Cryptographic Extraction and Key Derivation: The HKDF Scheme

Hugo Krawczyk

IBM T.J. Watson Research Center, Hawthorne, New York
hugo@ee.technion.ac.il
http://eprint.iacr.org/2010/264

Abstract. In spite of the central role of *key derivation functions (KDF)* in applied cryptography, there has been little formal work addressing the design and analysis of general multi-purpose KDFs. In practice, most KDFs (including those widely standardized) follow ad-hoc approaches that treat cryptographic hash functions as perfectly random functions. In this paper we close some gaps between theory and practice by contributing to the study and engineering of KDFs in several ways. We provide detailed rationale for the design of KDFs based on the *extract-then-expand* approach; we present the first general and rigorous definition of KDFs and their security that we base on the notion of *computational extractors*; we specify a concrete *fully practical* KDF based on the HMAC construction; and we provide an analysis of this construction based on the extraction and pseudorandom properties of HMAC. The resultant KDF design can support a large variety of KDF applications under suitable assumptions on the underlying hash function; particular attention and effort is devoted to minimizing these assumptions as much as possible for each usage scenario.

Beyond the theoretical interest in modeling KDFs, this work is intended to address two important and timely needs of cryptographic applications: (i) providing a single hash-based KDF design that can be standardized for use in multiple and diverse applications, and (ii) providing a conservative, yet efficient, design that exercises much care in the way it utilizes a cryptographic hash function.
(The HMAC-based scheme presented here, named HKDF, is being standardized by the IETF.)

1 Introduction

A Key derivation function (KDF) is a basic and essential component of cryptographic systems: Its goal is to take a *source of initial keying material,* usually containing some good amount of randomness, but not distributed uniformly or for which an attacker has some partial knowledge, and derive from it one or more *cryptographically strong* secret keys. We associate the notion of "cryptographically strong" keys with that of *pseudorandom* keys, namely, indistinguishable by feasible computation from a random uniform string of the same length. In particular, knowledge of part of the bits, or keys, output by the KDF should

T. Rabin (Ed.): CRYPTO 2010, LNCS 6223, pp. 631–648, 2010.

not leak information on the other generated bits. Examples of initial keying material include the output of an imperfect physical random number generator, a bit sequence obtained by a statistical sampler (such as sampling system events or user keystrokes), system PRNGs that use renewable sources of randomness, and the less obvious case of a Diffie-Hellman value computed in a key exchange protocol.

The main difficulty in designing a KDF relates to the form of the initial keying material (which we refer to as source keying material). When this key material is given as a uniformly random or pseudorandom key K then one can use K to seed a pseudorandom function (PRF) or pseudorandom generator (PRG) to produce additional cryptographic keys. However, when the source keying material is not uniformly random or pseudorandom then the KDF needs to first "extract" from this "imperfect" source a first pseudorandom key from which further keys can be derived using a PRF. Thus, one identifies two logical modules in a KDF: a first module that takes the source keying material and extracts from it a fixed-length pseudorandom key K, and a second module that expands K into several additional pseudorandom cryptographic keys.[1]

The expansion module is standard in cryptography and can be implemented on the basis of any secure PRF. The *extraction* functionality, in turn, is well modeled by the notion of *randomness extractors* [31,30] as studied in complexity theory and related areas (informally, an extractor maps input probability distributions with sufficient entropy into output distributions that are statistically close to uniform). However, in many cases the well-established extractors (e.g., via universal hashing) fall short of providing the security and/or functionality required in practice in the KDF context. Here we study randomness extraction from the cryptographic perspective and specifically in the context of KDFs (building upon and extending prior work [26,14,6,5]). A main objective is to develop a basis for designing and analyzing secure key derivation functions following the above natural extract-then-expand approach. We are interested in the *engineering* of practical designs that can serve a variety of applications and usage scenarios and hence can be standardized for wide use. In particular, we need to be able to design extractors that will be well-suited for a large variety of sources of keying material (see detailed examples in [28]) and work in liberal as well as constrained environments. For this we resort to the use of cryptographic functions, especially cryptographic hash functions, as the basis for such multi-purpose extraction.

We identify *computational extractors*, namely randomness extractors where the output is only required to be pseudorandom rather than statistically close to uniform, as the main component for extraction in cryptographic applications, and build the notion of a KDF and its implementations on the basis of such extractors. Computational extractors are well-suited for the crypto setting where

[1] KDF is sometimes used only with the meaning of expanding a given *strong* key into several additional keys (e.g., [33]); this ignores the extract functionality which is central to a general *multi-purpose* KDF.

attackers are computationally bounded and source entropy may only exist in a computational sense. In particular, one can build such extractors in more efficient and practical ways through the use of cryptographic functions *under suitable assumptions*. Advantages of such cryptographic extractors range from purely practical considerations, such as better performance and the operational advantage of re-using functions (e.g., cryptographic hash functions) that are already available in cryptographic applications, to their more essential use for bypassing some of the inherent limitations of statistical extractors. We use the ability of cryptographic hash functions to be keyed as a way to include a *salt* value (i.e., a random but non-secret key) which is essential to obtain *generic* extractors and KDFs that can extract randomness from arbitrary sources with sufficiently high entropy.

We then study the requirements from computational extractors in the cryptographic setting, ranging from applications where the source key material has large entropy (and a good source of public randomness is also available) to the much more constrained scenarios encountered in practice where these resources (entropy and randomness) are much more limited. On this basis, we offer a KDF design that accommodates these different scenarios under suitable assumptions from the underlying cryptographic functions. In some cases, well-defined combinatorial assumptions from the hash functions will suffice while in others one has to resort to idealized modeling and "random oracle" abstractions. Our goal is to *minimize such assumptions as much as possible for each usage scenario,* but for this we need to first develop a good understanding of the properties one can expect from cryptographic hash functions as well as an understanding of the extraction functionality and the intrinsic limitations of unconditional statistical extractors in practical settings. We provide a detailed account of these issues throughout the paper.

Based on the notion of computational extractors (and on a better understanding of the complexities and subtleties of the use of KDFs in practice), we present a formal definition of the key derivation functionality suitable for capturing multiple uses of KDFs and a basis for judging the quality of general KDF designs such as those considered here. Somewhat surprisingly, in spite of being one of the most central and widely used cryptographic functionalities (in particular, specified in numerous standards), there appears to be little formal work on the specific subject of multi-purpose key derivation functions. Ours seems to be the first general definition of the KDF functionality in the literature. Our definitions include a formalization of what is meant by a "source of keying material" and they spell the security requirements from the KDF taking into account realistic adversarial capabilities such as the possession by the attacker of side information on the input to the KDF. In our formulation, KDFs accept four inputs: a sample from the source of keying material from which the KDF needs to extract cryptographic keys, a parameter defining the number of key bits to be output, an (optional) randomizing salt value as mentioned before, and a fourth "contextual information" field. The latter is an important parameter for the KDF intended to include key-related information that needs to be uniquely and

cryptographically bound to the produced key material (e.g., a protocol identifier, identities of principals, timestamps, etc.).

We then use the above theoretical background, including results from [14,12], to describe and analyze a concrete practical design of a multi-purpose KDF based on cryptographic hash functions. The scheme (denoted HKDF), that uses HMAC as the underlying mode of operation, supports multiple KDF scenarios and strives to minimize the required assumptions from the underlying hash function *for each such scenario*. For example, in some applications, assuming that the underlying hash function has simple combinatorial properties, e.g., universal hashing, will suffice while in the most constrained scenarios we will need to model the hash function as a random oracle. *The important point is that we will be able to use the same KDF scheme in all these cases as required for a standardized multi-purpose KDF.*[2]

We end by observing that most of today's standardized KDFs (e.g., [3,4,32,23]) do not differentiate between the extract and expand phases but rather combine the two in ad-hoc ways under a single cryptographic hash function (refer to [28] for a description and discussion of these KDF schemes and their shortcomings). This results in ad-hoc designs that are hard to justify with formal analysis and which tend to "abuse" the hash function, requiring it to behave in an "ideally random" way even when this is not strictly necessary in most KDF applications (these deficiencies are present even in the simple case where the source of keying material is fully random). In contrast, we formulate and analyze a fully practical KDF scheme based on current theoretical research as well as on sound engineering principles. The end result is a well-defined hash-based KDF scheme applicable to a wide variety of scenarios and which exercises much care in the way it utilizes cryptographic hash functions. Our view is that given the current (healthy) skepticism about the strength of our hash functions we must strive to design schemes that use the hash function as prudently as possible. Our work is intended to fulfill this principle in the context of key derivation functions (especially at a time that new standards based on hash functions are being developed, e.g., [33]).

Related Work. As already mentioned, in spite of their importance and wide use, there is little formal work on the specific subject of multi-purpose key derivation functions. The first work to analyze KDFs in the context of cryptographic hash functions and randomness extractors appears to be [14], which was followed-up in the context of random oracles by [12]. The former work laid the formal foundations for the HMAC-based KDF scheme presented here. This scheme, in turn, is based on the KDF originally designed by this author for the IKE protocols [19,24] and which put forth the extract-then-expand paradigm in the context of practical KDFs. A variant of the expansion stage of HKDF has also been adopted elsewhere, e.g. into TLS [13] (however, TLS does not use the extract approach; for example, keys from a DH exchange are used directly as PRF keys without any extraction operation). The extract-then-expand approach has subsequently been taken in [5] in the context of designing "system random number generators"; that work shares many elements with ours although the papers

[2] The proposed HKDF scheme is being standardized by the IETF as RFC 5869 [27].

differ significantly in emphasis and scope. Another related work is [6] which proposes the use of statistical extractors in the design of *physical* random-number generators and points out to the potential practicality of these extractors in this specific setting. Both [5,6] offer interesting perspectives on the use of randomness extractors in practice that complement our work; our HKDF design is well suited for use also in the settings studied by these works. A good discussion of extraction issues in the context of KDFs in the Diffie-Hellman setting can be found in [11] where a dedicated deterministic extractor for specific DH groups is presented. Another such extractor (very different in techniques and applicability) is presented in [16]. See more on related work in [28].

Full version. Due to space limitations we have omitted some important material that complements this presentation. Please refer to the full version [28] for expanded rationale, a treatment of the role of random oracles in the KDF setting, comparison with the most commonly used KDFs in practice, discussion of additional KDF applications, and more.

2 Statistical and Computational Extractors

This section is intended to introduce the basic notions behind the abstract randomness extraction functionality; in particular we define "computational extractors" that are central in our treatment.

The goal of the extract part of a KDF scheme is to transform the input source (seen as a probability distribution) into a close-to-uniform output. This corresponds to the functionality of *randomness extractors* which have been extensively studied in complexity theory and related areas [31]. Informally, a randomness extractor is a family of functions indexed by a public, i.e., non-secret, parameter (which we refer to as "salt") with the property that on any input distribution with sufficiently large entropy, if one chooses a salt value at random (and independently of the source distribution) the output of the extractor is statistically close to uniform (see below for a formal definition). Moreover, this statistical closeness holds even if conditioned on the salt value. Extractors with the latter property are called *strong* randomness extractors but since we only consider this type we often omit both the "strong" and "randomness" qualifiers. On the other hand, we often add the qualifier "statistical" to differentiate these extractors from computational ones (defined below) that are an essential part of our work.

Before presenting a formal definition of statistical extractors, we recall the notion of entropy considered in this context, called *min-entropy*, that captures a "worst case" notion of entropy different than the traditional average notion of Shannon's entropy (it is not hard to see that Shannon's notion is insufficient in the context of randomness extraction).

Background definitions and notation. Refer to Appendix A for some background definitions and notation used throughout the paper (e.g., the notion of δ-close).

Definition 1. *A probability distribution $\mathcal{X}$ has* min-entropy (at least) *m if for all a in the support of $\mathcal{X}$ and for random variable X drawn according to $\mathcal{X}$, $Prob(X = a) \leq 2^{-m}$.*

Definition 2. *Let $\mathcal{X}$ be a probability distribution over $\{0,1\}^n$. A function* ext : $\{0,1\}^t \times \{0,1\}^n \rightarrow \{0,1\}^{m'}$ *is called a δ-*statistical extractor *with respect to $\mathcal{X}$ if the distribution of pairs (r, y), where r is chosen with uniform probability over $\{0,1\}^t$ and $y = \mathsf{ext}_r(x)$ for x chosen according to distribution $\mathcal{X}$, is δ-close to the distribution of pairs (r, z) where z is chosen with uniform probability from $\{0,1\}^{m'}$. If* ext *is a δ-statistical extractor with respect to all distributions over $\{0,1\}^n$ with min-entropy m, then we say that* ext *is a (m, δ)-*statistical extractor.

This notion was first defined in [31]; see [30,38] for surveys.

Randomization of the extractor function via the parameter r (the salt) is mandatory if the same extractor function is to be able to extract randomness from *any* high min-entropy distribution. Indeed, for any deterministic function one can construct a high min-entropy source on which the function will produce very non-uniform outputs. On the other hand, one may consider randomness extractors that are suited for a *specific* source (or family of sources). In the latter case, one can consider *deterministic extractors*. Examples of such source-specific extractors in the cryptographic setting include the well-known hard-core schemes for RSA [2,15] and for discrete-log based functions [21,34], and the recent elegant extraction functions specific to some Diffie-Hellman groups in [11,16]. For most of our study we focus on generic extractors, i.e., those that can extract randomness from *any* source with sufficient min-entropy, and hence require some non-secret salt.

A natural (and practical) question is whether common KDF applications may have a randomness source from which to obtain salt. After all, the whole purpose of extractors is to generate randomness, so if one already has such a random salt why not use it directly as a PRF key? The answer is that this randomness needs *not* be secret while in KDF applications we want the output of the extractor to be secret. Obtaining public randomness is much easier than producing secret bits, especially since in most applications the extractor key (or salt) can be used repeatedly with many (independent) samples from the same source (hence it can be chosen in an out-of-band or setup stage and be repeatedly used later). For example, a random number generator (RNG) that requires an extractor to "purify" its possibly imperfect output can simply have a random, non-secret, extractor key built-in; the same extractor key is used to purify each output from the RNG [6]. In other cases, such as key-exchange protocols, extraction keys can be generated as part of the protocol (e.g., by using random nonces exchanged in the clear [19,24]). See [28] for further elaboration on the issue of randomization in extractors, in particular as a means to enforce independence between the source distribution and the extractor.

Efficient constructions of generic (hence randomized) statistical extractors exist such as those built on the basis of universal hash functions [10]. However, in spite of their simplicity, combinatorial and algebraic constructions present significant limitations for their practical use in generic KDF applications. For example,

statistical extractors require a significant difference (called the gap) between the min-entropy m of the source and the required number m' of extracted bits (in particular, no statistical extractor can achieve a statistical distance, on arbitrary sources, better than $2^{-\frac{m-m'}{2}}$ [35,38]). That is, one can use statistical extractors (with its provable properties) only when the min-entropy of the source is significantly higher than the length of output. These conditions are met by some applications, e.g., when sampling a physical random number generator or when gathering entropy from sources such as system events or human typing (where higher min-entropy can be achieved by repeated sampling). In other cases, very notably when extracting randomness from computational schemes such as the Diffie-Hellman key exchange, the available gap may not be sufficient (for example, when extracting 160 bits from a DH over a 192-bit group). In addition, depending on the implementation, statistical extractors may require from several hundred bits of randomness (or salt) to as many bits of salt as the number of input bits.

To obtain more practical instantiations of extractors we relax their requirements in several ways. Most significantly, we will not require that the output of the extractor be statistically close to uniform but just "computationally close", i.e., pseudorandom. The following notion is implicit in [17,14].

Definition 3. *A* (t, ε)-computational extractor *with respect to a probability distribution* $\mathcal{X}$ *is defined as in Definition 2 except that the requirement for statistical closeness between the distributions* (r, y) *and* (r, z) *is replaced with* (t, ε)*-computational indistinguishability*[3]*. An extractor that is* (t, ε)*-computational with respect to all distributions with min-entropy* m *is called* (m, t, ε)*-computational.*

This relaxed notion will allow for more practical instantiations of extractors, particularly well-suited for the key derivation setting. Computational extractors fit the cryptographic settings where attackers are assumed to be computationally bounded, and they allow for constructions based on cryptographic hash functions. In addition, computational extraction is natural in settings such as the Diffie-Hellman protocol where the input g^{xy} to the extractor is taken from a source that has zero statistical entropy (since an attacker that knows g^x, g^y has full information to compute g^{xy}), yet may contain a significant amount of "computational min-entropy" [20] as defined next.

Definition 4. *A probability distribution* $\mathcal{X}$ *has* (t, ε)-computational min-entropy m *if there exists a distribution* $\mathcal{Y}$ *with min-entropy* m *such that* $\mathcal{X}$ *and* $\mathcal{Y}$ *are* (t, ε)*-computationally indistinguishable.*

Note: In our application of extraction to the KDF setting, where an attacker often has some a-priori information about the source (e.g., it knows the public DH values g^x, g^y from which the source key material g^{xy} is derived), we use a notion of min-entropy (statistical or computational) that is *conditioned* on such a-priori information (see following section).

[3] See Appendix A for the definition of computational indistinguishability.

3 Formalizing Key Derivation Functions

We present a formal definition of (secure) key derivation functions and a formalization of what is meant by a "source of keying material". To the best of our knowledge, no such general definitions have been given in the literature.[4] We start with a definition of KDF in terms of its inputs and outputs (consistent with the KDF description in Section 4). Later, after introducing the notion of sources of keying material, we define what it means for a KDF to be secure.

Definition 5. *A* key derivation function (KDF) *accepts as input four arguments: a value σ sampled from a source of keying material (Def. 6), a length value ℓ, and two additional arguments, a salt value r defined over a set of possible salt values and a context variable c, both of which are optional, i.e., can be set to the null string or to a constant. The KDF output is a string of ℓ bits.*[5]

The security and quality of a KDF depends on the properties of the "source of keying material", defined next, from which the input σ is chosen (see [28] for more examples of such sources.)

Definition 6. *A* source of keying material *(or simply* source*) Σ is a two-valued probability distribution (σ, α) generated by an efficient probabilistic algorithm.* (We will refer to both the probability distribution as well as the generating algorithm by Σ.)

This definition does not specify the input to the Σ algorithm (but see below for a discussion related to potential adversary-chosen inputs to such an algorithm). It does specify the form of the output: a pair (σ, α) where σ (the "sample") represents the (secret) source key material to be input to a KDF, while α represents some *auxiliary knowledge* about σ (or its distribution) that is available to the attacker. For example, in a Diffie-Hellman application the value σ will consist of a value g^{xy} while α could represent a quintuple (p, q, g, g^x, g^y). In a different application, say a random number generator that works by hashing samples of system events in a computer system, the value α may include some of the sampled events used to generate σ. The importance of α in our formal treatment is that we will require a KDF to be secure on inputs σ *even when the knowledge value α is given to the attacker.* The restriction to sources that can be generated efficiently represents our interest in sources that can arise (and be used/sampled) in practice.

Next, we define the security of a KDF with respect to a *specific* source Σ. See Definition 9 for the generic case.

Definition 7. *A key derivation function* KDF *is said to be* (t, q, ε)-secure with respect to a source of key material *Σ if no attacker $\mathcal{A}$ running in time t and making*

[4] Yao and Yin [40] provide a formal definition of KDFs specific to the password setting which is different from and inapplicable to the general setting treated here (see [28]).

[5] The values σ, ℓ, r, c correspond to the values $SKM, L, XTS, CTXinfo$ in the description of Section 4.

at most q queries can win the following distinguishing game with probability larger than $1/2 + \varepsilon$:

1. *The algorithm Σ is invoked to produce a pair σ, α.*
2. *A salt value r is chosen at random from the set of possible salt values defined by* KDF *(r may be set to a constant or a null value if so defined by* KDF*).*
3. *The attacker $\mathcal{A}$ is provided with α and r.*
4. *For $i = 1, \ldots, q' \leq q$: $\mathcal{A}$ chooses arbitrary values c_i, ℓ_i and receives the value $\mathsf{KDF}(\sigma, r, c_i, \ell_i)$ (queries by $\mathcal{A}$ are adaptive, i.e., each query may depend on the responses to previous ones).*
5. *$\mathcal{A}$ chooses values c and ℓ such that $c \notin \{c_1, \ldots, c_{q'}\}$.*
6. *A bit $b \in_R \{0,1\}$ is chosen at random. If $b = 0$, $\mathcal{A}$ is provided with the output of $\mathsf{KDF}(\sigma, r, c, \ell)$, else $\mathcal{A}$ is given a random string of ℓ bits.*
7. *Step 4 is repeated for up to $q - q'$ queries (subject to the restriction $c_i \neq c$).*
8. *$\mathcal{A}$ outputs a bit $b' \in \{0,1\}$. It wins if $b' = b$.*

It is imperative for the applicability of this definition that the attacker is given access to both α and r. This models the requirement that the KDF needs to remain secure even when the side-information α and salt r are known to the attacker (in particular, note that the choice of the c's and ℓ's by the attacker may depend on α and r). Allowing for multiple values of c_i to be chosen by the attacker under the same input σ to KDF ensures that even if an attacker can force the use of the same input σ to the KDF in two different contexts (represented by c), the outputs from the KDF in these cases are computationally independent (i.e., leak no useful information on each other).

The following definition extends the min-entropy definitions from Section 2 to the setting of keying material sources (for a detailed treatment of *conditional* (computational) entropy as used in the next definition see [36,37,22]).

Definition 8. *We say that Σ is a* statistical m-entropy source *if for all s and a in the support of the distribution Σ, the conditional probability $Prob\,(\sigma = s \mid \alpha = a)$ induced by Σ is at most 2^{-m}.*
We say that Σ is a computational m-entropy source *(or simply an m-*entropy source*) if there is a statistical m-entropy source Σ' that is computationally indistinguishable from Σ.*

We note that in the above definition we can relax the "for all a" to "all but a negligible fraction of a". That is, we can define $\alpha = a$ to be "bad" (for a given value m) if there is s such that $Prob\,(\sigma = s \mid \alpha = a) > 2^{-m}$ and require that the joint probability induced by Σ on bad a's be negligible.

Definition 9. *A KDF function is called (t, q, ε) m-*entropy secure *if it is (t, q, ε)-secure with respect to all (computational) m-entropy sources.*

We note that for the most part of this paper the (implicit) notion of security of a KDF corresponds to this last definition, namely, we think of KDFs mainly as a *generic* function that can deal with different sources as long as the source has enough computational min-entropy. We stress that this notion of security

can only be achieved for randomized KDFs where the salt value r is chosen at random from a large enough set. Yet, this work also touches on deterministic KDFs (see more in [28]) that may be good for specific applications and sources and whose security is formalized in Definition 7.

On adversarially-chosen inputs to Σ. Please refer to [28] for a discussion on extending the above definitions to incorporate possiblly adversarial inputs to the generation of the source Σ.

4 Extract-Then-Expand KDF and an HMAC-Based Instantiation

In this section we first describe an abstract KDF that implements the *extract-then-expand* approach discussed throughout this paper, and then specify an instantiation solely based on HMAC [7].

An extract-then-expand key derivation function KDF comprises two modules: a randomness extractor XTR and a variable-length output pseudorandom function PRF^* (the latter is usually built on the basis of a regular PRF with output extension via counter mode, feedback mode, etc.). The extractor XTR is assumed to produce "close-to-random", in the statistical or computational sense, outputs on inputs sampled from the source key material distribution (this should be the case also when the *SKM* value includes auxiliary knowledge α, per Definition 6, that is provided to the distinguisher). XTR may be deterministic or keyed via an optional "salt value" (i.e., a non-secret random value) that we denote by *XTS* (for *extractor salt*). The key to PRF^* is denoted by *PRK* (pseudorandom key) and in our scheme it is the output from XTR; thus, we are assuming that XTR produces outputs of the same length as the key to PRF^*. The function PRF^* also gets a length parameter indicating the number of bits to be output by the function. In all, KDF receives four inputs: the source key material *SKM*, the extractor salt *XTS* (which may be null or constant), the number L of key bits to be produced by KDF, and a "context information" string *CTXinfo* (which may be null). The latter string should include key-related information that needs to be uniquely (and cryptographically) bound to the produced key material. It may include, for example, information about the application or protocol calling the KDF, session-specific information (session identifiers, nonces, time, etc.), algorithm identifiers, parties identities, etc. The computation of the extract-then-expand KDF proceeds in two steps; the L-bit output is denoted *KM* (for "key material"):

$$\begin{aligned} 1.\quad PRK &= \mathsf{XTR}(XTS,\ SKM) \\ 2.\quad KM &= \mathsf{PRF}^*(PRK,\ CTXinfo,\ L) \end{aligned}$$

The following theorem establishes the security of a KDF built using this extract-then-expand approach. The proof, presented in [28], follows from the definition of computational extractors (Definition 3), the security definition of variable-length-output pseudorandom functions (Definition 14 in Appendix A), and the definition of KDF security (Definition 7).

Theorem 1. *Let* XTR *be a* (t_X, ε_X)*-computational extractor w.r.t. a source* Σ *and* PRF^* *a* $(t_P, q_P, \varepsilon_P)$*-secure variable-length-output pseudorandom function family, then the above extract-then-expand* KDF *scheme is* $(\min\{t_X, t_P\}, q_P, \varepsilon_X + \varepsilon_P)$*-secure w.r.t. source* Σ*.*

An HMAC-based instantiation. For the sake of implementation in real applications we propose to instantiate the above general scheme with HMAC serving as the PRF underlying PRF^* as well as the XTR function. We denote the resultant scheme by HKDF.

We use the following notational conventions: (i) the variable k denotes the output (and key) length of the hash function used with HMAC; (ii) we represent HMAC as a two-argument function where the first argument always represents the HMAC key; (iii) the symbol $\|$ denotes string concatenation. Thus, when writing $\mathsf{HMAC}(a,\ b \parallel c)$ we mean the HMAC function (using a given hash function) keyed with the value a and applied to the concatenation of the strings b and c.

The scheme HKDF is specified as:

$$\mathsf{HKDF}(\mathit{XTS},\ \mathit{SKM},\ \mathit{CTXinfo},\ L) \ = \ K(1) \parallel K(2) \parallel \ldots \parallel K(t)$$

where the values $K(i)$ are defined as follows:

$$\begin{aligned} \mathit{PRK} \ &= \mathsf{HMAC}(\mathit{XTS},\ \mathit{SKM}) \\ K(1) &= \mathsf{HMAC}(\mathit{PRK}, \mathit{CTXinfo} \parallel 0), \\ K(i+1) &= \mathsf{HMAC}(\mathit{PRK},\ K(i) \parallel \mathit{CTXinfo} \parallel i), \quad 1 \le i < t, \end{aligned}$$

where $t = \lceil L/k \rceil$ and the value $K(t)$ is truncated to its first $d = L \bmod k$ bits; the counter i is non-wrapping and of a given fixed size, e.g., a single byte. Note that the length of the HMAC output is the same as its key length and therefore the scheme is well defined.

When the extractor salt *XTS* is not provided (i.e., the extraction is deterministic) we set $\mathit{XTS} = 0$.

Example: Let HMAC-SHA256 be used to implement KDF. The salt *XTS* will either be a provided 256-bit random (but not necessarily secret) value or, if not provided, *XTS* will be set to 0. If the required key material consists of one AES key (128 bits) and one HMAC-SHA1 key (160 bits), then we have $L = 288$, $k = 256$, $t = 2$, $d = 32$ (i.e., we will apply HMAC-SHA256 with key *PRK* twice to produce 512 bits but only 288 are output by truncating the second output from HMAC to its first 32 bits). Note that the values $K(i)$ do not necessarily correspond to individual keys but they are concatenated to produce as many key bits as required.

Practical Notes. Please refer to [28] for several notes on the use of the HKDF in practice, including some variants such as replacing HMAC with a block-cipher based construct or with other "multi-property preserving" hash schemes, and using hybrid schemes where the extract and expand modules are implemented with separate components. [28] also contains a discussion on the use of feedback mode in the expansion stage of HKDF.

5 The Security of HKDF

Theorem 1 from Section 4 allows us to argue the security of HKDF on the basis of the properties of the HMAC scheme both as extractor and PRF. In this section we review results concerning these properties of HMAC and use them to prove the security of HKDF. These results demonstrate that the structure of HMAC works well in achieving the basic functionalities that underline HKDF including PRF, extraction, and random-oracle domain extension. In particular, they exploit the versatility of HMAC that supports working with a secret key, a random non-secret key (salt), or deterministically (i.e., with a fixed-value key). We also note that the security analysis of HKDF uses in an essential way the structure of HMAC and *would not hold* if one simply replaces HMAC with a plain (Merkle-Damgard) hash function.

Notation. We use H to denote a Merkle-Damgard hash function and h the underlying compression function. We also consider these as keyed families, where h_κ and H_κ denote, respectively, the compression function and the Merkle-Damgard hash with their respective IVs set to κ; the key and output lengths of these functions is denoted by k. We will abuse notation and talk about "the family h_κ" instead of the more correct $\{h_\kappa\}_{\kappa\in\{0,1\}^k}$; same for H_κ. When we say that "the family h_κ is random", we mean that each of the functions h_κ is chosen at random (with the corresponding input/output lengths). When we talk about HMAC (or NMAC), we assume underlying functions h and H (or their keyed versions).

The properties of HMAC as a pseudorandom function family are well established [8,9] and are based on the assumed pseudorandomness of the underlying compression function family h_κ.[6] It is not hard to see that the use of HMAC in "feedback mode" in HKDF (for realizing PRF^*) results in a secure variable-length-output pseudorandom function family. Indeed, the latter is a generic transformation from fixed-length output PRF into a variable-length output PRF^* (see more on this transformation and the rationale for the use of feedback mode in [28]).

The suitability of HMAC as a computational extractor is more complex and is treated in detail below. These results show the extraction properties of HMAC for a wide variety of scenarios under suitable assumptions on the underlying hash function, ranging from purely combinatorial properties, such as universality, to the idealized modeling of compression functions as random oracles.

We first state the following general theorem.

Theorem 2. *(informal) Let H be a Merkle-Damgard hash function built on a family of pseudorandom compression functions $\{h_\kappa\}_\kappa$. Let $\mathcal{S}$ be a collection of probability distributions acting as sources of keying material. Assume that the instantiation of HMAC with the family $\{h_\kappa\}_\kappa$ is a secure computational extractor w.r.t. sources in $\mathcal{S}$, then* HKDF *is a secure KDF w.r.t. sources in $\mathcal{S}$.*

[6] Although the security of HMAC as PRF degrades quadratically with the number of queries, such attack would require the computation of the PRF (by the owner of the key) over inputs totalizing $2^{k/2}$ blocks. This is not a concern in typical KDF applications where the number of applications of the PRF is relatively small.

The theorem follows from Theorem 1 applied to the collection of sources $\mathcal{S}$ and the fact, discussed above, that HMAC is a secure PRF when instantiated with a pseudorandom family of compression functions h_κ. *Each of the lemmas presented below provides a condition on HMAC extraction that can be plugged into this theorem to obtain a proof of security for* HKDF *for the appropriate sources of key material in a* well-defined and quantifiable *way.*

The results below involve the notion of "almost universal (AU)" hash functions [10,39]: A family h_κ is δ-AU if for any inputs $x \neq y$ and for random κ, $Prob(h_\kappa(x) = h_\kappa(y)) \leq \delta$. This is a natural (combinatorial) property of hash functions and also one that any (even mildly) collision resistant hash family must have and then a suitable assumption for cryptographic hash functions. Specifically, if the hash family h_κ is δ-collision-resistant against linear-size circuits (i.e., such an attacker finds collisions in h_κ with probability at most δ) then h_κ is δ-AU [14]. For results that apply to the most constrained scenarios (as those discussed in the Random Oracles section of [28]) we need to resort to stronger, idealized assumptions, in which we model functions as *random oracles (RO)*, namely, random functions which the attacker can only query on a limited number, q, of queries.

NMAC as extractor. The following lemmas are adapted from [14] and apply directly to the NMAC scheme underlying HMAC (recall that $\mathsf{NMAC}_{\kappa_1,\kappa_2}(x) = H_{\kappa_2}(H_{\kappa_1}(x))$, where H_{κ_2} is called the "outer function" and H_{κ_1} the "inner function"). The results extend to HMAC as explained below. They show that HMAC has a structure that supports its use as a generic extractor and, in particular, it offers a much better design for extraction than the plain hash function H used in many of the existing KDFs.

Lemma 1. *If the outer function is modeled as a RO and the inner function is δ-AU then NMAC applied to an m-entropy source produces an output that is $\sqrt{q(2^{-m}+\delta)}$-close to uniform where q is a bound on the number of RO queries.*

The above modeling of the outer function as a random oracle applies to the case where the outer function is a single (fixed) random function (in which case the source distribution needs to be independent of this function) or when it is represented as a keyed family of random functions (in which case only the key, or salt, needs to be chosen independently of the source distribution).

One natural question is whether one can ensure good extraction properties for NMAC based on the extraction properties of the underlying compression functions and *without idealized assumptions.* The following result from [14] provides an affirmative answer for *m-blockwise sources*, namely, where each k-bit input block has min-entropy m when conditioned on other blocks. Denote by $\hat{h}_\kappa$ a family identical to the compression function family h_κ but where the roles of key and input are swapped relative to the definition of h_κ.

Lemma 2. *If h_κ is a (m,δ)-statistical extractor and $\hat{h}_\kappa$ is a (t,q,ε)-pseudorandom family for $q = 1$, then NMAC is a $(t, n\delta+\varepsilon)$-computational extractor for m-blockwise sources with n input blocks.*

An example of a practical application where this non-idealized result can be used is the IKE protocol [19,24] where all the defined "mod p" DH groups have the required block-wise (computational) min-entropy. In particular, the output from HKDF is guaranteed to be pseudorandom in this case without having to model h as a random oracle.

Truncated NMAC. Stronger results can be achieved if one truncates the output of NMAC by c bits to obtain $k' = k - c$ bits of output (e.g., one computes NMAC with SHA-512 but only outputs 256 bits). In this case one can show that NMAC is a good *statistical* extractor (not just computational) under the following sets of assumptions:

Lemma 3. *If h_κ is a family of random compression functions (with k bits of output) then NMAC truncated by c bits is a $(k, \sqrt{(n+2)2^{-c}})$-statistical extractor where n is the number of input blocks.*

Lemma 4. *If the inner function is δ_1-AU and the outer is $(2^{-k'} + \delta_2)$-AU then truncated NMAC (with k' bits of output) is a $(m, \sqrt{2^{k'}(2^{-m} + \delta_1 + \delta_2)})$-statistical extractor.*

The latter lemma is particularly interesting as its guarantee is "unconditional": it does not depend on hardness assumptions or idealized assumptions, and it ensures statistical extraction. Moreover, it fits perfectly the HKDF setting if one implements the extract phase with, say, SHA2-512 (with output truncated to 256 bits) and the PRF part with SHA2-256 (as discussed in the practical notes section of [28]).

In particular, we have the following significant case:

Corollary 1. *If the family of compression functions h_κ is strongly universal (or pairwise independent) and the family H_κ is generically collision resistant against linear-size circuits, then NMAC truncated by c bits is a $(k, (n+2)2^{-c/2})$-statistical extractor on n-block inputs.*

Indeed, the assumption that the family h_κ is strongly universal means $\delta_2 = 0$; and it is not hard to see that if there is no trivial (linear size) algorithm to find collisions in the family H_κ better than guessing then $\delta_1 \leq n2^{-k}$. Putting these values and $m = k$ into Lemma 4 the corollary follows.

Applying the corollary to the above example of SHA2-512 truncated to 256 bits we get a statistical distance of $(n + 2)2^{-128}$. And there is plenty room to get good security even if the δ values deviate from the above numbers; e.g., for $\delta_1 = \delta_2 = 2^{100}2^{-k}$ we would get a statistical closeness of $(n + 2)2^{-78}$. Finally, we note that a min-entropy of $m = k$ is a condition satisfied by many common distributions such as statistical samplers and Diffie-Hellman groups modulo safe primes.

From NMAC to HMAC. To use the above results with HMAC one needs to assume the computational independence of the values $h(\kappa \oplus \text{opad})$ and $h(\kappa \oplus \text{ipad})$ for random κ (i.e., each of these values is indistinguishable from uniform even if

the other is given). In the cases where h is modeled as a random function this requires no additional assumption.

HMAC as a random oracle. In [28] (Random Oracles section) we point out to various scenarios that require the modeling of the extraction functionality through random oracles. This may be due to the stringent requirements of an application (e.g., when all of the min-entropy of the source is to be extracted), when extraction can be solely based on cryptographic hardness without assuming additional min-entropy (the "hard core" case), or when the application itself assumes the KDF to be a random oracle (as in certain key exchange protocols). In all these cases we are interested to model the extract part of HKDF as a random oracle. Fortunately, as shown in [12] (using the framework of indifferentiability from [29]), the HMAC structure preserves randomness in the sense that if the underlying compression function (computed on fixed length inputs) is modeled as a random oracle so is HMAC on variable length inputs ([12] claims the result for a variant of HMAC but it applies to HMAC itself).[7] This result together with the random-oracle-extraction lemma from [28] implies:

Lemma 5. *If the compression function h is modeled as a RO, then the application of HMAC to an m-entropy source produces an output that is $q2^{-m}$-close to uniform where q is a bound on the number of RO queries.*

As explained in [28], the above holds for source distributions that are (sufficiently) *independent* from the function h. To obtain a generic extractor one needs to randomize the scheme by keying HMAC with a salt value.

Finally, we point out that using random-oracle-hardcore lemma from [28], one obtains that if h_κ is a RO family then HMAC over the family H_κ is a generic hard-core family. This is needed when extraction is to be based on cryptographic hardness (or unpredictability) only without assuming additional min-entropy (e.g., in a Diffie-Hellman exchange where only CDH is assumed – see more in [28]).

References

1. Adams, C., Kramer, G., Mister, S., Zuccherato, R.: On The Security of Key Derivation Functions. In: Zhang, K., Zheng, Y. (eds.) ISC 2004. LNCS, vol. 3225, pp. 134–145. Springer, Heidelberg (2004)
2. Alexi, W., Chor, B., Goldreich, O., Schnorr, C.-P.: RSA and Rabin Functions: Certain Parts are as Hard as the Whole. SIAM J. Comput. 17(2), 194–209 (1988)
3. ANSI X9.42-2001: Public Key Cryptography For The Financial Services Industry: Agreement of Symmetric Keys Using Discrete Logarithm Cryptography

[7] This "RO preserving" property does not hold for the plain Merkle-Damgard hash which is susceptible to extension attacks. Moreover, even if one considers fixed-length inputs (to avoid extension attacks), the Merkle-Damgard family H_κ built on random compression functions is not a good statistical extractor (e.g., [14] show that the output of such family on any distribution for which the last block of input is fixed is statistically far from uniform).

4. ANSI X9.63-2002: Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport
5. Barak, B., Halevi, S.: A model and architecture for pseudo-random generation with applications to /dev/random. In: ACM Conference on Computer and Communications Security (2005)
6. Barak, B., Shaltiel, R., Tromer, E.: True random number generators secure in a changing environment. In: Walter, C.D., Koç, Ç.K., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 166–180. Springer, Heidelberg (2003)
7. Bellare, M., Canetti, R., Krawczyk, H.: Keying hash functions for message authentication. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 1–15. Springer, Heidelberg (1996)
8. Bellare, M., Canetti, R., Krawczyk, H.: Pseudorandom Functions Revisited: The Cascade Construction and Its Concrete Security. In: Proc. 37th FOCS, pp. 514–523. IEEE, Los Alamitos (1996)
9. Bellare, M.: New Proofs for NMAC and HMAC: Security Without Collision-Resistance. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 602–619. Springer, Heidelberg (2006)
10. Carter, L., Wegman, M.N.: Universal Classes of Hash Functions. JCSS 18(2) (1979)
11. Chevassut, O., Fouque, P.-A., Gaudry, P., Pointcheval, D.: The twist-aUgmented technique for key exchange. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T.G. (eds.) PKC 2006. LNCS, vol. 3958, pp. 410–426. Springer, Heidelberg (2006)
12. Coron, J.-S., Dodis, Y., Malinaud, C., Puniya, P.: Merkle-Damgard Revisited: How to Construct a Hash Function. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 430–448. Springer, Heidelberg (2005)
13. Dierks, T., Allen, C. (eds.): The TLS Protocol – Version 1. Request for Comments 2246 (1999)
14. Dodis, Y., Gennaro, R., Håstad, J., Krawczyk, H., Rabin, T.: Randomness Extraction and Key Derivation Using the CBC, Cascade and HMAC Modes. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 494–510. Springer, Heidelberg (2004)
15. Fischlin, R., Schnorr, C.-P.: Stronger Security Proofs for RSA and Rabin Bits. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 267–279. Springer, Heidelberg (1997)
16. Fouque, P.-A., Pointcheval, D., Stern, J., Zimmer, S.: Hardness of Distinguishing the MSB or LSB of Secret Keys in Diffie-Hellman Schemes. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 240–251. Springer, Heidelberg (2006)
17. Gennaro, R., Krawczyk, H., Rabin, T.: Secure Hashed Diffie-Hellman over Non-DDH Groups. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 361–381. Springer, Heidelberg (2004)
18. Goldwasser, S., Micali, S.: Probabilistic Encryption. JCSS 28(2), 270–299 (1984)
19. Harkins, D., Carrel, D. (eds.): The Internet Key Exchange (IKE). RFC 2409 (November 1998)
20. Hastad, J., Impagliazzo, R., Levin, L., Luby, M.: Construction of a Pseudorandom Generator from any One-way Function. SIAM. J. Computing 28(4), 1364–1396 (1999)
21. Hastad, J., Schrift, A., Shamir, A.: The Discrete Logarithm Modulo a Composite Hides O(n) Bits. J. Comput. Syst. Sci. 47(3), 376–404 (1993)
22. Hsiao, C.-Y., Lu, C.-J., Reyzin, L.: Conditional Computational Entropy, or Toward Separating Pseudoentropy from Compressibility. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 169–186. Springer, Heidelberg (2007)

23. IEEE P1363A: Standard Specifications for Public Key Cryptography: Additional Techniques, Institute of Electrical and Electronics Engineers
24. Kaufman, C. (ed.): Internet Key Exchange (IKEv2) Protocol. RFC 4306 (December 2005)
25. Krawczyk, H., Bellare, M., Canetti, R.: HMAC: Keyed-Hashing for Message Authentication. RFC 2104 (February 1997)
26. Krawczyk, H.: SIGMA: The 'SiGn-and-MAc' Approach to Authenticated Diffie-Hellman and Its Use in the IKE Protocols. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 400–425. Springer, Heidelberg (2003)
27. Krawczyk, H., Eronen, P.: HMAC-based Extract-and-Expand Key Derivation Function (HKDF), RFC 5869 (to appear)
28. Krawczyk, H.: Cryptographic Extraction and Key Derivation: The HKDF Scheme (full version of this paper), `http://eprint.iacr.org/2010/264`
29. Maurer, U.M., Renner, R., Holenstein, C.: Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 21–39. Springer, Heidelberg (2004)
30. Nisan, N., Ta-Shma, A.: Extracting Randomness: A Survey and New Constructions. JCSS 58, 148–173 (1999)
31. Nisan, N., Zuckerman, D.: Randomness is linear in space. J. Comput. Syst. Sci. 52(1), 43–52 (1996)
32. NIST Special Publication (SP) 800-56A, Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography (March 2006)
33. NIST Special Publication (SP) 800-108, Recommendation for Key Derivation Using Pseudorandom Functions (October 2009)
34. Patel, S., Sundaram, G.: An Efficient Discrete Log Pseudo Random Generator. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 304–317. Springer, Heidelberg (1998)
35. Radhakrishnan, J., Ta-Shma, A.: Tight bounds for depth-two superconcentrators. SIAM J. Discrete Math. 13(1), 2–24 (2000)
36. Renner, R., Wolf, S.: Smooth Renyi entropy and applications. In: Proceedings of IEEE International Symposium on Information Theory (2004)
37. Renner, R., Wolf, S.: Simple and tight bounds for information reconciliation and privacy amplification. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 199–216. Springer, Heidelberg (2005)
38. Shaltiel, R.: Recent developments in Extractors. Bulletin of the European Association for Theoretical Computer Science 77, 67–95 (2002), `http://www.wisdom.weizmann.ac.il/~ronens/papers/survey.ps`
39. Douglas, R.: Stinson: Universal Hashing and Authentication Codes. Des. Codes Cryptography 4(4), 369–380 (1994)
40. Yao, F.F., Yin, Y.L.: Design and Analysis of Password-Based Key Derivation Functions. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 245–261. Springer, Heidelberg (2005)

A Background Definitions

In this section we recall basic formal definitions for some of the notions used throughout this work.

Notation. In the sequel $\mathcal{X}$ and $\mathcal{Y}$ denote two (arbitrary) probability distributions over a common support set A; X and Y denote random variables drawn from $\mathcal{X}$ and $\mathcal{Y}$, respectively.

Definition 10. *We say that probability distributions $\mathcal{X}$ and $\mathcal{Y}$ have* statistical distance δ *(or are* δ-close*) if* $\sum_{a \in A} |Prob(X = a) - Prob(Y = a)| \leq \delta$.

Definition 11. *An algorithm D is an* ε-distinguisher *between distributions $\mathcal{X}$ and $\mathcal{Y}$ if* $|Prob(D(X) = 1) - Prob(D(Y) = 1| < \varepsilon$.

We note that two distributions $\mathcal{X}$ and $\mathcal{Y}$ are δ-close iff there is no ε-distinguisher between $\mathcal{X}$ and $\mathcal{Y}$ for $\varepsilon > \delta$.

By restricting the computational power of the distinguisher in the above definition one obtains the following well-known definition of "computational indistinguishability" [18] (we formulate definitions using the "concrete security" (t, ε) approach as a non-asymptotic alternative to the classical polynomial-time treatment; we also take the liberty of omitting the (t, ε) notation when appropriate).

Definition 12. *Two probability distributions $\mathcal{X}$, $\mathcal{Y}$ are* (t, ε)-computationally indistinguishable *if there is no ε-distinguisher between $\mathcal{X}$ and $\mathcal{Y}$ that runs in time t.*

Definition 13. *A probability distribution $\mathcal{X}$ over the set $\{0,1\}^n$ is called* (t, ε)-pseudorandom *if it is (t, ε)-computationally indistinguishable from the uniform distribution over $\{0,1\}^n$.*

Next we recall the definition of security for a variable-length output pseudorandom function family. Such a family consists of a collection of keyed functions which on input a key κ, an input c and a length parameter ℓ, outputs ℓ bits.

Definition 14. *A variable-length output pseudorandom function family* PRF^* *is* (t, q, ε)-secure *if no attacker $\mathcal{A}$ running in time t and making at most q queries can win the following distinguishing game with probability larger than $1/2 + \varepsilon$:*

1. *For $i = 1, \ldots, q' \leq q$: $\mathcal{A}$ chooses arbitrary values c_i, ℓ_i and receives the value $\mathsf{PRF}^*(\kappa, c_i, \ell_i)$ (queries by $\mathcal{A}$ are adaptive, i.e., each query may depend on the responses to previous ones).*
2. *$\mathcal{A}$ chooses values c and ℓ such that $c \notin \{c_1, \ldots, c_q\}$.*
3. *A bit $b \in_R \{0,1\}$ is chosen at random. If $b = 0$, $\mathcal{A}$ is provided with the output of $\mathsf{PRF}^*(\kappa, c, \ell)$, else $\mathcal{A}$ is given a random string of ℓ bits.*
4. *Step 4 is repeated for up to $q - q'$ queries.*
5. *$\mathcal{A}$ outputs a bit $b' \in \{0,1\}$. It wins if $b' = b$.*