Randomness Extraction and Key Derivation Using the CBC, Cascade and HMAC Modes[*]

Yevgeniy Dodis[1], Rosario Gennaro[2], Johan Håstad[3], Hugo Krawczyk[4], and Tal Rabin[2]

[1] New York University
dodis@cs.nyu.edu
[2] IBM Research
{rosario,talr}@watson.ibm.com
[3] Royal Institute, Sweden
johanh@nada.kth.se
[4] Technion, Israel, and IBM Research
hugo@ee.technion.ac.il

Abstract. We study the suitability of common pseudorandomness modes associated with cryptographic hash functions and block ciphers (CBC-MAC, Cascade and HMAC) for the task of "randomness extraction", namely, the derivation of keying material from semi-secret and/or semi-random sources. Important applications for such extractors include the derivation of strong cryptographic keys from non-uniform sources of randomness (for example, to extract a seed for a pseudorandom generator from a weak source of physical or digital noise), and the derivation of pseudorandom keys from a Diffie-Hellman value.
Extractors are closely related in their applications to pseudorandom functions and thus it is attractive to (re)use the common pseudorandom modes as randomness extractors. Yet, the crucial difference between pseudorandom generation and randomness extraction is that the former uses random *secret* keys while the latter uses random *but known* keys. We show that under a variety of assumptions on the underlying primitives (block ciphers and compression functions), ranging from ideal randomness assumptions to realistic universal-hashing properties, these modes induce good extractors. Hence, these schemes represent a more practical alternative to combinatorial extractors (that are seldom used in practice), and a better-analyzed alternative to the common practice of using SHA-1 or MD5 (as a single un-keyed function) for randomness extraction. In particular, our results serve to validate the method of key extraction and key derivation from Diffie-Hellman values used in the IKE (IPsec's Key Exchange) protocol.

1 Introduction

1.1 Key Derivation and Randomness Extractors

Key derivation is a central functionality in cryptography concerned with the process of deriving secret and random cryptographic keys from some source of

[*] Extended abstract. Full version available at eprint.iacr.org/2004/

© International Association for Cryptologic Research 2004

semi-secret randomness. In general, it is sufficient to derive a single random and secret key (say of length 128) which can then be used to key a pseudorandom function (or a pseudorandom generator) to obtain further pseudorandom keys as needed. Thus, a basic question, which motivates the work presented here, is how to derive such a random and secret key when all that is given is an imperfect source of randomness which contains some good amount of secret (computational) entropy, but this entropy is not presented in a direct form of uniformly (or pseudorandomly) distributed secret bits. This problem arises in a variety of scenarios such as when deriving keys from a non-uniform source of noise (as used, for example, by physical random generators) or from semi-random data (say, coming from user's input or the sampling of computer events, etc.). This is also the case when deriving keys from a Diffie-Hellman (DH) exchange. Let us elaborate on the latter case.

Let's assume that two parties run a DH protocol in order to agree on a shared secret key, namely, they exchange DH exponentials g^x and g^y and compute the DH value g^{xy}. In this case, g^x and g^y as seen by the attacker fully determine g^{xy}. Yet, it is assumed (by the Decisional Diffie-Hellman, DDH, assumption) that a computationally-bounded attacker cannot distinguish g^{xy} from a random element in the group generated by g. Thus, one can assume that g^{xy} contains $t = \log_2(order(g))$ bits of computational entropy relative to the view of the attacker (for a formal treatment of computational entropy in the DH context see [GKR04]). However, this entropy is spread over the whole value g^{xy} which may be significantly longer than t[1]. Thus, we are in a situation similar to that of an imperfect source of randomness as discussed above. In particular, g^{xy} cannot be used directly as a cryptographic key, but rather as a source from which to *extract* a shorter string (say, of length 128) of full computational entropy which can then be used as a cryptographic key.

The tools used to derive a uniform key from these sources of imperfect randomness are often referred to as *randomness extractors*. The amount of theoretical results in this area is impressive; moreover, some of the constructions that have proven extraction guarantees are also efficient (see [Sha02] for a recent survey). One such example is the so called "pairwise independent universal hash functions" (also called "strongly universal") [CW79] which have quite efficient implementations and provable extraction properties. In particular, [HILL99] shows (see also [Lub96,Gol01]) that if an input distribution has sufficient min-entropy (meaning that no single value is assigned a too-large probability even though the distribution may be far from uniform) then hashing this input into a (sufficiently) shorter output using a function chosen at random from a family of strongly universal hash functions results in an output that is statistically-close to uniform. (This result is often referred to as the "Leftover Hash Lemma".)

[1] For example, consider that g is an element of prime order q in Z_p^* (i.e., p and q are primes and $q/p-1$), and that $|p| = 1024$ and $|q| = 512$. In this case the DDH assumption guarantees that the value g^{xy} hides (from the attacker) 512 bits of computational entropy, yet these bits are spread in some unknown way among the 1024 bits of g^{xy}.

Yet, in spite of these results and an extensive literature studying their application to real cryptographic problems (such as those mentioned earlier, and in particular for the DH case [GKR04]) one seldom encounters in practice the use of strong universal hashing or other proven extractors. Instead, the common practice is to use cryptographic hash functions (such as MD5 and SHA-1) for the purpose of randomness extraction. A main reason for this practice, justified by engineering considerations, is that cryptographic hash functions are readily available in software and hardware implementations, and are required by most cryptographic applications for purposes other than randomness extraction (e.g., as pseudorandom functions). Therefore, it is attractive and convenient to use them for key extraction as well. Also, the common perception that these hash functions behave as random functions (formalized via the notion of "random oracles") make them intuitively appealing for the extraction applications.

1.2 Randomness Extraction via Common Chaining Modes

In this paper we attempt to bridge between the world of provable extraction and the common practice of relying on idealized hash functions. The question that we ask is what is the best way to use cryptographic hash functions, or other widely available cryptographic tools such as block ciphers, for the task of randomness extraction. Specifically, we consider three common modes of operation: CBC chaining, cascade (or Merkle-Damgard) chaining, and HMAC, and analyze the appropriateness of these modes as extraction tools. Since the goal is to provide as general (and generic) as possible results, we do not investigate the extraction properties of specific functions (say SHA-1 or AES) but rather abstract the basic primitives (the compression functions in the case of the cascade and HMAC modes, and block ciphers in the case of CBC), as random functions or permutations[2].

Before going on with the description of our results, it is worth considering the following issue. Given that the common practice is to extract randomness using a hash function modeled as a random oracle, then how much do we gain by analyzing the above modes under the weaker, but still idealized, randomness assumption on the underlying basic primitives. There are several aspects to this question.

The first thing to note is that modeling the *compression function of SHA-1*, for example, as a random function, or as a family of random functions, is a strict relaxation to modeling *SHA-1* (as a single un-keyed function) as a random function. This is easily seen from the fact that even if one starts with a random function as the compression function the result of the cascade chaining (which is how SHA-1 is derived) *is not a random function*. (For example, in the cascade construction, the probability that two L-block inputs that differ only in their first block are mapped to the same k-bit output is $L/2^k$, while for

[2] In the case of HMAC we obtain results based on non-ideal assumptions on the underlying basic primitives (see Section 1.3).

a random function this probability is $1/2^k$). Another important point is that cryptographic design work focuses on building the basic blocks, i.e. a compression function or a block cipher. Thus, making an assumption on these primitives will represent the design goals which there can be an attempt to satisfy. Also analysis of the type presented here, rather than implying the security of any specific implementation of these functions, serves to validate the suitability of the corresponding chaining modes for some defined goal (in our case the goal is randomness extraction). Indeed, the common approach for analyzing such modes (e.g., [Dam89,BKR94,BCK96a,BCK96b]) is to make some assumption on the basic primitive (for example, assuming the underlying compression function to be a pseudorandom function, or a secure MAC, or a collision-resistant hash function) and then proving that these or other properties are preserved or implied by the chaining operation.

In addition, the "monolithic" randomness assumption on a single (unkeyed) function such as SHA-1 is inappropriate for the setting of randomness extraction as no single function (even if fully random) can extract a close-to-uniform distribution from arbitrary high-entropy input distributions. This is so, since once the function is fixed (even if to purely random values) then there are high-entropy input distributions that will be mapped to small subsets of outputs[3]. Therefore, the viable approach for randomness extraction is to consider a *family* (or collection) of functions indexed by a set of keys. When an application requires the hashing of an input for the purpose of extracting randomness, then a random element (i.e., a function) from this family is chosen and the function is applied to the given input. While there may be specific input distributions that interact badly with specific functions in the family, a good randomness-extraction family will make this "bad event" happen with very small probability. Universal hash families, mentioned before, are examples of this approach. An *important point* here is that, while the choice of a function from the family is done by selecting a random index, or key, this key does not need to be kept secret (this is important in applications that use extraction to generate secret keys; otherwise, if we required this index to be secret then we would have a "chicken and egg" problem).

In our setting, families of keyed functions come up naturally with block ciphers and compression functions (for the latter we consider, as in HMAC, the variable IV as the key to the function). These functions are defined on fixed length inputs (e.g., 512 bits in the case of compression function of SHA-1, or 128 in the case of AES). Then, to hash arbitrarily long inputs, we extend these families by the appropriate chaining mode: cascade chaining (or HMAC) for compression functions, and CBC-MAC in the case of block ciphers. What makes the analysis of these functions challenging (in the setting of randomness extraction) is that, as discussed before, the key to the function is random but known. For ex-

[3] For example, let F be a random function from ℓ to k bits and let S denote the subset of $\{0,1\}^\ell$ that is mapped by F to outputs with a low-order bit of zero. If we consider the uniform distribution on S as the input distribution, then this distribution has almost full entropy, yet the output of F on S is trivially distinguishable from uniform.

ample, the fact that the above functions are widely believed to be pseudorandom does not help much here since, once the key is revealed, the pseudorandom properties may be lost completely (see full paper). Yet, as we will see in Section 4.2, we do use the pseudorandom property in some of our analysis. Also worth noting is that using families that are pseudorandom for extraction is particularly convenient since these same functions can then be used by the same application (for example, a key-exchange protocol, a random generator, etc.) for further key derivation (using the extracted key to key the pseudorandom function).

The last question is how to generate the random known keys used by the extractor. Technically this is not hard, as the parties can generate the appropriate randomness, but the exact details depend on the application. For example, in the DH key exchange discussed earlier, the parties exchange in the clear randomly chosen values, which are then combined to generate a single key κ for the extractor family $\{H_\kappa\}$ (e.g. HMAC-SHA1). The shared key is set to $H_\kappa(g^{xy})$. We note that this is substantially the procedure in place in the IKE protocol [RFC2409,IKEv2] (see also [Kra03]), and this paper presents the first formal analysis of this design.

A similar DH key extraction step is required in non-interactive scenarios, such as ElGamal or Cramer-Shoup encryption. There the extractor key κ can be chosen either by the encryptor and appended to the ciphertext, or chosen by the decryptor and included in the public key (this choice is mandatory in case we want CCA-security, as we don't want to leave the choice of κ in the hands of the adversary). For a different example, consider a cryptographic hardware device, containing a physical random generator that samples some imperfect source of noise. In this case the application can choose a random hash function in the family and wire-in its key into the randomness generation circuit [BST03]. Notice that by using our results, it will be possible to perform the extraction step using circuitry (such as a block-cipher or a cryptographic hash function) which is very likely to already be part of the device.

1.3 Our Results

The Extraction Properties of CBC-MAC Mode. We show, in Section 3, that if f is a random permutation over $\{0,1\}^k$ and $\mathcal{X}$ is an input distribution with min-entropy of at least $2k$, then the statistical distance between $F(\mathcal{X})$ (where F represents the function f computed in CBC-MAC mode over L blocks) and the uniform distribution on $\{0,1\}^k$ is $L \cdot 2^{-k/2}$. As an example, in the application (discussed before) in which we use the CBC-MAC function F to hash a Diffie-Hellman value computed over a DDH group of order larger than 2^{2k}, we get that the output distribution $F(g^{xy})$ is computationally indistinguishable from a distribution whose distance from uniform is at most $L2^{-k/2}$, hence proving (under DDH) that the k-bit output from $F(g^{xy})$ is computationally indistinguishable from uniform (and thus suitable for use as a cryptographic key). Note that if one works over Z_p^* for 1024-bit p and $k = 128$, then all we need to assume is a min-entropy of 256 out of the 1024 bits of g^{xy}. In the full paper we show that for input distributions with particularly high entropy (in particular those that contain a k-bit block of almost-full entropy) the CBC-MAC mode guarantees an almost-uniform output for *any* family of permutations.

The Extraction Properties of Cascade Chaining. In Section 4 we study the cascade (or Merkle-Damgard) chaining used in common hash functions such as MD5 and SHA-1. We show these families to be good extractors when modeling the underlying compression function as a family of random functions. However, in this case we need a stronger assumption on the entropy of the input distribution. Specifically, if the output of the compression function is k-bit long (typically, $k = 128$ or 160) we assume a min-entropy of $2k$ over the whole input, and "enough" min-entropy over the distribution induced on the last block of input (typically of length 512 bits). For example, if the last block has k bits of min-entropy, and we assume L blocks, then the statistical distance between the output of the cascade construction and the uniform distribution (on $\{0,1\}^k$) is at most $L \cdot 2^{-k/2}$. We note that the above restriction on the last-block distribution is particularly problematic in the case of practical functions such as MD5 and SHA since the input-padding conventions of these functions may cause a full *fixed* block to be added as the last block of input. In this case, the output distribution is provably far from uniform. Fortunately, we show that our analysis is applicable also to the padded-input case. However, instead of proving a negligible statistical distance, what we show is that the output of the "padded cascade" is *computationally indistinguishable* from uniform, a result that suffices for the cryptographic applications of extraction. Finally, we prove that when every block of input has large-enough min-entropy (conditioned on the distribution of previous blocks), then the above extraction results hold under the sole (*and non-ideal*) assumption that the underlying compression function is a family of δ-AU functions (for sufficiently small δ).

The Extraction Properties of HMAC. HMAC is the most widely used pseudorandom mode based on functions such as MD5 or SHA, thus proving its extraction properties is extremely important. Our main result concerning the good extraction properties of HMAC is proven on the basis of a high min-entropy ($2k$ bits) in the input distribution $\mathcal{X}$, without relying on any particular entropy in the last block of input. Specifically, let F denote the keyed hash function underlying an instantiation of HMAC (e.g., F is SHA-1 with random IV) and let f be the corresponding outer compression function. Then we show that if F is collision resistant and f is modeled as a random function then the output of HMAC (on input drawn from the distribution $\mathcal{X}$) is indistinguishable from uniform for any attacker that is restricted in the number of queries to the function f. Moreover, if the compression function itself is a good extractor, then HMAC is a good extractor too. However, in this latter case if we are interested in an output of ℓ close-to-uniform bits then the key to the underlying compression function needs to be sufficiently larger than ℓ. As a concrete example, if $\ell = 160$ (e.g., we need to generate a pseudorandom key of length 160) then we can use HMAC with SHA2-512. Note that this result is particularly interesting in the sense that it uses no idealized assumptions, and yet the output of HMAC is provably close to uniform (even against completely unbounded attackers, including attackers that can break the collision resistance of F).

Remark *(Pseudorandom functions with known keys).* It is tempting to use the pseudorandomness properties enjoyed by the modes studied here as a basis to claim good (computational) extraction properties. For example, in spite of the fact that the output of these functions may be statistically very-far from uniform, it is still true that no (efficient) *standard* statistical test will be able to tell apart this output from random (simply because such a test does not use the knowledge of the key even if this key is known.) Yet, for cryptographic applications using a family of functions as extractors, based solely on the assumption that the family is pseudorandom, may be totally insecure. We illustrate this point by showing, in the full paper, an example of a secure pseudorandom family whose output is trivially distinguishable from randomness once the key is known.

All proofs appear in the full version of the paper.

2 Universal Hashing and Randomness Extraction

Preliminaries. For a probability distribution $\mathcal{X}$ we use the notation $x \in_{\mathrm{R}} \mathcal{X}$ to mean that x is chosen according to the distribution $\mathcal{X}$. For a set S, $x \in_{\mathrm{R}} S$ is used to mean that x is chosen from S with uniform probability. Also, for a probability distribution $\mathcal{X}$ we use the notation $\Pr_{\mathcal{X}}(x)$ to denote the probability assigned by $\mathcal{X}$ to the value x. (We often omit the subscript when the probability distribution is clear from the context.) Throughout the paper, we will use $d(L)$ to denote the maximal numbers of divisors of any number smaller or equal to L. As a *very* crude upper bound, we will sometimes use the fact that $d(L) \leq 2\sqrt{L}$.

Min-Entropy and Collision probability. For a probability distribution $\mathcal{X}$ over $\{0,1\}^{\ell}$, we define its *min-entropy* as the minimum integer m such that for all $x \in \{0,1\}^{\ell}, \Pr_{\mathcal{X}}(x) \leq 2^{-m}$. We denote the min-entropy of such $\mathcal{X}$ by $\mathbf{H}_{\infty}(\mathcal{X})$. The *collision probability* of $\mathcal{X}$ is $\mathsf{Col}(\mathcal{X}) = \Pr_{x,x' \in_{\mathrm{R}} \mathcal{X}}(x = x') = \sum_x \Pr(\mathcal{X} = x)^2$, and the *Renyi (or collision) entropy* of $\mathcal{X}$ is $\mathbf{H}_2(\mathcal{X}) = -\log_2 \mathsf{Col}(\mathcal{X})$. It is easy to see that these two notions of entropy are related: $\mathbf{H}_{\infty}(\mathcal{X}) \leq \mathbf{H}_2(\mathcal{X}) \leq 2\mathbf{H}_{\infty}(\mathcal{X})$. In particular, we will frequently use the fact that $\mathsf{Col}(\mathcal{X}) = 2^{-\mathbf{H}_2(\mathcal{X})} \leq 2^{-\mathbf{H}_{\infty}(\mathcal{X})}$.

Statistical Distance. Let $\mathcal{X}_1, \mathcal{X}_2$ be two probability distributions over the set S. The *statistical distance* between the distributions $\mathcal{X}_1$ and $\mathcal{X}_2$ is defined as $\mathbf{SD}(\mathcal{X}_1, \mathcal{X}_2) = \frac{1}{2}\sum_{s \in S} |\Pr_{\mathcal{X}_1}(s) - \Pr_{\mathcal{X}_2}(s)|$. If two distributions have statistical distance of (at most) ϵ, then we refer to them as ϵ-close. We note that ϵ-close distributions cannot be distinguished with probability better than ϵ even by a computationally unbounded adversary. It is also well known that if $\mathcal{Y}$ has support on some set S and U is the uniform distribution over this set, then

$$\mathbf{SD}(\mathcal{Y}, U) \leq \frac{1}{2}\sqrt{|S| \cdot \mathsf{Col}(\mathcal{Y}) - 1} \qquad (1)$$

Definition 1. *Let k and ℓ be integers, and $\{h_\kappa\}_{\kappa\in\mathcal{K}}$ be a family of hash functions with domain $\{0,1\}^\ell$, range $\{0,1\}^k$ and key space $\mathcal{K}$. We say that the family $\{h_\kappa\}_{\kappa\in\mathcal{K}}$ is δ-*almost universal (δ-AU) *if* for every pair of different *inputs x, y from $\{0,1\}^\ell$ it holds that $\Pr(h_\kappa(x) = h_\kappa(y)) \le \delta$, where the probability is taken over $\kappa \in_R \mathcal{K}$. For a given probability distribution $\mathcal{X}$ on $\{0,1\}^\ell$, we say that $\{h_\kappa\}_{\kappa\in\mathcal{K}}$ is δ-*AU w.r.t. *$\mathcal{X}$ if $\Pr(h_\kappa(x) = h_\kappa(y)) \le \delta$ where the probability is taken over $\kappa \in_R \mathcal{K}$ and $x, y \in_R \mathcal{X}$ conditioned to $x \ne y$.*

Clearly, a family $\{h_\kappa\}_{\kappa\in\mathcal{K}}$ is δ-AU if it is δ-AU w.r.t. all distributions on $\{0,1\}^\ell$. The notion of universal hashing originates with the seminal papers by Carter and Wegman [CW79,WC81]; the δ-AU variant used here was first formulated in [Sti94]. The main usefulness of this notion comes from the following lemma whose proof is immediately obtained by conditioning on whether the two independent samples from $\mathcal{X}$ collide or not (below $\mathbf{E}$ denotes the expected value).

Lemma 1. *If $\{h_\kappa\}_{\kappa\in\mathcal{K}}$ is δ-AU w.r.t. $\mathcal{X}$, then $\mathbf{E}_\kappa[\mathsf{Col}(h_\kappa(\mathcal{X}))] \le \mathsf{Col}(\mathcal{X}) + \delta$.*

Now, using the above lemma and Eq. (1), the lemma below extends the well-known "Leftover Hash Lemma" (LHL) from [HILL99] in two ways. First, it relaxes the pairwise-independence condition assumed by that lemma on the family of hash functions, and allows for "imperfect" families in which the collision probability is only ϵ-close to perfect (i.e., $2^{-k} + \epsilon$ instead of 2^{-k}). Second, it allows for the collision probability to depend on the input distribution rather than being an absolute property of the family of hash functions. We use these straightforward extensions of the LHL in an essential way for achieving our results. We also note that the standard LHL can be obtained from Lemma 2 below by setting $\epsilon = 0$.

Lemma 2. *Let ℓ and k be integers, let $\mathcal{X}$ be a probability distribution over $\{0,1\}^\ell$, and let $\{h_\kappa\}_{\kappa\in\mathcal{K}}$ be a family of hash function with domain $\{0,1\}^\ell$ and range $\{0,1\}^k$. If $\{h_\kappa\}_{\kappa\in\mathcal{K}}$ is $(\frac{1}{2^k} + \epsilon)$-almost universal w.r.t. $\mathcal{X}$, U is uniform over $\{0,1\}^k$ and κ is uniform over $\mathcal{K}$, then*

$$\mathbf{SD}((\kappa, h_\kappa(\mathcal{X})), (\kappa, U)) \le \frac{1}{2} \cdot \sqrt{2^k \cdot (\mathsf{Col}(\mathcal{X}) + \epsilon)} \le \frac{1}{2} \cdot \sqrt{2^k \cdot (2^{-\mathbf{H}_\infty(\mathcal{X})} + \epsilon)} \quad (2)$$

Remark 1. It is important to note that for the above lemma to be useful one needs $\epsilon \ll 1/2^k$, or otherwise the derived bound on the statistical closeness approaches 1. Moreover, this fact is not a result of a sub-optimal analysis but rather there are examples of families with $\epsilon = 1/2^k$ (i.e., $(2/2^k)$-AU families) that generate outputs that are easily distinguishable from uniform. For example, if $\{h_\kappa\}$ is a family of pairwise independent hash functions with k-bit outputs, and we define a new family $\{h'_\kappa\}$ which is identical to $\{h_\kappa\}$ except that it replaces the last bit of output with 0, then the new family has collision probability of $2/2^k$ yet its output (which has a fixed bit of output) is trivially distinguishable from uniform. The fact that we need $\epsilon \ll 1/2^k$ (say, $\epsilon \approx 1/2^{2k}$) makes the analysis of CBC and the cascade construction presented in the next sections non-trivial. In particular, the existing analyses of these functions (such as [BKR94,BCK96b]) are too weak for our purposes as they yield upper bounds on the collision probability of these constructions that are larger than $2/2^k$.

3 The CBC-MAC Construction

Here we study the suitability of the CBC-MAC mode as a randomness extractor. Recall that for a given permutation f on $\{0,1\}^k$, the CBC-MAC computation of f on an input $x = (x_1, x_2, \dots x_L)$, with L blocks in $\{0,1\}^k$, is defined as $\bar{x}_L$ where the latter value is set by the recursion: $\bar{x}_0 = 0$, $\bar{x}_i = f(x_i \oplus \bar{x}_{i-1})$ for $1 \le i \le L$. We denote the output of the above CBC-MAC process by $F(x)$.

Our main result in this section states the extraction properties of CBC-MAC for a *random* permutation f on $K = 2^k$ elements. To state it more compactly, we let $\epsilon(L,K) = (d(L))^2 L K^{-2} + L^6 K^{-3}$, and notice that $\epsilon(L,K) = O(L^2/K^2)$ when $L < K^{1/4}$ (here we use the fact that $d(L) \le 2\sqrt{L}$).

Theorem 1. *Let F denote the CBC-MAC mode over a* random *permutation f on $\{0,1\}^k$, and let $\mathcal{X}$ be an input distribution to F defined over L-block strings. Then the statistical distance between $F(\mathcal{X})$ and the uniform distribution on $\{0,1\}^k$ is at most*

$$\sqrt{K \cdot 2^{-\mathbf{H}_\infty(\mathcal{X})} + O(K \cdot \epsilon(L,K))}$$

In particular, assuming $L < 2^{k/4}$ and $\mathbf{H}_\infty(\mathcal{X}) \ge 2k$, the above statistical distance is at most $O(L/2^{k/2})$.

The proof of the theorem follows from Lemma 2 in combination with the following lemma that shows that CBC-MAC mode with a random permutation is δ-AU for sufficiently small δ.

Lemma 3. *Let F denote the CBC-MAC mode over a permutation f on $\{0,1\}^k$. For any $x, y \in \{0,1\}^{Lk}$, if $x \ne y$ then $\Pr[F(x) = F(y)] \le \frac{1}{K} + O(\epsilon(L,K))$ where the probability is over the choice of a* random *permutation f.*

4 The Cascade Construction

We first recall the Merkle-Damgard approach to the design of cryptographic hash functions and introduce some notation and terminology. For given integers k and b, $b > k > 0$, let $\{f_\kappa : \kappa \in \mathcal{K}\}$ be a family of functions such that $\mathcal{K} = \{0,1\}^k$, and for all $\kappa \in \mathcal{K}$ the function f_κ maps b bits into k bits. On the basis of this family we build another family $\{F_\kappa : \kappa \in \mathcal{K}\}$ that works on inputs of any length which is a multiple of b and produces a k-bit output. For each $\kappa \in \mathcal{K}$, the function F_κ is defined as follows. Let $x = (x_1, \dots, x_L)$, for some $L \ge 1$ and $x_i \in \{0,1\}^b$ (for all i), denote the input to F_κ; we define L variables (each of length k)

$\bar{x}_1, \ldots, \bar{x}_L$ as $\bar{x}_0 = \kappa, \bar{x}_{i+1} = f_{\bar{x}_i}(x_{i+1})$, and set $F_\kappa(x) = \bar{x}_L$. For processing inputs of arbitrary length one needs a rule for padding inputs to a length that is a multiple of b. Specific functions that build on the above approach, such as MD5 and SHA-1, define their specific padding; we expand on this issue in Section 4.2. For the moment we assume inputs of length Lb for some L. Some more notation: Sometimes we use F to denote the family $\{F_\kappa\}_{\kappa \in \mathcal{K}}$, and we write $F(x)$ to denote the random variable $F_\kappa(x)$ for $\kappa \in_{\mathrm{R}} \mathcal{K}$. Finally, we use K to denote 2^k.

The family $\{f_\kappa\}_{\kappa \in \mathcal{K}}$ is called the "compression function(s)" and the family $\{F_\kappa\}_{\kappa \in \mathcal{K}}$ is referred to as the "cascade construction" (over the compression function $\{f_\kappa\}_{\kappa \in \mathcal{K}}$). Typical values for the parameters of the compression function are $b = 512$ and $k \in \{128, 160\}$.

4.1 The Basic Cascade Construction

The main result of this section is the following. Assume $\mathcal{X}$ is an input distribution with $2k$ bits of (overall) min-entropy and "enough" bits of min-entropy in its last (b-bit) block ("enough" will be quantified below). For the cascade constructions we model the underlying family of compression functions as a family of random functions (with k-bit outputs). Then the output of F on the distribution $\mathcal{X}$ is statistically close to uniform. This result is formalized in the following theorem. As in Section 3, we let $\epsilon(L, K) = (d(L))^2 L K^{-2} + L^6 K^{-3}$, and notice that $\epsilon(L, K) = O(L^2/K^2)$ when $L < K^{1/4}$.

Theorem 2. *Let $F = \{F_\kappa\}$ be the cascade construction defined, as above, over a family of* random functions *$\{f_\kappa\}$. Let $\mathcal{X}$ be the input distribution to F defined over L-block strings, and $\mathcal{X}_L$ denote the probability distribution induced by $\mathcal{X}$ on the last block x_L for $x \in_R \mathcal{X}$. Then, if U is the uniform distribution over $\{0,1\}^k$, we have*

$$\mathbf{SD}(F(\mathcal{X}),\ U) \leq \sqrt{K \cdot 2^{-\mathbf{H}_\infty(\mathcal{X})} + L \cdot 2^{-\mathbf{H}_\infty(\mathcal{X}_L)} + O(K \cdot \epsilon(L, K))} \qquad (3)$$

In particular, if $\mathbf{H}_\infty(\mathcal{X}) \geq 2k$, $\mathbf{H}_\infty(\mathcal{X}_L) \geq k$, and $L \leq 2^{k/4}$, then $\mathbf{SD}(F(\mathcal{X}), U) \leq O(L/2^{k/2})$.

The proof of the theorem follows from Lemma 2 in combination with the following lemma that shows that the cascade construction with a random family of compression functions is δ-AU for sufficiently small δ.

Lemma 4. *Let $F = \{F_\kappa\}$ be the cascade construction defined over a family of random functions $\{f_\kappa\}$. Let $\mathcal{X}$ be an input distribution as assumed in Theorem 2, where $\mathbf{H}_\infty(\mathcal{X}_L) > \log L$. Then, the family F is $(\frac{1}{K} + \frac{L}{K 2^{\mathbf{H}_\infty(\mathcal{X}_L)}} + O(\epsilon(L, K)))$-AU w.r.t. $\mathcal{X}$.*

The proof of this lemma is based on the following two propositions: the first analyzes the collision probability of the cascade function F over random compression functions on inputs that differ (at least) in the last block (Proposition 1);

then we extend the analysis to the general case, i.e. for any two different x and y (Proposition 2). All proofs appear in the final paper.

Proposition 1. *Let $F = \{F_\kappa\}$ be the cascade construction defined over a family of random functions $\{f_\kappa\}$. Let x, y be two inputs to F that differ (at least) in the last block, namely, $x_L \neq y_L$, and let $\kappa \in \mathcal{K}$ be any value of the initial key. Then $\Pr_F(F_\kappa(x) = F_\kappa(y)) \leq \frac{1}{K} + O(\epsilon(L, K))$, where the probability is taken over the choice of random functions in F.*

Proposition 2. *Let F be defined as above, let x, y be two different inputs to F, and let κ be any value of the initial key. Then $\Pr_F(F(x) = F(y)) \leq \frac{L}{K} + O(L\epsilon(L, K))$, where the probability is taken over the choice of random functions in F.*

The Value of the Initial Key κ. We note that the above analysis holds for any value of the initial key κ, when the functions are truly random, which means that in principle κ can be fixed to a constant. However, in practice not all the functions of the function family $\{f_\kappa\}$ satisfy this requirement. Thus, choosing κ at random allows, for example, to extend our analysis to the situation where a negligible fraction of functions in F are not close to random functions.

Necessity of Min-Entropy in the Last Block. We argue that assuming non-trivial min-entropy in the last block is required, even if the family $\{f_\kappa\}$ of compression functions is completely random. Assume an input distribution in which the last block is fixed to a value B. The first $L-1$ blocks induce some distribution for the last key in the sequence. Examining the distribution on $f_K(B)$, induced by (any) distribution on κ it is easy to see that this distribution is statistically far from the uniform distribution. In particular, we expected with high probability a constant fraction of elements will not appear in the distribution.

4.2 The Cascade Construction with Input Padding

The conditions imposed by Theorem 2 on the input distribution $\mathcal{X}$ conflict with a technical detail of the practical implementations of the cascade paradigm (such as MD5 and SHA-1): rather than applying the cascade process to the input $x \in_{\mathrm{R}} \mathcal{X}$, these functions modify x by concatenating enough padding bits as to obtain a new input x' whose length is a full multiple of the block length. In some cases this padding results in a full *fixed* block appended to x. Therefore, even if $\mathcal{X}$ has the property that the last block of input has relatively high entropy (as required by Theorem 2) the actual input x' to the cascade does not have this property any more. This fact is sufficient to make our main result from Section 4.1 irrelevant to these real-world functions; luckily, however, we show here that this serious obstacle can be lifted.

In order to better understand this problem we first describe the actual way this padding is performed. We consider the concrete value $b = 512$ used as the block length in these functions. Let n denote the length of the input x, and let

$n' = n \bmod 512$. If $n' < 448$ then x is padded with the binary string $10^{447-n'}$ followed by a 64-bit representation of n. If $n' \geq 448$ then *a whole new block* is added with a padding similar to the one described above (with the binary representation of n occupying the last 64 bits of the added block). From this description, we see that if, for example, the original input x had a length which was an exact multiple of 512, then $x' = x \parallel B$ where B is a whole new block (and $\parallel$ represents the concatenation operation). Moreover, the value of B is the same (and fixed) for all inputs of the length of x. In particular, if we consider the case in which we hash Diffie-Hellman values of length of 1024 or 2048 bits (this is the common case when working over Z_p^* groups), then we get that the padded input x' will always have a fixed last block. In other words, regardless of the entropy existing in the original input x the actual input to the cascade process now has a zero-entropy last block.

For this case, in which the last block is fixed, we show here that a somewhat weaker (but still very useful) result holds. Specifically, combining Theorem 2 with the assumption that the family of (compression) functions $\{f_\kappa\}$ is pseudorandom, we can prove that the output from the cascade construction is *pseudorandom*, i.e., computationally indistinguishable from uniform (and thus sufficient for most cryptographic applications) This result holds even though the key to the cascade function F is revealed! We note that the assumption that the family $\{f_\kappa\}$ is pseudorandom is clearly implied by the modeling (from the previous subsection) of these functions as random. But we also note that assuming the compression function (family) of actual schemes such as MD5 or SHA-1 to be pseudorandom is a standard and widely-used cryptographic assumption (see [BCK96b] for some analytical results on these PRF constructions).

Lemma 5. *Let $\{f_\kappa\}_{\kappa\in\mathcal{K}}$ be a family of pseudorandom functions which is $\epsilon_p(T)$-indistinguishable from random for attackers restricted to time T and a single query. Let $F = \{F_\kappa\}_{\kappa\in\mathcal{K}}$ denote the cascade construction over the family $\{f_\kappa\}_{\kappa\in\mathcal{K}}$. Further, let $\mathcal{X}$ be a probability distribution on L-block strings from which the inputs to F are chosen, and B be a fixed b-bit block. If the output distribution $F_\kappa(\mathcal{X})$, with random but known key κ, is ϵ_d-statistically close to uniform, then the distribution $F_\kappa(\mathcal{X} \parallel B)$ (for random but known κ) is $(\epsilon_d + \epsilon_p(T))$-indistinguishable from uniform by attackers that run time T.*

The above lemma together with Theorem 2 show that if $\{f_\kappa\}$ is a family of random functions then the cascade construction with a fixed last block block is indistinguishable from random, provided that the original input distribution (before padding!) satisfies the conditions of Theorem 2.

It is also worth noting that Lemma 5 can be generalized to input distributions $\mathcal{X}^*$ that can be described as the concatenation of two probability distributions $\mathcal{X}$ and $\mathcal{Y}$, where $\mathcal{X}$ satisfies the conditions of Theorem 2, and $\mathcal{Y}$ is an arbitrary (polynomial-time samplable) distribution *independent* from $\mathcal{X}$.

A PRACTICAL CONSIDERATION. Note that the application of Lemma 5 on an input distribution $\mathcal{X}$ still requires the last block of $\mathcal{X}$ (before padding) to have relatively high min-entropy. To maximize this last-block min-entropy it is advisable that any input x whose length is not a multiple of $b = 512$ be "shifted to the right" (to a block boundary) by prepending a sufficient number of bits (say, all zeros) to the beginning of x. This way, the resultant string x' is of length a multiple of b and, more importantly, its last block contains the full entropy of the last b bits in x. Also, this shifting forces the appended padding described earlier to add a full block as assumed in Lemma 5[4].

4.3 Modeling the Compression Function as a δ-AU Family

In Section 4.1 we presented an analysis of the basic cascade construction under the modeling of the compression function as a family of random functions. Here we study the question of what can be guaranteed on the output distribution of the cascade under the simple assumption that *the family of compression functions is a good extractor* (or more generally that this family is δ-AU). Clearly this is a more realistic assumption on the underlying compression function. On the other hand, in order to prove a close-to-uniform output in this case we are going to require a stronger assumption on the input distribution. Specifically, we are going to assume that the distribution on every block of input has a high min-entropy (e.g., $2k$ bits of min-entropy out of the b bits in the block), conditioned on the distribution of the previous blocks. We prove below that under these conditions the output of the cascade function is statistically close to uniform.

We note that the above requirement on the input distribution, while stringent, is met in some important cases, such as applications that extract keys from a Diffie-Hellman value computed over a high-entropy group. In particular, this requirement is satisfied by the DH groups in use with the IKE protocol.

Conditional Entropy. Let $\mathcal{X}$ and $\mathcal{Y}$ be two probability distributions over $\{0,1\}^a$ and $\{0,1\}^b$ respectively. If $y \in \{0,1\}^b$ we denote with $\mathcal{X}|y$ the distribution $\mathcal{X}$ conditioned to the event that the string y is selected according to $\mathcal{Y}$. Then we can define the conditional min-entropy of $\mathcal{X}|y$ (and denote it as $\mathbf{H}_\infty(\mathcal{X}|y)$) as the minimum integer m such that for all $x \in \{0,1\}^a$, $\Pr_{\mathcal{X}|y}(x) \le 2^{-m}$.

We define the conditional min-entropy of $\mathcal{X}$ with respect to $\mathcal{Y}$ as the expectation over $\mathcal{Y}$ of $\mathbf{H}_\infty(\mathcal{X}|y)$: $\mathbf{H}_\infty(\mathcal{X}|\mathcal{Y}) = \sum_{y\in\{0,1\}^b} \Pr_{\mathcal{Y}}(y) \cdot \mathbf{H}_\infty(\mathcal{X}|y)$.

Lemma 6. *Assume that the family of compression functions $\{f_\kappa\}_{\kappa\in\mathcal{K}}$ from b to k bits has the property that for any probability distribution $\mathcal{B}$ defined over $\{0,1\}^b$ with min-entropy of m, the output distribution $f_\kappa(B)$, for $\kappa \in_R \mathcal{K}$ and $B \in_R \mathcal{B}$, is ϵ-close to uniform (for some given $\epsilon = \epsilon(b,k,m)$). Further, assume that $\mathcal{X}$ is an input distribution on L b-bit blocks with the property that for each $i = 1, \ldots, L$,*

[4] For example, assume the inputs from the distribution $\mathcal{X}$ to be of length 1800 bits. Given such an input x we prepend to it 248 '0's resulting in a 4-block string $x' = 0^{248} \parallel x$. Now when this input is processed by, say, SHA-1 an additional fifth block is added to x'. The important thing is that the last block of x' receives as much entropy from the last 512 bits of x as possible.

the distribution $\mathcal{X}_i$ induced by $\mathcal{X}$ on the i-th block has conditional min-entropy m with respect to the distribution induced by $\mathcal{X}$ on blocks $1, \ldots, i-1$, and that $\mathcal{X}_i$ Then, the cascade construction over the family $\{f_\kappa\}$ applied to the distribution $\mathcal{X}$ is $(L \cdot \epsilon)$-close to uniform.

In particular, if we assume that the family $\{f_\kappa\}$ is $(2^{-k} + 2^{-2k})$-AU and the min-entropy of each input block (as defined above) is at least $m = 2k$, then we get (using Lemma 2) a statistical distance between the cascade construction on L blocks and the uniform distribution on $\{0,1\}^k$ of at most $L2^{-k/2}$.

Combining Lemmas 6 and 5 we get that, under the above assumption on the input distribution, if the family of compression functions is both δ-AU and pseudorandom then the output of the padded cascade (see Section 4.2) is pseudorandom (i.e. indistinguishable from uniform).

5 HMAC Construction

We now turn to the study of HMAC [BCK96a] as a randomness extraction family. HMAC (and its underlying family NMAC) is defined using the cascade construction over a family of compression functions $\{f_\kappa\}_{\kappa \in \mathcal{K}}$ with domain $\{0,1\}^b$, range $\{0,1\}^k$ and $\mathcal{K} = \{0,1\}^k$ (as usual we denote $K = 2^k$). The family of functions NMAC uses two independent keys drawn from $\mathcal{K}$ and is defined over $\{0,1\}^*$ as $\text{NMAC}_{\kappa_1,\kappa_2}(x) = f_{\kappa_2}(F_{\kappa_1}(x))$ where both x and the result from $F_{\kappa_1}(x)$ are padded as described in Section 4.2. On the basis of NMAC one defines the family HMAC as $\text{HMAC}_{\kappa_1,\kappa_2}(x) = \text{NMAC}_{\kappa'_1,\kappa'_2}(x)$ where $\kappa'_1 = f_{iv}(\kappa_1 \oplus pad_1)$ and $\kappa'_2 = f_{iv}(\kappa_2 \oplus pad_2)$, the value iv is fixed to the IV defined by the underlying hash function, and pad_1, pad_2 are two different fixed strings of length b. The analysis of HMAC is based on that of NMAC under the specialized assumption that the keys κ'_1 and κ'_2 are "essentially independent". We keep this assumption and develop our analysis on NMAC. (The reason of this form of derivation of the keys κ'_1, κ'_2 in HMAC is to allow for the use, without modification, of the underlying hash function; in particular, without having to replace the fixed IV with a variable value.)

We start by observing that if one considers the family $\text{NMAC}_{\kappa_1,\kappa_2}$ as a δ-AU family then we get $\delta > 2/K$. This is so since for any two inputs x, y the probability that NMAC sends both values to the same output is the sum of the probability that $F_{\kappa_1}(x) = F_{\kappa_1}(y)$ (which is at least $1/K$) plus the probability that $F_{\kappa_1}(x) \neq F_{\kappa_1}(y)$ but f_{κ_2} maps these two different results to the same value (which is also at least $1/K$). Therefore we cannot apply the results of Section 2 directly to the analysis of NMAC.

However, we provide three analyses, which, under different assumptions, establish the security of NMAC as a randomness extractor.

Dropping Some Output Bits. Specifically, we assume the "outer" function f_{κ_2} outputs $k' = k - c$ bits (e.g., in case f_{κ_2} is a random function outputting k bits, one can simply drop the last c bits and view it as a random function outputting k' bits). In this case, a straightforward application of Lemma 1 and Lemma 2 shows that if the family $\{F_{\kappa_1}\}$ is δ_1-AU w.r.t. $\mathcal{X}$ and $\{f_{\kappa_2}\}$ is $(1/2^{k'} + \delta_2)$-AU, then NMAC extracts k' bits of statistical distance at most $\sqrt{2^{k'}(\mathsf{Col}(\mathcal{X}) + \delta_1 + \delta_2)}$ from uniform. Assuming now that both families consist of random functions, then $\delta_2 = 0$ and Proposition 2 implies that $\delta_1 \leq L/K + O(L\epsilon(L, K))$. This means that if $\mathbf{H}_\infty(\mathcal{X}) \geq k$, $k' < k - \log L - 2\log(1/\gamma)$ and $L < 2^{k/4}$, then NMAC extracts k' bits which are γ-close to uniform. In fact, the same is true even if the outer function family $\{f_{\kappa_2}\}$ is merely a good extractor (e.g., if it is pairwise independent). In any case, we get that dropping roughly $(\log L + 160)$ bits from the output of the NMAC construction makes it a good extractor. To make this result meaningful, however, we must consider compression functions whose key is non-trivially larger than 160 bits, such as the compression function for SHA2-512.

Computational Security. Our second approach to analyzing NMAC is similar to the analysis of the padded cascade from Lemma 5. We will present it in the full version.

Modeling f_{κ_2} as a Random Oracle. As we remarked, even if f_{κ_2} is truly random, the value $f_{\kappa_2}(F_{\kappa_1}(\mathcal{X}))$ cannot be statistically close to uniform, even if $F_{\kappa_1}(\mathcal{X})$ was perfectly uniform. This was argued under an extremely strong distinguisher that can evaluate $f = f_{\kappa_2}$ at all of its 2^k inputs. This is different from the typical modeling of f as a random *oracle*. Namely, in the random oracle model it is assumed that the adversary can evaluate f at most a bounded number of points, $q \ll 2^k$. This assumption can be seen as restrictive, but in fact a realistic characterization of the adversary's capabilities. Thus, we show that when we model the outer function of NMAC, f, as a random oracle then the construction is a good randomness extractor. We start by showing the general result about the quality of using a random oracle f as an extractor, and then apply it to the NMAC construction.

5.1 Random Oracle as an Extractor

In this section, we show that by utilizing the power of the random oracle to the fullest, we can provide some provable guarantees on the quality of the random oracle as a randomness extractor. Our precise modeling of the random oracle $f : \{0,1\}^b \to \{0,1\}^k$ is the following. The adversary is allowed to adaptively query the random oracle f at upto q points, and possibly make the distribution $\mathcal{X}$ depend on these q queries. However, we assume that the remaining "unqueried" $(2^b - q)$ values of f are chosen randomly and *independently* of $\mathcal{X}$, and are never given to the adversary[5]. Finally, given a distribution $\mathcal{X}$ and a number q, we let $W_q(\mathcal{X})$ denote the probability mass of the q heaviest elements under $\mathcal{X}$.

[5] We stress that this is very different from our modeling of a random *function* from before, where the adversary first chooses the distribution $\mathcal{X}$, after which f is chosen at random (independently from $\mathcal{X}$) and given to the adversary *in its entirety*.

Lemma 7. *Assume f is a random oracle from b bits to k bits, and the adversary can evaluate f in at most q points. Then, for any distribution $\mathcal{X}$ on $\{0,1\}^b$, the maximal probability the adversary can distinguish $f(\mathcal{X})$ from the uniform distribution over $\{0,1\}^k$ is at most*

$$W_q(\mathcal{X}) \leq \min\Big(\ q \cdot 2^{-\mathbf{H}_\infty(\mathcal{X})},\ \sqrt{q \cdot \mathsf{Col}(\mathcal{X})}\ \Big) \tag{4}$$

Remark 2. We already remarked that no single function can be a universally good extractor, which means that one has to use a function family instead, indexed by some key κ. On the other hand, in the idealized random oracle model, we manage to use a single random oracle f in Lemma 7. This is not a contradiction since we critically assumed that the adversary cannot read the entire description of the random oracle. In essence, the choice of the random oracle f can be viewed as a key κ, but the distinguisher cannot read the entire key (although it has a choice which parts of it to read) and therefore cannot adversarially choose a bad distribution $\mathcal{X}$. Put differently, in our analysis we could assume that a large part of the key (i.e., f) is chosen independently of $\mathcal{X}$, which is consistent with the conventional extractors such as those obtained by the LHL. However, unlike the conventional extractors, we (restrictively) assume that the adversary never learns the entire description of the key (i.e., the unqueried parts of f), which allowed us to get a much stronger bound that what we could get with the LHL. For example, LHL required $\mathsf{Col}(\mathcal{X}) \ll 2^{-k}$, while Lemma 7 only requires $\mathsf{Col}(\mathcal{X}) \ll 1/q$.

We will also use the following Corollary of Eq. (4) and Lemma 1.

Corollary 1. *If a family of functions $\{h_\kappa\}_{\kappa\in\mathcal{K}}$ is δ-almost universal w.r.t. $\mathcal{X}$, f is a random oracle, U is the uniform distribution of $\{0,1\}^k$, and the adversary can make at most q queries to the random oracle, then the maximal probability the adversary can distinguish the pair $(\kappa, f(h_k(\mathcal{X})))$ from (κ, U) is at most $\sqrt{q \cdot (\mathsf{Col}(\mathcal{X}) + \delta)}$.*

The above corollary implies that the composition of a collision-resistant hash function and a random oracle could be viewed as a relatively good extractor. This is because a (computational) collision-resistant function must be (information-theoretically) almost universal. More precisely, if a function family $\mathcal{H} = \{h_\kappa\}$ is collision-resistant with exact security δ against non-uniform adversaries running in linear time, it must also be δ-almost universal. For uniform adversaries running in time T, $\mathcal{H}$ must be δ-almost universal w.r.t. any $\mathcal{X}$ which is samplable in time $T/2$.

Application to NMAC. We can now apply Corollary 1 to the case of NMAC assuming that the outer function $f = f_{\kappa_2}$ is a random oracle which can be evaluated in at most q places. By Proposition 2, the family $\{F_{\kappa_1}\}$ is δ-AU when the function family $\{f_{\kappa_1}\}$ is chosen at random, for $\delta < (L+1)/2^k$ (when $L < 2^{k/4}$). Thus, Corollary 1 implies that NMAC extracts k bits which cannot be distinguished from random with probability more than $\sqrt{q(\mathsf{Col}(\mathcal{X}) + (L+1)2^{-k})}$, which is negligible if $q \ll \min[2^{\mathbf{H}_2(\mathcal{X})}, 2^k/(L+1)]$.

Acknowledgments

We thank David Wagner for valuable discussions.

References

[BST03] B. Barak, R. Shaltiel, and E. Tromer. True Random Number Generators Secure in a Changing Environment. In *CHESS '03*, pages 166–180, 2003.

[BCK96a] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In *Crypto '96*, pages 1–15, 1996. LNCS No. 1109.

[BCK96b] M. Bellare, R. Canetti, and H. Krawczyk. Pseudorandom Functions Revisited: The Cascade Construction and Its Concrete Security. In *Proc. 37th FOCS*, pages 514–523. IEEE, 1996.

[BKR94] M. Bellare, J. Kilian, and P. Rogaway. The Security of Cipher Block Chaining. In *Crypto '94*, pages 341–358, 1994. LNCS No. 839.

[CW79] L. Carter and M. N. Wegman. Universal Classes of Hash Functions. *JCSS*, 18(2):143–154, April 1979.

[Dam89] I. Damgard. A Design Principle for Hash Functions. In *Crypto '89*, pages 416–427, 1989. LNCS No. 435.

[GKR04] R. Gennaro, H. Krawczyk, and T. Rabin. Secure Hashed Diffie-Hellman over Non-DDH Groups. In *Eurocrypt '04*, 2004. LNCS No.

[Gol01] Oded Goldreich. *Foundations of Cryptography.* Cambridge University Press, 2001. Preliminary version *http://philby.ucsd.edu/cryptolib.html/*.

[HILL99] J. Håstad, R. Impagliazzo, L. Levin, and M. Luby. Construction of a Pseudo-random Generator from any One-way Function. *SIAM. J. Computing*, 28(4):1364–1396, 1999.

[IKEv2] IKEv2. Internet Key Exchange (IKEv2) Protocol, draft-ietf-ipsec-ikev2-13.txt, (to be published as an RFC). Editor: C. Kaufman, March 2004.

[Kra03] H. Krawczyk. SIGMA: The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and Its Use in the IKE Protocols. In *Crypto '03*, pages 400–425, 2003. LNCS No. 2729.

[Lub96] Michael Luby. *Pseudorandomness and Cryptographic Applications.* Princeton Computer Science Note, Princeton University Press, January 1996.

[RFC2409] RFC2409. The Internet Key Exchange (IKE). Editors: D. Harkins and D. Carrel, Nov 1998.

[Sha02] R. Shaltiel. Recent developments in Extractors. Bulletin of the European Association for Theoretical Computer Science, Volume 77, June 2002, pages 67-95. Available at:
http://www.wisdom.weizmann.ac.il/~ronens/papers/survey.ps

[Sti94] D. R. Stinson. Universal Hashing and Authentication Codes. Designs, Codes and Cryptography 4 (1994), 369-380.

[WC81] M.N. Wegman and L. Carter. New Hash Functions and Their Use in Authentication and Set Equality. *JCSS*, 22(3):265–279, July 1981.