A shortened version of this paper appears in the proceedings of the *44th Annual International Conference on the Theory and Applications of Cryptographic Techniques (Eurocrypt 2025)*.

Key Derivation Functions Without a Grain of Salt

Matilda Backendal[1], Sebastian Clermont[2], Marc Fischlin[2], and Felix Günther[3]

[1]Department of Computer Science, ETH Zurich, Zurich, Switzerland
[2]Cryptoplexity, Technische Universität Darmstadt, Darmstadt, Germany
[3]IBM Research Europe – Zurich, Rüschlikon, Switzerland

mbackendal@inf.ethz.ch, {sebastian.clermont,marc.fischlin}@tu-darmstadt.de, mail@felixguenther.info

April 10, 2025

Abstract

Key derivation functions (KDFs) are integral to many cryptographic protocols. Their functionality is to turn raw key material, such as a Diffie–Hellman secret, into a strong cryptographic key that is indistinguishable from random. This guarantee was formalized by Krawczyk together with the seminal introduction of HKDF (CRYPTO 2010), in a model where the KDF only takes a single key material input. Modern protocol designs, however, regularly need to combine multiple secrets, possibly even from different sources, with the guarantee that the derived key is secure as long as at least one of the inputs is good. This is particularly relevant in settings like hybrid key exchange for quantum-safe migration. Krawczyk's KDF formalism does not capture this goal, and there has been surprisingly little work on the security considerations for KDFs since then.

In this work, we thus revisit the syntax and security model for KDFs to treat multiple, possibly correlated inputs. Our syntax is assertive: We do away with salts, which are needed in theory to extract from arbitrary sources in the standard model, but in practice, they are almost never used (or even available) and sometimes even misused, as we argue. We use our new model to analyze real-world multi-input KDFs—in Signal's X3DH protocol, ETSI's TS 103 744 standard, and MLS' combiner for pre-shared keys—as well as new constructions we introduce for specialized settings—e.g., a purely blockcipher-based one. We further discuss the importance of collision resistance for KDFs and finally apply our multi-input KDF model to show how hybrid KEM key exchange can be analyzed from a KDF perspective.

1 Introduction

Key derivation functions (KDFs) are a common ingredient of many cryptographic applications. They operate as a versatile tool, taking highly entropic sources and generating an almost infinite amount of strong cryptographic keys, potentially bound to some context. KDFs are used to derive keys in many widely-used protocols such as TLS, IPsec, Bluetooth, or Signal. Several standards describe KDFs based on common cryptographic algorithms and deployed in many widely-used protocols. At the forefront, this includes the IETF RFC 5869 [46] for HKDF, based on the seminal work of Krawczyk [48] which describes how HMAC can be used to build a KDF. Equally important are NIST's special publications concerning KDFs, notably SP800-56C [6] on key derivation in key establishment schemes, in particular using hash functions, HMAC, and KMAC, as well as SP800-108 [24] concerning KDFs based on pseudorandom functions like HMAC, CMAC, and the hash function SHA-3.

1.1 State-of-the-Art KDFs

HKDF AND THE EXTRACT-THEN-EXPAND PARADIGM. The most commonly used key derivation function in practice is arguably the HMAC-based function HKDF, proposed by Krawczyk at CRYPTO 2010 [48] and subsequently standardized by the IETF in RFC 5869 [46]. HKDF follows the *extract-then-expand* (**XtX**) paradigm, also by Krawczyk, which suggests that key derivation functions should operate in two

stages: The first stage *extracts* uniform randomness from the (entropic, but not necessarily uniform) input key material, generating an intermediate pseudorandom key PRK. The second stage takes the key PRK and *expands* it using a variable-output-length PRF to generate a key of the desired length.

In the case of HKDF, the extract (HKDF.Extract) and expand (HKDF.Expand) modules are both based on HMAC [9]. HMAC: $\{0,1\}^* \times \{0,1\}^* \to \{0,1\}^{8hl}$, as standardized in [45, 53], takes an arbitrary-length key $K \in \{0,1\}^*$ and input $x \in \{0,1\}^*$ and applies a (nested) computation of a Merkle–Damgård hash function H: $\{0,1\}^* \to \{0,1\}^{8hl}$ to obtain an hl-byte output.[1] The extract function HKDF.Extract is defined as

$$\mathsf{HKDF.Extract}(S, \sigma) := \mathsf{HMAC}(S, \sigma),$$

where σ is the source key material and S is a salt value. The output is a pseudorandom key PRK of length hl. The expansion function HKDF.Expand takes as input PRK, a label L for the derived key, and an output length ℓ as a value up to $255 \cdot hl$. The evaluation $\mathsf{HKDF.Expand}(PRK, L, \ell)$ iterates HMAC, computing

$$\begin{aligned} &T(0) \leftarrow \varepsilon, \\ &T(i) \leftarrow \mathsf{HMAC}(PRK, T(i-1)||L||\,[i]_1) \quad \text{for } i = 1, 2, \ldots, N = \lceil \ell/hl \rceil. \end{aligned}$$

The counter value i is encoded as a single byte, allowing at most 255 iterations. The procedure eventually outputs the first ℓ bytes of $T(1)\|T(2)\|\ldots\|T(N)$. Finally, HKDF is defined as $\mathsf{HKDF}(\sigma, S, L, \ell) := \mathsf{HKDF.Expand}(\mathsf{HKDF.Extract}(S, \sigma),\, L,\, \ell)$.

Extendable output, or not. Note that the output length parameter ℓ does not enter the HMAC calls but only determines how the $T(i)$ values are truncated. This means that HKDF is a so-called *extendable-output function* (XOF). That is, if one fixes the inputs σ, L, S, then the output for some length parameter ℓ_1 is a prefix of the output for length parameter $\ell_2 \geq \ell_1$. We call this type of KDF a XOF-KDF to distinguish it from KDFs that produce independent keys for distinct length parameters (which we instead refer to as *not output-extendable*, NOF-KDFs); we treat both types in this work.

Single-input KDF syntax and security. Together with the design of HKDF and the **XtX** paradigm, Krawczyk also formalized the syntax of a key derivation function and established its expected security. As can be seen in the function signature of HKDF, a (single-input) KDF in Krawczyk's syntax takes as input the raw key material σ generated by some randomness source Σ, a salt S used for smoothing the raw key material, a label L binding key-related information (and possibly context about the input key material) to the derived key, and finally the desired output length ℓ:

$$K \leftarrow \mathrm{KDF}(\sigma, S, L, \ell).$$

For example, σ could be a Diffie–Hellman secret, S some independent random nonce, and L a string like `"c hs traffic"` or `"c ap traffic"` as used in TLS 1.3 [57] for computing the client handshake resp. application traffic secret.

Krawczyk's security model in [48] considers an indistinguishability game in which the adversary has oracle access to a single KDF instance for random, but fixed, key material σ and salt S, on labels L and length ℓ values of its choice. That is, the game samples σ and S, and the adversary can then adaptively query $\mathrm{KDF}(\sigma, S, \cdot, \cdot)$ on arbitrary label L and length ℓ and learn the derived key. For a single challenge query on a label distinct from those in all prior and later queries, the adversary either gets the real KDF value on the chosen inputs or a random string of the requested length and has to distinguish which it is.

Key combiners. Following Krawczyk's initial work, most formalizations of KDFs—including standards [24, 46]—cover single-input KDFs: the KDF takes a single key material input σ to derive the key. In practice, however, many protocols deal with multiple key material inputs $\sigma_1, \ldots, \sigma_n$ when deriving keys, possibly from different sources. This includes the Transport Layer Security (TLS) protocol in version 1.3 [57], where cryptographic keys may be derived by combining a pre-shared key (established in a previous connection) and a fresh Diffie–Hellman (DH) secret. Similarly, Signal's X3DH key exchange [49] derives its key from three to four correlated DH secrets computed from several ephemeral and static DH shares. In the Messaging Layer Security (MLS) protocol [7], an arbitrary number n of pre-shared keys can be injected into the key schedule (e.g., to add entropy or authenticate members of prior epochs) and for that purpose are first combined into a single key value using HKDF extraction and expansion.

Combining keys in this fashion aims to achieve security for the derived keys as long as at least one input key is secure. Yet, no existing formalism for KDFs with multiple key material inputs captures

[1] HKDF and HMAC interpret lengths in bytes (octets), hence the factor 8 here.

this security goal. Hence, all of these protocols resort to using a single-input KDF, applied either to the concatenation $\sigma_1, \ldots, \sigma_n$ of the key material inputs or by repeatedly feeding key material into both the salt and key input in a chaining fashion. Since neither of these approaches follows the intended usage of the KDF, each construction necessitates a tailored security analysis, and a formal definition of the joint security goal remains lacking.

The need for a thorough formal understanding of multi-input KDFs becomes even more pressing with the quantum-safe transition and the increasing deployment of hybrid protocols that combine keys from classical and post-quantum schemes. Advancements in quantum key distribution (QKD) may further increase the urgency, as a hybrid deployment of QKD with both post-quantum and classical key exchange protocols might be desirable for risk-hedging purposes. This development is accompanied by the appearance of the first standards on KDFs for these new scenarios, such as TS 103 744 [36, 37] of the European Telecommunications Standards Institute (ETSI), which so far lack a matching, formal definition.

This is the gap that we aim to fill. We begin by defining the syntax and security of multi-input KDFs, thereby providing a formal underpinning for the key combiners appearing in deployed systems and recent standards. We use our definition to analyze these practical constructions, providing proofs and exposing shortcomings, and devise new constructions targeting dedicated usage scenarios. Finally, looking at applications, we discuss how the security of a hybrid key exchange can be viewed through the lens of multi-input KDFs.

1.2 Our Contributions

Multi-input KDFs. Our first contribution is to extend the syntax of a KDF. Instead of only taking a single key material input, a KDF now accepts n key material inputs $\{\sigma_i\}_{i\in[1,n]}$, each associated with some context information c_i.[2] We call such a function an "n-KDF", to highlight the multiple inputs. As before, an n-KDF also takes a label L and a length parameter ℓ specifying the intended usage and desired length of the derived key, respectively:

$$K \leftarrow n\text{-KDF}((\sigma_1, c_1), \ldots, (\sigma_n, c_n), L, \ell).$$

Note that for single-input KDFs, the distinction between context information c and the label L is often blurred. Some standards, like NIST SP800-108 [24], make both explicit, but then simply concatenate c and L in all proposed constructions. RFC 5869 [46], in contrast, specifies HKDF only with an `info` field for capturing both context and label data, following [48]. For n-KDFs, the separation between context and label is clearer: context is associated with the individual source key material. Hence, there are n separate such inputs, whereas the label L is associated with the derived key; hence, there is only a single such input regardless of the value of n.

The attentive reader will have noticed the absence of salt in the inputs to the n-KDF. This is on purpose and our second contribution to the syntax of KDFs. As it turns out, almost without exception, practical approaches to building multi-input KDFs—as well as applications of single-input KDFs—*do not use salts at all*. Instead, if a single-input KDF is used as a building block, the salt input field is simply populated with a fixed constant, or one of the multiple key inputs. For some prominent examples, we refer to Hybrid Public-Key Encryption (HPKE) [8], iMessage PQ3 [63], MLS [7], Noise [54], Signal's X3DH [49] and PQXDH [50] handshakes, TLS 1.3 [57] and its hybrid key exchange draft [64], and XWing [5].[3] This is not due to ignorance of the concepts of salts in extractor theory, but due to practical limitations. A random and independent salt can be unavailable, primarily due to latency or communication constraints. In these cases demanding a salt input may be outright dangerous: The multi-input KDF construction suggested by ETSI in version 1.1.1 of TS 103 744 [36] turns out to be *insecure* when used as a general KDF without restrictions, due to a misuse of labels as salts.

In some multi-input KDFs, extraction is handled in a separate step (e.g., via privacy amplification of a reconciled key in QKD) before inputs are combined. In either case, salts, in practice, are not used in the construction or for security of multi-input KDFs, so we take this opportunity to remove them. This both simplifies the syntax and exposes the true security achieved by practical constructions.

Security model. The goal of a multi-input KDF remains similar to that of a single-input KDF: to generate keys of the desired length, which are computationally indistinguishable from random. In

[2] Context here refers to information associated to the *generation* of the source key material, such as public parameters or the transcript of a key exchange. Generally, including as much context as possible in the KDF evaluation is advisable.

[3] Indeed, the only example of a widely deployed protocol actually using salts in a KDF is the Internet Key Exchange (IKE) version 2 protocol [43] underlying IPsec.

contrast to single-input KDFs, this guarantee is now extended to encompass keys derived from partially adversarially-controlled inputs. A secure n-KDF *combines* the entropy of the inputs and returns a key that is indistinguishable from random as long as one of the n key material inputs is unknown to the adversary.

In our new KDF security notion (Section 4), we capture this by distinguishing between "honest" key material inputs, which contain entropy, and "dishonest" inputs chosen by the adversary. The adversary can ask to see multiple challenge outputs of the n-KDF, for chosen labels L and output lengths ℓ, and should not be able to distinguish the outputs from random, under the condition that in each key derivation, at least one key material input is honest. The adversary may also ask for real outputs from the KDF—however, we show that this does not add to security; the notions with and without such queries are logically and tightly equivalent. Our model further distinguishes between extendable (XOF) and non-extendable (NOF) output KDFs, capturing the expected security for both types, via appropriate query restrictions.

The honest key material inputs, also referred to as "secrets", are generated by a collection $\mathbf{\Sigma}$ of sources, where each source $\Sigma_i \in \mathbf{\Sigma}$ outputs a vector of potentially correlated secrets σ_i with some context information c_i, as well as some leakage information α_i which is handed to the adversary. This allows us to consider, for example, related Diffie–Hellman secrets $g^{x_i x_j}$, arising from pairwise combinations from a collection of DH shares $\{g^{x_1}, \ldots, g^{x_n}\}$.

Analyses of real-world proposals. We then turn to study real-world constructions of multi-input KDFs in our security model in Section 5. To illustrate the versatility of our model, we study three constructions with distinct internal designs, input key material settings, and output behaviors.

The first construction captures the approach used in MLS [7] to combine a sequence of n pre-shared keys into one, fixed-length key. This scheme, which we call MLS-PSK-KDF, uses a chain of HKDF extraction and expansion steps. We show that it is a secure n-KDF in our model when used to combine uniform pre-shared keys. The proof only relies on the standard-model assumptions that HMAC (underlying HKDF) is a dual-PRF [3] and collision resistant.

The second construction, which we call ETSI-CatKDF, is proposed by ETSI in their TS 103 744 standard; of which we analyze version 1.1.1 [36]. ETSI-CatKDF concatenates the key material inputs and applies HKDF to derive multiple keys (XOF style). We first identify that this proposal is *insecure* when used as a general, unrestricted KDF in the sense of our model. The reason is a misuse of labels as salt inputs to the extraction step HKDF.Extract, allowing an adversary to produce identical outputs by choosing different labels appropriately. If we restrict the choice of labels and assume that all key material inputs have fixed length—which the standard does not enforce in version 1.1.1—then we can prove ETSI-CatKDF secure as a XOF-KDF, modeling HMAC as a random oracle [33]. Notably, the recently published version 1.2.1 of the standard [37] restricts the choice of labels; a similar revision is under consideration for version 1.1.1 [36] following discussions of our results with ETSI. Furthermore, the key material from hybrid key exchanges targeted by ETSI TS 103 744 is fixed-length in nature. Arguably, the ETSI design could be strengthened to achieve general-purpose multi-input KDF security. Instead, the revised version 1.2.1 restricts ETSI-CatKDF to $2 + 1$ key inputs (compared of $n + 1$ in version 1.1.1 which we analyze), barring it from use as a general purpose n-KDF.

The third construction, used in Signal's X3DH [49] key exchange, derives a single key from 3 or 4 correlated Diffie–Hellman secrets using (plain) concatenation without context and HKDF. We show that the corresponding X3DH-KDF scheme is a secure 4-KDF (with the fourth key material input being optional) via a somewhat lossy reduction to the security of the underlying correlated Diffie–Hellman source, modeling HMAC as a random oracle. Notably, we can show that the loss in the reduction could be reduced significantly if X3DH-KDF were to include *context* (i.e., the involved DH public keys) in the key derivation—which it currently does not.[4]

The Combine-then-Expand paradigm. From our analyses of real-world constructions of multi-input KDFs, a general pattern emerges. We call it the "Combine-then-Expand" (**CtX**) paradigm, where—in accordance with existing designs—the extract step is replaced by a generic combine step, which may or may not involve extraction. **CtX** can be seen as a counterpart to the Extract-then-Expand (**XtX**) paradigm for multiple inputs. In brief, the **CtX** paradigm constructs an n-KDF as follows, illustrated in Figure 1: First, the source key material and context is combined into a single, fixed-length, pseudorandom key PRK via a Combine function. Then, PRK is used to key an Expand function, taking the label L and desired output length ℓ as input to produce the output key K.

[4] This has been assumed in a tight security analysis of a variant of X3DH by Kiltz et al. [44] and was suggested for independent reasons in analyses of Signal's post-quantum extension PQXDH [12, 38].

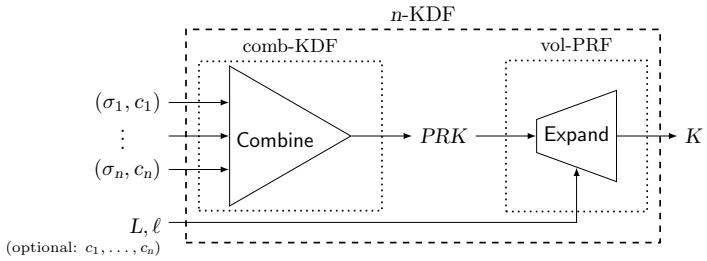

Figure 1: Combine-then-Expand (**CtX**) paradigm for building an n-KDF from a comb-KDF Combine (with *empty* label and *fixed* output length) and a variable–output-length PRF Expand.

In Section 6 we show that the **CtX** paradigm yields a secure n-KDF if the Combine function is a secure "combiner n-KDF" (comb-KDF) with *fixed, empty* label and *fixed* output length $|PRK|$ and the Expand function is a secure variable-output-length PRF. Observing that a comb-KDF with empty label and fixed output length is a notably simpler object to construct (given that it has only to produce a single, fixed-length output key), the **CtX** paradigm provides a blueprint for constructing multi-input KDFs from simpler building blocks. In particular, candidate variable-output-length PRFs for the Expand step readily exist, with HKDF.Expand being a prime example that practical constructions already use in this paradigm.

New constructions. Equipped with the **CtX** paradigm, we give new constructions of multi-input KDFs for specialized settings in Section 7. So far, the proposed n-KDFs we studied are exclusively based on hash functions. Yet, for single-input KDFs, standards also specify blockcipher-based constructions [24]. Accordingly, we propose an n-KDF that is purely based on a blockcipher BC such as AES. Besides the pseudorandomness of BC, we also require BC to be key-collision resistant, meaning that it is infeasible to find keys $k \neq k'$ and an input x such that $\mathsf{BC}(k, x) = \mathsf{BC}(k', x)$. Inspired by the **CtX** paradigm, we show that these two properties suffice to combine multiple keys into a single one (without touching context information or labels yet). Running a variable-output-length PRF (e.g., the CMAC-based construction by NIST [24]) on this merged key (now including context and label) yields a secure, fully blockcipher-based multi-input KDF.

We also look into the combination of computationally secure key material with information-theoretically secure key material. The need for such a combiner arises, for example, to combine a quantum key distribution protocol with classical and/or post-quantum key exchange. The resulting key should inherit the information-theoretic security if the corresponding input source is, in fact, information-theoretically secure. Likewise, it should provide computational security if one of the other sources is computationally secure or if the "information-theoretic" source merely provides computational security. We show that combining key material from all sources into a multi-input KDF and then adding another portion of the alleged information-theoretic source to the output provides this feature.

Collision resistance as a feature. We argue (and discuss in detail in Appendix E) that, ideally, KDFs should also be collision resistant. This means that even if the adversary has full control over the inputs to the KDF (i.e., we allow all inputs to be maliciously chosen), it should be hard to find two inputs for which the KDF output collides. Note that the pseudorandomness of the KDF does not guarantee this because the adversary fully determines the key material when attempting to produce a collision.

Collision resistance is sometimes explicitly mentioned in standards (e.g., in TLS 1.3 [57]) and in scientific works (e.g., in [19]) as a requirement for KDFs. We revisit the example of unknown key share (UKS) attacks on key exchange protocols [15] and how they can be easily prevented if the underlying KDF is collision resistant.

One may think that HKDF, i.e., HMAC, is collision-resistant if the underlying hash function is. The nuisance in HMAC, however, lies in the initial processing of the input key, which is padded with 0 bytes (if too short) or first hashed (if too long). This means HKDF is not collision-resistant for arbitrary inputs, but only with appropriate restrictions, which—as seen for the ETSI-CatKDF construction—can lead to insecurities if neglected. We stress that this appears to be widely known among practitioners; here, we simply show what it means for constructions and how to avoid the potential pitfalls. In particular, we discuss the constructions we analyze. To give evidence that this property is not achieved in general, especially for blockcipher-based constructions, we look at the CMAC-based single-input KDF in Bluetooth Low Energy [16] and show that it is *not* collision-resistant.

Application. We finally illustrate how our security notion of KDFs can facilitate the analysis of higher-

level protocols. In Section 8, we demonstrate this with the example of a KEM+KEM-combiner key exchange protocol and security against passive adversaries, where the parties combine keys from one ephemeral and one static KEM. This is used, for example, to build a hybrid combiner of a classically secure and a post-quantum secure KEM. Ultimately, the key exchange steps can be considered to be the means to derive key material, and we are only interested in the KDF eventually putting the two keys together. It is therefore natural to look at the security of the protocol from the angle of the KDF (instead of the KEMs as in, say, [14]); the key exchange protocol data only enters via the context information for the KDF. We show that the key exchange security of the combiner indeed immediately follows from KDF security when the KEMs constitute secure sources. We leave it to future work to apply our model to more complex protocols and to consider active security.

1.3 Related Work

As mentioned above, key derivation with multiple inputs is common in practice, often as part of complex protocols. One of the most prominent examples of multi-input key derivation is the latest TLS 1.3 standard [57], combining a pre-shared key *psk* with a DH secret to derive multiple session keys. TLS 1.3's key schedule uses HKDF and can be seen as an example of a 2-KDF. It has been cryptographically analyzed, explicitly or implicitly, in several works [34, 31, 29, 28, 20]. The results usually model the underlying hash function as a random oracle, but also rely on the dual-PRF security of HMAC which was only recently studied by Backendal et al. [3].

In secure messaging, Signal's X3DH key exchange protocol [49] derives a key from several (correlated) Diffie–Hellman secrets, basically using concatenation for combing the secrets. Signal's X3DH KDF has been analyzed as part of the Signal protocol by Cohn-Gordon et al. [26] in the random oracle model. Post-quantum versions of Signal's X3DH appear in [18, 32, 17]. Signal also introduced a post-quantum key exchange PQXDH [50], adding a KEM shared secret to the DH secrets of X3DH as input for key derivation, which was analyzed in [12, 38]. An analysis of hybrid solutions of Perrin's Noise framework as a full channel protocol appears in [1]. The post-quantum hybrid version of the Noise-inspired WireGuard protocol has been analyzed by Hülsing et al. [41] as a key exchange protocol, relying on the dual-PRF security of the key derivation function.

The Messaging Layer Security (MLS) protocol [7] also uses HKDF-based combiners to derive keys from multiple inputs. Brzuska et al. [19] provide an analysis of the key scheduling. Brzuska and Winkelmann (BW) [21] consider multi-input pseudorandom functions (n-PRFs) as key combiners in the context of MLS. While being related to multi-input KDFs, there are significant differences. The goal of n-PRFs is to combine several (pseudo)random keys in a robust way. In contrast, multi-input KDFs target arbitrary and even correlated sources. Notably, while n-PRFs take context inputs, these are required to be unique, while key material (and contexts) may be reused in a multi-input KDF. BW give a construction called NPRF based on HMAC, which computes the xor of PRF outputs for each (extracted) key and the unique context information. A similar proposal with a comparable goal appears in a recent work by Aviram et al. [2]. We note that BW [21] mention collision resistance as a desirable goal for multi-input PRFs and argue, based on the collision-resistance of HMAC, that their enhanced construction crNPRF achieves this property.

Password-based key derivation functions such as PBKDF2 [42] need to process low-entropic inputs, in contrast to the sources we consider here. This introduces special requirements for the design of such schemes, e.g., impeding exhaustive searches. While our approach is, in principle, applicable to this setting as well, the constructions are usually fundamentally different. We note that Nair and Song [52] recently considered multi-factor key derivation protocols based on Krawczyk's KDF framework. Although their protocols are also called key derivation functions, they are located at a higher abstraction level, aiming to be able to deal with lost secrets and, therefore, requiring some form of setup steps. Indeed, their constructions themselves apply (password-based) KDFs. Furthermore, they were recently shown to have severe vulnerabilities [59].

Combiners for key encapsulation mechanisms (KEMs) have been considered in [40, 14, 35, 55]. Bindel et al. [14] consider several suggestions close to the TLS 1.3 key scheduling and its hybrid key exchange draft [64], Schwabe et al. [60, 61] and Celi et al. [23] analogously investigate the combiner used in KEMTLS. Campagna and Petcher [22] investigated the security of the key derivation functions proposed by ETSI from a hybrid KEM viewpoint. All these results refer to security from a KEM perspective, via IND-CPA/CCA type games. We, in contrast, are interested in the security of the key derivation step itself for general key material sources, multiple challenge queries, and applications of derived keys. Indeed, our application to a KEM+KEM-combiner key exchange protocol (see Section 8) is an example of analyzing

a hybrid (KEM) key exchange through the lens of (multi-input) KDFs rather than KEM combiners.

2 Notation and Conventions

As is common in standards, we sometimes work with bytes (octets) instead of bits as atomic values. We denote by $\mathbb{O} = \{\texttt{0x00}, \ldots, \texttt{0xFF}\}$ the set of all possible values of an octet, in hexadecimal form. To capture both bit-oriented as well as octet-oriented settings simultaneously, we sometimes use $\mathbb{B}$ for the underlying character set, with the understanding that $\mathbb{B} = \{0,1\}$ for bit strings and $\mathbb{B} = \mathbb{O}$ for byte strings.

For a string $a \in \mathbb{B}^*$ we denote by $|a|$ its length (in $\mathbb{B}$ characters) and by $a[i]$ and $a[i..j]$ the characters at position i, resp. i through j (inclusive), starting with index 1; if $i > j$ then $a[i..j]$ denotes the empty string ε. We write $a_1 \preccurlyeq a_2$ for a_1 being a prefix of a_2. The symbol $\|$ denotes string concatenation and we write $a^n = a\| \ldots \|a$ for the n-fold repetition of a string a. We write $a = [x]_n$ for the canonical big-endian encoding of integer $x < 2^n$ with a fixed number $|a| = n$ of $\mathbb{B}$ characters. We denote by $\langle x, y\rangle$ an injective encoding of strings which ensures that, if $x \neq x'$ or $y \neq y'$, then $\langle x, y\rangle \neq \langle x', y'\rangle$.

For $i \in \mathbb{N}$, $[i]$ denotes the set $\{1, \ldots, i\}$. For a vector $\mathbf{x}$, usually written in boldface, $|\mathbf{x}|$ denotes its length and for $i \in [|\mathbf{x}|]$, $\mathbf{x}[i]$ or $\mathbf{x}_i$ denotes the i^{th} element. By $x \leftarrow\!\!\$\ \mathcal{S}_1$ we denote sampling an element uniformly at random from $\mathcal{S}_1$ and assigning it to x. Similarly, we write $X \leftarrow\!\!\$\ \mathcal{X}$ to indicate sampling a random variable X from a probability distribution $\mathcal{X}$. By [[cond]] we denote the boolean result of evaluating cond.

The distinguished symbol $\perp$ is used as a placeholder value for uninitialized variables and to signal errors. Unless otherwise defined, sets, strings, counters, and boolean variables are assumed to be initialized to the empty set $\emptyset$, the empty string ε, the value 0, and 0, respectively. Vectors—we often use the term tables synonymously—such as $\mathbf{a}[\cdot]$ are initialized to $\perp$ in all positions. We follow an object-oriented approach to read and write vector entries. That is, for vector $\mathbf{a}$ we denote by $\mathbf{a}[i].x$ the value of entry $\mathbf{a}[i]$ at attribute x. Likewise, $\mathbf{a}[i].x \leftarrow x$ denotes the assignment of variable x to attribute x of the ith entry in $\mathbf{a}$ (where we often use identical variable and attribute names). We also use $\mathbf{a}[i].(x,y,z)$ to read all entries x, y, z at once and the notation $\mathbf{a}[i].(x,y,z) \leftarrow (x, y, z)$ as shortcut for the simultaneous assignment. We use (query) lists $\mathcal{Q}$ to store queries, adding element (x, y, z) to the initially empty list via $\mathcal{Q}.\texttt{append}(x, y, z)$. The ith element is denoted $\mathcal{Q}[i]$. We denote by $\mathcal{Q}_1||\mathcal{Q}_2$ the concatenation of two lists.

We use the game-playing framework of [11]. By $\mathbf{G}^{\text{Sec-}d}_{\mathsf{S}}(\mathcal{A})$, we denote running the game for security experiment Sec, parameterized by scheme S and (optionally) bit $d \in \{0,1\}$, with adversary $\mathcal{A}$. Games have procedures, also called oracles. In particular, there may be an initialization procedure Initialize and a finalization procedure Finalize. If oracles Initialize or Finalize are present, only one query to each is allowed, and this must be the first resp. last query the adversary makes. The output of the game execution is defined as the output of Finalize if the latter is present and the output of the adversary otherwise. By $\mathbf{G}(\mathcal{A}) \Rightarrow y$ we denote the event that the execution of game $\mathbf{G}$ with adversary $\mathcal{A}$ results in output y. We associate the boolean values true to 1 and false to 0 and write $\Pr[\mathbf{G}(\mathcal{A})]$ as shorthand for $\Pr[\mathbf{G}(\mathcal{A}) \Rightarrow 1]$, the probability that the execution returns 1.

3 Key Material Sources

The security of a key produced by a key derivation function inevitably depends on the amount of entropy in the source(s) from which the inputs are drawn. Here, we recall the definition of a key material source, introduce the concept of a source collection, and give measures of source security. Further relevant background definitions are in Appendix A.

Key material sources. In the following, we will be concerned with different kinds of sources that provide the raw input key material to the key derivation functions. We will assume that the sources generate key material with high entropy, but not necessarily that this source key material is unbiased and independent. In particular, we want a definition that is general enough to cover the most common types of "secrets" used for key derivation in cryptographic applications. This includes (in order of decreasing expected entropy): information-theoretically secure QKD keys, computationally secure uniform keys, shared keys from Diffie–Hellman key exchange with high computational min-entropy, and passphrases or passwords from some distribution with low guessing probability. In particular, the latter two examples illustrate the need to cover sources which generate *correlated* secrets; in a Diffie–Hellman protocol, the public key of user A can be used to exchange keys with multiple other users, generating a set of shared

keys that are all correlated, and in the password setting, users are known to choose highly correlated secrets.

We adapt the definition of a source of key material from Krawczyk's HKDF paper [48] to also encompass sources of correlated key material as follows: A source Σ samples a vector of tuples, (σ, c, α), where each tuple consists of "secret" key material $\sigma \in \mathsf{Skm}$, context information $c \in \mathsf{Ctx}$ about the generated secret, and auxiliary information $\alpha \in \mathsf{Aux}$ (which is assumed to be available to the adversary as leakage). As a convention, the auxiliary information always includes the context, meaning that the context is also publicly known. The distinction between context and leakage is that context is provided as input to the key derivation function, whereas leakage is given to the adversary. As such, the auxiliary information may contain more information than just the context, as long as it does not leak information about the secret key material that would allow an adversary to mount trivial attacks on the underlying scheme.

To illustrate the difference between context and auxiliary information, consider a simple Diffie–Hellman key exchange in which the parties exchange shares g^x and g^y and use $\sigma = g^{xy}$ as the key material. The context information c could be empty according to the protocol specification; this is for example the approach taken by Signal [49] in the multi-input setting. However, the adversary would nonetheless learn the auxiliary data $\alpha = (g^x, g^y)$, reflecting the information observed in the key exchange.[5]

Definition 1 (Key material source)**.** A source of key material (or, simply, source) Σ is a correlated probability distribution over a support set $\mathsf{Skm} \times \mathsf{Ctx} \times \mathsf{Aux}$ which is generated by an efficient probabilistic algorithm that we will also refer to as Σ. Associated with Σ is an integer parameter u called the *sample size*, such that when called, Σ outputs a vector $\mathbf{z}$ of length u, where each element consists of a triple $(\sigma, c, \alpha) \in \mathsf{Skm} \times \mathsf{Ctx} \times \mathsf{Aux}$. We assume that c is contained in α. That is, for each $(\sigma, c, \alpha) \in \mathbf{z}$, there exists an efficient way of extracting c from α.

The notion of key material sources is purposefully generic and allows for a wide range of instantiations. Let us consider some illustrating examples.

Uniform keys. For a source of uniform keys of length ℓ, the sampler Σ returns a vector of uniformly chosen strings in $\{0,1\}^\ell$, with context and auxiliary information being empty (or possibly including some public identifier for the key material). This models key generation for cryptographic primitives with uniformly distributed fix-length keys. In practice, this could capture pre-shared keys or the usage of a QKD scheme.

KEMs. A source can also capture the shared secrets established by a key encapsulation mechanism $\mathsf{KEM} = (\mathsf{KGen}, \mathsf{Encaps}, \mathsf{Decaps})$. Here, each element in $\mathbf{z}$ is a tuple $(K, \langle pk, c\rangle, \alpha)$, where $(pk, sk) \leftarrow\!\!\$\ \mathsf{KEM.KGen}()$ is a KEM key pair and K is the result of an encapsulation $(c, K) \leftarrow\!\!\$\ \mathsf{KEM.Encaps}(pk)$. That is, the secret key material is the KEM secret K and the context is an encoding of the KEM public key and ciphertext $\langle pk, c\rangle$. Recall that, as per our definition, this context is also included in the auxiliary information α.

Diffie–Hellman. To model a Diffie–Hellman key exchange protocol, the source key material might be a distribution over the group G of prime order p with an associated generator g. Here, each element in the key material vector has the form $(g^{xy}, \langle G, p, g, g^x, g^y\rangle, \alpha)$, where $x, y \in \mathbb{Z}_p^*$. That is, the secret key material is the Diffie–Hellman secret and the context information is an encoding of the group, its order, the generator, and the DH shares; per definition, this context is also included in α.

In some source distributions the secrets σ in the vector $\mathbf{z} \leftarrow\!\!\$\ \Sigma$ may be correlated. This is the case if, for example, the same Diffie–Hellman or KEM public key is used to establish multiple secrets. We assume that the distribution is public, and hence that such correlation patterns are known to the adversary. For example, for the Diffie–Hellman source capturing the inputs to Signal's KDF in X3DH, the vector $\mathbf{z}$ contains all pair-wise combinations of users' long-term/static and ephemeral Diffie–Hellman public keys (see Section 5.4 for details). We then assume that the adversary knows which elements in $\mathbf{z}$ are associated with the public key of each user, without having to first inspect the context information of each secret.

Source collections. We define security of key derivation functions with respect to the key material sources that provide the inputs to the KDF. That is, the game that defines security of an n-input KDF (n-KDF) will be parameterized by a *collection* of sources, $\boldsymbol{\Sigma}$, from which the inputs are drawn.

[5] Other protocols like TLS 1.3 put the data of the key exchange in c such that it is immediately available to the adversary by our assumption about α superseding c.

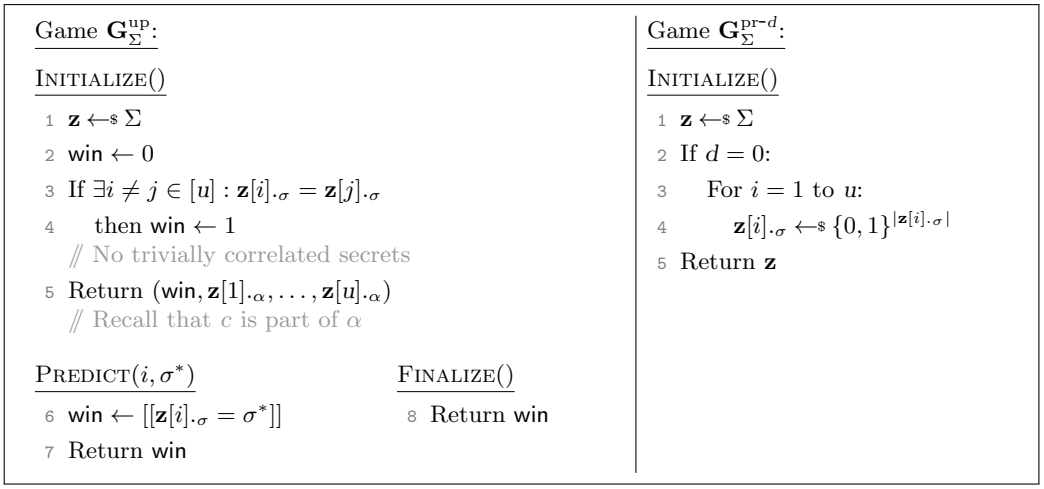

Figure 2: Games defining multi-challenge unpredictability (left) and pseudorandomness (right) of a source Σ.

To accommodate that correlated secrets from the same source may be used across multiple input positions in the KDF, we associate to the source collection a surjective function $\Sigma\text{-}\mathsf{map}\colon [n] \to [r]$ which maps an integer representing the position of a KDF input to an index of a source in the collection of r sources. That is, an (n, r)-source collection maps the n input positions of a n-KDF to the r sources, such that each position in the KDF is assigned exactly one source. We define the mapping $\Sigma\text{-}\mathsf{map}$ to be surjective to ensure that we only consider sources which contribute to the key derivation process.

Consider, for example, a collection $\mathbf{\Sigma} = (\Sigma_1, \Sigma_2)$ consisting of a source Σ_1 outputting a vector of Diffie-Hellman shares $\sigma_i = g^{xy_i}$ and a source Σ_2 generating a vector of random keys K_i representing ephemeral KEM shares. Then a 2-KDF which expects the DH secrets as the first input and the KEM values at the second position has $\Sigma\text{-}\mathsf{map}(1) = 1$ and $\Sigma\text{-}\mathsf{map}(2) = 2$.

Definition 2 (Source collection)**.** An (n, r)-source collection $\mathbf{\Sigma} = (\Sigma_1, \ldots, \Sigma_r)$ is a tuple of $r \leq n$ sources with an associated surjective function $\Sigma\text{-}\mathsf{map}\colon [n] \to [r]$.

Note that the sources Σ_i of a collection $\mathbf{\Sigma} = (\Sigma_1, \ldots, \Sigma_r)$ are considered to be independent, meaning that one can sample each source individually. Correlations of key material, as in the Diffie–Hellman example of sampling related secrets $\sigma_i = g^{xy_i}$ for the same share g^x are captured within a single source. Maliciously chosen key material is later dealt with in our security model by giving the adversary the possibility to set key inputs for the KDF.

ENTROPY MEASURES. In order to quantify the security that a KDF provides with respect to a source collection, we need a measure of the amount of entropy provided by the sources. In particular, due to the leakage of the auxiliary information, we measure the unpredictability of the secrets in a sample given this leakage.

We use a game-based approach to define average min-entropy. In the game, shown in Figure 2, a vector $\mathbf{z}$ of key material is sampled according to the source Σ. The predictor receives the auxiliary data for $\mathbf{z}$ and can make multiple attempts to guess a secret σ_i in the vector via an oracle PREDICT, winning if at least one attempt is correct. To capture cases where the secrets are trivially correlated due to being identical, we let the adversary immediately win if identical secrets occur in the sample. If the predictor $\mathcal{A}$ is unbounded, the advantage measures the statistical unpredictability of the source; if $\mathcal{A}$ is computationally bounded, it measures the computational source unpredictability.

Definition 3 (Unpredictable source)**.** We define the advantage of an adversary $\mathcal{A}$ against the unpredictability ofa key material source Σ to be

$$\mathbf{Adv}^{\mathrm{up}}_{\Sigma}(\mathcal{A}) = \Pr\left[\, \mathbf{G}^{\mathrm{up}}_{\Sigma}(\mathcal{A}) \Rightarrow 1 \,\right] ,$$

where game $\mathbf{G}^{\mathrm{up}}_{\Sigma}$ is given in Figure 2. We say that Σ is (t, ε)-computationally unpredictable if for any adversary running in time at most t we have $\mathbf{Adv}^{\mathrm{up}}_{\Sigma}(\mathcal{A}) \leq \varepsilon$.

Sometimes we can make stronger assumptions about the key material sources than unpredictability. For example, when the secrets are generated by a KEM, the IND-CPA-security of the KEM implies that

the key material not only has high entropy but is pseudorandom. That is, each secret generated by the source is computationally indistinguishable from a uniformly random string of some fixed length. We refer to this type of source as a *pseudorandom source* and show in Appendix A.3 that pseudorandom sources are also unpredictable.

Definition 4 (Pseudorandom source)**.** Let Σ be a key material source. We define the advantage of an adversary $\mathcal{A}$ against the pseudorandomness of Σ to be

$$\mathbf{Adv}^{\mathrm{pr}}_{\Sigma}(\mathcal{A}) = \mathbf{G}^{\mathrm{pr}\text{-}1}_{\Sigma}(\mathcal{A}) - \mathbf{G}^{\mathrm{pr}\text{-}0}_{\Sigma}(\mathcal{A}) \ ,$$

where game $\mathbf{G}^{\mathrm{pr}\text{-}d}_{\Sigma}$ is given in Figure 2. The source Σ is (t, ε)-pseudorandom if for any adversary running in time at most t we have $\mathbf{Adv}^{\mathrm{pr}}_{\Sigma}(\mathcal{A}) \leq \varepsilon$.

4 Multi-Input Key Derivation Functions

In this section, we define multi-input key derivation functions and their security.

4.1 Syntax

An n-KDF extends the basic notion of a KDF by taking up to n key material inputs $\sigma_i \in \mathsf{Skm}_i$ together with the context information $c_i \in \mathsf{Ctx}_i$ of each input (cf. Definition 1). Syntactically, an n-KDF always has a fixed number n of inputs. However, our definition also supports *optional* inputs by letting $\perp \in \mathsf{Skm}_i$, where $\perp$ is a distinguished symbol denoting that this input is null. In addition to the source key material and their contexts, the n-KDF takes a label $L \in \mathsf{Lbls}$ for the output key and a parameter $\ell \in \mathsf{KLout}$ declaring the desired key output length. Here, the length may be denoted in bits or in octets. Recall that we use $\mathbb{B}$ as the generic character set, capturing both bits $\mathbb{B} = \{0,1\}$ and octets $\mathbb{B} = \mathbb{O}$.

Definition 5 (Key derivation function)**.** An n-input key derivation function (n-KDF) $\mathsf{F} : (\mathsf{Skm}_1 \times \mathsf{Ctx}_1) \times \ldots \times (\mathsf{Skm}_n \times \mathsf{Ctx}_n) \times \mathsf{Lbls} \times \mathsf{KLout} \to \mathbb{B}^*$ is a deterministic algorithm with an associated key output length set $\mathsf{KLout} \subseteq \mathbb{N}$. Via $K \leftarrow \mathsf{F}((\sigma_1, c_1), \ldots, (\sigma_n, c_n), L, \ell)$, algorithm F on input n tuples of secrets and associated contexts, a label string L and an output length $\ell \in \mathsf{KLout}$ outputs a key $K \in \mathbb{B}^{\ell}$.

We call an n-KDF with fixed, empty label and fixed output length (i.e., $\mathsf{Lbls} = \{\varepsilon\}$ and $\mathsf{KLout} = \{\ell\}$) a "combiner n-KDF" (comb-KDF).

4.2 Security

We define the security notion for an n-KDF F through the real-or-random indistinguishability game $\mathbf{G}^{\mathrm{kdf}\text{-}d}_{\mathsf{F},\boldsymbol{\Sigma},req}$ shown in Figure 3. The game is parameterized by a bit $d \in \{0,1\}$, a source collection $\boldsymbol{\Sigma}$, and a predicate *req* which helps to exclude trivial attacks such as challenge queries for dishonest keys only. For comb-KDFs, the empty label and fixed output length (in gray boxes) can essentially be ignored; we call this simplified notion "fixed-label-and-length" (fix$L\ell$-kdf) security.

The game's parameter d determines the setting, with $d = 1$ indicating the "real" game and $d = 0$ the "random" game, $\boldsymbol{\Sigma}$ determines which key material sources may be used in which position of the KDF, and the predicate *req* determines the requirements on queries made by the adversary. We list possible requirements in Table 1, discussed in Section 4.3 below. One purpose of *req* is to prevent the adversary from mounting trivial attacks, but it also allows us to capture different settings with no changes to the main game. For example, *req* encodes whether the game captures security for extendable-output n-KDFs (XOF-n-KDFs), when $req \implies req_X$, or for non-extendable-output n-KDFs (NOF-n-KDFs), namely when $req \implies req_N$. The task of the adversary is to distinguish real outputs of the KDF F from uniformly random strings of the same length.

The game begins by sampling a vector $\mathbf{z}_i$ (of length u) of source key material from each $\Sigma_i \in \boldsymbol{\Sigma}$. It then runs the adversary with oracle access to the procedures NewKey, SetKey, Ro$-KDF, and KDF.

Oracle NewKey takes as input two indexes i, j: the first one specifies from which source $\Sigma_i \in \boldsymbol{\Sigma}$ the key material should be taken, and the second one the position $j \in [u]$ of the desired key material in the vector $\mathbf{z}_i$. The oracle assigns a new "honest" input for F with source key material, context, and auxiliary information $(\sigma, c, \alpha) = \mathbf{z}_i[j]$, where "honest" indicates that the key material is not influenced by the adversary. The new key is stored in a vector $\mathbf{k}[\cdot]$, together with the index i of its source. The game keeps track of the next available position in $\mathbf{k}$ by using the counter ν. Finally, the used key material is deleted from $\mathbf{z}_i$, preventing it from being assigned again, and the auxiliary information is returned to the

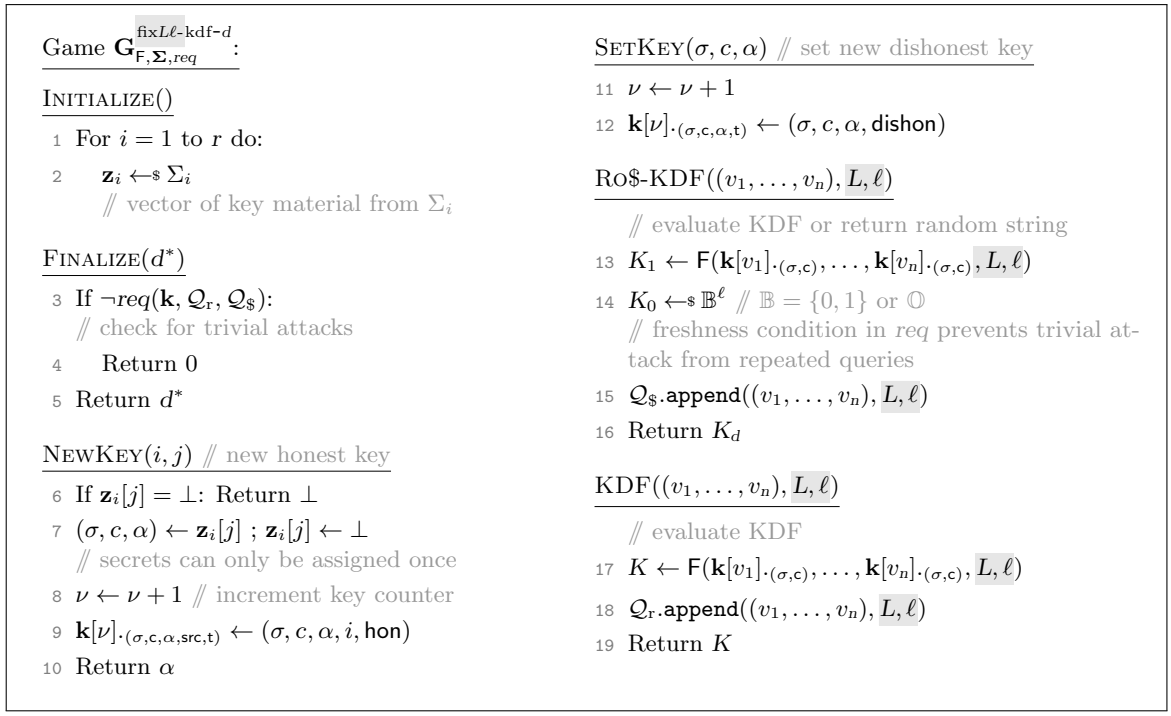

Figure 3: Games $\mathbf{G}^{\text{kdf-}d}_{\mathsf{F},\boldsymbol{\Sigma},req}$ and $\mathbf{G}^{\text{fix}L\ell\text{-kdf-}d}_{\mathsf{F},\boldsymbol{\Sigma},req}$ defining kdf security for an n-input key derivation function F, an (n, r)-source collection $\boldsymbol{\Sigma}$, and requirement req; see Table 1 for the requirement definition. The "fixed-label-and-length" (fix$L\ell$-kdf) version of the game is for a KDF with fixed label L and output length ℓ (in gray boxes, which can essentially be ignored).

adversary as leakage. Note that the key material can only be assigned once to a position in $\mathbf{k}$, but then the value may be used multiple times to derive keys.

Complementing NEWKEY, oracle SETKEY allows the adversary to register (maliciously chosen) input key material of its choice. These keys are also stored in the vector $\mathbf{k}[\cdot]$ at the next available position ν. To distinguish them from honest keys, keys registered using oracle SETKEY are marked with $\mathbf{k}[\nu].\mathsf{t} = \mathsf{dishon}$. This oracle also allows the adversary to void an optional key material input of the KDF by registering a new key with σ set to the dedicated symbol $\bot$. Such void optional inputs are marked as dishon since they do not contribute to the entropy of the derived key.

Finally, in oracles RO$-KDF and KDF the adversary can request outputs from the n-KDF. The oracles take as input a vector $(v_1, \ldots, v_n)$ of indexes, where v_p indicates which source key material and context from vector $\mathbf{k}$ to use in position p. The compliance of these inputs is checked in the requirement req at the end of the game. (For example, req_{ValidPos} ensures that honest inputs are only used in positions associated with the source from which the key material was drawn.) The other arguments passed to the oracle are the label L and the output key length ℓ. In the RO$-KDF oracle, the final key may be replaced by a random string of length ℓ, depending on the game bit d. The game stores query inputs for oracle RO$-KDF in the list $\mathcal{Q}_{\$}$ and queries to oracle KDF in $\mathcal{Q}_{\mathrm{r}}$.

The game ends when the adversary runs oracle FINALIZE with its bit guess d^* as input. If requirement req is fulfilled (indicating that, at a minimum, there have been no trivial attacks), FINALIZE returns d^*; otherwise, it returns 0. The parameter req additionally allows us to put further restrictions on the adversary's success, thereby obtaining multiple different security notions from one game, as explained next.

We define the following security notion for n-KDFs, where $\mathbf{G}^{\text{kdf}}_{\mathsf{F},\boldsymbol{\Sigma},req}(\mathcal{A})$ is as defined in Figure 3.

Definition 6 (KDF security)**.** Let F be an n-KDF. We define the advantage of an adversary $\mathcal{A}$ against the KDF security of F with source collection $\boldsymbol{\Sigma}$ and requirement req as

$$\mathbf{Adv}^{\text{kdf}}_{\mathsf{F},\boldsymbol{\Sigma},req}(\mathcal{A}) = \Pr\left[\,\mathbf{G}^{\text{kdf-1}}_{\mathsf{F},\boldsymbol{\Sigma},req}(\mathcal{A})\,\right] - \Pr\left[\,\mathbf{G}^{\text{kdf-0}}_{\mathsf{F},\boldsymbol{\Sigma},req}(\mathcal{A})\,\right].$$

Note that for this definition to be meaningful, the requirements must include all the trivial attack restrictions. That is, we demand that req implies either req_N or req_X. We elaborate on the relations between the security notions induced by the different requirements below.

Name	Description	Predicate
req_{1HKey}	At least one honest key	$(\forall((v_1,\ldots,v_n),L,\ell)\in\mathcal{Q}_\$)(\exists p\in[n]):\mathbf{k}[v_p].{}_t=\mathsf{hon}$
req_{XOF}	XOF freshness	$(\forall q\le\lvert\mathcal{Q}_\$\rvert)\ (\forall q'\le\lvert\mathcal{Q}_\$\Vert\mathcal{Q}_\mathrm{r}\rvert),(\mathbf{v}_q,L_q,*)\leftarrow\mathcal{Q}_\$[q],(\mathbf{v}_{q'},L_{q'},*)\leftarrow(\mathcal{Q}_\$\Vert\mathcal{Q}_\mathrm{r})[q']:q\neq q'\implies(\mathbf{v}_q,L_q)\neq(\mathbf{v}_{q'},L_{q'})$
req_{NOF}	NOF freshness	$(\forall q\le\lvert\mathcal{Q}_\$\rvert)\ (\forall q'\le\lvert\mathcal{Q}_\$\Vert\mathcal{Q}_\mathrm{r}\rvert):q\neq q'\implies\mathcal{Q}_\$[q]\neq(\mathcal{Q}_\$\Vert\mathcal{Q}_\mathrm{r})[q']$
req_{NoDColl}	No dishonest key collisions	$(\forall v,v'\le\nu):v\neq v'\wedge\mathbf{k}[v].{}_\mathsf{t}=\mathbf{k}[v'].{}_\mathsf{t}=\mathsf{dishon}\implies\mathbf{k}[v].{}_{(\sigma,\mathsf{c})}\neq\mathbf{k}[v'].{}_{(\sigma,\mathsf{c})}$
req_{ValidPos}	Honest keys in valid positions	$(\forall((v_1,\ldots,v_n),L,\ell)\in\mathcal{Q}_\$\cup\mathcal{Q}_\mathrm{r})(\forall p\in[n]):\mathbf{k}[v_p].{}_t=\mathsf{hon}\implies\mathbf{k}[v_p].{}_\mathsf{src}=\Sigma\text{-}\mathsf{map}(p)$
req_X	XOF KDF	$req_{\mathsf{XOF}}\wedge req_{\mathsf{1HKey}}\wedge req_{\mathsf{NoDColl}}\wedge req_{\mathsf{ValidPos}}$
req_N	NOF KDF	$req_{\mathsf{NOF}}\wedge req_{\mathsf{1HKey}}\wedge req_{\mathsf{NoDColl}}\wedge req_{\mathsf{ValidPos}}$
$req_{1\times\mathsf{Key}}(\mathcal{S})$	One-time keys for sources $\mathcal{S}$	$(\forall q,q'\le\lvert\mathcal{Q}_\$\Vert\mathcal{Q}_\mathrm{r}\rvert),((v_1,\ldots,v_n),*,*)\leftarrow(\mathcal{Q}_\$\Vert\mathcal{Q}_\mathrm{r})[q],((v'_1,\ldots,v'_n),*,*)\leftarrow(\mathcal{Q}_\$\Vert\mathcal{Q}_\mathrm{r})[q'],(\forall p\in[n]\mid\Sigma\text{-}\mathsf{map}(p)\in\mathcal{S}):q\neq q'\implies v_p\neq v'_p$
$req_{\mathsf{HybridKR}}(\mathcal{S})$	Honest keys for sources $\mathcal{S}$	$(\forall p\in[n]\mid\Sigma\text{-}\mathsf{map}(p)\in\mathcal{S})(\forall((v_1,\ldots,v_n),L,\ell)\in\mathcal{Q}_\$):\mathbf{k}[v_p].{}_t=\mathsf{hon}$
$req_{\mathsf{Hybrid}}(\mathcal{S})$	Honest keys for sources $\mathcal{S}$ (no key reuse)	$req_{1\times\mathsf{Key}}(\mathcal{S})\wedge req_{\mathsf{HybridKR}}(\mathcal{S})$
req_{NoReal}	No queries to KDF oracle	$\lvert\mathcal{Q}_\mathrm{r}\rvert=0$

Table 1: Requirements *req* for games $\mathbf{G}^{\text{kdf-}d}_{\mathsf{F},\mathbf{\Sigma},req}$ and $\mathbf{G}^{\text{fix}L\ell\text{-kdf-}d}_{\mathsf{F},\mathbf{\Sigma},req}$ (the latter fixes the label and length inputs in the gray boxes) defined in Figure 3. The upper predicates are necessary to rule out trivial attacks, the lower predicates are optional and implement various useful refinements.

4.3 Requirements

The parameter *req* flexibly controls the admissible actions of the adversary, penalizing any violations at the end of the game. The possible requirements are listed in Table 1.

Fundamental Requirements. The first core requirement is called req_{1HKey} and ensures that there is at least one "honest" key among the secrets input to the n-KDF in an RO$-KDF query. Otherwise, the adversary could call oracle RO$-KDF on only maliciously chosen keys and easily determine the secret game bit d. Additionally, in order to prevent trivial attacks, *req* must also include one of the two "freshness" requirements req_{XOF} and req_{NOF}. These ensure that queries to the challenge oracle RO$-KDF are fresh, in the sense that the same inputs have not been used in a prior computation of F. The difference between the two is whether the output length ℓ is considered to contribute to freshness or not. req_{XOF} caters to extendable-output functions (XOF) and hence does not take the output length into account when checking freshness. In contrast, with req_{NOF}, two queries are considered fresh with respect to one another even if they only differ in the desired output length ℓ and should hence produce independent outputs, even if all other inputs are equal, in order for the n-KDF to be secure.

The last requirement to prevent trivial attacks is called req_{NoDColl}. It disallows the adversary from registering the same input key material (σ, c, α) more than once in oracle SetKey. Without this check, the adversary could call oracle RO$-KDF on distinct inputs $((v_1,..,v_i,..v_n),L,\ell)$ and $((v_1,..,v'_i,..,v_n),L,\ell)$, where $v_i\neq v'_i$ both refer to dishonest inputs holding the same key material (σ,c,α). That is, the indexes v_i, v'_i are distinct, but $\mathbf{k}[v_i].{}_{\sigma,\mathsf{c},\alpha}=\mathbf{k}[v'_i].{}_{\sigma,\mathsf{c},\alpha}$. This would allow the adversary to bypass the freshness check, making two identical queries that look distinct to the game, and hence to trivially distinguish real from random answers of the oracle, because in the real case the responses would be identical, whereas in

the random case this rarely happens. For optional inputs, the adversary must—and can—point to a fixed index v_i holding symbol $\perp$ to satisfy req_{NoDColl}.

Finally, we also use req to ensure that the honest inputs to the KDF computation are used in the correct positions. Recall that the game is parameterized by a source collection $\mathbf{\Sigma} = (\Sigma_1, \ldots, \Sigma_r)$ which associates the $r \leq n$ sources to the input positions of the KDF via the mapping $\Sigma\text{-}\mathsf{map}\colon [n] \to [r]$ associated to $\mathbf{\Sigma}$. Requirement req_{ValidPos} ensures that there are no KDF computations in which honest keys are used in a position not associated to their source. (For most n-KDF constructions, the source collection either consists of a single source covering all inputs—in which case this check is not needed—or of n sources associated to one input position each—in which case $\Sigma\text{-}\mathsf{map}$ is the identity function.)

For XOF-n-KDFs and NOF-n-KDFs, we conveniently collect the minimum necessary requirements in the predicates req_X and req_N, respectively.

Optional Requirements. It is sometimes convenient to make further restrictions beyond what is strictly necessary. For example, req_{NoReal} restricts the adversary from making any queries to oracle KDF. That is, there are no "real" queries—only challenge queries to oracle RO$-KDF are allowed. We can also ask that keys in certain positions (corresponding to a set $\mathcal{S}$ of sources) are only used once in queries with requirement $req_{1\times\mathsf{Key}}(\mathcal{S})$. The latter is useful if one considers information-theoretically secure key material that should be used only once. Note that the predicate still allows the same key input to appear at multiple positions within a single query.

The Hybrid Requirement(s). Our fundamental requirements req_X resp. req_N provide very strong security guarantees. With theses requirements, security refers to indistinguishability of challenge values from random, as long as one of the key material inputs in each query is genuine (requirement req_{1HKey}). The positions of the good inputs, however, may vary with each challenge query. For instance, for the first real-or-random query the key material $\mathbf{k}[v_1]_{\cdot(\sigma,\mathsf{c})}$ may be honest and the material $\mathbf{k}[v_2]_{\cdot(\sigma,\mathsf{c})}$ may be controlled by the adversary, and for the second real-or-random query it may be the opposite case. This captures cases where individual outputs of sources may be weak.

In contrast, the common notion of hybrid key derivation, e.g., combining classical and quantum-resistant sources, demands that at least one *source* is good and that *all* key material inputs from this source are good. This contemplates that the underlying classical assumption or the post-quantum assumption (or both) cannot be broken, such that all input key material from such a source contains sufficiently high entropy. This is a more restrictive stipulation, summarized in the requirement $req_{\mathsf{HybridKR}}(\mathcal{S})$, for a designated set $\mathcal{S}$ of good sources: it demands that all keys from sources in $\mathcal{S}$ are good (i.e., honestly generated), and the other ones in $[r] \setminus \mathcal{S}$ should be treated as providing no security.

To see that the hybrid requirement is restricting the adversary more than the regular one, note that $req_{\mathsf{HybridKR}}(\mathcal{S}) \implies req_{\mathsf{1HKey}}$ for a non-empty set $\mathcal{S}$ because, by surjectivity of $\Sigma\text{-}\mathsf{map}$, a good source $i \in \mathcal{S}$ according to predicate $req_{\mathsf{HybridKR}}(\mathcal{S})$ must be assigned to one or more positions $p_1, p_2, \ldots$. But then the honest key material inputs of this source i must appear at the (same) positions in all real-or-random queries, such that each query contains at least one genuine key material inputs, implying predicate req_{1HKey}. The converse does not hold in general, as the above example with two queries switching the position of the honest contribution shows.

However, even when imposing the stricter requirement $req_{\mathsf{HybridKR}}(\mathcal{S})$, our model still captures stronger guarantees than commonly expected in hybrid scenarios. Namely, $req_{\mathsf{HybridKR}}(\mathcal{S})$ allows for key re-use from good sources in $\mathcal{S}$ (hence the "KR"), in arbitrary combinations with other keys (even from sources deemed insecure). Usually, hybrid combiners are however used with ephemeral/one-time-use key material, so that any good key is combined only once with possibly insecure keys. This can be captured in our model by additionally requiring the predicate $req_{1\times\mathsf{Key}}(\mathcal{S})$, and we hence ultimately capture this common hybrid approach with the (even) stricter requirement $req_{\mathsf{Hybrid}}(\mathcal{S})$ combining $req_{\mathsf{HybridKR}}(\mathcal{S})$ and $req_{\mathsf{1HKey}}(\mathcal{S})$.

The difference between the two scenarios, our strong main notions and the hybrid setting, becomes discernible when showing security of constructions. The fundamental requirements req_X and req_N are less restrictive for the adversary. Since the adversary in each real-or-random query may vary the position of the genuine key material input, security proofs require all sources to be secure. This results typically in security statements where the advantage against the multi-input KDF is bounded by the sum of the security advantages against the sources. For instance, assuming that the KDF behaves like a random oracle and assuming a suitable input encoding, we obtain (Theorem 8 in Section 5.1):

$$\mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{RO\text{-}KDF}_n[\mathsf{H}_\mathrm{T}],\mathbf{\Sigma},req}(\mathcal{A}) \leq 2 \cdot \sum_{i=1}^{r} \mathbf{Adv}^{\mathrm{up}}_{\Sigma_i}(\mathcal{B}_i),$$

with respect to the unpredictability of all sources.[6] For the hybrid setting, with the more demanding restriction for the adversary, one usually requires only security of one source such that the bounds become tighter. Once more, for the random-oracle-based KDF we obtain (Theorem 9 in Section 5.1):

$$\mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{RO\text{-}KDF}_n[\mathsf{H}_\mathrm{T}],\boldsymbol{\Sigma},req\wedge req_{\mathsf{Hybrid}}(\mathcal{S})}(\mathcal{A}) \leq 2 \cdot \min_{i\in\mathcal{S}} \left(\mathbf{Adv}^{\mathrm{up}}_{\Sigma_i}(\mathcal{B}_i)\right) .$$

and the advantage against the random-oracle-based KDF is thus bounded by the minimum over the unpredictability advantages of all good sources. This indeed captures the expected the intuition behind hybrid multi-input key derivation: derived keys should inherit the strength of the *unbroken* key material source(s) entering the KDF, as long as good keys are used only once. The proofs for the random-oracle-based KDF are in the full version [4].

We emphasize that, whenever a hybrid setting uses static/multi-use key material (e.g., long-term user keys), then the predicate $req_{\mathsf{HybridKR}}(\mathcal{S})$ allowing key reuse is appropriate to capture arbitrary combinations with other keys. Security bounds for such settings then need to account for collisions in the honest key material of sources: If such values collides in some position i, then the adversary in the key-reuse setting can query RO$-KDF on distinct inputs $((v_1,..,v_i,..v_n),L,\ell)$ and $((v_1,..,v'_i,..,v_n),L,\ell)$, where $v_i \neq v'_i$ point to such a colliding key material, yielding matching answers if $d = 1$. To render this attack useless, one needs to exclude the possibility of collisions in each of the good sources, resulting in the sum over all probabilities.

4.4 Model Simplification and Relations

Our KDF security notion provides the adversary with both a challenge oracle (RO$-KDF) and a "real" function oracle (KDF). This is in contrast to, for example, PRF security, where the game only has a real-or-random challenge oracle. Here, we show that providing the real oracle in addition to the challenge oracle does not make a qualitative difference to the KDF security notion: one can use the RO$-KDF to first replace all RO$-KDF and KDF query responses with random, then again to change the KDF queries back to real (the formal proof is in Appendix C). That is, KDF security with both real and real-or-random oracle is equivalent to KDF security with only the real-or-random oracle.

Proposition 7. *Let* F *be an* n*-KDF and let adversary* $\mathcal{A}$ *be an adversary against the KDF security of* F*. Then for any source collection* $\boldsymbol{\Sigma}$*, there exists an adversary* $\mathcal{B}$ *such that*

$$\mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{F},\boldsymbol{\Sigma},req}(\mathcal{A}) \leq 2 \cdot \mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{F},\boldsymbol{\Sigma},req\wedge req_{\mathsf{NoReal}}}(\mathcal{B}) .$$

Adversary $\mathcal{B}$ *has query count* $\mathbf{q}_{\mathrm{OR}}(\mathcal{B}) = \mathbf{q}_{\mathrm{OR}}(\mathcal{A})$ *for any oracle* OR *from the set* $\{\textsc{NewKey}, \textsc{SetKey}\}$*, and* $\mathbf{q}_{\text{RO\$-KDF}}(\mathcal{B}) = \mathbf{q}_{\text{RO\$-KDF}}(\mathcal{A}) + \mathbf{q}_{\mathrm{KDF}}(\mathcal{A})$*. The running time of* $\mathcal{B}$ *is roughly the same as that of* $\mathcal{A}$*.*

In Appendix B, we further relate the security of XOF- and NOF-n-KDFs, and also formally justify the approach taken, e.g., by TLS 1.3 [57] to bind the input key material to its intended output length by including the latter in the label, effectively transforming a XOF-KDF to a NOF-KDF.

5 Analyzing Real-World Constructions

We now apply our security model to analyze real-world constructions of multi-input KDFs. To illustrate the versatility of our model, we study three constructions with distinct internal designs, input key material settings, and output behaviors: the approach in MLS [7] to combine pre-shared keys for injection into its key schedule, an ETSI proposal based on concatenation [36], and the key derivation in Signal's X3DH handshake protocol [49]. Beforehand, we also discuss the generic random-oracle-based construction.

5.1 Random-Oracle-Based Construction

We discuss here that for hash-based constructions, if one views the hash function as a unitary random oracle, then any reasonable encoding of the inputs will result in a secure KDF for multiple inputs, as long as one is unpredictable. If one uses a hash function with fixed output length, e.g., one of the members of the SHA-3 family, then one obtains a NOF-n-KDF; if the hash function has extendable output length, e.g., one of the SHAKE versions, then one obtains a XOF-n-KDF. Formally, we thus distinguish between

[6] The factor 2 accounts for adversarial evaluation queries made to KDF in addition to the ones made to RO$-KDF, as we show generally in the next section.

$\underline{\mathsf{RO\text{-}KDF}_n[\mathsf{H}_\mathrm{T}]((\sigma_1, c_1), (\sigma_2, c_2), \ldots, (\sigma_n, c_n), L, \ell)}$

1 $K \leftarrow \mathsf{H}_\mathrm{T}(\langle \sigma_1, c_1, \sigma_2, c_2, \ldots, \sigma_n, c_n, L \rangle)$
// parameter ℓ only for T = XOF
2 Return $K[1..\ell]$

Figure 4: KDF $\mathsf{RO\text{-}KDF}_n[\mathsf{H}_\mathrm{T}]$ based on random oracle H with some recoverable encoding $\langle \cdot \rangle$.

random oracles with non-extendable (fixed) output length *hl*, denoted as $\mathsf{H}_{\mathrm{NOF}}$, and those with extendable outputs, denoted $\mathsf{H}_{\mathrm{XOF}}$. The latter takes an additional length parameter ℓ as input and outputs a string of length ℓ. A formal definition of such random oracles appears in Appendix A.2. In the NOF-n-KDF case we assume that the output length parameter ℓ is bounded from above by *hl* and truncate the hash output to ℓ bits.

We assume some recoverable encoding $\langle \sigma_1, c_1, \sigma_2, c_2, \ldots, \sigma_n, c_n, L \rangle$ of the input data. One option to encode the values σ_i, c_i, and L is to prepend each value with its length, encoded in a fixed-space entry. For instance, if the length of each entry is less than 2^{32} bits, then four octets suffice. We would then set

$$\langle \sigma_1, c_1, \sigma_2, c_2, \ldots, \sigma_n, c_n, L \rangle = [|\sigma_1|]_{32} \,\|\sigma_1\|\, [c_1]_{32} \,\|c_1\| \ldots \| \,[|L|]_{32} \,\|L$$

where $[i]_{32}$ denotes the encoding of integer $i < 2^{32}$ with 32 bits. Then, one can parse the string from left to right and recover the entries again.

The following theorem shows KDF security based on the unpredictability of the source collection:

Theorem 8 (KDF security based on random oracle)**.** *Let* H_T *be a random oracle (with output length hl in case of* T = NOF *resp. unbounded output length in case of* T = XOF*). Consider* $\mathsf{RO}\text{-}KDF_n[\mathsf{H}_\mathrm{T}]$ *as in Figure 4 (with output length* $\mathsf{KLout} = \{\ell\}$ *in case of* T = NOF *resp. output length* $\mathsf{KLout} = [hl]$ *in case of* T = XOF*). Let req be* req_N *for* T = NOF *and be* req_X *for* T = XOF*. Then for any source collection* $\boldsymbol{\Sigma} = (\Sigma_1, \ldots, \Sigma_r)$ *with mapping* Σ-map*, each source outputting at most u elements, and any adversary* $\mathcal{A}$ *against the* kdf *security of* $\mathsf{RO}\text{-}KDF_n[\mathsf{H}_\mathrm{T}]$*, making at most* $\mathbf{q}_{\textsc{NewKey}}(\mathcal{A})$*,* $\mathbf{q}_{\textsc{SetKey}}(\mathcal{A})$ *and* $\mathbf{q}_{\mathrm{RO}}(\mathcal{A})$ *queries to oracles* NewKey*, and* SetKey *and to the random oracle* H_T*, we have*

$$\mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{RO}\text{-}KDF_n[\mathsf{H}_\mathrm{T}],\boldsymbol{\Sigma},req}(\mathcal{A}) \leq 2 \cdot \sum_{i=1}^{r} \mathbf{Adv}^{\mathrm{up}}_{\Sigma_i}(\mathcal{B}_i),$$

for adversaries $\mathcal{B}_i$ *with roughly the same running time as* $\mathcal{A}$*, and making at most* $\mathbf{q}_{\textsc{Predict}}(\mathcal{B}_i) = \mathbf{q}_{\textsc{NewKey}}(\mathcal{A}) \cdot \mathbf{q}_{\textsc{SetKey}}(\mathcal{A}) + n \cdot u \cdot \mathbf{q}_{\mathrm{RO}}(\mathcal{A})$ *queries to their prediction oracle.*

The above bound requires that all sources are unpredictable. This is because our security model gives strong security guarantees, namely, that even highly correlated inputs to the Ro$-KDF result in quasi-independent random answers. If the key material σ_j for some key with index v for only one source at some position p was predictable, then the adversary could register a dishonest key under a fresh index v' with $\sigma' = \sigma_j$ and identical context information, and repeat the query to the Ro$-KDF with v' replacing v. If the oracle returns real KDF outputs, then the answers would be identical; if the oracle returns fresh random values, then the answers would be different with overwhelming probability. This would allow the adversary to distinguish the two cases.

Proof. We assume that $\mathcal{A}$ does not make queries to oracle KDF. By Proposition 7, such queries can be simulated via Ro$-KDF oracle queries, increasing the advantage by a factor 2. It remains to argue that any queries to oracle Ro$-KDF result in random-looking answers even for the "real" game ($d = 1$) unless $\mathcal{A}$ makes a random oracle query H_T to the actual input of the KDF evaluation. We will argue that this contradicts unpredictability of the source collection, considering a family of adversaries $\mathcal{B}_1, \ldots, \mathcal{B}_r$ simultaneously. The adversary $\mathcal{B}_i$ against unpredictability receives a sample $\mathbf{z}[i].\alpha$ from source Σ_i. Its goal is to predict some value $\mathbf{z}.\sigma$ by running a black-box simulation of $\mathcal{A}$, simulating all steps of the KDF security game (sampling the vectors for the other sources) and the random oracle, with one exception: If the adversary $\mathcal{A}$ makes a query $(v_1, \ldots, v_n, L, \ell)$ to the Ro$-KDF then $\mathcal{B}_i$ for $d = 1$ simply gives a random answer. This is necessary since $\mathcal{B}_i$ may not know the key material if the evaluation involves source Σ_i. For $d = 0$, it proceeds according to the game.

Consider a query $(v_1, \ldots, v_n, L, \ell)$ of $\mathcal{A}$ to the Ro$-KDF oracle. We first argue that the input values $\langle \sigma_1, c_1, \sigma_2, c_2, \ldots, \sigma_n, c_n, L \rangle$ to the random oracle resulting from these oracle input values are always fresh. For this, observe first that requirement req_{NOF} resp. req_{XOF} implies that the adversary $\mathcal{A}$ can never repeat

queries for the same values $(v_1, \ldots, v_n, L)$. For T = NOF this follows immediately from req_{NOF} and the single admissible length input ℓ, in the case of T = XOF the freshness condition req_{XOF} disallows queries with identical $(v_1, \ldots, v_n, L)$ but different length values $\ell \neq \ell'$. Hence, since encoding is recoverable, we must always have different input values for the oracle.

Consider another (distinct) query $(v'_1, \ldots, v'_n, L', \ell')$ to the oracle. There are three cases in which repeated inputs to the underlying KDF can occur:

1. Assume that there exists a position index p such that $v_p \neq v'_p$ but such that the keys are honest, $\mathbf{k}[v_p].\mathsf{t} = \mathbf{k}[v'_p].\mathsf{t} = \mathsf{hon}$, but contain the same secret, $\mathbf{k}[v_p].\sigma = \mathbf{k}[v'_p].\sigma$. The requirement req_{ValidPos} ensures that both keys at position p must originate from the same source $i = \Sigma\text{-}\mathsf{map}(p)$. But then this gives a straightforward reduction to unpredictability for algorithm $\mathcal{B}_i$, because $\mathcal{B}_i$ immediately wins the unpredictability game if such σ-collisions occur within a source.

2. Assume that there exists an index p such that $v_p \neq v'_p$ and one key is honest, $\mathbf{k}[v_p].\mathsf{t} = \mathsf{hon}$, and one is dishonest, $\mathbf{k}[v'_p].\mathsf{t} = \mathsf{dishon}$, but they contain the same secret, $\mathbf{k}[v_p].\sigma = \mathbf{k}[v'_p].\sigma$. Then we can construct an algorithm $\mathcal{B}_i$ against the unpredictability of source $i = \Sigma\text{-}\mathsf{map}(p)$, by letting it immediately call its prediction oracle on the index in the source sample corresponding to the honest key, and the key material $\sigma^* = \mathbf{k}[v'_p].\sigma$ of the dishonest key.

   More specifically, note that any adversary $\mathcal{B}_i$ can keep track of all key materials of dishonest keys by observing $\mathcal{A}$'s call to oracle SetKey, and knows all sources and positions of (unknown) honest key material by observing the NewKey queries of $\mathcal{A}$. When $\mathcal{A}$ sets a new honest key for source i and index j, then $\mathcal{B}_i$ will call oracle Predict on position j and all dishonest key materials σ^* set before, resulting in at most $\mathbf{q}_{\textsc{SetKey}}(\mathcal{A})$ calls. When $\mathcal{A}$ sets a new dishonest key to value σ^*, then $\mathcal{B}_i$ will look up all the previously set honest keys and their position j in the source sample and call oracle Predict about j and σ^*, resulting in at most $\mathbf{q}_{\textsc{NewKey}}(\mathcal{A})$ calls for each of the $\mathbf{q}_{\textsc{SetKey}}(\mathcal{A})$ calls to SetKey. In either case, if an index p as above in a call to oracle RO$-KDF occurs, then $\mathcal{B}_i$ successfully predicts the secret key material of the corresponding source $i = \Sigma\text{-}\mathsf{map}(p)$.

3. Assume that there exists an index p such that $v_p \neq v'_p$, both keys are dishonest, $\mathbf{k}[v_p].\mathsf{t} = \mathbf{k}[v'_p].\mathsf{t} = \mathsf{dishon}$, but where the key material and context information are identical $\mathbf{k}[v_p].(\sigma,\mathsf{c}) = \mathbf{k}[v'_p].(\sigma,\mathsf{c})$. Note that such queries for the same position p are disallowed according to requirement req_{NoDColl}.

We can thus conclude that either one of our algorithms $\mathcal{B}_i$ breaks unpredicatability, or $\mathcal{A}$ loses because of requirement req_{NoDColl}, if we have different inputs $(v_1, \ldots, v_n, L, [\ell])$ mapping to the same encoding $\langle \sigma_1, c_1, \sigma_2, c_2, \ldots, \sigma_n, c_n, L, [\ell] \rangle$. Furthermore, the adversary cannot ask for distinct length values ℓ in the case of the NOF KDF because the set $\mathsf{KLout} = \{\ell\}$ of admissible output lengths is a singleton. For a XOF KDF, this follows from requirement req_{XOF}, disallowing length-prefix queries of otherwise identical values.

Now that we have established that all inputs $\langle \sigma_1, c_1, \sigma_2, c_2, \ldots, \sigma_n, c_n, L \rangle$ to the random oracle in RO$-KDF queries are distinct, the only possibility for the adversary to note a difference between the game with $d = 1$ and the simulated game with random responses is to query the random oracle on the input in question. The requirement req_{1HKey} guarantees that at least one honest key must be used in queries to RO$-KDF. Hence, a successfully distinguishing random oracle query by $\mathcal{A}$ must predict such a secret value.

Once more, we can construct adversaries $\mathcal{B}_i$ against unpredictability. For each query $\langle \sigma_1, c_1, \sigma_2, c_2, \ldots, \sigma_n, c_n, L \rangle$ of $\mathcal{A}$ to its random oracle, for each $p \in [n]$, query oracle Predict on all values $j \in [u]$ and σ_p. If $\mathcal{A}$ ever makes such a random oracle query, then $\mathcal{B}_i$ wins the unpredictability game.

We can now conclude that $\mathcal{A}$ cannot distinguish the games for $d = 0$ and $d = 1$ anymore because, in both cases, the oracle RO$-KDF returns fresh random answers. □

The following theorem shows that $\mathsf{RO}\text{-}\mathrm{KDF}_n[\mathsf{H}_\mathrm{T}]$ is a secure hybrid n-KDF. That is, that it is secure as long as at least one of the sources of key material is unpredictable. For this, we use the requirement req_{Hybrid}, combining req_{HybridKR} and $req_{1\times\mathsf{Key}}$ (cf. Table 1).

Theorem 9 (Hybrid KDF security based on random oracle)**.** *Let* H_T *be a hash function, which we model as a random oracle (with output length hl in case of* T = NOF *resp. unbounded output length in case of* T = XOF*, see Figure 15). Consider* $\mathsf{RO}\text{-}\mathit{KDF}_n[\mathsf{H}_\mathrm{T}]$ *as in Figure 4 (with output length* $\mathsf{KLout} = \{\ell\}$ *in case of* T = NOF *resp. output length* $\mathsf{KLout} \in [hl]$ *in case of* T = XOF*). Let req be* req_N *for* T = NOF *and be* req_X *for* T = XOF*. Then for any source collection* $\mathbf{\Sigma} = (\Sigma_1, \ldots, \Sigma_r)$ *with mapping* $\Sigma\text{-}\mathsf{map}$*, and*

any non-empty set $\mathcal{S} \subseteq [r]$ *there exist adversaries* $\mathcal{B}_i$ *for* $i \in \mathcal{S}$ *such that*

$$\mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{RO}\text{-}KDF_n[\mathsf{H}_T],\boldsymbol{\Sigma},req\wedge req_{\mathsf{Hybrid}}(\mathcal{S})}(\mathcal{A}) \leq 2 \cdot \min_{i\in\mathcal{S}} \left(\mathbf{Adv}^{\mathrm{up}}_{\Sigma_i}(\mathcal{B}_i)\right).$$

Adversaries $\mathcal{B}_i$ *have roughly the same running time as* $\mathcal{A}$*, and make at most* $\mathbf{q}_{\textsc{Predict}}(\mathcal{B}_i) = n \cdot u \cdot \mathbf{q}_{\mathrm{RO}}(\mathcal{A})$ *queries to their prediction oracles.*

Proof. We may again assume that adversary $\mathcal{A}$ never queries oracle KDF, causing a factor 2 in the security bound. Let $i \in \mathcal{S}$ be an index of a good source. We construct an adversary $\mathcal{B}_i$ against the unpredictability of source Σ_i. Adversary $\mathcal{B}_i$ receives a sample $\mathbf{z}_{\cdot\alpha}$ but does not have access to the secret key material $\mathbf{z}_{\cdot\sigma}$. Let $p_1, p_2, \ldots, p_k \in [n]$ be the positions in the KDF that Σ-map maps to source i; by assumption about the surjectivity of Σ-map at least one such position must exist. Then $\mathcal{B}_i$ does not know any of the key material inputs for these positions when simulating the oracles for $\mathcal{A}$. However, $\mathcal{B}_i$ samples all the data for the other sources in the simulation, including the ones in $\mathcal{S} \setminus \{i\}$. It can therefore map all indexes at other positions p (different from $p_1, p_2, \ldots, p_k$) for queries $(v_1, \ldots, v_n, L, \ell)$ of $\mathcal{A}$ to the simulated RO$-KDF oracle to the actual values $\mathbf{k}[v_p]_{\cdot(\sigma,\mathsf{c})}$; for all positions $p_1, p_2, \ldots, p_k$ adversary $\mathcal{B}_i$ simply leaves the value v_{p_j}. We call this the partial key-mapping of $(v_1, \ldots, v_n, L, \ell)$.

Adversary $\mathcal{B}_i$ now responds to $\mathcal{A}$'s oracle queries to RO$-KDF by simply returning a random key. All queries to SetKey and NewKey are handled locally by $\mathcal{B}_i$, keeping track of key values. The auxiliary information about keys from source i are given as input to $\mathcal{B}_i$ in the unpredictability game, and for keys from other sources $\mathcal{B}_i$ knows them from the internal sampling step, such that $\mathcal{B}_i$ can return these information to $\mathcal{A}$ for NewKey queries. In addition, for each query of $\mathcal{A}$ to its random oracle H about $\langle \sigma_1, c_1, \sigma_2, c_2, \ldots, \sigma_n, c_n, L\rangle$—we assume that $\mathcal{A}$ never repeats such a query for the same data—adversary $\mathcal{B}_i$ answers with a fresh random value. When $\mathcal{A}$ eventually stops, adversary $\mathcal{B}_i$ calls its oracle Predict on all values $j \in [u]$ and all inputs σ_{p_j} for $j = 1, 2, \ldots, k$ that have occurred in a query $\langle \sigma_1, c_1, \sigma_2, c_2, \ldots, \sigma_n, c_n, L\rangle$ to H. Overall these are at most $n \cdot u$ queries for each of the $\mathbf{q}_{\mathrm{RO}}(\mathcal{A})$ random oracle queries of $\mathcal{A}$. Then $\mathcal{B}_i$ also stops.

The simulation is perfect, unless

1. $\mathcal{A}$ makes a random oracle query about some actual input $\langle \sigma_1, c_1, \sigma_2, c_2, \ldots, \sigma_n, c_n, L\rangle$ used in the RO$-KDF oracle, possibly noting that the random oracle answer does not match, or
2. $\mathcal{A}$ manages to make distinct queries $(v_1, \ldots, v_n, L, \ell)$ and $(v'_1, \ldots, v'_n, L', \ell')$ to oracle RO$-KDF that map to the same value under the partial key-mapping.

For the former case note that this eventually results in a successful prediction attack, such that we can bound the probability of this happening by $\mathbf{Adv}^{\mathrm{up}}_{\Sigma_i}(\mathcal{B}_i)$. The reason is that $req_{\mathsf{HybridKR}}(\mathcal{S})$—which is part of $req_{\mathsf{Hybrid}}(\mathcal{S})$—stipulates that all key inputs in oracle RO$-KDF at positions $p_1, p_2, \ldots, p_k$ need to come from the good source $i \in \mathcal{S}$. Hence, if $\mathcal{A}$ ever makes such a random oracle query, $\mathcal{B}_i$ eventually makes the corresponding call to oracle Predict and wins the unpredictability game.

For the other case, that $\mathcal{A}$ makes colliding queries to oracle RO$-KDF, we argue that this would violate requirement $req_{1\times\mathsf{Key}}(\mathcal{S})$—which is part of $req_{\mathsf{Hybrid}}(\mathcal{S})$. Recall that this requirement stipulates that each key index v_{p_j} at the positions $p_1, p_2, \ldots, p_k$ corresponding to source Σ_i can only occur once among all queries to oracle RO$-KDF. Therefore, the partial key mapping, keeping the values v_{p_i} as is, can never map to the same input twice. If, on the other hand, two distinct indexes $v_{p_j} \neq v'_{p_j}$ would result in the same secret key material for source Σ_i, then this meant a collision in the key material and is also covered by the unpredictability term $\mathbf{Adv}^{\mathrm{up}}_{\Sigma_i}(\mathcal{B}_i)$.

Overall, the simulation of each adversary $\mathcal{B}_i$ is close to an actual attack of $\mathcal{A}$, with the difference in probabilities bounded from above by $\mathbf{Adv}^{\mathrm{up}}_{\Sigma_i}(\mathcal{B}_i)$. But in the simulation the secret bit d of oracle RO$-KDF is not used at all and thus completely hidden, such that $\mathcal{A}$ cannot obtain an advantage over the guessing probability. Since this holds for all adversaries $\mathcal{B}_i$, we get the minimum over all unpredictability advantages as an upper bound. □

5.2 MLS: n-KDF Combining Pre-Shared Keys

The MLS [7, Section 8.4] protocol uses the n-KDF given in Figure 5 to combine n pre-shared keys into a single key using HKDF (i.e., based on HMAC). The combiner, which we denote MLS-PSK-KDF, takes the n pre-shared keys psk_i, each associated with some identifying information called `PSKLabel` in [7] which we treat as context c_i; it does not take any label as input (i.e., $L = \varepsilon$), and the output length ℓ is fixed to hl, the output length of the hash function underlying HKDF. The KDF first derives a psk_input_i

MLS-PSK-KDF$((psk_1, c_1), \ldots, (psk_n, c_n), L = \varepsilon, \ell = hl)$

1 For $i = 1..n$:
2 $\quad psk\_extracted_i \leftarrow \mathsf{HKDF.Extract}(0, psk_i)$
3 $\quad psk\_input_i \leftarrow \mathsf{HKDF.Expand}(psk\_extracted_i, \ell || \texttt{"MLS 1.0 derived psk"} || c_i, hl)$
4 $psk\_secret_0 \leftarrow 0$
5 For $i = 1..n$:
6 $\quad psk\_secret_i \leftarrow \mathsf{HKDF.Extract}(psk\_input_i, psk\_secret_{i-1})$
7 $K \leftarrow psk\_secret_n$
8 Return K

Figure 5: MLS n-KDF combiner for pre-shared keys psk_i, based on HMAC.

value from each psk_i input, binding the key and context together. Then, in a cascading application of HKDF.Extract (i.e., HMAC), it combines the n psk_input_i values into a final value psk_secret_n, which is also the final output key K.

Assuming pre-shared keys function as a pseudorandom source, we show that MLS-PSK-KDF is a secure (NOF) KDF based on the dual-PRF security of HKDF.Extract (i.e., regular and swap-PRF security of HMAC), PRF security of HKDF.Expand (i.e., of HMAC, as the output length is fixed to hl), and the collision resistance of HMAC. Note that HMAC achieves dual-PRF security and collision resistance for the fixed key-input lengths used in MLS-PSK-KDF, as shown in [3].

Theorem 10 (KDF security of MLS-PSK-KDF)**.** *Let* MLS-PSK-KDF *be the NOF-n-KDF in Figure 5. Let Σ be a pseudorandom source and $\boldsymbol{\Sigma}$ be the $(n, 1)$-collection based on Σ. Then for any adversary $\mathcal{A}$ against the KDF security of* MLS-PSK-KDF *there exist adversaries $\mathcal{B}_1, \ldots, \mathcal{B}_6$ such that*

$$\begin{aligned}\mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{MLS\text{-}PSK\text{-}KDF},\boldsymbol{\Sigma},req_N}(\mathcal{A}) \leq 2 \cdot \Big(&\mathbf{Adv}^{\mathrm{pr}}_{\Sigma}(\mathcal{B}_1) + \mathbf{Adv}^{\text{swap-prf}}_{\mathsf{HKDF.Extract}}(\mathcal{B}_2) \\ &+ \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{HKDF.Expand}}(\mathcal{B}_3) + \mathbf{Adv}^{\mathrm{cr}}_{\mathsf{HMAC}}(\mathcal{B}_4) + \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{HKDF.Extract}}(\mathcal{B}_5) \\ &+ (n-1) \cdot \mathbf{Adv}^{\text{swap-prf}}_{\mathsf{HKDF.Extract}}(\mathcal{B}_6) + \frac{n \cdot (\mathbf{q}_{\text{RO\$-KDF}}(\mathcal{A}) + \mathbf{q}_{\text{KDF}}(\mathcal{A}))}{2^{8hl}}\Big).\end{aligned}$$

The adversaries $\mathcal{B}_1, \ldots, \mathcal{B}_6$ have running time roughly the same as $\mathcal{A}$ and the following query counts: $\mathbf{q}_{\text{New}}(\mathcal{B}_2) = \mathbf{q}_{\text{Fn}}(\mathcal{B}_2) = \mathbf{q}_{\text{NewKey}}(\mathcal{A})$*;* $\mathbf{q}_{\text{New}}(\mathcal{B}_3) = \mathbf{q}_{\text{NewKey}}(\mathcal{A})$ *and* $\mathbf{q}_{\text{Fn}}(\mathcal{B}_3) \leq n \cdot (\mathbf{q}_{\text{RO\$-KDF}}(\mathcal{A}) + \mathbf{q}_{\text{KDF}}(\mathcal{A}))$*;* $\mathbf{q}_{\text{New}}(\mathcal{B}_5), \mathbf{q}_{\text{Fn}}(\mathcal{B}_5) \leq n \cdot (\mathbf{q}_{\text{RO\$-KDF}}(\mathcal{A}) + \mathbf{q}_{\text{KDF}}(\mathcal{A}))$*;* $\mathbf{q}_{\text{New}}(\mathcal{B}_6), \mathbf{q}_{\text{Fn}}(\mathcal{B}_6) \leq \mathbf{q}_{\text{RO\$-KDF}}(\mathcal{A}) + \mathbf{q}_{\text{KDF}}(\mathcal{A})$*.*

Proof. We first apply Proposition 7, losing a factor of 2, to move to the KDF security game without real KDF oracle queries for some adversary $\mathcal{A}'$ with $\mathbf{q}_{\text{Or}}(\mathcal{A}') = \mathbf{q}_{\text{Or}}(\mathcal{A})$ for $\text{Or} \in \{\text{NewKey}, \text{SetKey}\}$ and $\mathbf{q}_{\text{RO\$-KDF}}(\mathcal{A}') = \mathbf{q}_{\text{RO\$-KDF}}(\mathcal{A}) + \mathbf{q}_{\text{KDF}}(\mathcal{A})$. We then bound the advantage of $\mathcal{A}'$ through a series of game hops, beginning with the "real" ($d = 1$) game $\mathbf{G}_0 := \mathbf{G}^{\text{kdf-1}}_{\mathsf{MLS\text{-}PSK\text{-}KDF},\boldsymbol{\Sigma},req_N \wedge req_{\mathsf{NoReal}}}$ and ending with the "random" ($d = 0$) game.

We begin by replacing in game $\mathbf{G}_0$ the source key material in the vector $\mathbf{z}_1$ with uniformly random values, a step we straightforwardly bound by the pseudorandomness of source Σ:

$$\Pr[\mathbf{G}_0] - \Pr[\mathbf{G}_1] \leq \mathbf{Adv}^{\mathrm{pr}}_{\Sigma}(\mathcal{B}_1).$$

In the second hop, game $\mathbf{G}_2$, we replace the $psk\_extracted_i$ values computed from honest source keys psk_i by uniformly random values. To bound this hop, consider the following reduction $\mathcal{B}_2$ to the multi-user swap-PRF security of HKDF.Extract. Upon each NewKey call of $\mathcal{A}'$, $\mathcal{B}_2$ registers a new key in the multi-user swap-PRF game, taking the place of psk_i. It calls Fn on input 0 for each of these keys and substitutes the results for the $psk\_extracted_i$ values. The rest of the game is simulated truthfully, in particular, $\mathcal{B}_2$ handles all dishonest secrets registered via SetKey itself. We have

$$\Pr[\mathbf{G}_1] - \Pr[\mathbf{G}_2] \leq \mathbf{Adv}^{\text{swap-prf}}_{\mathsf{HKDF.Extract}}(\mathcal{B}_2).$$

For game $\mathbf{G}_3$, we next replace HKDF.Expand with a random function where it is used in line 3 of Figure 5 to compute values psk_input_i from "honest" $psk\_extracted_i$ (i.e., those derived from honest psk_i and replaced by random in $\mathbf{G}_2$). Note that this in particular leads to the computed "honest" psk_input_i

values being independently and randomly sampled, except when some psk_i key is reused. We bound this by a reduction $\mathcal{B}_3$ to the multi-user (regular) PRF security of HKDF.Expand. Upon each NewKey call of $\mathcal{A}'$, $\mathcal{B}_3$ register a new $psk\_extracted_i$ key in the multi-user regular PRF game. It calls Fn on input $\ell||$"MLS 1.0 derived psk"$||c_i||$0x01 whenever a Ro$-KDF call involves $psk\_extracted_i$; this amounts to at most n calls to Fn per Ro$-KDF call of $\mathcal{A}'$ (which translates to n calls per KDF or Ro$-KDF call of $\mathcal{A}$). Again $\mathcal{B}_3$ handles dishonest secrets from the SetKey oracle itself. We have

$$\Pr[\mathbf{G}_2] - \Pr[\mathbf{G}_3] \leq \mathbf{Adv}^{\text{prf}}_{\mathsf{HKDF.Expand}}(\mathcal{B}_3).$$

In game $\mathbf{G}_4$, we abort the game (by immediately returning 0), if $\mathcal{A}'$ creates a collision in any psk_input_i value for some position i computed from *dishonest* psk_i values (registered via the SetKey). That is, we abort if within or across any Ro$-KDF query/queries, there are two sets of inputs $(psk_i, c_i) \neq (psk'_i, c'_i)$ such that

$$\begin{aligned}&\mathsf{HKDF.Expand}(\mathsf{HKDF.Extract}(0, psk_i), \ell||\texttt{"MLS 1.0 derived psk"}||c_i)\\ =\ &\mathsf{HKDF.Expand}(\mathsf{HKDF.Extract}(0, psk'_i), \ell||\texttt{"MLS 1.0 derived psk"}||c'_i).\end{aligned}$$

Rewriting HKDF in terms of its HMAC building blocks, this means

$$\begin{aligned}&\mathsf{HMAC}(\mathsf{HMAC}(0, psk_i), \ell||\texttt{"MLS 1.0 derived psk"}||c_i||\texttt{0x01})\\ =\ &\mathsf{HMAC}(\mathsf{HMAC}(0, psk'_i), \ell||\texttt{"MLS 1.0 derived psk"}||c'_i||\texttt{0x01}).\end{aligned}$$

We bound this step by a reduction $\mathcal{B}_4$ to the collision resistance of HMAC, noting that in the reduction, $\mathcal{B}_4$ performs the relevant computations itself and can hence detect such collisions. We have

$$\Pr[\mathbf{G}_3] - \Pr[\mathbf{G}_4] \leq \mathbf{Adv}^{\text{cr}}_{\mathsf{HMAC}}(\mathcal{B}_4).$$

Next, in game $\mathbf{G}_5$, within each Ro$-KDF query, we replace HKDF.Extract with a random function where it is used in line 6 of Figure 5 to compute values psk_secret_i from "honest" psk_input_i (i.e., those derived from honest psk_i and replaced by outputs of a random function in $\mathbf{G}_3$). Similar to before, a reduction $\mathcal{B}_5$ to the PRF security of HKDF.Extract registers a new key for each of these honestly-derived psk_input_i values (which are independent and random per $\mathbf{G}_3$, except for repeated psk_i key material); there are at most n keys per Ro$-KDF call of $\mathcal{A}'$ (if in each call all keys are honest and previously unused). It then calls its Fn oracle on psk_secret_{i-1} to obtain psk_secret_i, computing the steps involving dishonest keys itself; this again amounts to at most n calls to Fn per Ro$-KDF call of $\mathcal{A}'$. We have

$$\Pr[\mathbf{G}_4] - \Pr[\mathbf{G}_5] \leq \mathbf{Adv}^{\text{prf}}_{\mathsf{HKDF.Extract}}(\mathcal{B}_5).$$

Finally, in game $\mathbf{G}_6$, we replace the final output key $K = psk\_secret_n$ by a uniformly random value in any Ro$-KDF query. We let a reduction $\mathcal{B}_6$ against the swap-PRF security of HKDF.Extract "complete" the HKDF.Extract derivation chain from the last psk_input_i being derived from an honest psk_i (and hence an output of a random function by game $\mathbf{G}_5$). Iterating over $i = 1$ to $n-1$, $\mathcal{B}_6$ given i registers a new key in place of each such psk_input_i (at most one per Ro$-KDF call of $\mathcal{A}'$) and computes psk_input_{i+1} using its Fn (again at most once per Ro$-KDF call of $\mathcal{A}'$).

What remains to be argued is that in completing the chain, there is no collision in the derivation chain affecting the independence of the final psk_secret_n value. Denoting this such collision event as coll, and summing the reductions over $i \in [1, n-1]$, we have

$$\Pr[\mathbf{G}_5] - \Pr[\mathbf{G}_6] \leq (n-1) \cdot \mathbf{Adv}^{\text{swap-prf}}_{\mathsf{HKDF.Extract}}(\mathcal{B}_6) + \Pr[\mathsf{coll}].$$

We complete this step by bounding the remaining collision probability.

Recall that by definition, the inputs $((j_1, \ldots, j_n), L, \ell)$ between any two Ro$-KDF must differ somewhere. Given that both the label $L = \varepsilon$ and the length $\ell = hl$ are fixed in the MLS n-KDF combiner, we can ignore them here. Hence, between any two queries, the key indexes (j_i) must differ for some position pos. If both of them are honest, then by $\mathbf{G}_3$ the resulting psk_input_i values are derived from independent random keys, making them independent, too. If both of them are dishonest, then by the req_{NoDColl} requirement either the key material σ or the context c values must be distinct. So we have by $\mathbf{G}_4$ that the resulting psk_input_i values do not collide and hence distinct PRF inputs are used in this step (in the application of the swap-PRF game). Finally, the probability of a dishonest psk_input_i value

$\underline{\mathsf{ETSI\text{-}CatKDF}((k_1,(c_1,MA_1,MB_1)),\ldots,(k_n,(c_n,MA_n,MB_n)),(psk,\varepsilon),L,\ell)}$

1 $secret \leftarrow psk||k_1||\ldots||k_n$
2 $c \leftarrow \langle c_1,\ldots,c_n\rangle$, $MA = \langle MA_1,\ldots,MA_n\rangle$, $MB = \langle MB_1,\ldots,MB_n\rangle$
3 $f_{\text{context}} \leftarrow f(c, MA, MB)$
4 $K \leftarrow \mathsf{HKDF}(secret, L, f_{\text{context}}, \ell)$ // label L used as salt input in HKDF
5 Return K

Figure 6: ETSI-CatKDF based on HKDF.

colliding with an honestly derived one in that position, replaced with random in $\mathbf{G}_3$, is $\frac{1}{2^{8hl}}$. Hence, across all n positions across the $\mathbf{q}_{\text{RO\$-KDF}}(\mathcal{A}') = \mathbf{q}_{\text{RO\$-KDF}}(\mathcal{A}) + \mathbf{q}_{\text{KDF}}(\mathcal{A})$ oracle queries, we can upper bound such collisions by

$$\frac{n \cdot (\mathbf{q}_{\text{RO\$-KDF}}(\mathcal{A}) + \mathbf{q}_{\text{KDF}}(\mathcal{A}))}{2^{8hl}}.$$

Observing that $\mathbf{G}_6$ equals the "random" game $\mathbf{G}^{\text{kdf-0}}_{\mathsf{MLS\text{-}PSK\text{-}KDF},\boldsymbol{\Sigma},req_N \wedge req_{\mathsf{NoReal}}}$ completes the proof. □

5.3 ETSI-CatKDF: Concatenation Combiner for Generic Sources

In 2020, ETSI published a technical specification describing their take on building hybrid key exchange systems: TS 103 744 version 1.1.1 [36]. The specification assumes that several key exchange protocols are executed, each deriving a shared secret k_i. Parties may also hold a pre-shared secret *psk* which—if not present—is replaced by an empty string. The keys are combined via an n-KDF that we call ETSI-CatKDF by concatenating all secret key material. Notably, the standard only considers single-round KEM-based key exchange protocols to derive keys k_i,resulting in messages MA_i and MB_i per key, which we treat as context. The ETSI proposal employs a context formatting function $f$which hashes its length-encoded inputs.

While ETSI proposes ETSI-CatKDF specifically for single time use as part of a hybrid key exchange, the interface of the KDF suggests that a much broader usage is possible. In fact, for example, the German Federal Office for Information Security points readers of its technical specification to ETSI-CatKDF for the general purpose use case of hybridisation [39]. For this reason, we analyze ETSI-CatKDF is the more general context of a generic n-KDF.

The concatenation construction ETSI-CatKDF is displayed in Figure 6. It concatenates the source key materials to get *secret*, then encodes the context information via function f to derive f_{context}, and finally applies HKDF to key *secret*, salt input L, context f_{context}, and length parameter ℓ. To match our formalization of an n-KDF we assume that we have n sources, where the ith source Σ_i generates key material $\sigma_i = k_i$, context information (c_i, MA_i, MB_i), and some auxiliary information α_i. If the optional pre-shared key *psk* is available, then we assume that this is output by another source Σ_{n+1} as $\sigma_{n+1} = psk$, with empty context $c_{n+1} = \varepsilon$ and some auxiliary information α_{n+1}.

Our first observation is that the construction ETSI-CatKDF is *insecure* according to our security model when interpreting it as a generic n-KDF. The reason is that any queries to oracle RO\$-KDF that differ only in the labels L (for $|L| < B$) and $L' = L\|\texttt{0x00}$ yield the same output, allowing an adversary to immediately win. Recall that the initial extraction step of HKDF pads the labels with sufficiently many zero-bytes such that both padded labels in the queries become identical. A similar effect occurs if one label is longer than B octets and gets hashed first: Then $L = \mathsf{H}(L')$ of length B and L' of more than B octets would yield the same processed salt input and thus identical answers of RO\$-KDF for different inputs. Since these attacks depend on the possible choices of the value L, whether or not they translate into practical vulnerabilities depends on how ETSI-CatKDF is used in real implementations. Yet, from our viewpoint, these attacks reveal a misconception in the deployment of the HMAC salt values in ETSI-CatKDF, rendering the construction's security brittle. Another issue, which we do not further explore here, is that keys in *secret* are simply concatenated, without length encoding. Then keys k_1, k_2 and $k_1' = k_1\|k_2$, $k_2' = \varepsilon$ would yield the same value *secret*.

We do not formalize the attacks above but instead turn to a restricted security statement for fixed-size labels and keys. Notably, the recently published version 1.2.1 of ETSI's TS 103 744 [37] mandates labels to have fixed length; the key material arising in the targeted hybrid key exchange setting are also fixed-length in nature. We also restrict ourselves here to the case of an empty pre-shared secret *psk*, such that we really have an n-KDF.

Theorem 11 (KDF security of ETSI-CatKDF)**.** *Consider* ETSI-CatKDF *with empty pre-shared key* ($psk = \varepsilon, c_{n+1} = \varepsilon$) *and fixed-length labels* $L \in \mathbb{O}^{hl}$*, and model* HMAC *with octet output size* hl *as a random oracle. Let* Σ *be a pseudorandom source with sample size* u *and* $\mathsf{Skm} = \mathbb{O}^m$ *and* $\mathbf{\Sigma} = (\Sigma)$ *be the collection with* $\Sigma\text{-}\mathsf{map}(i) = 1$ *for all* $i \in [n]$*. For any adversary* $\mathcal{A}$ *against the* kdf *security of* ETSI-CatKDF*, making at most* $\mathbf{q}_{\mathsf{HMAC}}(\mathcal{A})$ *to random oracle* HMAC*, there exists an adversary* $\mathcal{B}$ *such that*

$$\begin{aligned}\mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{ETSI\text{-}CatKDF},\mathbf{\Sigma},req_X}(\mathcal{A}) \leq 2 \cdot \Big(&\mathbf{Adv}^{\mathrm{pr}}_{\Sigma}(\mathcal{B}) + u^2 \cdot 2^{-8m} \\ &+ \mathbf{q}_{\mathsf{HMAC}}(\mathcal{A}) \cdot \mathbf{q}_{\text{RO\$-KDF}+KDF}(\mathcal{A}) \cdot (2^{-8m} + 2^{-8hl}) \\ &+ (\mathbf{q}_{\text{RO\$-KDF}+KDF}(\mathcal{A}))^2 \cdot 2^{-8hl}\Big).\end{aligned}$$

where $\mathbf{q}_{\text{RO\$-KDF}+KDF}(\mathcal{A}) = \mathbf{q}_{\text{RO\$-KDF}}(\mathcal{A}) + \mathbf{q}_{\mathrm{KDF}}(\mathcal{A})$*. Adversary* $\mathcal{B}$ *has roughly the same running time as* $\mathcal{A}$*.*

The proof can be easily adapted to the case that either all labels are strictly shorter than hl, or that all labels are strictly larger than hl, following an argument analogously to [3] for dual-PRFs.

Proof. We may first assume that $\mathcal{A}$ does not make any queries to oracle KDF. This can always be ensured and increases the advantage of $\mathcal{A}$ by at most a factor of 2 according to Proposition 7. It also moves the queries to oracle KDF to queries to RO\$-KDF. We next bound the advantage of $\mathcal{A}$ through a series of game hops, beginning with the "real" ($d = 1$) game $\mathbf{G}_0 := \mathbf{G}^{\text{kdf-1}}_{n\text{-}\mathsf{KDF}_{\mathsf{ETSI\text{-}CatKDF},\mathbf{\Sigma},req_X \wedge req_{\mathsf{NoReal}}}}$, and finishing with the "random" ($d = 0$) game.

We begin by replacing in game $\mathbf{G}_0$ all u values σ output by the source Σ at the beginning of the game by independent random values σ of m octets each. Call this $\mathbf{G}_1$. We can reduce this to the pseudorandomness of the source, $\mathbf{Adv}^{\mathrm{pr}}_{\mathbf{\Sigma}}(\mathcal{B})$, by introducing adversary $\mathcal{B}$:

$$\Pr[\,\mathbf{G}_0\,] - \Pr[\,\mathbf{G}_1\,] \leq \mathbf{Adv}^{\mathrm{pr}}_{\mathbf{\Sigma}}(\mathcal{B}).$$

In the next game hop to game $\mathbf{G}_2$ we immediately abort (and declare $\mathcal{A}$ to lose) if it queries its random oracle HMAC about some input $(L, secret)$ matching the extraction input in any of the $\mathbf{q}_{\text{RO\$-KDF}}(\mathcal{A}) + \mathbf{q}_{\mathrm{KDF}}(\mathcal{A})$ challenge queries to oracle RO\$-KDF. Note that $secret = k_1 \| \ldots \| k_n$ in all such oracle queries, and according to requirement $req_{1\mathsf{HKey}}$ there must be at least one honest input key k_i. Furthermore, this key is now an unknown random string of m octets by game $\mathbf{G}_1$. This can be seen as follows: For each query to the RO\$-KDF we map the vector $(v_1, \ldots, v_n)$ of indexes of input keys to a vector of entries, either the index j_i for honest keys, or the actual value k_i for dishonest keys. We call this the *sanitized secret*. Then we keep a random oracle table for such vectors (with the label in the first entry) and pick a fresh value for each new query. It follows that the answer only depends on the index of honest keys (and honest keys can only be assigned once, such that the index of honest keys points to a specific dedicated content). Thus, the probability that any of the $\mathbf{q}_{\mathsf{HMAC}}(\mathcal{A})$ queries to the random oracle coincides with any of these inputs, is at most

$$\Pr[\,\mathbf{G}_1\,] - \Pr[\,\mathbf{G}_2\,] \leq \mathbf{q}_{\mathsf{HMAC}}(\mathcal{A}) \cdot (\mathbf{q}_{\text{RO\$-KDF}}(\mathcal{A}) + \mathbf{q}_{\mathrm{KDF}}(\mathcal{A})) \cdot 2^{-8m}.$$

In the next game hop to $\mathbf{G}_3$, we abort if there are two honest keys mapping to the same value. Since there are at most u honest keys, we get by the birthday bound:

$$\Pr[\,\mathbf{G}_2\,] - \Pr[\,\mathbf{G}_3\,] \leq u^2 \cdot 2^{-8m}.$$

We say that two queries $((v_1, \ldots, v_n), L)$, $((v'_1, \ldots, v'_n), L)$ of $\mathcal{A}$ to oracle RO\$-KDF for the same label are *equivalent* if $(v_1, \ldots, v_n)$ and $(v'_1, \ldots, v'_n)$ map to the same sanitized vector. Note that this means that they coincide on indexes for honest keys and on the actual content of dishonest keys, even though the indexes for dishonest keys may differ. Note that we can decide if two queries are equivalent by observing the queries to oracle SETKEY. Now, any two vectors $(v_1, \ldots, v_n)$, $(v'_1, \ldots, v'_n)$ not in the same equivalence class yield an independent random oracle value PRK output by HKDF.Extract. The reason is that, if the vectors differ in indexes of honest keys, then they also differ in the *secret* value by the previous game. And if they only differ in dishonest keys, then, as they do not lie in the same equivalence class, they must have different values in the entries of dishonest keys. Furthermore, since $\mathcal{A}$ has never queried the random oracle about the value *secret*, the adversary is perfectly oblivious about each PRK.

In game $\mathbf{G}_4$ we let $\mathcal{A}$ lose if it either queries the random oracle HMAC about one of the extracted keys PRK at some point or if two extracted keys collide. There are at most $\mathbf{q}_{\mathsf{HMAC}}(\mathcal{A})$ random oracle queries and

$\underline{\mathsf{X3DH\text{-}KDF}((dh_1, c_1), (dh_2, c_2), (dh_3, c_3), (dh_4, c_4), L = \varepsilon, \ell = 256)}$

1 If $dh_4 = \perp$: $secret \leftarrow dh_1 || dh_2 || dh_3$
2 Else: $secret \leftarrow dh_1 || dh_2 || dh_3 || dh_4$
3 $K \leftarrow \mathsf{HKDF}(F || secret, 0^{hl}, \texttt{info}, \ell)$
4 Return K

Figure 7: Key derivation function in Signal's X3DH handshake protocol, based on HKDF. F is a curve-dependent constant, `info` is a fixed per-application label.

---

at most $(\mathbf{q}_{\text{Ro\$-KDF}}(\mathcal{A}) + \mathbf{q}_{\text{KDF}}(\mathcal{A}))$ extracted keys in total, each of octet size hl, such that the probability of querying HMAC about such a value is at most $\mathbf{q}_{\mathsf{HMAC}}(\mathcal{A}) \cdot (\mathbf{q}_{\text{Ro\$-KDF}}(\mathcal{A}) + \mathbf{q}_{\text{KDF}}(\mathcal{A})) \cdot 2^{-8hl}$. Furthermore, the independent random extracted keys collide with probability at most $(\mathbf{q}_{\text{Ro\$-KDF}}(\mathcal{A}) + \mathbf{q}_{\text{KDF}}(\mathcal{A}))^2 \cdot 2^{-8hl}$. Therefore,

$$\begin{aligned}\Pr[\,\mathbf{G}_3\,] - \Pr[\,\mathbf{G}_4\,] \leq\ & \mathbf{q}_{\mathsf{HMAC}}(\mathcal{A}) \cdot (\mathbf{q}_{\text{Ro\$-KDF}}(\mathcal{A}) + \mathbf{q}_{\text{KDF}}(\mathcal{A})) \cdot 2^{-8hl} \\ & + (\mathbf{q}_{\text{Ro\$-KDF}}(\mathcal{A}) + \mathbf{q}_{\text{KDF}}(\mathcal{A}))^2 \cdot 2^{-8hl}.\end{aligned}$$

Note that collisions for different context information are impossible because f_{context} injectively encodes any context information. Furthermore, each evaluation in the iterations of HKDF.Expand appends a fixed-size octet $[i]_1$ for the counter, such that collisions within one expansion step cannot happen either. It follows that all evaluations of HMAC in the expansion step are for different inputs. Given in addition that $\mathcal{A}$ never queries the random oracle HMAC about the extracted keys, the outputs of random oracle HMAC in the expansion step are thus all independently and randomly distributed. It follows that the winning probability of $\mathcal{A}$ in game $\mathbf{G}_4$ is upper bounded by the probability of winning the "random" game ($d = 0$). Collecting the probabilities now yields the claim. □

5.4 Signal X3DH: 4-KDF Combining Diffie–Hellman Secrets

Next, we take a look at the approach employed in Signal's X3DH key exchange protocol [49] to combine multiple Diffie–Hellman secrets. This combiner, which we denote as X3DH-KDF, is a 4-KDF (with optional fourth input) built on top of HKDF as shown in Figure 7. Its purpose is to derive one shared secret from three to four correlated elliptic-curve Diffie-Hellman secrets, all originating from the same DH source (concretely, elliptic-curve DH secrets, all from either X25519 or X448). X3DH-KDF first concatenates the DH secrets into a string *secret*, which is prepended with a constant F identifying the curve and then used as the (single) input key material for HKDF. (Note that *secret* is uniquely encoded thanks to the fixed length of the DH secrets.) The remaining inputs to HKDF are fixed to a constant-zero salt 0^{hl}, an application-specific label `info`, and an output length of $\ell = 32$ bytes.

In X3DH, parties hold three types of Diffie–Hellman keys: long-term keys (g^a, g^b, ...), semi-static keys (g^s, ...), and ephemeral keys (g^x, g^y, ...). A session key between two parties Alice and Bob is derived using X3DH-KDF from $dh_1 = g^{as}$, $dh_2 = g^{xb}$, $dh_3 = g^{xs}$, and (optionally) $dh_4 = g^{xy}$, where g^a, g^x are Alice's long-term and ephemeral shares and g^b, g^s, g^y are Bob's long-term, semi-static, and ephemeral shares (where g^y might not be available). We denote the Diffie–Hellman source underlying X3DH for some group G of order q between p parties, each running at most s sessions and using at most t many semi-static keys as $\Sigma_{\mathsf{X3DH}}^{G,p,s,t}$. A sample from $\Sigma_{\mathsf{X3DH}}^{G,p,s,t}$ contains all pair-wise combinations of the parties' long-term keys and semi-static keys (like $\sigma = g^{as}$) as well as those between ephemeral keys and long-term (g^{xb}), semi-static (g^{xs}), and ephemeral (g^{xy}) keys, respectively; i.e., $u = p^2t + p^2s + p^2st + ps^2$ many entries. We assume that the order of these keys is known to an adversary, who can hence directly index into the $\Sigma_{\mathsf{X3DH}}^{G,p,s,t}$ sample to select any particular DH combination of its choice. X3DH does not use context, so $c = \varepsilon$ in tuples produced by $\Sigma_{\mathsf{X3DH}}^{G,p,s,t}$, but the auxiliary information α includes the DH shares (e.g., g^a, g^s for a long-term–semi-static combination), reflecting that the adversary observes those in the X3DH handshake.

The source $\Sigma_{\mathsf{X3DH}}^{G,p,s,t}$ is unpredictable, following for adversaries making a *single* prediction query only from the computational Diffie–Hellman (CDH) assumption in G, and for general adversaries from the gap Diffie–Hellman (GapDH) assumption in G, modulo a term accounting for colliding DH secrets. We prove this in Appendix D.1.

Theorem 12 (KDF security of X3DH-KDF)**.** *Let* X3DH-KDF *be the* 4-KDF *in Figure 7. Let* $\Sigma_{\mathsf{X3DH}}^{G,p,s,t}$ *be the Diffie–Hellman source underlying X3DH as defined above, let* $\mathbf{\Sigma}$ *be the* $(4,1)$*-source collection based*

on $\Sigma_{\mathsf{X3DH}}^{G,p,s,t}$*, and model* HMAC *as a random oracle. Then for any adversary* $\mathcal{A}$ *against the KDF security of* X3DH-KDF*, there exists an adversary* $\mathcal{B}$ *(making one* PREDICT *query) such that*

$$\mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{X3DH\text{-}KDF},\mathbf{\Sigma},req_N}(\mathcal{A}) \leq 4\mathbf{q}_{\textsc{NewKey}}(\mathcal{A}) \cdot \big(\mathbf{q}_{\textsc{SetKey}}(\mathcal{A}) + 4\mathbf{q}_{\mathrm{RO}}(\mathcal{A})\big) \cdot \mathbf{Adv}^{\mathrm{up}}_{\Sigma_{\mathsf{X3DH}}^{G,p,s,t}}(\mathcal{B}).$$

Proof. Recall that HKDF is defined based on HMAC, so given the fixed output length $\ell = hl$, the key derivation on line 3, Figure 7 can be rewritten as

$$K \leftarrow \mathsf{HMAC}(\ \mathsf{HMAC}(0^{hl}, F||secret),\ \texttt{info}||\texttt{0x01}).$$

Since X3DH-KDF does not use a label (permitting only a single RO$-KDF call for any combination of secrets) and the outer HMAC call uses a constant second input, we can focus for security on the inner HMAC call: if that inner call yields a value *PRK*, independent and indistinguishable from random for each RO$-KDF call, then the output of the outer call is also indistinguishable from random, since HMAC is modeled as a random oracle and inner and outer calls are domain-separated (via F and `info`).

The proof proceeds via a sequence of games G_0–G_3. Game G_0 is identical to $\mathbf{G}_1^{\text{kdf-X3DH-KDF},\mathbf{\Sigma},req_N}$, i.e., the "real" n-KDF game, with HMAC modeled as a random oracle.

The first step of the proof is to apply Proposition 7, losing a factor of 2, to move to game G_1, which is the KDF security game without "real" KDF oracle queries, for an adversary $\mathcal{A}'$ with $\mathbf{q}_{\text{RO\$-KDF}}(\mathcal{A}') = \mathbf{q}_{\text{RO\$-KDF}}(\mathcal{A}) + \mathbf{q}_{\text{KDF}}(\mathcal{A})$.

Next, let game G_2 be identical to G_1, except that of computing the inner HMAC values $PRK = \mathsf{HMAC}(0^{hl}, F||secret)$, these are replaced with independent and uniform values $PRK \leftarrow\!\!\$\ \mathbb{O}^{hl}$. This change can be detected by $\mathcal{A}'$ only through (1) forcing a collision between the *secret* values in two RO$-KDF calls, or (2) making a random oracle query of the form $(0^{hl}, F||secret)$ involving the value *secret* used in an RO$-KDF call. Call these two events bad_1 and bad_2, respectively, then $\Pr[\mathrm{G}_1] \leq \Pr[\mathrm{G}_2] + \Pr[\mathsf{bad}_1 \vee \mathsf{bad}_2]$. Observe that due to the format of *secret* and collisions among dishonest key material being disallowed by req_{NoDColl}, bad_1 can only be triggered by colliding honest key material, or across dishonest and honest key material.

We bound $\Pr[\mathsf{bad}_1 \vee \mathsf{bad}_2]$ via a reduction $\mathcal{B}$ to the unpredictability of $\Sigma_{\mathsf{X3DH}}^{G,p,s,t}$ as follows. Instead of sampling a source key material itself, $\mathcal{B}$ uses the auxiliary information obtained in the unpredictability game to simulate NEWKEY queries. It answers all RO$-KDF queries by sampling the inner HMAC value *PRK* at random. When $\mathcal{A}'$ stops, $\mathcal{B}$ picks at random $b \in \{1, 2\}$ as well as one of the key material indexes j which $\mathcal{A}'$ registered via the NEWKEY oracle.

- If $b = 1$, $\mathcal{B}$ further picks at random one of the SETKEY queries $\mathcal{A}'$ made, with source key material σ_i. It queries (j, σ_i) to it PREDICT oracle. If $\mathcal{A}'$ triggers bad_1 via colliding dishonest and honest key material, then $\mathcal{B}$ will pick these with probability $\frac{1}{\mathbf{q}_{\textsc{NewKey}}(\mathcal{A}) \cdot \mathbf{q}_{\textsc{SetKey}}(\mathcal{A})}$. If honest key material collides, $\mathcal{B}$ immediately wins the unpredictability anyway.

- If $b = 2$, $\mathcal{B}$ further picks at random one of the random oracle queries of the form $\mathsf{HMAC}(0^{hl}, F||secret)$ that $\mathcal{A}'$ made. It parses $secret = dh_1||dh_2||dh_3||dh_4$ (where dh_4) is optional, picks one of the dh values at random, and queries (j, dh) to its PREDICT oracle.

  If $\mathcal{A}'$ triggers bad_2, then we know that one random oracle call is on a value $secret = dh_1||dh_2||dh_3||dh_4$ (with dh_4 optional) used in an RO$-KDF call. By req_{1HKey}, we know that one of the involved dh_i secrets must be honest, i.e., corresponding to the secret registered via some NEWKEY query. Hence with probability $\frac{1}{4 \cdot \mathbf{q}_{\textsc{NewKey}}(\mathcal{A}) \cdot \mathbf{q}_{\mathrm{RO}}(\mathcal{A})}$, $\mathcal{B}$ will pick the index of the honest secret involved and the random oracle call in which $\mathcal{A}$ queries it, winning the unpredictability game.

Combining both cases, we have

$$\begin{aligned}\Pr[\mathsf{bad}_1 \vee \mathsf{bad}_2] &\leq \Pr[\mathsf{bad}_1] + \Pr[\mathsf{bad}_2] \\ &\leq 2\mathbf{q}_{\textsc{NewKey}}(\mathcal{A}) \cdot (\mathbf{q}_{\textsc{SetKey}}(\mathcal{A}) + 4\mathbf{q}_{\mathrm{RO}}(\mathcal{A})) \cdot \mathbf{Adv}^{\mathrm{up}}_{\Sigma_{\mathsf{X3DH}}^{G,p,s,t}}(\mathcal{B}).\end{aligned}$$

Finally, in game G_3, we replace all responses to RO$-KDF queries with random, uniform strings of octet length hl. Since, in the outer HMAC computation, the first inputs at this point are independent random strings of octet length hl and the second inputs are a constant, this does not change the view of $\mathcal{A}'$, hence $\Pr[\mathrm{G}_2] = \Pr[\mathrm{G}_3]$. Observing that G_3 is equivalent to to the "random" n-KDF game $\mathbf{G}_0^{\text{kdf-X3DH-KDF},\mathbf{\Sigma},req_N}$ establishes the bound. □

$\underline{\mathbf{CtX}[\mathsf{Combine},\mathsf{Expand}]((\sigma_1,c_1),\ldots,(\sigma_n,c_n),L,\ell)}$

$PRK \leftarrow \mathsf{Combine}((\sigma_1,c_1),\ldots,(\sigma_n,c_n),\varepsilon,\ell_{PRK})$

$K \leftarrow \mathsf{Expand}(PRK,L,\ell)$

Return K

Figure 8: Combine-then-Expand (**CtX**) construction $\mathbf{CtX}[\mathsf{Combine},\mathsf{Expand}]$ of an n-KDF from comb-KDF Combine and function family Expand.

Remark 13. We note that Signal's X3DH-KDF *could* achieve tighter KDF security if it included the Diffie–Hellman shares as *context* in the inner HMAC call (i.e., HKDF.Extract), in some unambiguous way. (This is akin to the random-oracle–based n-KDF construction discussed in Section 5.1.) That way, the reduction to unpredictability in the proof of Proposition 12 would not need to guess which honest key corresponds to which random oracle query made by the adversary: Seeing the DH shares as context in an HMAC random oracle call, it could directly identify the corresponding key material and make one PREDICT query on that key material (per random oracle query of $\mathcal{A}$). This strategy avoids the $4\mathbf{q}_{\text{NEWKEY}}(\mathcal{A})\mathbf{q}_{\text{RO}}(\mathcal{A})$ loss in the bound, which can be notable in practical terms, relying on the GapDH instead of CDH assumption for multi-challenge unpredictability (cf. Lemma 31).

This approach would mirror tight proof strategies in the random oracle model for Diffie–Hellman key exchange [27]. Context being included in the key derivation has also been assumed in a tight security analysis of a variant of X3DH by Kiltz et al. [44]. It is further in line with recommendations from recent analyses of Signal's post-quantum handshake PQXDH [12, 38].

6 The Combine-then-Expand Paradigm

From the analyses of real-world constructions of multi-input KDFs in the previous section, we can observe a general two-step approach to building n-KDFs.

Combine. First, the multiple input key materials are *combined* into a single, pseudorandom value PRK (taking the context into account). In the MLS n-KDF (see Figure 5), this is done by cascading applications of HMAC to produce a value psk_secret_n. In ETSI-CatKDF and Signal's X3DH-KDF (Figures 6 and 7), this is done via first concatenating the source key material (and in ETSI-CatKDF hashing the context) and then applying HKDF.Extract inside HKDF, producing a pseudorandom key.

Expand. Second, PRK is *expanded* into the desired output-key length, incorporating the label. This is analogous to the expand step in HKDF's Extract-then-Expand (**XtX**) paradigm, and indeed both ETSI-CatKDF and Signal use HKDF.Expand for this purpose, keyed with PRK. MLS, deriving only a single key of fixed length without any label input, skips this step and directly uses PRK derived in Combine as the output (i.e., Expand is simply the identity function).

We call this paradigm "Combine-then-Expand" (**CtX**), and formalize it as a generic construction $\mathbf{CtX}[\mathsf{Combine},\mathsf{Expand}]$ of an n-KDF from two functions Combine and Expand, as shown in Figure 8 (see also Figure 1 for an illustration). Syntactically, Combine is what we call a "comb-KDF": an n-KDF with no label input and fixed output length $\ell_{PRK} = |PRK|$ (see Definition 5), and $\mathsf{Expand}\colon \{0,1\}^{\ell_{PRK}} \times \mathsf{Lbls} \times \mathsf{KLout} \to \mathbb{B}^*$ is a function family.

The careful reader will notice that the real-world constructions we have seen do not always fully align with the formalized version of **CtX** in Figure 8. MLS and Signal both derive only a single output key of fixed length, hence the Expand step is somewhat degenerate: in Signal, it only takes a hard-coded "`info`" string and fixed output length; in MLS, there is no Expand step at all (i.e., it is the identity function). ETSI-CatKDF indeed uses HKDF.Expand in the general sense to derive multiple keys from the combined key material, via label and length inputs, but also includes the context values there, not in the Combine step along with the source key material. Consequently, we gave tailored analyses for the n-KDF constructions used in MLS, Signal, and ETSI-CatKDF in the prior section. Nevertheless, going forward, we deem the **CtX** paradigm a valuable blueprint for designing new multi-input KDFs.

Concretely, we show that the $\mathbf{CtX}[\mathsf{Combine},\mathsf{Expand}]$ construction yields a secure XOF- or NOF-n-KDF if the Combine function is a secure fix$L\ell$-KDF (fixed label and length) and the Expand function is a (multi-user) secure variable-output-length PRF (volPRF, see Appendix A.2 for the formal definition) of type XOF or NOF, respectively. Importantly, the requirement for Combine, being an n-KDF that produces

a single, fixed-length key (with no label input), is notably easier to achieve (and can be constructed from other primitives than hash functions, as we explore in the next section). For the Expand step, candidates for variable-output-length PRFs readily exist, with HKDF.Expand being a prime example of an XOF volPRF.

Theorem 14 (n-KDF security of **CtX**). *Let* Combine *be a comb-KDF with output length* $\ell_{PRK} \in \mathbb{N}$*, let* $\mathsf{Expand}\colon \{0,1\}^{\ell_{PRK}} \times \mathsf{Lbls} \times \mathsf{KLout} \to \{0,1\}^*$ *be a function family and let* $\mathsf{C} = \mathbf{CtX}[\mathsf{Combine}, \mathsf{Expand}]$ *be the n-KDF built from* Combine*,* Expand *as defined in Figure 8. Then for any source collection* $\boldsymbol{\Sigma}$ *and requirement* $req \implies req_X \vee req_N$*, and any adversary* $\mathcal{A}$ *against the* kdf *security of* C*, there exist adversaries* $\mathcal{B}_1$*,* $\mathcal{B}_2$ *such that*

$$\mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{C},\boldsymbol{\Sigma},req}(\mathcal{A}) \le 2 \cdot \big(\mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{Combine},\boldsymbol{\Sigma},req\wedge req_{\mathsf{NoReal}}}(\mathcal{B}_1) + \mathbf{Adv}^{\text{vol-prf}}_{\mathsf{Expand},\mathrm{type}}(\mathcal{B}_2)\big),$$

where type $=$ XOF *if* $req \implies req_X$ *and* type $=$ NOF *if* $req \implies req_N$*. Adversaries* $\mathcal{B}_1, \mathcal{B}_2$ *have roughly the same running time as* $\mathcal{A}$*. Adversary* $\mathcal{B}_1$ *makes the same number of queries to oracles* NewKey, SetKey *as* $\mathcal{A}$*, and at most* $\mathbf{q}_{\text{Ro\$-KDF}}(\mathcal{A}) + \mathbf{q}_{\mathrm{KDF}}(\mathcal{A})$ *to oracle* Ro\$-KDF*. Adversary* $\mathcal{B}_2$ *makes at most* $\mathbf{q}_{\text{Ro\$-KDF}}(\mathcal{A}) + \mathbf{q}_{\mathrm{KDF}}(\mathcal{A})$ *queries to each of its oracles.*

Proof. We assume without loss of generality that adversary $\mathcal{A}$ does not violate requirement req; if it did, it would have an advantage of 0, making the claimed bound trivially true. To begin with, we apply Proposition 7 to add requirement req_{NoReal} to req, effectively removing oracle KDF from the game. The reason is that the intermediate key PRK might be shared across calls to Ro\$-KDF and KDF (if they differ only in the label or length inputs), but the volPRF security game does not provide a "real" function oracle. Hence, the final reduction to the volPRF security of Expand would not be able to simulate responses to queries to oracle KDF in these cases. Proposition 7 gives us an adversary $\mathcal{B}$ making the same number of queries as $\mathcal{A}$ to $\{\text{NewKey}, \text{SetKey}\}$ and $\mathbf{q}_{\text{Ro\$-KDF}}(\mathcal{A}) + \mathbf{q}_{\mathrm{KDF}}(\mathcal{A})$ queries to oracle Ro\$-KDF, such that

$$\mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{C},\boldsymbol{\Sigma},req}(\mathcal{A}) \le 2 \cdot \mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{C},\boldsymbol{\Sigma},req\wedge req_{\mathsf{NoReal}}}(\mathcal{B})\ .$$

Adversary $\mathcal{B}$ additionally does not violate $req \wedge req_{\mathsf{NoReal}}$, by assumption on $\mathcal{A}$.

The proof now proceeds through a sequence of games G_0–G_2, which all run adversary $\mathcal{B}$ and provide the oracles named in the n-KDF game in Figure 3. Let $req^* := req \wedge req_{\mathsf{NoReal}}$. Game G_0 is identical to $\mathbf{G}^{\text{kdf-1}}_{\mathsf{C},\boldsymbol{\Sigma},req^*}$, that is, the "real" n-KDF game without oracle KDF. Hence $\Pr[\mathrm{G}_0] = \Pr[\mathbf{G}^{\text{kdf-1}}_{\mathsf{C},\boldsymbol{\Sigma},req^*}]$. Game G_1 only differs from G_0 in oracle Ro\$-KDF, where instead of computing the key K as

$$\begin{aligned} PRK &\leftarrow \mathsf{Combine}((\sigma_1, c_1), \ldots, (\sigma_n, c_n), \varepsilon, \ell_{PRK}) \\ K &\leftarrow \mathsf{Expand}(PRK, L, \ell), \end{aligned}$$

the key PRK for Expand is replaced by a consistently sampled random string in $\mathbb{B}^{\ell_{PRK}}$, with consistency maintained with a table $\mathsf{T}[(v_1, \ldots, v_n)]$ indexed by the source key material indexes input to oracle Ro\$-KDF by the adversary.

We construct an adversary $\mathcal{B}_1$ against the fix$L\ell$-kdf security of Combine such that

$$\Pr[\mathrm{G}_0] - \Pr[\mathrm{G}_1] \le \mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{Combine},\boldsymbol{\Sigma},req^*}(\mathcal{B}_1), \tag{1}$$

as follows. Adversary $\mathcal{B}_1$ runs adversary $\mathcal{B}$, acting as the challenger in game G_0, with the following differences. $\mathcal{B}_1$ does not sample any source key material. Instead, it simulates queries to oracle NewKey by forwarding them to its own oracle NewKey. It similarly relays queries to oracle SetKey. When $\mathcal{B}$ makes a query Ro\$-KDF$((v_1, \ldots, v_n), L, \ell)$, adversary $\mathcal{B}_1$ uses its own Ro\$-KDF to populate table $\mathsf{T}[(v_1, \ldots, v_n)]$. That is, it checks if the table entry is initialized; if not, it initializes it with the key returned from Ro\$-KDF$((v_1, \ldots, v_n), \varepsilon, \ell_{PRK})$, i.e., the forwarded query from $\mathcal{B}$ to its own oracle, stripped of label and with fixed length. It then sets $PRK \leftarrow \mathsf{T}[(v_1, \ldots, v_n)]$, proceeding as the challenger in game G_0.

This way, adversary $\mathcal{B}_1$ perfectly simulates game G_{1-d} when playing the fix$L\ell$-kdf game with bit parameter d. Adversary $\mathcal{B}_1$ has query count $\mathbf{q}_{\mathrm{OR}}(\mathcal{B}_1) = \mathbf{q}_{\mathrm{OR}}(\mathcal{B}) = \mathbf{q}_{\mathrm{OR}}(\mathcal{A})$ for oracles $\mathrm{OR} \in \{\text{NewKey}, \text{SetKey}\}$, and $\mathbf{q}_{\text{Ro\$-KDF}}(\mathcal{B}_1) \le \mathbf{q}_{\text{Ro\$-KDF}}(\mathcal{B}) = \mathbf{q}_{\text{Ro\$-KDF}}(\mathcal{A}) + \mathbf{q}_{\mathrm{KDF}}(\mathcal{A})$ Furthermore, $\mathcal{B}_1$ satisfies requirements $req \wedge req_{\mathsf{NoReal}}$, assuming that $\mathcal{B}$ also does. For all requirements except req_{NOF} or req_{XOF}, this follows directly from $\mathcal{B}$ not violating the requirement. For the freshness condition, we observe that thanks to the table T, adversary $\mathcal{B}_1$ never repeats a query to oracle Ro\$-KDF, hence fulfilling both req_{NOF} and req_{XOF}. This proves Equation (1).

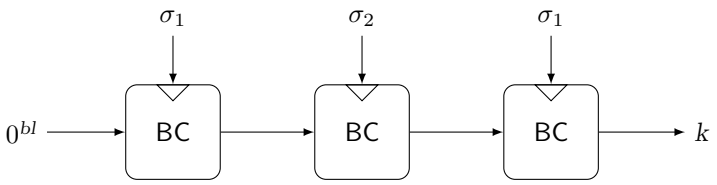

Figure 9: Key combiner for two keys based on blockcipher BC.

We proceed with the final game hop. Game G_2 is identical to G_1, except that instead of computing K as $K \leftarrow \mathsf{Expand}(PRK, L, \ell)$ in oracle RO$-KDF, K is sampled consistently at random in $\mathbb{B}^\ell$. We construct an adversary $\mathcal{B}_2$ against the volPRF security of Expand such that

$$\Pr[G_1] - \Pr[G_2] \leq \mathbf{Adv}^{\text{vol-prf}}_{\mathsf{Expand},\text{type}}(\mathcal{B}_1), \tag{2}$$

where type = XOF if $req \implies req_X$ and type = NOF if $req \implies req_N$. Adversary $\mathcal{B}_2$ runs $\mathcal{B}$, acting as the challenger in game G_1, with two exceptions when adversary $\mathcal{B}$ issues an RO$-KDF$((v_1, \ldots, v_n), L, \ell)$ query. First, instead of sampling PRK in $\mathbb{B}^{\ell_{PRK}}$ to populate the table $\mathsf{T}[(v_1, \ldots, v_n)]$ (if not already populated), adversary $\mathcal{B}_2$ queries oracle NEW in its own game to initialize a new key for Expand and stores the index of the key in $\mathsf{T}[(v_1, \ldots, v_n)]$. Second, instead of computing $\mathsf{Expand}(PRK, L, \ell)$ itself, $\mathcal{B}_2$ queries oracle FN$(\mathsf{T}[(v_1, \ldots, v_n)], L, \ell)$ to compute K.

This way, adversary $\mathcal{B}_2$ perfectly simulates game G_1 and G_2 for $\mathcal{B}$ when the bit parameter d in the volPRF game is 1 and 0, respectively. Furthermore, if the queries by $\mathcal{B}$ satisfy req_X, adversary $\mathcal{B}_2$ will never make two queries FN(i, L, ℓ) and FN(i, L, ℓ') such that $\ell \neq \ell'$ (as such a query would only be triggered by a query from $\mathcal{B}$ which violated req_{XOF}). This proves Equation (2). Adversary $\mathcal{B}_2$ has query count $\mathbf{q}_{\text{NEW}}(\mathcal{B}_2) \leq \mathbf{q}_{\text{RO\$-KDF}}(\mathcal{B}) = \mathbf{q}_{\text{RO\$-KDF}}(\mathcal{A}) + \mathbf{q}_{\text{KDF}}(\mathcal{A})$ and $\mathbf{q}_{\text{FN}}(\mathcal{B}_2) = \mathbf{q}_{\text{RO\$-KDF}}(\mathcal{B}) = \mathbf{q}_{\text{RO\$-KDF}}(\mathcal{A}) + \mathbf{q}_{\text{KDF}}(\mathcal{A})$. □

7 New Constructions

We discuss two new approaches to building multi-input KDFs. The first construction uses a simple key-mixing step, which can be implemented with key-collision-resistant blockciphers, and only requires a variable-output-length pseudorandom function on top, enabling a purely blockcipher-based design. The second approach shows how to build a multi-input KDF that achieves information-theoretic security if one of the key material sources is statistically secure, e.g., if generated via a quantum key distribution protocol. Both approaches do not make use of hash functions, addressing scenarios where only a limited set of building blocks is available due to system limitations or design requirements.

7.1 Blockcipher-Based Construction

The idea for the two-input blockcipher-based construction is to first define a simple key combiner function, mixing the two (pseudorandom) keys σ_1 and σ_2 via blockcipher $\mathsf{BC}\colon \{0,1\}^{bl} \times \{0,1\}^{kl} \to \{0,1\}^{bl}$ into a single, strong key. Only then we apply a simple variable-output-length pseudorandom function to this derived key and the context information, label, and desired output length. This variable-output-length pseudorandom function can, for example, be the CMAC construction by NIST [24], yielding a solution that is entirely based on a blockcipher. Note that we thus slightly diverge from our general **CtX** paradigm discussed in Section 6 in the sense that we mix in the context information only in the expansion step, owned to the fact that we use a blockcipher in the combining step. For "empty" context information security of our construction here already follows from the more general **CtX** result.

Besides the pseudorandomness, the security of our construction hinges on the key-collision resistance of the blockcipher [13]. This means that it should be infeasible to find an input $x \in \{0,1\}^{bl}$ and distinct keys $k, k' \in \{0,1\}^{kl}$ such that $\mathsf{BC}(k, x) = \mathsf{BC}(k', x)$. The absence of such collisions is also commonly used in practical schemes for so-called *key check values* (KCVs), with the goal to ensure the integrity of the key by comparing (truncated) blockcipher outputs for constant inputs like 0^{bl}. We note that key-collision resistance already follows from collision-resistance of the blockcipher used in Davies–Meyer mode [56], $h(k, x) = \mathsf{BC}(k, x) \oplus x$. Any key collision as above for the same input value x thus also yields a collision for the Davies–Meyer hash function.

$\underline{\mathsf{BC\text{-}KDF}[\mathsf{BC},\mathsf{Expand}]((\sigma_1,c_1,\varepsilon),(\sigma_2,c_2,\varepsilon),L,\ell)}$

1 $k \leftarrow \mathsf{BC}(\sigma_1,\mathsf{BC}(\sigma_2,\mathsf{BC}(\sigma_1,0^{bl})))$
2 $K \leftarrow \mathsf{Expand}(k,\langle L,c_1,c_2\rangle,\ell)$
3 Return K

Figure 10: Construction BC-KDF[BC, Expand] based on blockcipher BC and variable-output-length Expand. Note that context information is part of the label.

Definition 15 (Key-collision resistance of blockcipher)**.** For an adversary $\mathcal{A}$ denote by $\mathbf{Adv}^{\mathrm{kcr}}_{\mathsf{BC}}(\mathcal{A})$ the probability that $\mathcal{A}$ outputs $(k,k',x) \in \{0,1\}^{kl} \times \{0,1\}^{kl} \times \{0,1\}^{bl}$ such that $\mathsf{BC}(k,x) = \mathsf{BC}(k',x)$ and $k \neq k'$.

We describe the construction formally as a 2-KDF for combining two uniform keys, illustrated in Figure 9. We assume that the source collection $\mathbf{\Sigma}$ outputs pseudorandom strings σ of kl bits, and that the auxiliary information α is empty. Other sources may require another intermediate step to first derive such (pseudo)random key material. The key mixing step applies the blockcipher three times in a row, starting with input 0^{bl} and computing the key $k \leftarrow \mathsf{BC}(\sigma_1, \mathsf{BC}(\sigma_2, \mathsf{BC}(\sigma_1, 0^{bl})))$ for the key material σ_1 and σ_2. Then it runs the underlying 1-KDF F on this key k and the joint context information (and salts, if existing), the label, and the length parameter.

The idea of the key mixing approach is that if the middle entry σ_2 is honest, then the block cipher input to the last round is pseudorandom and thus is k. If the adversary queries for the same material σ_2 again, but for a different dishonest key σ_1', then key-collision-resistance ensures that the input to the inner evaluation $\mathsf{BC}(\sigma_2,\cdot)$ is new, and that the evaluation of the blockcipher for σ_2 yields a fresh pseudorandom string. If, vice versa, the inner key σ_2 is dishonest but σ_1 is honest, then key-collision-resistance of the inner evaluation guarantees distinct inputs to the outer evaluation $\mathsf{BC}(\sigma_1,\cdot)$, ensuring pseudorandomness of the output. If the adversary queries for a different dishonest key σ_2' for the same value σ_1, then once more key-collision resistance ensures that the input to the final evaluation is new and hence the output a fresh pseudorandom value.

To formalize this, we also require pseudorandomness of the blockcipher for secret key k. To simplify, and since we only need to compute the blockcipher in the forward direction, we define pseudorandomness via indistinguishability from pseudorandom *functions*, implicitly including a PRP/PRF switch [11]. See Definition 20 in Appendix A.1 for the formalization of PRF security. In addition, we consider the source (collection) Σ which outputs a vector consisting of pseudorandom key material σ_i of bit size kl and empty auxiliary data α_i. Note that the context information is arbitrary.

The following theorem shows that the construction is secure if BC is a pseudorandom, key-collision-resistant blockcipher and Expand is a secure volPRF. The resulting KDF inherits the type (XOF/NOF) from Expand. For technical reasons in the reduction, we need to move the context information into the label field when evaluating Expand, as it is done, for instance, in TLS 1.3 [57] and also in the NIST proposals [24] anyway.

Theorem 16 (KDF security based on Blockcipher)**.** *Let* Expand *be a volPRFof type* $\mathrm{T} \in \{\mathrm{XOF},\mathrm{NOF}\}$ *and* BC *be a blockcipher. Let req be* req_X *for* $\mathrm{T} = \mathrm{XOF}$ *and be* req_N *for* $\mathrm{T} = \mathrm{NOF}$*. Consider* BC-KDF[BC, Expand] *as in Figure 10. Then for a pseudorandom source (collection)* $\mathbf{\Sigma} = (\Sigma)$ *with output length* u *and any adversary* $\mathcal{A}$ *against the* kdf *security of* BC-KDF[BC, Expand]*, making at most* $\mathbf{q}_{\textsc{SetKey}}(\mathcal{A})$ *queries to oracle* SetKey *and at most* $\mathbf{q}_{\text{Ro\$-KDF}}(\mathcal{A}) + \mathbf{q}_{\text{KDF}}(\mathcal{A})$ *queries to oracles* Ro\$-KDF *and* KDF *in total, there exists adversaries* $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3$*, and* $\mathcal{B}_4$ *such that*

$$\mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{BC\text{-}KDF}[\mathsf{BC},\mathsf{Expand}],\mathbf{\Sigma},req}(\mathcal{A}) \leq 2 \cdot \left(\mathbf{Adv}^{\mathrm{pr}}_{\Sigma}(\mathcal{B}_1) + \mathbf{Adv}^{\mathrm{kcr}}_{\mathsf{BC}}(\mathcal{B}_2) + \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{BC}}(\mathcal{B}_3) + \mathbf{Adv}^{\text{vol-prf}}_{\mathsf{Expand},\mathrm{T}}(\mathcal{B}_4) \right),$$

where adversaries $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3, \mathcal{B}_4$ *have roughly the same running time as* $\mathcal{A}$*. Furthermore, adversary* $\mathcal{B}_3$ *against the pseudorandomness of* BC*, as well as adversary* $\mathcal{B}_4$ *against the variable-output-length pseudorandomness of* Expand*, each make at most* $u \cdot \mathbf{q}_{\textsc{SetKey}}(\mathcal{A})$ *calls to their oracle* New *and at most* $\mathbf{q}_{\text{Ro\$-KDF}}(\mathcal{A}) + \mathbf{q}_{\text{KDF}}(\mathcal{A})$ *calls to their oracle* Fn*.*

Proof. To simplify the proof we first assume that $\mathcal{A}$ does not make any queries to oracle KDF, costing us a factor 2 in the bound, but keeping the sum $\mathbf{q}_{\text{Ro\$-KDF}}(\mathcal{A}) + \mathbf{q}_{\text{KDF}}(\mathcal{A})$ of queries to oracles Ro\$-KDF and KDF unchanged. We continue to bound the advantage of $\mathcal{A}$ through a sequence of game hops, beginning

with the "real" ($d = 1$) game $\mathbf{G}_0 := \mathbf{G}^{\text{kdf-1}}_{\mathsf{BC\text{-}KDF}[\mathsf{BC},\mathsf{Expand}],\boldsymbol{\Sigma},req}$, and finishing with the "random" game for $d = 0$.

In game $\mathbf{G}_1$ we replace the pseudorandom outputs of the source (collection) $\boldsymbol{\Sigma} = (\Sigma)$ by truly random key material σ_i of the corresponding length. The pseudorandomness of the source gives us

$$\Pr[\,\mathbf{G}_0\,] - \Pr[\,\mathbf{G}_1\,] \leq \mathbf{Adv}^{\text{pr}}_{\Sigma}(\mathcal{B}_1)$$

where $\mathcal{B}_1$ simulates $\mathcal{A}$'s attack with the help of its (random or pseudorandom) sample $\mathbf{z}$, outputting 1 if and only if $\mathcal{A}$ succeeds in the simulated attack.

In game $\mathbf{G}_2$ we immediately stop if the adversary, during the attack, ever generates a key-collision in the step for computing k (line 1 of Figure 10) in an oracle call to RO$-KDF. That is, for different keys σ_b, σ'_b and the same input x (where $x = 0^{bl}$ or $x = \mathsf{BC}(\sigma_1, 0^{bl})$) we have $\mathsf{BC}(\sigma_b, x) = \mathsf{BC}(\sigma'_b, x)$. We implement this in the game by keeping track of all intermediate values when evaluating the BC chain and abort if we find such a collision. To bound the probability of $\mathcal{A}$ causing such a key-collision in game $\mathbf{G}_2$ we argue via the key-collision resistance of BC:

$$\Pr[\,\mathbf{G}_1\,] - \Pr[\,\mathbf{G}_2\,] \leq \mathbf{Adv}^{\text{kcr}}_{\mathsf{BC}}(\mathcal{B}_2).$$

In game $\mathbf{G}_3$ we next replace all evaluations of BC in the key mixing step (line 1) for each honest key material σ by a random function (but consistently, i.e., for same key σ and same input with the same value as before). The random functions are implemented via tables $\mathsf{T}_\sigma[\cdot]$ and efficient look-ups. Note that for dishonest σ we still compute BC as before. Furthermore, we still adhere to the checking for collisions in the intermediate values, now for the modified evaluations of the blockcipher. It follows from the pseudorandomness of BC that this is indistinguishable, i.e., since this corresponds exactly to the difference in the pseudorandomness game, there exists an algorithm $\mathcal{B}_3$ such that

$$\Pr[\,\mathbf{G}_2\,] - \Pr[\,\mathbf{G}_3\,] \leq \mathbf{Adv}^{\text{prf}}_{\mathsf{BC}}(\mathcal{B}_3).$$

Note that there are at most u honest key material values σ, and each value can be assigned at most once. In total, adversary $\mathcal{B}_3$ thus makes at most u calls to its oracle NEW in the (multi-user) PRF game, and at most $\mathbf{q}_{\text{RO\$-KDF}}(\mathcal{A}) + \mathbf{q}_{\text{KDF}}(\mathcal{A})$ evaluation requests to oracle FN.

Now we observe that each call to oracle RO$-KDF for indexes v_1, v_2 must point to at least one honest key according to requirement req_{1HKey}. Hence, the evaluation of the most outer honest key (either σ_2 for a dishonest σ_1, or σ_1 if σ_2 is honest or dishonest) in each such call is always for a new input value by the previous games, if the other key changes. It follows that the output of such an evaluation is an independent random value, denoted as $K_{(\sigma_1,\sigma_2)}$. This also holds for a dishonest outer value σ_1 because we then apply the final blockcipher evaluation $\mathsf{BC}(\sigma_1, \cdot)$ for the chosen key to the random output $\mathsf{T}_{\sigma_2}[\cdot]$ for honest key σ_2. Here we use the fact that σ_2 is set by the adversary before the first call to RO$-KDF and thus independent of the random function value.

We would now like to proceed to $\mathbf{G}_4$ and replace the evaluation results of the variable-output-length PRF Expand by random variables in calls to oracle RO$-KDF, using the fact that the key for Expand is already random according to the previous game, i.e., that each pair of key material (σ_1, σ_2) with one honest key maps to a random string $K_{(\sigma_1,\sigma_2)}$. However, the adversary may register identical (dishonest) key material σ under different key indexes ν and ν' via oracle SETKEY; we only disallow the adversary to duplicate entire inputs (σ, c) via requirement req_{NoDColl}, but here only the context information may, for instance, change. Fortunately, our reduction $\mathcal{B}_4$ to the pseudorandomness of Expand can easily track such cases, since the key material contents of dishonest keys are known. Hence, adversary $\mathcal{B}_4$ can define a mapping of indexes (v_1, v_2) passed to oracle RO$-KDF to the corresponding key pair identifier, $\mathsf{Kmap}(v_1, v_2) = (j_1, j_2)$; for dishonest keys it even knows the key material σ in addition to the index.

With game $\mathbf{G}_4$ we next replace all evaluations of the variable-output-length PRF Expand in the simulated calls of $\mathcal{A}$ to oracle RO$-KDF by random values (but consistently). To argue the validity of this modification, we can build an adversary $\mathcal{B}_4$ against the (multi-user) security of Expand. Adversary $\mathcal{B}_4$ creates a new key in its game via a call for NEW for each pair (v_1, v_2) mapping to a new pair $\mathsf{Kmap}(v_1, v_2)$. Since each of the entries in the source vector $\mathbf{z}$ can be assigned only once to a key position, and adversary $\mathcal{A}$ sets at most $\mathbf{q}_{\text{SETKEY}}(\mathcal{A})$ dishonest keys, there can be at most $u \cdot \mathbf{q}_{\text{SETKEY}}(\mathcal{A})$ keys in $\mathcal{B}_4$'s game. Adversary $\mathcal{B}_4$ then maps the pair $\mathsf{Kmap}(v_1, v_2)$ in $\mathcal{A}$'s attack to one of these generated keys. Algorithm $\mathcal{B}_4$ simply uses oracle access to the volPRF security game to simulate the game of $\mathcal{A}$ now, encoding the (public) context information as part of the label. It eventually outputs 1 iff $\mathcal{A}$ wins.

Note that $\mathcal{B}_4$ has another stipulation (only) in case of a XOF, namely, that it never repeats the same query (i, L) for key index i and label L for different length parameters. We argue that this must be the

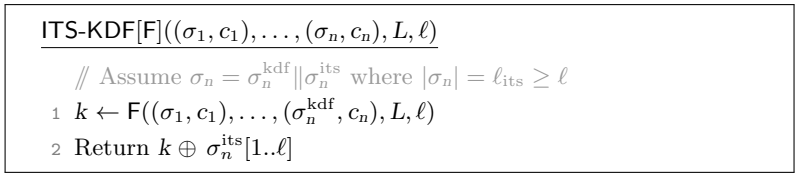

Figure 11: Construction ITS-KDF[F] based on KDF F, achieving information-theoretic security.

case if $\mathcal{A}$ successfully plays against the KDF of type XOF. To this end we observe that requirement req_{XOF} for $\mathcal{A}$ demands that in any two calls to the RO$-KDF oracle the pairs $(\mathbf{v}, L)$, $(\mathbf{v}', L')$ differ. If the labels L, L' in such queries are distinct, then the label field in $\mathcal{B}_4$'s calls (encoding L, c_1, and c_2 in a recoverable way) must also be distinct. Assume next that the labels $L = L'$ match, such that the indexes must differ. Then, adversary $\mathcal{B}_4$ will only map this to the same key index i in its simulation via Kmap if both indexes include dishonest indexes $v_b \neq v'_b$ at the same position b, holding the same key material σ. But then requirement req_{NoDColl} for XOF KDFs ensures that the context information c for both keys must differ (because position and key material already match). This, however, implies that $\mathcal{B}_4$ encodes different values in its label $\langle L, c_1, c_2 \rangle$ for both queries, such that it does not violate the XOF property in its pseudorandomness game.

We conclude that pseudorandom answers of $\mathcal{B}_4$'s evaluation oracle correspond then to $\mathbf{G}_3$, and random answers to game $\mathbf{G}_4$, and that $\mathcal{B}_4$ successfully distinguishes the cases if $\mathcal{A}$ wins in the corresponding game:

$$\Pr[\,\mathbf{G}_3\,] - \Pr[\,\mathbf{G}_4\,] \leq \mathbf{Adv}^{\text{vol-prf}}_{\mathsf{Expand},\mathrm{T}}(\mathcal{B}_4).$$

Note that in the final game, all of the responses of oracle RO$-KDF are random and correspond to $\mathbf{G}^{\text{kdf-0}}_{\mathsf{BC\text{-}KDF[BC,Expand]},\boldsymbol{\Sigma},req}$. This proves the theorem. □

We observe that the above construction can be generalized to accommodate more keys. Given that one has already applied the key mixing step for $n-1$ keys, then one processes the resulting key with the final key in an additional key mixing step. That is, one first combines the first two keys and then combines the result with the third key, and so on. The more direct alternative is to use a scheduling of keys in order $\sigma_1, \sigma_2, \ldots, \sigma_n, \sigma_1, \ldots, \sigma_{n-1}$ and apply the blockcipher for this sequence.

7.2 Achieving Information-Theoretic Security

We propose here a simple construction for combining two or more cryptographic keys. This construction might be of special interest to Quantum Key Distribution (QKD) networks and similar scenarios where one source generates truly random output but is hedged by also using classical and/or post-quantum cryptographic key exchange mechanisms on top. The interesting feature of our construction is that the resulting key is truly random, as long as at least one dedicated source (say, Σ_n for the sake of concreteness) of keying material is truly random. Needless to say, this key material can only be used once. Besides achieving this stronger security guarantee, the construction preserves the common feature of combiners: If at least one source is (computationally) secure, then so is the combined key, even if the key material is used multiple times.

The construction is depicted in Figure 11. The idea is to split the potential information-theoretically secure key material σ_n into two parts, one short part σ_n^{kdf} of ℓ_{kdf} bits entering the key combiner with the other key material, and the other part σ_n^{its} of ℓ_{its} which is then added to the output of the combiner. Note that it is necessary to include some input of the source σ_n into the combiner evaluation, otherwise the adversary could leave the honest key material σ_n fixed and change the other $n-1$ keys using the SetKey oracle to distinguish two answers by from random.

The proof follows by reduction to the security of the underlying n-KDF, using only the portion σ_n^{kdf} of this source as input. For a source collection $\boldsymbol{\Sigma}$, we thus define a collection $\boldsymbol{\Sigma}'$ where we (a) truncate the outputs $\mathbf{z}_i[j].\sigma$ for $i = 1, 2, \ldots, u$ of each source Σ_i with $i = \Sigma\text{-map}(n)$ to the first ℓ_{kdf} bits, and (b) augment the respective auxiliary information $\mathbf{z}_i[j].\alpha$ by the remaining ℓ_{its} bits. Note that, in general, the key material parts can be correlated, although they are independent in the information-theoretic case. This is why we make the additional data available in α.

Theorem 17 (KDF security of ITS-KDF)**.** *Let* F *be a* n*-KDF with requirements* req*, and consider* ITS-KDF[F]*. Then, for any source collection* $\boldsymbol{\Sigma}$ *and any adversary* $\mathcal{A}$ *against the* kdf *security of* ITS-KDF[F]*,*

there exists an adversary $\mathcal{B}$ such that

$$\mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{ITS\text{-}KDF}[\mathsf{F}],\boldsymbol{\Sigma},req}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{F},\boldsymbol{\Sigma}',req}(\mathcal{B}) \ ,$$

where $\boldsymbol{\Sigma}'$ is defined as above. Adversary $\mathcal{B}$ has roughly the same running time as $\mathcal{A}$.

Proof. Consider adversary $\mathcal{A}$ attacking ITS-KDF[F]. Assume that, for each oracle query of $\mathcal{A}$ to Ro$-KDF we instead give a random answer. We construct an adversary $\mathcal{B}$ against F and source collection $\boldsymbol{\Sigma}'$ showing that this is indistinguishable. The main task of the reduction is the translation between $\boldsymbol{\Sigma}$ and $\boldsymbol{\Sigma}'$, which is dependent on the position where the key material is used. To this end, $\mathcal{B}$ forwards all of $\mathcal{A}$'s queries to the NewKey oracle as well as the SetKey oracle to its own respective oracles, but retains a local copy of the queries.

When $\mathcal{B}$ receives a query to the Ro$-KDF or the KDF, it first checks which key material is used in position n (which we fixed as the position containing the possibly uniform key material). This results in one of two cases:

- If $\mathcal{A}$ is using key material it set using the SetKey, $\mathcal{B}$ retrieves this key material from its local copy of earlier queries and translates it to $\boldsymbol{\Sigma}'$ by splitting σ_n into the first ℓ_{kdf} bits σ_n^{kdf} and the remaining ℓ_{its} bits σ_n^{its}. $\mathcal{B}$ then queries its own SetKey oracle with $(\sigma_n^{\mathrm{kdf}}, c, (\alpha, \sigma_n^{\mathrm{its}}))$. It also modifies $\mathcal{A}'s$ query such that it now refers to this newly set key in position n.
- If $\mathcal{A}$ is using honestly generated key material in position n, then $\mathcal{B}$ forwards all buffered queries to SetKey oracle that are required for this query. Per definition of σ', $\mathcal{B}$ receives as answer the leakage α, which contains the key material σ_n^{its} used for the one time pad

In both cases, $\mathcal{B}$ has obtained the value σ_n^{its} required for simulating the one-time pad. Now, $\mathcal{B}$ forwards the (potentially modified) Ro$-KDF or KDF query to its own corresponding oracle, receiving the output k. Before forwarding this to $\mathcal{A}$, $\mathcal{B}$ augments k by calculating $k' \leftarrow k \oplus \sigma_n^{\mathrm{its}}$ and finally answers $\mathcal{A}$'s query with k'.

Once Algorithm $\mathcal{A}$ halts and returns a bis d^*, $\mathcal{B}$ also halts and outputs the same bit. Note that $\mathcal{B}$ satisfies the requirements *req* iff $\mathcal{A}$ does. Furthermore, if the Ro$-KDF returns the correct F values ($d = 1$ in $\mathcal{B}$'s attack), then $\mathcal{B}$ perfectly simulates $\mathcal{A}$'s attack for genuine answers of ITS-KDF[F]. If, on the other hand, $\mathcal{B}$'s oracle Ro$-KDF gives random answers instead, then $\mathcal{B}$ adds the value σ^{its} to this value, but this remains an independent and random response. Hence, $\mathcal{B}$ perfectly simulates the modified game above in which we replace all answers of Ro$-KDF in $\mathcal{A}$'s attack by random. Hence,

$$\mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{ITS\text{-}KDF}[\mathsf{F}],\boldsymbol{\Sigma},req}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{F},\boldsymbol{\Sigma}',req}(\mathcal{B}),$$

as claimed. □

We can also show security for unbounded adversaries $\mathcal{A}$ if we assume that σ_n is truly uniform. In this case we would require $\mathcal{A}$ to adhere to predicate $req_{1\times\mathsf{Key}}(\{i\})$ where source i is placed in position n. The desired result follows immediately, as the xor operation with a one-time uniform key is a one-time pad, making the output of ITS-KDF indistinguishable from a randomly sampled string of the same length.

8 From KDF Security to Key Exchange Security

We conclude by putting our KDF notion to action, demonstrating that it can elegantly cover the core of a passive security proof of a KEM+KEM-combiner key exchange protocol, such as the one given in Figure 12. We first give a definition of KEM sources and show (in Appendix A.4) that IND-CPA-secure KEMs yield pseudorandom sources. Then, we reduce the key exchange security of the combiner to KDF security.

Definition 18 ($p \times s$ KEM source)**.** A $p \times s$ KEM source for a key encapsulation mechanism $\mathsf{KEM} = (\mathsf{KGen}, \mathsf{Encaps}, \mathsf{Decaps})$ with key length m is the correlated probability distribution over $\{0,1\}^m \times \{0,1\}^* \times \mathsf{Aux}$ generated by a sampler Σ with sample size $u = p \cdot s$ for $p, s \in \mathbb{N}$, with the following properties:

1. Σ samples $(pk_i, sk_i) \leftarrow\!\!\$\ \mathsf{KEM.KGen}()$, for $i \in [p]$.
2. Σ computes $(c_{i,j}, K_{i,j}) \leftarrow\!\!\$\ \mathsf{KEM.Encaps}(pk_i)$, for $(i,j) \in [p] \times [s]$.
3. Σ outputs $\big((K_{1,1}, (pk_1, c_{1,1}), \alpha_{1,1}), (K_{1,2}, (pk_1, c_{1,2}), \alpha_{1,2}), \ldots, (K_{p,s}, (pk_p, c_{p,s}), \alpha_{p,s})\big)$.

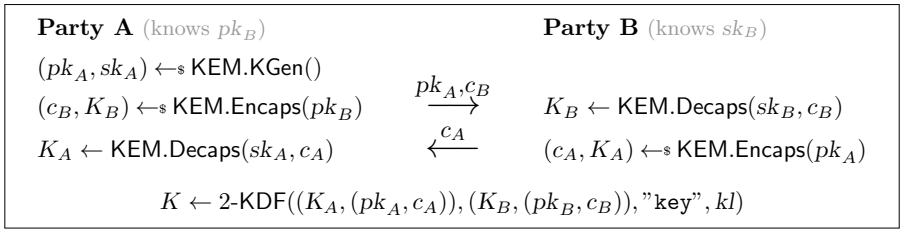

Figure 12: Simple KEM+KEM-combiner key exchange protocol $\mathsf{KE}^{\text{KEM+KEM}}$ applying a 2-KDF for key derivation to combine two keys K_A and K_B resulting from an ephemeral resp. static KEM.

We call a KEM source "single-pk" if $p = 1$ and its auxiliary information "publicly computable" if there is a function f such that $f(i, j, pk_i, c_{i,j}) = \alpha_{i,j}$.

Consider a passive key exchange security game $\mathbf{G}^{\text{psv-ke-}d}_{\mathsf{KE},p}$, where an adversary interacts with p parties (whose public keys it is given at the outset of the game) and has access to three oracles: a Run oracle running the key exchange protocol KE for users of its choice and returning the communication transcript to the adversary, a Reveal oracle that returns the established session key of such run, and a Test oracle that returns either the real session key or a random key for a run, depending on the hidden challenge bit d. (Calling both Reveal and Test is forbidden, being a trivial attack.) The adversary's task is to determine d, measured as $\mathbf{Adv}^{\text{psv-ke}}_{\mathsf{KE}} = \Pr\left[\mathbf{G}^{\text{psv-ke-1}}_{\mathsf{KE}}\right] - \Pr\left[\mathbf{G}^{\text{psv-ke-0}}_{\mathsf{KE}}\right]$. We provide the formal game in Figure 16 in Appendix A.5.

We show that the security of the KEM+KEM-combiner key exchange in Figure 12 directly reduces to the security of the used 2-KDF for KEM sources.

Theorem 19 (KEM+KEM-combiner key exchange security)**.** *Let Σ_e be a $n \times 1$ KEM source and Σ_s be a $p \times s$ KEM source. Let $\mathsf{KE}^{\text{KEM+KEM}}$ be the key exchange protocol given in Figure 12 using a KDF* 2-KDF *admitting output length kl and defined wrt. $(2,2)$-source collection $\mathbf{\Sigma} = (\Sigma_e, \Sigma_s)$ with* Σ-map *being the identity function.*

Then, for any adversary $\mathcal{A}$ in the passive key exchange security game for $\mathsf{KE}^{\text{KEM+KEM}}$, where p is the number of parties, s the number of protocol runs any single party is involved in, and $n = \mathbf{q}_{\text{Run}}(\mathcal{A}) \leq p \cdot s$ is the number of protocol runs overall, there exists an adversary $\mathcal{B}$ such that

$$\mathbf{Adv}^{\text{psv-ke}}_{\mathsf{KE}^{\text{KEM+KEM}},p} \leq \mathbf{Adv}^{\text{kdf}}_{\text{2-}\mathsf{KDF},\mathbf{\Sigma},req}(\mathcal{B}),$$

where the KDF can be of either type ($req \in \{req_X, req_N\}$). Adversary $\mathcal{B}$ makes $\mathbf{q}_{\text{NewKey}}(\mathcal{B}) \leq p + 2 \cdot \mathbf{q}_{\text{Run}}(\mathcal{A})$, $\mathbf{q}_{\text{KDF}}(\mathcal{B}) = \mathbf{q}_{\text{Reveal}}(\mathcal{A})$, and $\mathbf{q}_{\text{Ro\$-KDF}}(\mathcal{B}) = \mathbf{q}_{\text{Test}}(\mathcal{A})$ number of queries and runs in about the same time as $\mathcal{A}$.

The following proof shows essentially that the NewKey, KDF, and Ro$-KDF oracles can be used to simulate the key exchange security oracles Run, Reveal, and Test, respectively.

Proof. In simulating $\mathbf{G}^{\text{psv-ke-}d}_{\mathsf{KE}^{\text{KEM+KEM}}}$, adversary $\mathcal{B}$ first initializes an ephemeral-key counter ctr_e and a static-key counter ctr_u for each party $u \in [p]$. It then calls NewKey for source $i = 2$ and index $j = (u-1) \cdot s$ once for each party $u \in [p]$, remembering the returned ciphertext for later and taking the obtained public key pk_u as the public key for party u, which it then gives to $\mathcal{A}$.

Whenever $\mathcal{A}$ calls its Run oracle, $\mathcal{B}$ registers two new keys via NewKey: one in each position of the KDF, from Σ_e and Σ_s, respectively. For the first, it simply picks the next index $j_e = ctr_e$ via the counter ctr_e, for the second, it picks the index as $j_s = (u-1) \cdot s + ctr_u$, where $u \in [p]$ is an index associated with party B of the key exchange and ctr_u a per-party counter. It then increments both ctr_e and ctr_u by 1 each. (Note that for the very first query, $\mathcal{B}$ already queried that index during initialization and uses the remembered values from that call.) It uses the first's context as (pk_A, c_A) and the second's context ciphertext as c_B to form the transcript to be returned to $\mathcal{A}$. For each such Run oracle call, it records the used source key indexes (j_e, j_s).

When $\mathcal{A}$ makes a Reveal query for some protocol run, $\mathcal{B}$ computes the session key by calling its KDF oracle on inputs $((j_e, j_s), \text{"key"}, kl)$, where (j_e, j_s) are the source key indexes used for that protocol run. Likewise, $\mathcal{B}$ answers a Test query in the same way, except by using its Ro$-KDF oracle instead. The queries $\mathcal{B}$ makes are permissible wrt. both NOF and XOF requirements ($req \in \{req_N, req_X\}$), as $\mathcal{B}$ only uses honest source key indexes and does not reuse them between KDF and Ro$-KDF queries, since $\mathcal{A}$ is not allowed to make both a Reveal and Test query to the same protocol run.

This means means that $\mathcal{B}$ simulates $\mathbf{G}^{\text{psv-ke-}d}_{\mathsf{KE}^{\text{KEM+KEM}}}$ for $\mathcal{A}$ with $d = 1$ or $d = 0$ depending on the bit in the KDF security game, establishing the claim. □

Acknowledgments

We thank Mihir Bellare and Matteo Scarlata for insightful discussions in an early phase of the project, Matthew Campagna for discussions of ETSI TS 103 744, and the anonymous reviewers for their helpful comments. This work has been co-funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – SFB 1119 – 236615297 and the German Federal Ministry of Education and Research (BMBF) under reference 16KISQ074.

References

[1] Yawning Angel, Benjamin Dowling, Andreas Hülsing, Peter Schwabe, and Fiona Johanna Weber. Post quantum noise. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 97–109. ACM Press, November 2022.

[2] Nimrod Aviram, Benjamin Dowling, Ilan Komargodski, Kenneth G. Paterson, Eyal Ronen, and Eylon Yogev. Practical (post-quantum) key combiners from one-wayness and applications to TLS. Cryptology ePrint Archive, Report 2022/065, 2022.

[3] Matilda Backendal, Mihir Bellare, Felix Günther, and Matteo Scarlata. When messages are keys: Is HMAC a dual-PRF? In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part III*, volume 14083 of *LNCS*, pages 661–693. Springer, Cham, August 2023.

[4] Matilda Backendal, Sebastian Clermont, Marc Fischlin, and Felix Günther. Key derivation functions without a grain of salt. Cryptology ePrint Archive.

[5] Manuel Barbosa, Deirdre Connolly, João Diogo Duarte, Aaron Kaiser, Peter Schwabe, Karoline Varner, and Bas Westerbaan. X-wing. *CiC*, 1(1):21, 2024.

[6] Elaine Barker, Lily Chen, and Richard Davis. Recommendation for key-derivation methods in key-establishment schemes. NIST Special Publication (SP) 800-56C-r2, National Institute of Standards and Technology (NIST), Gaithersburg, MD, August 2020.

[7] R. Barnes, B. Beurdouche, R. Robert, J. Millican, E. Omara, and K. Cohn-Gordon. The Messaging Layer Security (MLS) Protocol. RFC 9420 (Proposed Standard), July 2023.

[8] R. Barnes, K. Bhargavan, B. Lipp, and C. Wood. Hybrid Public Key Encryption. RFC 9180 (Informational), February 2022.

[9] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 1–15. Springer, Berlin, Heidelberg, August 1996.

[10] Mihir Bellare and Anna Lysyanskaya. Symmetric and dual PRFs from standard assumptions: A generic validation of an HMAC assumption. Cryptology ePrint Archive, Report 2015/1198, 2015.

[11] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426. Springer, Berlin, Heidelberg, May / June 2006.

[12] Karthikeyan Bhargavan, Charlie Jacomme, Franziskus Kiefer, and Rolfe Schmidt. Formal verification of the PQXDH post-quantum key agreement protocol for end-to-end secure messaging. In Davide Balzarotti and Wenyuan Xu, editors, *USENIX Security 2024*. USENIX Association, August 2024.

[13] Eli Biham. How to decrypt or even substitute des-encrypted messages in 2^{28} steps. *Inf. Process. Lett.*, 84(3):117–124, 2002.

[14] Nina Bindel, Jacqueline Brendel, Marc Fischlin, Brian Goncalves, and Douglas Stebila. Hybrid key encapsulation mechanisms and authenticated key exchange. In Jintai Ding and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 10th International Conference, PQCrypto 2019*, pages 206–226. Springer, Cham, 2019.

[15] Simon Blake-Wilson and Alfred Menezes. Unknown key-share attacks on the station-to-station (STS) protocol. In Hideki Imai and Yuliang Zheng, editors, *PKC'99*, volume 1560 of *LNCS*, pages 154–170. Springer, Berlin, Heidelberg, March 1999.

[16] Bluetooth Special Interest Group (SIG). Bluetooth core specification, January 2023. Ver. 5.4.

[17] Jacqueline Brendel, Rune Fiedler, Felix Günther, Christian Janson, and Douglas Stebila. Post-quantum asynchronous deniable key exchange and the Signal handshake. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *PKC 2022, Part II*, volume 13178 of *LNCS*, pages 3–34. Springer, Cham, March 2022.

[18] Jacqueline Brendel, Marc Fischlin, Felix Günther, Christian Janson, and Douglas Stebila. Towards post-quantum security for Signal's X3DH handshake. In Orr Dunkelman, Michael J. Jacobson Jr., and Colin O'Flynn, editors, *SAC 2020*, volume 12804 of *LNCS*, pages 404–430. Springer, Cham, October 2020.

[19] Chris Brzuska, Eric Cornelissen, and Konrad Kohbrok. Security analysis of the MLS key derivation. In *2022 IEEE Symposium on Security and Privacy*, pages 2535–2553. IEEE Computer Society Press, May 2022.

[20] Chris Brzuska, Antoine Delignat-Lavaud, Christoph Egger, Cédric Fournet, Konrad Kohbrok, and Markulf Kohlweiss. Key-schedule security for the TLS 1.3 standard. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part I*, volume 13791 of *LNCS*, pages 621–650. Springer, Cham, December 2022.

[21] Chris Brzuska and Jan Winkelmann. NPRFs and their application to MLS. https://datatracker.ietf.org/meeting/interim-2020-cfrg-02/materials/slides-interim-2020-cfrg-02-sessa-nprfs-and-their-application-to-mls-00.pdf, July 2020. Visited on August 26, 2022.

[22] Matthew Campagna and Adam Petcher. Security of hybrid key encapsulation. Cryptology ePrint Archive, Report 2020/1364, 2020.

[23] Sofía Celi, Jonathan Hoyland, Douglas Stebila, and Thom Wiggers. A tale of two models: Formal verification of KEMTLS via Tamarin. In Vijayalakshmi Atluri, Roberto Di Pietro, Christian Damsgaard Jensen, and Weizhi Meng, editors, *ESORICS 2022, Part III*, volume 13556 of *LNCS*, pages 63–83. Springer, Cham, September 2022.

[24] Lily Chen. Recommendation for key derivation using pseudorandom functions. NIST Special Publication (SP) 800-108-r1-upd1, National Institute of Standards and Technology (NIST), Gaithersburg, MD, February 2024.

[25] Tristan Claverie and José Lopes-Esteves. Bluemirror: Reflections on bluetooth pairing and provisioning protocols. In *IEEE Security and Privacy Workshops, SP Workshops 2021, San Francisco, CA, USA, May 27, 2021*, pages 339–351. IEEE, 2021.

[26] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A formal security analysis of the signal messaging protocol. In *2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017*, pages 451–466. IEEE, 2017.

[27] Katriel Cohn-Gordon, Cas Cremers, Kristian Gjøsteen, Håkon Jacobsen, and Tibor Jager. Highly efficient key exchange protocols with optimal tightness. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 767–797. Springer, Cham, August 2019.

[28] Hannah Davis, Denis Diemert, Felix Günther, and Tibor Jager. On the concrete security of TLS 1.3 PSK mode. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 876–906. Springer, Cham, May / June 2022.

[29] Hannah Davis and Felix Günther. Tighter proofs for the SIGMA and TLS 1.3 key exchange protocols. In Kazue Sako and Nils Ole Tippenhauer, editors, *ACNS 21International Conference on Applied Cryptography and Network Security, Part II*, volume 12727 of *LNCS*, pages 448–479. Springer, Cham, June 2021.

[30] Cyprien Delpech de Saint Guilhem, Marc Fischlin, and Bogdan Warinschi. Authentication in key-exchange: Definitions, relations and composition. In Limin Jia and Ralf Küsters, editors, *CSF 2020 Computer Security Foundations Symposium*, pages 288–303. IEEE Computer Society Press, 2020.

[31] Denis Diemert and Tibor Jager. On the tight security of TLS 1.3: Theoretically sound cryptographic parameters for real-world deployments. *Journal of Cryptology*, 34(3):30, July 2021.

[32] Samuel Dobson and Steven D. Galbraith. Post-quantum signal key agreement from SIDH. In Jung Hee Cheon and Thomas Johansson, editors, *Post-Quantum Cryptography - 13th International Workshop, PQCrypto 2022, Virtual Event, September 28-30, 2022, Proceedings*, volume 13512 of *Lecture Notes in Computer Science*, pages 422–450. Springer, 2022.

[33] Yevgeniy Dodis, Thomas Ristenpart, John P. Steinberger, and Stefano Tessaro. To hash or not to hash again? (In)differentiability results for H^2 and HMAC. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 348–366. Springer, Berlin, Heidelberg, August 2012.

[34] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. A cryptographic analysis of the TLS 1.3 handshake protocol. *Journal of Cryptology*, 34(4):37, October 2021.

[35] Benjamin Dowling, Torben Brandt Hansen, and Kenneth G. Paterson. Many a mickle makes a muckle: A framework for provably quantum-secure hybrid key exchange. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*, pages 483–502. Springer, Cham, 2020.

[36] ETSI. Quantum-safe hybrid key exchanges. Technical Specification TS 103 744 v1.1.1, European Telecommunications Standards Institute (ETSI), December 2020.

[37] ETSI. Quantum-safe hybrid key exchanges. Technical Specification TS 103 744 v1.2.1, European Telecommunications Standards Institute (ETSI), March 2025.

[38] Rune Fiedler and Felix Günther. Security analysis of Signal's PQXDH handshake. In Tibor Jager, Jiaxin Pan, Bor de Kock, and Tjerand Slide, editors, *28th International Conference on Practice and Theory of Public-Key Cryptography (PKC 2025)*. Springer, May 2025. https://eprint.iacr.org/2024/702.

[39] German Federal Office for Information Security (BSI). Cryptographic mechanisms: Recommendations and key lengths, version: 2025-1. Technical Guideline TR-02102-1, January 2025.

[40] Federico Giacon, Felix Heuer, and Bertram Poettering. KEM combiners. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 190–218. Springer, Cham, March 2018.

[41] Andreas Hülsing, Kai-Chun Ning, Peter Schwabe, Fiona Johanna Weber, and Philip R. Zimmermann. Post-quantum WireGuard. In *2021 IEEE Symposium on Security and Privacy*, pages 304–321. IEEE Computer Society Press, May 2021.

[42] B. Kaliski. PKCS #5: Password-Based Cryptography Specification Version 2.0. RFC 2898 (Informational), September 2000. Obsoleted by RFC 8018.

[43] C. Kaufman, P. Hoffman, Y. Nir, and P. Eronen. Internet Key Exchange Protocol Version 2 (IKEv2). RFC 5996 (Proposed Standard), September 2010. Obsoleted by RFC 7296, updated by RFCs 5998, 6989.

[44] Eike Kiltz, Jiaxin Pan, Doreen Riepel, and Magnus Ringerud. Multi-user CDH problems and the concrete security of NAXOS and HMQV. In Mike Rosulek, editor, *CT-RSA 2023*, volume 13871 of *LNCS*, pages 645–671. Springer, Cham, April 2023.

[45] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104 (Informational), February 1997. Updated by RFC 6151.

[46] H. Krawczyk and P. Eronen. HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC 5869 (Informational), May 2010.

[47] Hugo Krawczyk. SIGMA: The "SIGn-and-MAc" approach to authenticated Diffie-Hellman and its use in the IKE protocols. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 400–425. Springer, Berlin, Heidelberg, August 2003.

[48] Hugo Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 631–648. Springer, Berlin, Heidelberg, August 2010.

[49] Moxie Marlinspike and Trevor Perrin. The X3DH key agreement protocol, April 2016. Revision 1.

[50] Moxie Marlinspike and Trevor Perrin. The PQXDH key agreement protocol, January 2024. Revision 3.

[51] Ueli M. Maurer and Stefan Wolf. Diffie-Hellman oracles. In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 268–282. Springer, Berlin, Heidelberg, August 1996.

[52] Vivek Nair and Dawn Song. Multi-factor key derivation function (MFKDF) for fast, flexible, secure, & practical key management. In Joseph A. Calandrino and Carmela Troncoso, editors, *USENIX Security 2023*, pages 2097–2114. USENIX Association, August 2023.

[53] National Institute of Standards and Technology. The keyed-hash message authentication code (HMAC). Federal Information Processing Standards Publications (FIPS PUBS) 198-1, U.S. Department of Commerce, Washington, D.C., 2008.

[54] Trevor Perrin. The noise protocol framework, July 2018.

[55] Bertram Poettering and Simon Rastikian. A study of KEM generalizations. In Felix Günther and Julia Hesse, editors, *Security Standardisation Research - 8th International Conference, SSR 2023, Lyon, France, April 22-23, 2023, Proceedings*, volume 13895 of *Lecture Notes in Computer Science*, pages 53–77. Springer, 2023.

[56] Bart Preneel, René Govaerts, and Joos Vandewalle. Hash functions based on block ciphers: A synthetic approach. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 368–378. Springer, Berlin, Heidelberg, August 1994.

[57] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446 (Proposed Standard), August 2018.

[58] Phillip Rogaway. Formalizing human ignorance. In Phong Q. Nguyen, editor, *Progressin Cryptology - VIETCRYPT 2006, First International Conference on Cryptology in Vietnam, Hanoi, Vietnam, September 25-28, 2006, Revised Selected Papers*, volume 4341 of *Lecture Notes in Computer Science*, pages 211–228. Springer, 2006.

[59] Matteo Scarlata, Matilda Backendal, and Miro Haller. MFKDF: Multiple factors knocked down flat. In Davide Balzarotti and Wenyuan Xu, editors, *USENIX Security 2024*. USENIX Association, August 2024.

[60] Peter Schwabe, Douglas Stebila, and Thom Wiggers. Post-quantum TLS without handshake signatures. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1461–1480. ACM Press, November 2020.

[61] Peter Schwabe, Douglas Stebila, and Thom Wiggers. More efficient post-quantum KEMTLS with pre-distributed public keys. In Elisa Bertino, Haya Shulman, and Michael Waidner, editors, *ESORICS 2021, Part I*, volume 12972 of *LNCS*, pages 3–22. Springer, Cham, October 2021.

[62] JH. Song, R. Poovendran, J. Lee, and T. Iwata. The AES-CMAC Algorithm. RFC 4493 (Informational), June 2006.

[63] Douglas Stebila. Security analysis of the iMessage PQ3 protocol. Cryptology ePrint Archive, Report 2024/357, 2024.

[64] Douglas Stebila, Scott Fluhrer, and Shay Gueron. Hybrid key exchange in TLS 1.3 – draft-ietf-tls-hybrid-design-10, April 2024.

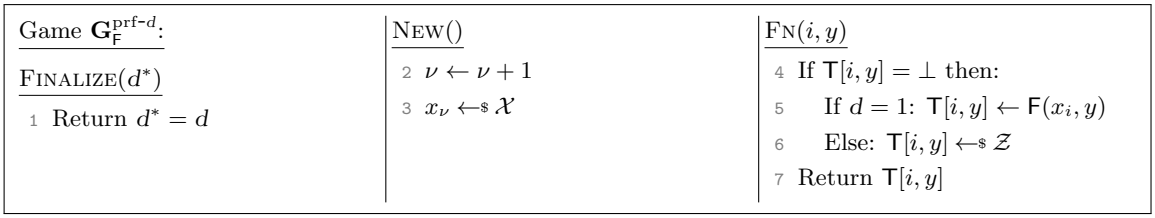

Figure 13: Game defining PRF security for function $\mathsf{F} : \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$.

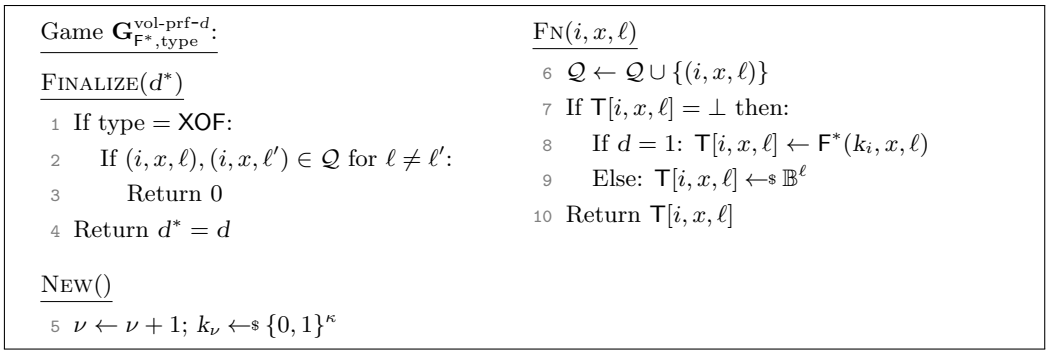

Figure 14: Game defining variable-output-length PRF security for function $\mathsf{F}^* \colon \{0,1\}^\kappa \times \mathcal{X} \times \mathbb{N} \to \{0,1\}^*$ for extendable (type = XOF) resp. non-extendable (type = NOF) outputs.

A Basic Lemmas and Definitions

A.1 PRF and swap-PRF Security

First, we recap the multi-user PRF and swap-PRF [10] security for a function $\mathsf{F} \colon \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$, where the *swapped* function $\bar{\mathsf{F}} : \mathcal{Y} \times \mathcal{X} \to \mathcal{Z}$ for F is defined as $\bar{\mathsf{F}}(y, x) = \mathsf{F}(x, y)$.

Definition 20 (PRF and swap-PRF security)**.** Let $\mathsf{F} \colon \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$ be a function and let $\bar{\mathsf{F}}(y, x) = \mathsf{F}(x, y)$. We define the advantage of an adversary $\mathcal{A}$ against the PRF and swap-PRF security of F as

$$\mathbf{Adv}_{\mathsf{F}}^{\text{prf}}(\mathcal{A}) = \Pr\left[\mathbf{G}_{\mathsf{F}}^{\text{prf-1}}(\mathcal{A})\right] - \Pr\left[\mathbf{G}_{\mathsf{F}}^{\text{prf-0}}(\mathcal{A})\right], \quad \text{resp.} \quad \mathbf{Adv}_{\mathsf{F}}^{\text{swap-prf}}(\mathcal{A}) = \mathbf{Adv}_{\bar{\mathsf{F}}}^{\text{prf}}(\mathcal{A}),$$

where game $\mathbf{G}_{\mathsf{F}}^{\text{prf-}d}$ is given in Figure 13.

A.2 Variable-Output-Length Pseudorandom Functions and Extendable-Output Random Oracles

We follow the idea in [48] to define variable-output-length pseudorandom functions (volPRF). These are functions $\mathsf{F}^* \colon \{0,1\}^\kappa \times \mathcal{X} \times \mathbb{N} \to \mathbb{B}^*$ that take as input a key from $\{0,1\}^\kappa$, some input x (such as the label in a KDF), and an output length parameter $\ell \in \mathbb{N}$, and return a pseudorandom string of length ℓ. The task of the adversary is to distinguish such outputs from random for adaptively chosen inputs (x, ℓ) when the key is random and secret. Analogously to KDFs, we can define volPRFs as XOF or NOF. The former additionally restricts the adversary from querying both (x, ℓ) and (x, ℓ') for $\ell \neq \ell'$.

Definition 21 (volPRF security)**.** Let $\mathsf{F}^* \colon \{0,1\}^\kappa \times \mathcal{X} \times \mathbb{N} \to \mathbb{B}^*$ be a function such that $\mathsf{F}^*(k, x, \ell) \in \mathbb{B}^\ell$ for any $k \in \{0,1\}^\kappa, x \in \mathcal{X}$ and $\ell \in \mathbb{N}$. We define the advantage of an adversary $\mathcal{A}$ against the vol-prf security of F^* for extendable (type = XOF) and non-extendable (type = NOF) outputs as

$$\mathbf{Adv}_{\mathsf{F}^*,\text{type}}^{\text{vol-prf}}(\mathcal{A}) = \Pr\left[\mathbf{G}_{\mathsf{F}^*,\text{type}}^{\text{vol-prf-1}}(\mathcal{A})\right] - \Pr\left[\mathbf{G}_{\mathsf{F}^*,\text{type}}^{\text{vol-prf-0}}(\mathcal{A})\right],$$

where game $\mathbf{G}_{\mathsf{F}^*,\text{type}}^{\text{vol-prf-}d}$ is given in Figure 14.

For an extendable-output version of a volPRF we penalize the adversary for querying a prefix of another query. An alternative approach, not punishing the adversary at all, would be to truncate previous answers (in case a query for a longer length parameter was made before) resp. to fill up the undetermined

$\underline{\mathsf{H}_{\mathrm{XOF}}(x, \ell)}$

$\ell_x \leftarrow |\mathsf{T}[x]|$ // initially empty entry $\mathsf{T}[x] = \varepsilon$

If $\ell_x < \ell$ then

  $y \leftarrow\!\!\$\ \mathbb{B}^{\ell - \ell_x}$

  $\mathsf{T}[x] \leftarrow \mathsf{T}[x] \| y$

Return $\mathsf{T}[x][1..\ell]$

Figure 15: Extendable-output random oracle $\mathsf{H}_{\mathrm{XOF}}$.

bits with random values instead (in case only shorter queries about the same key-input pair have been made before). This approach is equivalent via a reduction $\mathcal{B}$ which for each query (i, x, ℓ) of the adversary $\mathcal{A}$ first queries $(i, x, \ell_{\mathsf{max}}(\mathcal{A}))$ for an upper bound $\ell_{\mathsf{max}}(\mathcal{A})$ over all query lengths of $\mathcal{A}$ and then truncates for each answer. We also note that the idea of filling up unspecified parts of the output gives rise to a definition of extendable-output random oracles:

Definition 22 (Extendable-Output Random Oracles)**.** An extendable-output random oracle $\mathsf{H}_{\mathrm{XOF}}$ is defined in Figure 15.

A.3 Properties of Pseudorandom Sources

Next, we observe that pseudorandom sources, as introduced in Definition 4 are also unpredictable.

Lemma 23 (Pseudorandom sources are unpredictable)**.** *Let Σ be a key material source over $\mathsf{Skm} \times \mathsf{Ctx} \times \mathsf{Aux}$ for $\mathsf{Skm} \subseteq \{0,1\}^m$, with sample size u. Let $\mathcal{A}$ be an adversary against the predictability of Σ, making at most $\mathbf{q}_{\textsc{Predict}}(\mathcal{A})$ queries to oracle* PREDICT*. Then there exists an adversary $\mathcal{B}$, running in approximately the same time as $\mathcal{A}$, such that*

$$\mathbf{Adv}^{\mathrm{up}}_{\Sigma}(\mathcal{A}) \leq (\mathbf{q}_{\textsc{Predict}}(\mathcal{A}) + u^2) \cdot 2^{-m} + \mathbf{Adv}^{\mathrm{pr}}_{\Sigma}(\mathcal{B}).$$

Hence, any (t, ε)-pseudorandom source Σ is (t', ε')-unpredictable for $t' \approx t$ and $\varepsilon' = \varepsilon + (\mathbf{q}_{\textsc{Predict}}(\mathcal{A}) + u^2) \cdot 2^{-m}$,

Proof of Lemma 23. Adversary $\mathcal{B}$ initially receives a vector $\mathbf{z}$ of tuples $\mathbf{z}[i] = (\sigma_i, c_i, \alpha_i)$, where σ_i is either a genuine sample (if the secret bit d in $\mathcal{B}$'s game is 1) or a truly random string (if $d = 0$). It invokes adversary $\mathcal{A}$ against the unpredictability game in a black-box simulation by handing over all auxiliary data α_i. Whenever $\mathcal{A}$ calls its PREDICT about values (i, σ^*), then $\mathcal{B}$ checks if $\sigma^* = \sigma_i$ or not. Return the answer to $\mathcal{A}$ and continue the simulation. If $\mathcal{B}$ in the simulation encounters a match $\sigma^* = \sigma_i$ at some point or $\sigma_i = \sigma_j$ for $i \neq j$ at the beginning, then it outputs 1 as its guess in the pseudorandomness game, else it returns 0.

Adversary $\mathcal{B}$ clearly runs in about the same time as $\mathcal{A}$, and perfectly simulates the unpredictability game if $d = 1$ in its own game. Hence, $\Pr\left[\mathbf{G}^{\text{pr-1}}_{\Sigma}(\mathcal{B})\right] = \Pr\left[\mathbf{G}^{\mathrm{up}}_{\Sigma}(\mathcal{A}) \Rightarrow 1\right] = \mathbf{Adv}^{\mathrm{up}}_{\Sigma}(\mathcal{A})$. Furthermore, $\Pr\left[\mathbf{G}^{\text{pr-0}}_{\Sigma}(\mathcal{B})\right] = (\mathbf{q}_{\textsc{Predict}}(\mathcal{A}) + u^2) \cdot 2^{-m}$, since the secret value σ_i is sampled independently and uniformly at random from $\{0,1\}^m$ in $\mathbf{G}^{\text{pr-0}}_{\Sigma}$, such that the probability that there exists the first match in the i-th query is at most 2^{-m} for each i. Furthermore, the probability of a collision among the random values is at most $u^2 \cdot 2^{-m}$. Subtracting case $d = 0$ from case $d = 1$ gives the bound from the lemma. □

A.4 KEM Sources are Pseudorandom

We show that KEM sources are pseudorandom.

Lemma 24 (KEM sources are pseudorandom)**.** *Let Σ be a $p \times s$ KEM source with sample size $u = p \cdot s$ and publicly computable auxiliary information for a key encapsulation mechanism KEM with key length m. Let $\mathcal{A}$ be an adversary against the pseudorandomness of Σ. Then there exist adversaries $\mathcal{B}_1$, $\mathcal{B}_2$ such that*

$$\mathbf{Adv}^{\mathrm{pr}}_{\Sigma}(\mathcal{A}) \leq \mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{KEM}}(\mathcal{B}_1) \leq p \cdot s \cdot \mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{KEM}}(\mathcal{B}_2),$$

where $\mathcal{B}_1$ is a multi-user, multi-challenge IND-CPA adversary that makes $p \cdot s$ challenge queries to p users, and $\mathcal{B}_2$ is a (classic) single-user, single-challenge IND-CPA adversary.

Game $\mathbf{G}^{\text{psv-ke-}d}_{\mathsf{KE},p}$:

INITIALIZE()

1 $d \leftarrow\$ \{0,1\}$; $s \leftarrow 0$
2 For $i = 1$ to p do:
3 $\quad (pk_i, sk_i) \leftarrow\$ \mathsf{KE.KGen}()$
$\quad$ // generate parties' key pairs
4 Return $(pk_1, \ldots, pk_p)$

FINALIZE(d^*)

5 Return $d^* = d$

RUN(i, j)

6 $s \leftarrow s + 1$ // increment session counter
7 $(K_s, T_s) \leftarrow\$ \mathsf{KE.Run}(i, j, pk_i, sk_i, pk_j, sk_j)$ // execute the key exchange protocol KE between users i and j, resulting in a session key K_s and transcript T_s
8 Return T_s

REVEAL(s)

9 If $queried_s$ then: return $\perp$
10 $queried_s \leftarrow 1$ // mark sth session queried
11 Return K_s // reveal sth session key K_s

TEST(s)

12 If $queried_s$ then: return $\perp$
13 $queried_s \leftarrow 1$ // mark sth session queried
14 If $d = 1$:
15 $\quad$ Then: $K \leftarrow K_s$
16 $\quad$ Else: $K \leftarrow \{0,1\}^{|K_s|}$
17 Return K // real-or-random key

Figure 16: Game $\mathbf{G}^{\text{psv-ke-}d}_{\mathsf{KE},p}$ defining passive key exchange security for a key exchange protocol $\mathsf{KE} = (\mathsf{KGen}, \mathsf{Run})$.

Proof. The multi-user, multi-challenge reduction $\mathcal{B}_1$ creates p users, obtaining their public keys pk_i. It then issues s encapsulation challenge queries against each of them, obtaining real-or-random shared secrets $K_{i,j}$ and ciphertexts $c_{i,j}$. It populates the key material vector $\mathbf{z}$ with corresponding entries $(K_{i,j}, (pk_i, c_{i,j}), \alpha_{i,j})$, using that the auxiliary information is publicly computable. When $\mathcal{A}$ outputs its bit guess, $\mathcal{B}_1$ forwards this as its own. Depending on its challenge bit, $\mathcal{B}_1$ simulates $\mathbf{G}^{\text{pr-1}}_{\Sigma}$ or $\mathbf{G}^{\text{pr-0}}_{\Sigma}$, establishing the first part of the bound.

The second part follows via a standard hybrid argument from multi-user, multi-challenge IND-CPA security to the classic, single-user, single-challenge version. □

A.5 Passive Key Exchange Security

Figure 16 formalizes passive key exchange security for a key exchange protocol $\mathsf{KE} = (\mathsf{KGen}, \mathsf{Run})$, where $\mathsf{KE.KGen}()$ generates a user key pair (pk, sk) and $\mathsf{KE.Run}(i, j, pk_i, sk_i, pk_j, sk_j)$ executes the key exchange protocols between two users with identity i resp. j taking their key pairs as input.

B Extendable and Non-extendable Output Functions

Some applications such as TLS 1.3 [57] bind the input key material to its intended output length, effectively transforming a XOF-KDF to a NOF-KDF. We now provide a formal justification of this strategy and show that including ℓ in the label turns a XOF-KDF into a NOF-KDF. We finally argue that one can turn any NOF-KDF into a XOF-KDF if there is a reasonable upper bound $\ell_{\max}$ on the output length: simply generate the maximal-size output in each call and truncate to the desired length ℓ.

We will now formalize these definitions and provide proofs for our claims.

Definition 25 (XOF/NOF KDF)**.** Let F be an n-KDF and let $(\sigma_1, c_1), \ldots, (\sigma_n, c_n)$ be a tuple of secrets and contexts generated by the sources in a source collection and let $\ell_1, \ell_2 \in \mathsf{F.KLout}$. Let

$$K_1 \leftarrow \mathsf{F}((\sigma_1, c_1), \ldots, (\sigma_n, c_n), L, \ell_1) \text{ and}$$
$$K_2 \leftarrow \mathsf{F}((\sigma_1, c_1), \ldots, (\sigma_n, c_n), L, \ell_2)$$

If $\ell_1 \leq \ell_2 \implies K_1 \preccurlyeq K_2$ for all inputs, then we say that F is an *extendable-output* n-KDF (XOF-n-KDF). If there exists some inputs and lengths $\ell_1 \leq \ell_2$ such that K_1 is not a prefix of K_2, then we instead call F a *non-extendable output* n-KDF (NOF-n-KDF).

The security that we expect from NOF-n-KDFs is slightly stronger than for XOF-n-KDFs. For a NOF-n-KDF, we require that the output of the function is always indistinguishable from random, even

given outputs of the KDF on related inputs. For XOF-n-KDFs, this clearly will not be the case due to the extendable outputs. Hence, we weaken the security notion by imposing more restrictions on the adversary to obtain XOF KDF security. In the XOF version, we prohibit queries $(\mathbf{v}, L, *)$ to the RO$-KDF oracle that only differ in the output-length compared to previous queries to either the KDF or RO$-KDF oracle. Since we expect XOF KDF outputs to be prefixes of each other for such inputs, repeating them across oracles would trivially allow for distinguishing the outputs. Formally, we have modeled this by leveraging different freshness requirements req_{NOF} and req_{XOF} for NOF-n-KDF and XOF-n-KDF, respectively. Note that this freshness property alone does not rule out trivial attacks, such as an adversary evaluating the KDF exclusively on adversarially provided key material. A secure NOF-n-KDF fulfills the predicate req_N while a secure XOF-n-KDF fulfills the predicate req_X

The following proposition states that any secure NOF-n-KDF is also a secure XOF-n-KDF, whereas the converse is, in general, not true.

Proposition 26 (NOF-n-KDF-security is strictly stronger than XOF-n-KDF-security)**.**

1. *For any n-KDF F, any source collection $\mathbf{\Sigma}$, any req, and any adversary $\mathcal{A}$ we have* $\mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{F},\mathbf{\Sigma},req_{\mathsf{XOF}}\wedge req}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{F},\mathbf{\Sigma},req_{\mathsf{NOF}}\wedge req}(\mathcal{A})$.
2. *For any XOF-n-KDF XF with $\ell_1 < \ell_2 \in \mathsf{KLout}$ and any source collection $\mathbf{\Sigma}$, there exists an adversary $\mathcal{A}$ such that* $\mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{XF},\mathbf{\Sigma},req_N}(\mathcal{A}) \geq 1 - 2^{-\ell_1}$.

Proof.

1. Note that $req_{\mathsf{XOF}} \implies req_{\mathsf{NOF}}$, because if all entries $(\mathbf{v}, L_i, *)$, $(\mathbf{v}, L_j, *)$ are distinct for $i \neq j$ according to req_{XOF}, then they are distinct, too, when taking the length values ℓ_i, ℓ_j into account as required by req_{NOF}. That is, an adversary that satisfies XOF freshness also satisfies NOF freshness. Hence for any n-KDF F, source collection $\mathbf{\Sigma}$, any requirement req, and any $\mathcal{A}$,
$$\mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{F},\mathbf{\Sigma},req_{\mathsf{XOF}}\wedge req}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{F},\mathbf{\Sigma},req_{\mathsf{NOF}}\wedge req}(\mathcal{A}) \ .$$
2. Adversary $\mathcal{A}$ initializes n keys via oracle NewKey. He then queries oracle RO$-KDF using the generated keys in their according positions as mapped by Σ-map associated with $\mathbf{\Sigma}$. It makes the two valid queries $(\mathbf{v}, L, \ell_1)$ and then again on $(\mathbf{v}, L, \ell_2)$ where $\ell_1 < \ell_2 \in \mathsf{KLout}$.

   Then, it checks whether the first response is a prefix of the second one, and only outputs 1 if this is the case. Since the queries do not infringe predicate req_N, the adversary always outputs 1 if oracle RO$-KDF returns the actual KDF value, and with probability at most $2^{-\ell_1}$ if both answers are random. This yields a distinguishing advantage of $1 - 2^{-\ell_1}$. □

Note, however, that while any XOF-n-KDF that is secure under req_N is also secure under req_X, this does not make it a XOF-n-KDF, since it does not exhibit the extendable-output property. Yet, we observe that any XOF-n-KDF can be turned into a NOF-n-KDF by encoding the output length as part of the label. Formally, we show that there exists a transform XtoN such that for any secure XOF-n-KDF XF, the resulting $\mathsf{XtoN}[\mathsf{XF}]$ is a secure NOF-n-KDF. This transform matches, for instance, the strategy employed by TLS 1.3 [57] where the desired output length of HKDF becomes part of `Hkdflabel`. More formally, the label consists of the length value ℓ encoded with two octets, the encoding of the constant `"tls13 "` followed by context information. Our result for injective encodings thus confirms that this provably turns the XOF-n-KDF HKDF into a NOF-n-KDF-version.

Definition 27 (XOF-to-NOF transform XtoN)**.** Let XF be a XOF-n-KDF. Then for all inputs $(\sigma_1, c_1), \ldots, (\sigma_n, c_n), L, \ell$ and $\ell \in \mathsf{XF.KLout}$, we let
$$\mathsf{XtoN}[\mathsf{XF}]((\sigma_1, c_1), \ldots, (\sigma_n, c_n), L, \ell) := \mathsf{XF}((\sigma_1, c_1), \ldots, (\sigma_n, c_n), \langle L, \ell \rangle, \ell) \ ,$$
where $\langle L, \ell \rangle$ is an injective encoding which ensures that if $L \neq L'$ or $\ell \neq \ell'$, then $\langle L, \ell \rangle \neq \langle L', \ell' \rangle$. The resulting n-KDF $\mathsf{XtoN}[\mathsf{XF}]$ output length set $\mathsf{XF.KLout}$.

Proposition 28 (XtoN preserves KDF security)**.** *Let XF be a XOF-n-KDF and let $\mathsf{F} := \mathsf{XtoN}[\mathsf{XF}]$. Then for any source collection $\mathbf{\Sigma}$ and any adversary $\mathcal{A}$ against the* kdf *security of F, there exists an adversary $\mathcal{B}$ such that*
$$\mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{F},\mathbf{\Sigma},req_N\wedge req}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{XF},\mathbf{\Sigma},req_X\wedge req}(\mathcal{B}) \ .$$
where req can be one of the requirements given in Table 1 or `true`.

Adversary $\mathcal{B}$ has roughly the same running time as $\mathcal{A}$ and makes the same number of queries to each of the oracles.

Proof. Adversary $\mathcal{B}$ runs adversary $\mathcal{A}$, simulating access to the oracles in game $\mathbf{G}^{\text{kdf}}_{\mathsf{F},\mathbf{\Sigma},req_N \wedge req}$ as follows. For oracles NewKey and SetKey, $\mathcal{B}$ simply relays the queries to its own corresponding oracles. If $\mathcal{A}$ issues a query with inputs $(\mathbf{v}, L, \ell)$ to oracle Ro$-KDF or KDF, adversary $\mathcal{B}$ queries the corresponding oracle in its game on input $(\mathbf{v}, \langle L, \ell \rangle, \ell)$. When adversary $\mathcal{A}$ halts and outputs a bit d^*, adversary $\mathcal{B}$ also halts and returns d^*.

This way, $\mathcal{B}$ perfectly simulates game $\mathbf{G}^{\text{kdf}}_{\mathsf{F},\mathbf{\Sigma},req_N \wedge req}$ for $\mathcal{A}$. Additionally, if the queries issued by $\mathcal{A}$ fulfill requirement req_{NOF}, then the queries made by adversary $\mathcal{B}$ fulfill req_{XOF}. For this, we observe that for each query with inputs $(\mathbf{v}, L, \ell)$ by adversary $\mathcal{A}$ to oracle Ro$-KDF and each other query to oracle Ro$-KDF or KDF with inputs $(\mathbf{v}', L', \ell')$, req_{NOF} means that $(\mathbf{v}, L, \ell) \neq (\mathbf{v}', L', \ell')$, implying that either $\mathbf{v} \neq \mathbf{v}'$, $L \neq L'$, or $\ell \neq \ell'$. Thanks to the injectivity of the encoding scheme in the XtoN transform, this implies that $(\mathbf{v}, \langle L, \ell \rangle) \neq (\mathbf{v}', \langle L', \ell' \rangle)$, which is equivalent to req_{XOF}.

Since req_{NOF} vs. req_{XOF} is the only difference from req_N to req_X, $\mathcal{B}$ wins whenever $\mathcal{B}$ wins and hence $\mathbf{Adv}^{\text{kdf}}_{\mathsf{F},\mathbf{\Sigma},req_N}(\mathcal{A}) \leq \mathbf{Adv}^{\text{kdf}}_{\mathsf{XF},\mathbf{\Sigma},req_X}(\mathcal{B})$. For the (optional) additional requirement req, observe that an adversary $\mathcal{A}$ against $\mathbf{G}^{\text{kdf}}_{\mathsf{F},\mathbf{\Sigma},req_N \wedge req}$ must adhere to them in the same way adversary $\mathcal{B}$ against $\mathbf{G}^{\text{kdf}}_{\mathsf{F},\mathbf{\Sigma},req_X \wedge req}$. In combination with the previous observation that $\mathcal{B}$ fulfills req_N, this establishes the claim. □

The other direction holds conditionally. That is, a NOF-n-KDF F can be turned into a XOF-n-KDF XF if there exists a reasonable upper bound $\ell_{\max}$ on the output key length. For example, HKDF, despite being a XOF-n-KDF, returns keys of at most 255 hash outputs such that $\ell_{\max} = 255 \cdot hl$. The key derivation functions in NIST's standard SP800-108 [24], all NOF-n-KDFs, put an upper limit of $2^{32} - 1$ of pseudorandom function outputs, but allow applications to enforce smaller upper bounds. The transformed $\mathsf{NtoX}[\mathsf{NF}]$ now simply computes NF for length $\ell_{\max}$ and then truncates the result to the required output length.

Definition 29 (NOF-to-XOF transform)**.** Let NF be a NOF-n-KDF with maximal output length $\ell_{\max} \in \mathsf{NF.KLout}$. Then for all inputs$(\sigma_1, c_1), \ldots, (\sigma_n, c_n), L, \ell$ where $\ell \in \mathsf{NF.KLout}$, we let

$$\mathsf{NtoX}[\mathsf{NF}]((\sigma_1, c_1), \ldots, (\sigma_n, c_n), L, \ell) := \mathsf{NF}((\sigma_1, c_1), \ldots, (\sigma_n, c_n), L, \ell_{\max})[1..\ell].$$

The resulting n-KDF $\mathsf{NtoX}[\mathsf{NF}]$ has output length set $\mathsf{NF.KLout}$.

It is straightforward to see that the transformed function $\mathsf{NtoX}[\mathsf{NF}]$ adheres to the extendable output property. For any inputs $(\mathbf{v}, L, \ell_1)$, $(\mathbf{v}, L, \ell_2)$ for $\ell_1 \leq \ell_2$ the first output is a prefix of the second one since in both cases the function is evaluated for the same input $(\mathbf{v}, L, \ell_{\max})$ and truncates the output to ℓ_1 resp. ℓ_2 characters.

Proposition 30 (NtoX preserves KDF security)**.** *Let NF be a NOF-n-KDF with maximal output length ℓ_{max}, and let $\mathsf{F} := \mathsf{NtoX}[\mathsf{NF}]$. Then for any source collection $\mathbf{\Sigma}$, any requirements req and any adversary $\mathcal{A}$ against the* kdf *security of F, there exists an adversary $\mathcal{B}$ such that*

$$\mathbf{Adv}^{\text{kdf}}_{\mathsf{F},\mathbf{\Sigma},req_{\mathsf{XOF}}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{kdf}}_{\mathsf{NF},\mathbf{\Sigma},req_{\mathsf{NOF}}}(\mathcal{B}) \ .$$

Adversary $\mathcal{B}$ has roughly the same running time as $\mathcal{A}$ and makes the same number of queries to each of the oracles.

Proof. Adversary $\mathcal{B}$ once more runs adversary $\mathcal{A}$, simulating access to the oracles named in game $\mathbf{G}^{\text{kdf}}_{\mathsf{F},\mathbf{\Sigma},req_{\mathsf{XOF}}}$ via its own oracles. Adversary $\mathcal{B}$ keeps track of $\mathcal{A}$'s queries to oracles Ro$-KDF and KDF in lists $\mathcal{Q}_{\$}$ and $\mathcal{Q}_{\text{r}}$. Whenever $\mathcal{A}$ makes a query $(\mathbf{v}, L, \ell)$ to oracle Ro$-KDF or KDF, adversary $\mathcal{B}$ first checks if $(\mathbf{v}, L, *)$ has been recorded in the correspond list with response K; if so $\mathcal{B}$ returns the ℓ-bit prefix of this value K. Else, $\mathcal{B}$ calls its oracle about $(\mathbf{v}, L, \ell_{\max})$ to get a response K. It stores the query and the answer in the corresponding list $\mathcal{Q}_{\$}$ or $\mathcal{Q}_{\text{r}}$ and returns the ℓ-bit prefix of K. Adversary $\mathcal{B}$ eventually returns $\mathcal{A}$'s output bit upon termination.

Observe that $\mathcal{B}$ perfectly emulates the $\mathsf{NtoX}[\mathsf{NF}]$ transform. Furthermore, $\mathcal{B}$ only queries its oracles Ro$-KDF and KDF about $\mathcal{A}$'s values $(\mathbf{v}, L)$ for the same length parameter $\ell_{\max}$. Hence, if $\mathcal{A}$ adheres to the requirements req_X, then so does $\mathcal{B}$ for requirement req_N. The bound now follows. □

If the source collection $\mathbf{\Sigma}$ consists solely of uniform sources, then a more efficient transform exists. The idea, which already appears in many designs like HKDF, is to generate some fixed-length pseudorandom output via the NOF-n-KDF and to iterate the NOF-n-KDF sufficiently often to (deterministically) stretch this pseudorandom value to the desired length. Let κ be the length of the keys in $\mathsf{Skm}_1 \in \mathbf{\Sigma}$ and let $\hat{\ell} \in \mathbb{N}$ be some fixed length. To compute $\mathsf{XF}((\sigma_1, c_1), \ldots, (\sigma_n, c_n), L, \ell)$, first compute $PRK \leftarrow$

$\mathsf{F}((\sigma_1, c_1), \ldots, (\sigma_n, c_n), L, \kappa)$, then iterate $k_i \leftarrow \mathsf{F}((PRK, \varepsilon), (0, \varepsilon) \ldots, (0, \varepsilon), \langle L, i\rangle, \hat{\ell})$ for $i = 1$ to $\lceil \ell/\hat{\ell} \rceil$, and then return $k_1 \| k_2 \| \ldots$ truncated to ℓ bits. Note that if one is willing to assume the existence and availability of a PRF, then the iterated calls to F can be replaced by iterated computations of the PRF instead.

Specific Transformation for HKDF. Recall that the HKDF expansion function HKDF.Expand computes the output for the pseudorandom key material PRK and the context information c as

$$\begin{aligned} T(0) &= \varepsilon \\ T(i) &= \mathsf{HMAC}(PRK, T(i-1) || c || [i]_1) \quad \text{for } i = 1, 2, \ldots, N = \lceil \ell/hl \rceil. \end{aligned}$$

where the counter value i is encoded as a single octet. The procedure returns the first ℓ octets of $T(1) \| T(2) \| \ldots \| T(N)$, yielding an extendable output function.

One option to turn HKDF into a non-extendable output function is to use our general result and include the length parameter ℓ in the context information c. This route is, for example, taken by TLS 1.3 [57], carefully ensuring that the number of internal hash evaluations for HMAC is still optimal. Another option, which we suggest here and which is currently not covered by the standard, is to use the empty label $T(0)$ instead to encode the length parameter (with at most hl characters). That is, define the expansion step as

$$\begin{aligned} T(0) &= [\ell]_{hl} \\ T(i) &= \mathsf{HMAC}(PRK, T(i-1) || c || [i]_1) \quad \text{for } i = 1, 2, \ldots, N = \lceil \ell/hl \rceil. \end{aligned}$$

Because HKDF requires $\ell \leq 255 \cdot hl$ for the fixed-length encoding of the counter value i, encoding the length value with hl octets in $T(0)$ is feasible for any practical parameter choices. One can also use shorter encodings of ℓ in $T(0)$ if key derivation only requires shorter outputs. The advantage of this method is that the users do not need to put the length parameter themselves into the context information.

C Proof of Proposition 7 (Eliminating the KDF Oracle)

Proof. For this proof we have to distinguish between non-extendable output KDFs and extendable output KDFs, characterized by the predicates req_{NOF} or req_{XOF}, respectively. As discussed in Section 4.2, the requirements predicate always contains at least one of the two terms in order to obtain a meaningful definition.

NOF KDFs. We start with the case where req_{NOF} is included in req. Assume w.l.o.g. that adversary $\mathcal{A}$ never repeats the same query to oracle RO$-KDF or KDF. If it did, it would either receive the same response (for repeated queries to oracle KDF), or violate the NOF freshness condition req_{NOF} (for repeated queries to RO$-KDF). Hence, we can construct an almost as efficient adversary with the same or higher advantage which does not repeat queries. Furthermore, assume that $\mathcal{A}$ does not make any "unnecessary" queries to oracle KDF for which it already knows all of the input secrets. This is also w.l.o.g., as $\mathcal{A}$ could compute the response to such queries by itself. Finally, we assume that $\mathcal{A}$ adheres to the requirement req, as $\mathcal{A}$ would lose the game otherwise. In summary, we assume that all queries to oracles RO$-KDF and KDF by $\mathcal{A}$ are distinct, contain at least one honest input secret, use all honest secrets in valid positions, and that there are no dishonest key collisions.

The proof proceeds via a sequence of games G_0–G_2, where G_0 is $\mathbf{G}^{\text{kdf-1}}_{\mathsf{F},\boldsymbol{\Sigma},req}$ (that is, the "real" KDF game). Let G_1 be equivalent to $\mathbf{G}^{\text{kdf-0}}_{\mathsf{F},\boldsymbol{\Sigma},req}$ (the "random" KDF game), except that oracle KDF also responds to all valid queries with freshly sampled random strings of the required output length. That is, in G_1, both oracle RO$-KDF and oracle KDF return random responses. We construct an adversary $\mathcal{B}_1$ such that

$$\Pr[\,\mathrm{G}_0(\mathcal{A})\,] - \Pr[\,\mathrm{G}_1(\mathcal{A})\,] \leq \mathbf{Adv}^{\text{kdf}}_{\mathsf{F},\boldsymbol{\Sigma},req \wedge req_{\mathsf{NoReal}}}(\mathcal{B}_1). \tag{3}$$

Adversary $\mathcal{B}_1$ runs adversary $\mathcal{A}$, acting as the challenger in game G_0 and simulates access to oracles NewKey, SetKey, and RO$-KDF by relaying the queries to its own corresponding oracles. To simulate oracle KDF, adversary $\mathcal{B}_1$ forwards the query to oracle RO$-KDF in its own game. This gives adversary $\mathcal{B}_1$ query count $\mathbf{q}_{\mathrm{OR}}(\mathcal{B}_1) = \mathbf{q}_{\mathrm{OR}}(\mathcal{A})$ for $\mathrm{OR} \in \{\textsc{NewKey}, \textsc{SetKey}\}$, and $\mathbf{q}_{\text{RO\$-KDF}}(\mathcal{B}_1) = \mathbf{q}_{\text{RO\$-KDF}}(\mathcal{A}) + \mathbf{q}_{\text{KDF}}(\mathcal{A})$. When $\mathcal{A}$ halts and returns, $\mathcal{B}_1$ halts and returns the same output. Depending on the bit d in the game played by $\mathcal{B}_1$, this simulates either game G_0 (if $d = 1$) or G_1 (if $d = 0$).

By assumption, all of the queries by $\mathcal{A}$ satisfy the requirements req, hence adversary $\mathcal{B}_1$ also satisfies req. Furthermore, adversary $\mathcal{B}_1$ makes no queries to oracle KDF, thereby satisfying req_{NoReal}. Hence $\Pr[\mathbf{G}^{\text{kdf-}d}_{\mathsf{F},\boldsymbol{\Sigma},req \wedge req_{\mathsf{NoReal}}}(\mathcal{B}_1)] = \Pr[\,\mathrm{G}_{1-d}(\mathcal{A})\,]$. Subtracting case $d = 0$ from case $d = 1$ yields Equation (3).

Next, let game G_2 be identical to G_1, except that oracle KDF once again returns computations of F rather than random strings. We construct an adversary $\mathcal{B}_2$ such that

$$\Pr[\,G_1(\mathcal{A})\,] - \Pr[\,G_2(\mathcal{A})\,] \leq \mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{F},\boldsymbol{\Sigma},req\wedge req_{\mathsf{NoReal}}}(\mathcal{B}_2). \tag{4}$$

Adversary $\mathcal{B}_2$ acts like the challenger in game G_1.

$\mathcal{B}_2$ simulates oracles NewKey and SetKey by forwarding the queries to its own corresponding oracles and and returns their outputs. When receiving queries to Ro$-KDF, $\mathcal{B}_2$ returns uniformly random strings of appropriate length. To simulate responses to KDF queries, adversary $\mathcal{B}_2$ forwards them to its Ro$-KDF oracle and returns the output. When $\mathcal{A}$ halts and returns d^*, $\mathcal{B}_2$ also halts and outputs the same bit d^*.

Hence, $\mathbf{q}_{\mathrm{OR}}(\mathcal{B}_2) = \mathbf{q}_{\mathrm{OR}}(\mathcal{A})$ for $\mathrm{OR} \in \{\textsc{NewKey}, \textsc{SetKey}\}$, and $\mathbf{q}_{\text{Ro\$-KDF}}(\mathcal{B}_2) = \mathbf{q}_{\mathrm{KDF}}(\mathcal{A})$. This way, adversary $\mathcal{B}_2$ simulates game G_1 or G_2 for $\mathcal{A}$ depending on the value of the bit d in its own game. Since it makes no queries to oracle KDF, $\mathcal{B}_2$ trivially satisfies req_{NoReal}. Furthermore, in order to succeed adversary $\mathcal{A}$ has to satisfy req, which means $\mathcal{B}_2$ making the same queries also satisfies req.

Therefore, we obtain $\Pr[\mathbf{G}^{\text{kdf-0}}_{\mathsf{F},\boldsymbol{\Sigma},req\wedge req_{\mathsf{NoReal}}}(\mathcal{B}_2)] = \Pr[G_1(\mathcal{A})]$ as well as the equation $\Pr[\mathbf{G}^{\text{kdf-1}}_{\mathsf{F},\boldsymbol{\Sigma},req\wedge req_{\mathsf{NoReal}}}(\mathcal{B}_2)] = \Pr[G_2(\mathcal{A})]$. Subtracting the two values yields Equation (4).

Combining Equation (3) and (4) with standard rewriting of the definition of $\mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{F},\boldsymbol{\Sigma},req}(\mathcal{A})$, we obtain

$$\mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{F},\boldsymbol{\Sigma},req}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{F},\boldsymbol{\Sigma},req\wedge req_{\mathsf{NoReal}}}(\mathcal{B}_1) + \mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{F},\boldsymbol{\Sigma},req\wedge req_{\mathsf{NoReal}}}(\mathcal{B}_2)\ . \tag{5}$$

Finally, we construct adversary $\mathcal{B}$ by running one of $\mathcal{B}_1$ and $\mathcal{B}_2$ at random, such that

$$\mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{F},\boldsymbol{\Sigma},req\wedge req_{\mathsf{NoReal}}}(\mathcal{B}_1) + \mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{F},\boldsymbol{\Sigma},req\wedge req_{\mathsf{NoReal}}}(\mathcal{B}_2) = 2 \cdot \mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{F},\boldsymbol{\Sigma},req\wedge req_{\mathsf{NoReal}}}(\mathcal{B})\ , \tag{6}$$

with $\mathbf{q}_{\mathrm{OR}}(\mathcal{B}) = \max(\mathbf{q}_{\mathrm{OR}}(\mathcal{B}_1), \mathbf{q}_{\mathrm{OR}}(\mathcal{B}_2)) = \mathbf{q}_{\mathrm{OR}}(\mathcal{A})$ for oracle $\mathrm{OR} \in \{\textsc{NewKey}, \textsc{SetKey}\}$, $\mathbf{q}_{\text{Ro\$-KDF}}(\mathcal{B}) = \max(\mathbf{q}_{\text{Ro\$-KDF}}(\mathcal{B}_1), \mathbf{q}_{\text{Ro\$-KDF}}(\mathcal{B}_2)) = \mathbf{q}_{\text{Ro\$-KDF}}(\mathcal{A}) + \mathbf{q}_{\mathrm{KDF}}(\mathcal{A})$. Combining Equations (5) and (6) yields the claim for NOF KDFs.

XOF KDFs. Now we look at the second case where req contains req_{XOF}. This case works similarly to the NOF case, but we have to modify the behavior of the reductions to account for the prefix property. It is important to observe, that while querying the Ro$-KDF with identical inputs but different output lengths violates the freshness requirement req_{XOF}, such queries to the "regular" KDF are, in fact, admissible. For this case, we assume the existence of some maximum length for queries to Ro$-KDF and KDF, which we call $\ell_{\mathsf{max}}(\mathcal{A})$. While longer queries are possible in theory, the assumption is that no adversary will execute such a query. As a result, reductions can simulate answers to the oracle KDF by modifying the length to be $\ell_{\mathsf{max}}(\mathcal{A})$ and then relaying the query to its own Ro$-KDF oracle. The response is then stored and truncated to the length that was initially requested. For any future queries to Ro$-KDF or KDF with the same input but a different length, the reductions do not utilize their own oracles, but truncate the stored values to the appropriate length.

We assume again, w.l.o.g., that adversary $\mathcal{A}$ never repeats the same query to oracle Ro$-KDF or KDF $\mathcal{A}$ and does not make any "unnecessary" queries to oracle KDF for which it already knows all of the input secrets. Additionally, assume $\mathcal{A}$ does not query Ro$-KDF or KDF with $\ell > \ell_{\mathsf{max}}(\mathcal{A})$, i.e., we assume that $\ell_{\mathsf{max}}(\mathcal{A})$ is the maximum length value over all of $\mathcal{A}$'s queries (which is upper bounded by the maximal admissible key length $\ell_{\max}$ of the KDF, but can be smaller).

Once again, we use a sequence of games G_0–G_2, where G_0 is $\mathbf{G}^{\text{kdf-1}}_{\mathsf{F},\boldsymbol{\Sigma},req}$. Let G_1 be equivalent to $\mathbf{G}^{\text{kdf-0}}_{\mathsf{F},\boldsymbol{\Sigma},req}$, except that oracle KDF responds to all valid queries with freshly sampled random strings of the required output length. That is, in G_1, both oracle Ro$-KDF and oracle KDF return random responses while also respecting the XOF prefix property. Meaning, the output for queries of the type $(\mathbf{v}, L)$ are randomly sampled once for $\ell_{\mathsf{max}}(\mathcal{A})$, but for all following queries with a different length value, the outputs are prefixes of each other.

We construct an adversary $\mathcal{B}_1$ such that

$$\Pr[\,G_0(\mathcal{A})\,] - \Pr[\,G_1(\mathcal{A})\,] \leq \mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{F},\boldsymbol{\Sigma},req\wedge req_{\mathsf{NoReal}}}(\mathcal{B}_1), \tag{7}$$

Adversary $\mathcal{B}_1$ runs adversary $\mathcal{A}$, acting as the challenger in game G_0 and simulating access to oracles NewKey, SetKey, and Ro$-KDF by relaying the queries to its own corresponding oracles. Recall that repeated queries only differing in length to the Ro$-KDF are disallowed by req_{XOF}. As such we have to manually preserve the prefix property for such calls to KDF.

For queries to KDF, that by definition consist of the tuple $(\mathbf{v}, L, \ell)$, $\mathcal{B}_1$ first checks, if the same query, but with a different length, has been made before: $(\mathbf{v}, L, *) \stackrel{?}{\in} \mathcal{Q}_{\$} \cup \mathcal{Q}_{\mathrm{r}}$. If this is the case, $\mathcal{B}_1$ retrieves the

response it previously received from its own RO$-KDF oracle and truncates the value to the requested length ℓ. Otherwise, $\mathcal{B}_1$ modifies the ℓ input by replacing it with $\ell_{\mathsf{max}}(\mathcal{A})$ and forwards the resulting query $(\mathbf{v}, L, \ell_{\mathsf{max}}(\mathcal{A}))$ to its own RO$-KDF oracle. It stores the response but returns the ℓ-bit prefix. This procedure ensures that the freshness property for XOF is achieved while replies consistently exhibit the XOF property.

This gives adversary $\mathcal{B}_1$ query count $\mathbf{q}_{\mathrm{OR}}(\mathcal{B}_1) = \mathbf{q}_{\mathrm{OR}}(\mathcal{A})$ for oracle $\mathrm{OR} \in \{\textsc{NewKey}, \textsc{SetKey}\}$, and $\mathbf{q}_{\text{RO\$-KDF}}(\mathcal{B}_1) \leq \mathbf{q}_{\text{RO\$-KDF}}(\mathcal{A}) + \mathbf{q}_{\mathrm{KDF}}(\mathcal{A})$. When $\mathcal{A}$ halts and returns, $\mathcal{B}_1$ halts and returns the same output. Depending on the bit d in the game played by $\mathcal{B}_1$, this simulates either game G_0 (if $d = 1$) or G_1 (if $d = 0$).

By assumption, all of the queries by $\mathcal{A}$ to both RO$-KDF and KDF satisfy the provided requirement *req* Furthermore, $\mathcal{B}_1$ does not query the real KDF oracle KDF, thus additionally satisfying req_{NoReal}. We get $\Pr[\mathbf{G}^{\text{kdf-}d}_{\mathsf{F},\mathbf{\Sigma},req\wedge req_{\mathsf{NoReal}}}(\mathcal{B}_1)] = \Pr[\mathrm{G}_{1-d}(\mathcal{A})]$. Subtracting case $d = 0$ from case $d = 1$ yields Equation (7).

Next, we modify G_1 such that the KDF oracle always outputs real values, while the RO$-KDF oracle still outputs random values only and call the resulting game G_2. Note that G_2 is identical to $\mathbf{G}^{\text{kdf-1}}_{\mathsf{F},\mathbf{\Sigma},req}$.

Now we construct an adversary $\mathcal{B}_2$ such that

$$\Pr[\,\mathrm{G}_1(\mathcal{A})\,] - \Pr[\,\mathrm{G}_2(\mathcal{A})\,] \leq \mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{F},\mathbf{\Sigma},req\wedge req_{\mathsf{NoReal}}}(\mathcal{B}_2), \tag{8}$$

Adversary $\mathcal{B}_2$ acts like the challenger in game G_1, but simulates oracles NewKey and SetKey by forwarding the queries to its own corresponding oracles. For queries to the RO$-KDF oracle, it samples a uniformly random string of length $\ell_{\mathsf{max}}(\mathcal{A})$, stores the resulting string, and returns a truncation to the requested length. If the same (note that "same" in the context of XOF KDFs means the length might be different) input has already been queried $\mathcal{B}_2$ retrieves the corresponding random string it stored earlier and returns the truncated version. To simulate responses to KDF queries, adversary $\mathcal{B}_2$ forwards requests to its RO$-KDF.

Eventually, $\mathcal{A}$ terminates and outputs a bit b. $\mathcal{B}_2$ now evaluates *req* using the lists $\mathbf{k}', \mathcal{Q}'_{\$}, \mathcal{Q}'_{\mathrm{r}}$ it accumulated during $\mathcal{A}$'s execution. If the requirement evaluates to false, $\mathcal{B}_2$ return 0, otherwise it forwards b.

Hence, $\mathbf{q}_{\mathrm{OR}}(\mathcal{B}_2) = \mathbf{q}_{\mathrm{OR}}(\mathcal{A})$ for $\mathrm{OR} \in \{\textsc{NewKey}, \textsc{SetKey}\}$, and we also get $\mathbf{q}_{\text{RO\$-KDF}}(\mathcal{B}_2) \leq \mathbf{q}_{\mathrm{KDF}}(\mathcal{A})$.

$\mathcal{B}_2$ now simulates either $\mathbf{G}_1$ or $\mathbf{G}_2$ depending on the value d in its own game. By modifying the output length of queries by $\mathcal{A}$ to RO$-KDF and KDF, outputs of $\mathcal{B}_2$ also exhibit the prefix property, which we expect of its answers. Lastly, since no queries to the real KDF oracle KDF are made, $\mathcal{B}_2$ also satisfies req_{NoReal}.

Combining Equation (7) and (8) with standard rewriting of the definition of $\mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{F},\mathbf{\Sigma},req}(\mathcal{A})$, we obtain

$$\mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{F},\mathbf{\Sigma},req}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{F},\mathbf{\Sigma},req\wedge req_{\mathsf{NoReal}}}(\mathcal{B}_1) + \mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{F},\mathbf{\Sigma},req\wedge req_{\mathsf{NoReal}}}(\mathcal{B}_2)\ . \tag{9}$$

Finally, we construct adversary $\mathcal{B}$ from $\mathcal{B}_1$ and $\mathcal{B}_2$ (by running one of them at random) such that

$$\mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{F},\mathbf{\Sigma},req\wedge req_{\mathsf{NoReal}}}(\mathcal{B}_1) + \mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{F},\mathbf{\Sigma},req\wedge req_{\mathsf{NoReal}}}(\mathcal{B}_2) = 2 \cdot \mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{F},\mathbf{\Sigma},req\wedge req_{\mathsf{NoReal}}}(\mathcal{B})\ , \tag{10}$$

with, $\mathbf{q}_{\mathrm{OR}}(\mathcal{B}) = \max(\mathbf{q}_{\mathrm{OR}}(\mathcal{B}_1), \mathbf{q}_{\mathrm{OR}}(\mathcal{B}_2)) = \mathbf{q}_{\mathrm{OR}}(\mathcal{A})$ for oracle $\mathrm{OR} \in \{\textsc{NewKey}, \textsc{SetKey}\}$, as well as $\mathbf{q}_{\text{RO\$-KDF}}(\mathcal{B}), = \max(\mathbf{q}_{\text{RO\$-KDF}}(\mathcal{B}_1)$, and $\mathbf{q}_{\text{RO\$-KDF}}(\mathcal{B}_2)) = \mathbf{q}_{\text{RO\$-KDF}}(\mathcal{A}) + \mathbf{q}_{\mathrm{KDF}}(\mathcal{A})$. Combining Equations (5) and (6) yields the same bound as we have achieved in the case for NOF KDFs.

For the final bound we have to merge the bounds we have shown for the NOF and the XOF case. Since these are identical, they consitute the claimed bound. □

D Deferred Details of Constructions

D.1 Signal X3DH

Lemma 31 ($\Sigma^{G,p,s,t}_{\mathsf{X3DH}}$ is unpredictable)**.** *Let $\Sigma^{G,p,s,t}_{\mathsf{X3DH}}$ be the DH source underlying X3DH for group G with sample size $u = p^2t + p^2s + p^2st + ps^2$ for p parties with s sessions and t semi-static keys each, as described above. Let $\mathcal{A}_1$, $\mathcal{A}_2$ be adversaries against the multi-challenge unpredictability of $\Sigma^{G,p,s,t}_{\mathsf{X3DH}}$, with $\mathcal{A}_1$ making only a single query to* Predict*. Then there exist adversaries $\mathcal{B}_1$, $\mathcal{B}_2$ against the computational Diffie–Hellman and gap Diffie–Hellman problem in G, respectively, such that*

$$\mathbf{Adv}^{\mathrm{up}}_{\Sigma}(\mathcal{A}_1) \leq \mathbf{Adv}^{\mathrm{cdh}}_{G}(\mathcal{B}_1) + \tfrac{u^2}{|G|} \quad \textit{and} \quad \mathbf{Adv}^{\mathrm{up}}_{\Sigma}(\mathcal{A}_2) \leq \mathbf{Adv}^{\mathrm{gapdh}}_{G}(\mathcal{B}_2) + \tfrac{u^2}{|G|},$$

The adversaries $\mathcal{B}_1$, $\mathcal{B}_2$ have running time roughly the same as $\mathcal{A}_1$, $\mathcal{A}_2$, respectively, plus $\mathcal{O}(u)$, and $\mathbf{q}_{\mathrm{DDH}}(\mathcal{B}_2) = \mathbf{q}_{\textsc{Predict}}(\mathcal{A}_2)$.

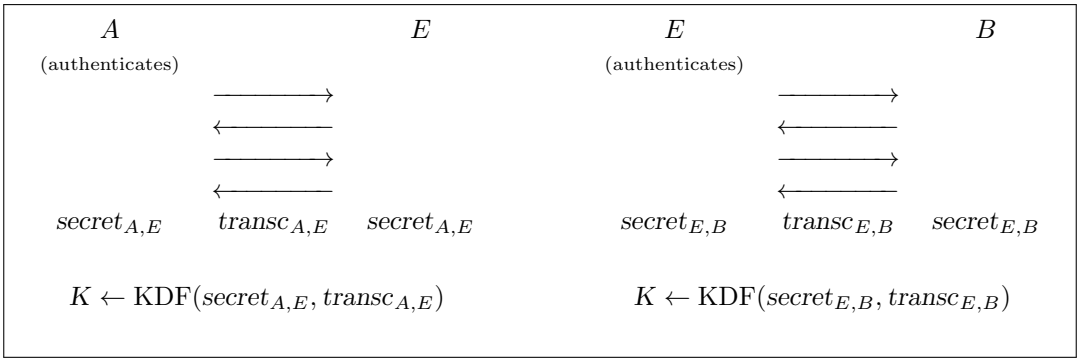

Figure 17: Unknown key share attack of adversary E on key exchange protocol: B assumes to share the key K with E but instead shares it with A.

Proof. We leverage the random self-reducibility of CDH and GapDH [51].

For the first case, given the CDH challenge g^x, g^y, algorithm $\mathcal{B}_1$ sets the DH shares making up the source key material of $\mathbf{z}[i]$ at position i as $(g^x)^{r_i}$, $(g^y)^{s_i}$, where it samples and stores $r_i, s_i \leftarrow\!\!\$\ \mathbb{Z}_q$, and outputs these shares as the auxiliary information to $\mathcal{A}_1$. (If any randomizer values collide (i.e., $r_i s_i = r_j s_j$ for $i \neq j$), we have $\mathcal{B}_1$ immediately abort; by the birthday bound, this happens with probability at most $\frac{u^2}{|G|}$.) Upon $\mathcal{A}_1$'s (single) $\textsc{Predict}(i, \sigma^*)$ query, algorithm $\mathcal{B}$ outputs $(\sigma^*)^{r_i^{-1} s_i^{-1}}$ as its CDH solution. If $\mathcal{A}_1$'s prediction was correct, then $(\sigma^*)^{r_i^{-1} s_i^{-1}} = (g^{x r_i y s_i})^{r_i^{-1} s_i^{-1}} = g^{xy}$, and $\mathcal{B}_1$ succeeds.

For the second case, algorithm $\mathcal{B}_2$ likewise re-randomizes the GapDH challenge g^x, g^y to form the key material vector $\mathbf{z}$ (and aborts upon collisions). Upon a $\textsc{Predict}(i, \sigma^*)$ query by $\mathcal{A}_2$, $\mathcal{B}_2$ calls its DDH on the triple $((g^x)^{r_i}, (g^y)^{s_i}, \sigma^*)$. If the answer is 1, $\mathcal{B}_2$ outputs $(\sigma^*)^{r_i^{-1} s_i^{-1}}$ as its solution, otherwise it returns $\mathsf{win} = 0$ to $\mathcal{A}_2$. If any of $\mathcal{A}_1$'s predictions are correct, then $\mathcal{B}_2$ succeeds. □

E Collision Resistance

We argue here that collision resistance is a desirable security feature of key derivation functions and, thus, of combiners. This property says that it is infeasible to find distinct inputs for the key derivation function resulting in the same key. Note that this property does not follow from the pseudorandomness of the key derivation function. The latter property only ensures random-looking outputs if the key material input has sufficient entropy. In the collision-resistant setting, however, there may be no entropy in the inputs. Collision resistance of HKDF.Expand is, for example, explicitly stated as a requirement in the TLS 1.3 standard [57]: "In some of the uses of HKDF in this document (e.g., for generating exporters and the resumption_master_secret), it is necessary that the application of HKDF-Expand be collision resistant".

E.1 Collision Resistance as a Useful Feature

We give an example of why collision resistance is a useful property for key derivation functions. We discuss this for the case of a plain key derivation function but the idea applies equally to combiners. Our example is based on implicitly authenticated key exchange protocols and *unknown key share* (UKS) attacks [15] (also known as *identity-misbinding* attacks [47]). In a UKS attack the adversary E aims to make a party B believe that it shares a derived key with E while it actually shares the key with another honest party A. A formalization of security against such attacks can be found in [30]: If a session of an honest party B accepts and the intended (possibly malicious) partner E has authenticated, then there cannot be any other session for a different honest party A which holds the same session key as B.

The attack is depicted in Figure 17. We assume that in each execution between two parties, only the left party authenticates (unilateral authentication). The adversary E engages in an execution with party A, resulting in an intermediate secret $secret_{A,E}$, like a Diffie-Hellman share, and a communication transcript $transc_{A,E}$. We assume that the key derivation function KDF now applies $secret_{A,E}$ with input $transc_{A,E}$ to derive the session key. The same happens (concurrently, intertwined, or afterwards) on the right-hand side in an execution between E and party B, yielding a derived key $\mathrm{KDF}(secret_{E,B}, transc_{E,B})$. The goal of the attacker E is to ensure that both executions end up with the same session key K.

Observe that E actively participates in both executions and may thus influence the inputs to the key derivation function, potentially being able to force colliding session keys despite different transcripts

Game $\mathbf{G}^{\text{cr}}_{\mathsf{F}}$:

FINALIZE$((\sigma_i, c_i)_{i=1..n}, L, \ell,\ (\sigma'_i, c'_i)_{i=1..n}, L', \ell')$

1 Return $\mathsf{F}((\sigma_i, c_i)_{i=1..n}, L, \ell) = \mathsf{F}((\sigma'_i, c'_i)_{i=1..n}, L', \ell')$
$\wedge\ ((\sigma_i, c_i)_{i=1..n}, L, \ell) \neq ((\sigma'_i, c'_i)_{i=1..n}, L', \ell')$

Figure 18: Game defining collision-resistance of an n-input KDF F.

$transc_{A,E}$ and $transc_{E,B}$ on both sides. If we now conservatively assume that the key derivation function is collision resistant then E's attack strategy is invalidated: It is reasonable to assume that the authenticating party transmits some identifying information like a certified public key with a unique serial number. Then the two transcripts must be distinct, and collision resistance of the key derivation function ensures that the session keys cannot match. Let us finally remark that key exchange protocols may still withstand UKS attacks even if the key derivation is not collision resistant. But then this must be ensured by the protocol itself and does not follow immediately from the security of the KDF.

E.2 Definition

We define collision resistance in a straightforward way, bounding the probability of an adversary $\mathcal{A}$ outputting two distinct lists of values

$$\big((\sigma_1, c_1), \ldots, (\sigma_n, c_n), L, \ell\big) \neq \big((\sigma'_1, c'_1), \ldots, (\sigma'_n, c'_n), L', \ell'\big)$$

which the n-KDF F maps to the same key output. Collision resistance applies to both NOF-n-KDF and XOF-n-KDF key derivation functions. Since there always exists an adversary outputting such collisions, we follow the human-ignorance approach [58] and bound the probability of a specific adversary finding such a collision. However, for a reasonable notion, one usually requires that the set KLout of admissible output lengths ℓ enforces a non-trivial lower bound.

Definition 32 (KDF Collision Resistance)**.** Let F be an n-KDF. We define the advantage of an adversary $\mathcal{A}$ against the collision resistance of F as

$$\mathbf{Adv}^{\text{cr}}_{\mathsf{F}}(\mathcal{A}) = \Pr\left[\,\mathbf{G}^{\text{cr}}_{\mathsf{F}}(\mathcal{A})\,\right].$$

E.3 Examples

We consider here the previously investigated examples in light of collision resistance. One may suspect that HKDF-based constructions inherit collision resistance from the underlying hash function, and this is indeed true if one restricts S inputs to fixed lengths (see also [33]). Blockcipher-based constructions like the ones based on CMAC are, in general, not collision-resistant, as we explain for the case of the key derivation function in Bluetooth Low Energy. As for our new constructions, the information-theoretical one is provably not collision resistant. For the blockcipher-based construction, the question remains open.

HKDF. Recall from the analysis of the proposed ETSI-CatKDF in Section 5.3 that HMAC and thus HKDF is not collision resistant. Any short or long salt value S in the extraction step $\mathsf{HKDF.Extract}(S, \sigma,) = \mathsf{HMAC}(S, \sigma)$ is either padded with 0-bytes or hashed first:

$$\mathsf{PoH}(S) := \begin{cases} S \| (\texttt{0x00})^{B-|S|} & \text{if } |S| \leq B \\ \mathsf{H}(S) \| (\texttt{0x00})^{B-hl} & \text{otherwise.} \end{cases}$$

Collisions like S and $S' = S\|\texttt{0x00}$ for short values S with $|S| < B$, and S and $S' = \mathsf{H}(S)$ for values S with $|S| > B$ can easily be constructed. This means that one needs to put further restrictions on inputs used as the key in HMAC, for example, only allowing fixed-length inputs or always hashing the data first.

For HKDF, the HMAC properties directly translate into HKDF being collision resistant if and only if salts are either *fixed-length* or *strictly longer* than block-length B bits. To see this, observe that $\mathsf{HKDF.Extract}(S, \sigma) = \mathsf{HMAC}(S, \sigma)$ maps inputs S and σ to a unique pseudorandom key PRK (unless a collision occurs in HMAC). This pseudorandom key is then used in $\mathsf{HKDF.Expand}(PRK, c, \ell)$, iterating $T(i) = \mathsf{HMAC}(PRK, T(i-1)||c||i)$ for $T(0) = \varepsilon$ and counter $i = 1, 2, \ldots$ represented as a fixed-length

octet. Once more, unless one finds a collision in the underlying hash function, all outputs $T(i)$ are unique if PRK and c are different. In summary, any collision $(\sigma, S, c) \neq (\sigma, S, c)$ for *"feasible" salts* S, S' yields a collision for the underlying hash function H in HMAC. This holds also for varying length parameters ℓ, since for identical $(\sigma, S, c) = (\sigma', S', c')$ but different length inputs $\ell \neq \ell'$ HKDF produces outputs of different length, which cannot collide.

MLS PSK COMBINER. The MLS n-KDF for PSK combination from Figure 5 is collision resistant if the hash function H underlying HMAC is. For distinct values psk_i, the extracted values $psk\_extracted_i$ must be distinct (since HMAC is applied in a collision-resistant way). Hence for distinct values $(psk_i, c_i, L) \neq (psk'_i, c'_i, L')$, we also have that the derived values differ, $psk\_input_i \neq psk\_input'_i$ (except for an HMAC collision and thus a H collision). Finally, distinct psk_input_i values being folded into the final psk_secret_n output also do not collide, except if an HMAC collision occurs.

ETSI-CatKDF. Let us now look into $\mathsf{ETSI\text{-}CatKDF}$, which computes the key as $\mathsf{HKDF}(secret, L, f_{\text{context}}, \ell)$. Note that $f_{\text{context}} = f(c, MA, MB)$ encodes the input tuple (c, MA, MB) injectively. In contrast, the source key material $secret = psk||k_1||\ldots||k_n$ simply concatenates the provided sub keys, with psk being potentially the empty string. The standard does not specify that the number of sub-keys is fixed, nor that the length of the keys is. We thus state this explicitly as a requirement to achieve collision resistance:

1. Labels L in $\mathsf{ETSI\text{-}CatKDF}$ shall be of fixed length.
2. The number n of sub keys k_i and their individual lengths must be fixed.

Under these stipulations, all inputs to HKDF are unique. It follows that protocol $\mathsf{ETSI\text{-}CatKDF}$ is collision-resistant due to the collision resistance of HKDF.

SIGNAL X3DH. Signal's key derivation function is generally not collision resistant since the context information does not enter the key derivation. If we assume that all context values c_i are always empty, then Signal inherits collision-resistance from HKDF. It merely inserts the concatenated key material values and constants, especially the all-zero salt, into the computations such that collision resistance of the underlying hash function ensures that distinct inputs are mapped to distinct outputs.

BLOCKCIPHER-BASED KEY DERIVATION IN BLUETOOTH LOW ENERGY. The Bluetooth Low Energy [16] key derivation function, denoted as $f5$, takes as input the 256-bit x-coordinate W of an elliptic-curve Diffie-Hellman share, computed jointly by the two parties during Secure Simple Pairing, and first computes

$$T = \mathsf{CMAC}(S, W)$$

for the constant 128-bit salt $S = \texttt{0x6C888391 AAF5A538 60370BDB 5A6083BE}$. Here, CMAC is the AES-CMAC algorithm according to RFC 4493 [62]. The algorithm computes a CBC-MAC for the all-zero initialization vector $IV = 0^{128}$, where the last message block is first xored with sub key K_1 if it is full block length resp. padded with 10^j and xored with sub key K_2 if it is not aligned to block length. The sub-keys are derived from the AES key K (the details are irrelevant to us here). More concretely, let $M = M_1||M_2||\ldots||M_n$ with $M_i \in \{0,1\}^{128}$ for $i = 1, 2, \ldots, n-1$ and $M_n \in \{0,1\}^{\leq 128}$. Set $M'_n = K_1 \oplus M_n$ if $|M_n| = 128$ and $M'_n = K_2 \oplus M_n||10^{|M_n|-1}$ if $|M_n| < 128$. Then

$$\begin{aligned}&\mathsf{CMAC}(K, M) = \mathsf{AES}(K, M'_n \oplus y_{n-1})\\ &\text{where } y_0 = 0^{128},\ y_i = \mathsf{AES}(K, M_i \oplus y_{i-1}) \text{ for } i = 1, 2, \ldots, n-1.\end{aligned}$$

In the second step, one computes two keys, one for message authentication during the Secure Simple Pairing, and one as the session key. Here we only consider the derivation of the session key (with prefix $\texttt{0x01}$ instead of $\texttt{0x00}$ for the MAC key):

$$k = \mathsf{CMAC}(T, \texttt{0x01}||keyID||N_1||N_2||A_1||A_2||\texttt{0x0100})$$

where $keyID = \texttt{0x62746C65}$ is the 32-bit ASCII representation of the string 'btle', N_1, N_2 are the 128-bit nonces exchanged in the Secure Simple Pairing, and A_1, A_2 are the 56-bit encodings of the Bluetooth addresses of the involved parties. Hence, the overall input length equals 424 bits, yielding four AES evaluations for inputs $M_1 = \texttt{0x01}||keyID||N_1[1..88]$, $M_2 = N_1[89..128]||N_2[1..88]$, $M_3 = N_2[89..128]||A_1||A_2[1..32]$ and $M_4 = A_2[33..56]||\texttt{0x0100}||10^{87}$.

The Bluetooth Low Energy key derivation is *not* collision-resistant due to the deployment of the block cipher. We can use the idea of the BlueMirror reflection attack on Bluetooth [25]—which considers mirroring the data sent by an honest user and thus does not touch collision-resistance immediately. An

adversary can first pick regular data W, N_1, N_2, A_1, A_2 and then computes T (with sub-keys T_1, T_2) and the session key k according to the scheme. In particular, the result of the first block in the computation equals $y_1 = \mathsf{AES}(T, \texttt{0x01}||\mathit{keyID}||N_1[1..88])$. Let y_2, y_3, y_4 denote the other intermediate blocks. Then the adversary picks a fresh $W^* \neq W$, and computes T^* with sub-keys T_1^*, T_2^*. It now computes $y_1^* = \mathsf{AES}(T^*, M_1)$ and then "backwards" $y_3^* = \mathsf{AES}^{-1}(T^*, y_4) \oplus T_2 * \oplus M_4$, $y_2^* = \mathsf{AES}^{-1}(T^*, y_3^*) \oplus M_3$, and $M_2^* = \mathsf{AES}^{-1}(T^*, y_2^*) \oplus y_1^*$. The adversary sets $N_1^*[89..128]||N_2^*[1..88] = M_2^*$, and otherwise $N_1^*, N_2^*, A_1^*, A_2^*$ coincide with the values N_1, N_2, A_1, A_2. In particular, the adversary only needs to adapt the input block M_2^*, but the other input blocks M_1, M_3, M_4 remain identical. If one now computes the values in forward direction for T^* then $y_1^* = \mathsf{AES}(T^*, M_1)$, $y_2^* = \mathsf{AES}(T^*, M_2^* \oplus y_1^*)$, $y_3^* = \mathsf{AES}(T^*, M_3 \oplus y_2^*)$ and $y_4^* = \mathsf{AES}(T^*, M_4 \oplus y_3^* \oplus T_2^*) = y_4$. Hence, the adversary is always able to find a collision.

Our blockcipher-based construction. Recall that we apply the blockcipher BC first via

$$k \leftarrow \mathsf{BC}(\sigma_1, \mathsf{BC}(\sigma_2, \mathsf{BC}(\sigma_1, 0^{bl}))$$

to compute a key, and then call the volPRF Expand for inputs k and (c_1, c_2), L and ℓ. Even if we assume collision resistance of the function Expand, then this implies that the only chance to find an input collision is to make two distinct key pairs (σ_1, σ_2), (σ_1', σ_2') collide to yield the same value k. We are not aware if this task is feasible for the chain evaluations of the blockcipher, and leave this as an open question.

Our Information-theoretically secure construction. The information-theoretically secure construction is certainly *not* collision resistant. Given arbitrary inputs $(\sigma_1, c_1), \ldots, (\sigma_n, c_n), L, \ell$ and $(\sigma_1', c_1'), \ldots, (\sigma_n', c_n'), L', \ell'$ with $\ell = \ell'$, one can set the information-theoretic part $\sigma_n^{\mathrm{its}'}$ in σ_n' to

$$\begin{aligned}\sigma_n^{\mathrm{its}'} \leftarrow \sigma_n^{\mathrm{its}} &\oplus \mathsf{F}((\sigma_1, c_1), \ldots, (\sigma_n^{\mathrm{kdf}}, c_n), L, \ell) \\ &\oplus \mathsf{F}((\sigma_1', c_1'), \ldots, (\sigma_n^{\mathrm{kdf}'}, c_n',), L', \ell')\end{aligned}$$

such that the outputs of both evaluations collide.