) \cdot \Delta_{\mathcal{N}} \approx (n + q)2^{-\kappa_{\mathcal{N}}}$. As $n + q \leq t_{\mathcal{A}}$, 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 $\mathsf{Adv}^{(q,q)\text{-}\mathsf{PRF}}_{\mathcal{C}_3,\mathsf{Expand}} \leq q^2/2^{N_h}$. 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 $N/2 \geq \kappa_{\mathcal{N}}$ 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. [3, Section 3]), studied for instance in [10]. 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 $q \leq 2^{N_n}$). 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 $q \leq 2^{N_n}$. 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 $t_{\mathcal{A}} \leq 2^{\kappa}$ and $q \leq 2^{N_n}$. 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.

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 DFG Cluster of Excellence 2092 CASA. Benjamin Lipp was supported by ERC CIRCUS (grant agreement n° 683032) and ANR TECAP (decision number ANR-17-CE39-0004-03). Doreen Riepel was supported by the Cluster of Excellence 2092 CASA.

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 (2001). https://doi.org/10.1007/3-540-45353-9_12

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
3. Alwen, J., Blanchet, B., Hauck, E., Kiltz, E., Lipp, B., Riepel, D.: Analysing the HPKE standard. Cryptology ePrint Archive, Report 2020/1499 (2020). https://eprint.iacr.org/2020/1499
4. 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, March 2020. https://tools.ietf.org/html/draft-ietf-mls-protocol-09
5. Barnes, R.L., Bhargavan, K., Lipp, B., Wood, C.A.: Hybrid Public Key Encryption. Internet-Draft draft-irtf-cfrg-hpke-08, IETF Secretariat, October 2020. https://tools.ietf.org/html/draft-irtf-cfrg-hpke-08
6. Bellare, M.: New proofs for NMAC and HMAC: security without collision resistance. J. Cryptol. **28**(4), 844–878 (2015)
7. Bellare, M., Canetti, R., Krawczyk, H.: Keying hash functions for message authentication. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 1–15. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68697-5_1
8. Bellare, M., Rogaway, P.: Code-based game-playing proofs and the security of triple encryption. Cryptology ePrint Archive, Report 2004/331 (2004). http://eprint.iacr.org/2004/331
9. 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, Cham (2020). https://doi.org/10.1007/978-3-030-45727-3_17
10. 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 (2016). https://doi.org/10.1007/978-3-662-53018-4_10
11. 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 (2006). https://doi.org/10.1007/11745853_14
12. 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
13. Blanchet, B.: A computationally sound mechanized prover for security protocols. IEEE Trans. Dependable Secure Comput. **5**(4), 193–207 (2008)
14. 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, Cham (2017). https://doi.org/10.1007/978-3-319-63697-9_22
15. Cramer, R., Shoup, V.: Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. SIAM J. Comput. **33**(1), 167–226 (2003)
16. Dent, A.W.: Hybrid signcryption schemes with insidersecurity. In: Boyd, C., González Nieto, J.M. (eds.) ACISP 2005. LNCS, vol. 3574, pp. 253–266. Springer, Heidelberg (2005). https://doi.org/10.1007/11506157_22
17. 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 (2005). https://doi.org/10.1007/11556992_15

18. Dent, A.W., Zheng, Y. (eds.): Practical Signcryption. Information Security and Cryptography. Springer, HeidelbergHeidelberg (2010). https://doi.org/10.1007/978-3-540-89411-7
19. 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). http://eprint.iacr.org/2013/382
20. Gayoso Martínez, V., Alvarez, F., Hernandez Encinas, L., Sánchez Ávila, C.: A comparison of the standardized versions of ECIES. In: 2010 6th International Conference on Information Assurance and Security, IAS 2010, August 2010
21. 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 (2004). https://doi.org/10.1007/978-3-540-24654-1_13
22. 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, April 2017
23. Krawczyk, H., Bellare, M., Canetti, R.: HMAC: Keyed-hashing for message authentication. RFC 2104, RFC Editor, February 1997. https://www.rfc-editor.org/rfc/rfc2104.html
24. 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
25. Langley, A., Hamburg, M., Turner, S.: Elliptic curves for security. RFC 7748, RFC Editor, January 2016. https://www.rfc-editor.org/rfc/rfc7748.html
26. Lipp, B.: An analysis of hybrid public key encryption. Cryptology ePrint Archive, Report 2020/243 (2020). https://eprint.iacr.org/2020/243
27. 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, Stockholm, Sweden, pp. 231–246. IEEE Computer Society, June 2019. https://hal.inria.fr/hal-02100345
28. National Institute of Standards and Technology: Digital Signature Standard (DSS). FIPS Publication 186-4, July 2013. https://doi.org/10.6028/nist.fips.186-4
29. 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, July 2020. https://tools.ietf.org/html/draft-ietf-mls-architecture-05
30. Rescorla, E., Oku, K., Sullivan, N., Wood, C.A.: TLS Encrypted Client Hello. Internet-Draft draft-ietf-tls-esni-07, IETF Secretariat, June 2020. https://tools.ietf.org/html/draft-ietf-tls-esni-07
31. Zheng, Y.: Digital signcryption or how to achieve cost(signature & encryption) $\ll$ cost(signature) + cost(encryption). In: Kaliski, B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 165–179. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0052234