# How to Extract and Expand Randomness: A Summary and Explanation of Existing Results*

Yvonne Cliff, Colin Boyd, and Juan Gonzalez Nieto

Information Security Institute, Queensland University of Technology
GPO Box 2434, Brisbane Qld 4001, Australia
y.cliff@isi.qut.edu.au, {c.boyd,j.gonzaleznieto}@qut.edu.au

**Abstract.** We examine the use of randomness extraction and expansion in key agreement (KA) protocols to generate uniformly random keys in the standard model. Although existing works provide the basic theorems necessary, they lack details or examples of appropriate cryptographic primitives and/or parameter sizes. This has lead to the large amount of min-entropy needed in the (non-uniform) shared secret being overlooked in proposals and efficiency comparisons of KA protocols. We therefore summarize existing work in the area and examine the security levels achieved with the use of various extractors and expanders for particular parameter sizes. The tables presented herein show that the shared secret needs a min-entropy of at least 292 bits (and even more with more realistic assumptions) to achieve an overall security level of 80 bits using the extractors and expanders we consider. The tables may be used to find the min-entropy required for various security levels and assumptions. We also find that when using the short exponent theorems of Gennaro et al., the short exponents may need to be much longer than they suggested.

**Keywords:** randomness extraction, randomness expansion, key agreement, key exchange protocols, pseudorandom function (PRF), universal hash function, leftover hash lemma (LHL).

## 1 Introduction

In this paper we examine the techniques available for extracting and expanding randomness in the context of key agreement (KA) protocols. In such protocols, an agreed secret key is often a random member of a given group, and not a string of bits distributed uniformly at random. However, when the key is used, e.g. as the key of a symmetric encryption scheme, it is likely that a key consisting of bits distributed uniformly at random will be necessary, requiring the use of randomness extraction, and possibly randomness expansion techniques.

* This is an extended abstract. The full version is available at http://eprint.iacr.org/2009/136. Research funded by Australian Research Council through Discovery Project DP0666065.

M. Abdalla et al. (Eds.): ACNS 2009, LNCS 5536, pp. 53–70, 2009.

Informally, a randomness extractor is a family of functions keyed by a random but public value, where the input to each function is a value with high entropy, and the output is indistinguishable from a uniformly random bit string. Unfortunately, the number of bits of entropy in the input must usually be much larger than the number of bits in the output for practical security parameters.

A randomness expander, or pseudo-random function family (PRFF), is a family of functions keyed by secret, uniformly random strings, with each function taking as input any publicly known value and outputting a value indistinguishable from one distributed uniformly at random.

When only one relatively short uniformly random key is required of a KA protocol, the output of a randomness extractor may be used as the required key. However, it is more likely that the output of the extractor will be used to key a randomness expander, to provide a longer key or multiple keys, e.g. when one random group member is used to derive a MAC (message authentication code) key for use in the KA protocol, as well as the final agreed secret key.

This step of converting a randomly chosen group member to a uniformly random string or strings of bits is often not discussed in papers proposing KA protocols. However, if the key derivation function is not modelled with the random oracle model, this step has a significant impact on how large the security parameter of the KA protocol needs to be to achieve proven security of a given level. As noted by Gennaro et al. [1, p.4] and Chevassut et al. [2, p.2], this point is often overlooked, particularly in protocol efficiency comparisons.

One reason randomness extraction and expansion and their effect on security parameter sizes is often overlooked may be the plethora of existing works that must be examined to obtain the necessary background knowledge, and the dearth numerical examples. Therefore, this paper provides:

- a summary of existing results on randomness extraction and expansion, including relevant definitions and theorems, and numerical examples,
- details of the short exponent discrete-log (DLSE) assumption and its use with randomness extraction and expansion (including numerical examples),
- an analysis of why assumptions made by Dodis et al. [3] in some justifications of the use of HMAC and cascade chaining (such as SHA) as randomness extractors are not realistic,
- a valuable resource for protocol designers and implementors to enable them to use security parameters of an appropriate size in efficiency comparisons and implementations, without having to examine all of the existing works,
- the observation, through the use of numerical examples for values of practical interest, that some of the theoretical results available are of limited practical value, due to the non-existence of underlying functions of an appropriate size or the availability of better methods,
- results for the standard model only; although use of a random oracle as a randomness extractor would mean that shorter parameters would be required in a protocol to achieve the same security level, making it more efficient, our aim is to describe solutions available for the standard model.

We will begin by examining the suitability of various candidates as randomness expanders, which will tell us how large a key needs to be provided by the randomness extractor. We will then examine randomness extractors, and the amount of entropy required for their input in order to extract a long enough key for the randomness expander.

Prior work includes that of Dodis et al. [3], the first to attempt to justify the use of CBC-MAC, cascade chaining and HMAC as randomness extractors in the standard model, and that of Gennaro et al. [1] who examined the use of universal hash functions as randomness extractors in conjunction with the DDH (decisional Diffie-Hellman) assumption and short exponents. Chevassut et al. [2] made some brief but interesting observations on randomness extraction and expansion in general, before providing methods of randomness extraction which are more efficient than those studied here, but are only applicable for groups of points over an elliptic curve (EC), and the group of prime order $q$ in $\mathbb{Z}_p^*$ where $p = 2q + 1$ and is prime. Their method for EC groups requires computations in the KA protocol to be carried out on an EC as well as its twist, instead of just on the curve, and so increases the number of computations required. However, the method may be advantageous as these computations on the EC and its twist will be in smaller groups than those necessary when using the methods studied in this paper in conjunction with computations on the EC only. Fouque et al. [4] showed that the lower order bits of a member of a subgroup of $\mathbb{Z}_p^*$ may be considered random in the right circumstances. Another work of Fouque et al. [5] examined the use of HMAC as a randomness extractor when the randomness is extracted from the HMAC key, and included an analysis of the cascade construction as a randomness extractor.

## 2 Notation and Basic Definitions

The notation mostly follows Dodis et al. [3] and Gennaro et al. [1]. For a probability distribution $\mathcal{X}$ over a set $A$, the notation $x \in_{\mathcal{X}} A$ indicates that $x$ is chosen from $A$ according to the distribution $\mathcal{X}$. The notation $x \in_{\mathrm{R}} A$ indicates that $x$ is chosen from $A$ according to the uniform distribution. $\Pr_{\mathcal{X}}[x]$ indicates the probability that distribution $\mathcal{X}$ assigns to the value $x \in A$. In some cases, definitions taken from other works have been modified to make the notation consistent.

This paper uses a concrete security approach, to allow determination of the size of the parameters needed in a protocol to achieve a given level of security. Following Gennaro et al. [1], we speak of circuits of size $S$ having a certain probability, $\epsilon$ of solving a particular problem. One may also think of a circuit of size $S$ as a programme running in time $t$, where 'time' actually includes the length of the description of the programme (to avoid trivializing hard problems through the use of large precomputed tables), as well as the actual execution time of that programme [6].

We now introduce computational indistinguishability, a refinement of the notion of statistical distance (or variation distance) from probability theory. If two distributions are statistically close, they are computationally indistinguishable, although the converse is not true [7, Sect. 3.2.2].

**Definition 1 ($(S,\epsilon)$-indistinguishability [1, p.19]).** *Let $\mathcal{X}, \mathcal{Y}$ be two probability distributions over $A$. Given a circuit $D$ (called the distinguisher) consider the following quantities:*

$$\delta_{D,\mathcal{X}} = \Pr_{x\in\mathcal{X}}[D(x) = 1] \quad \textit{and} \quad \delta_{D,\mathcal{Y}} = \Pr_{y\in\mathcal{Y}}[D(y) = 1] \tag{1}$$

*We say that the probability distributions $\mathcal{X}$ and $\mathcal{Y}$ are $(S,\epsilon)$-indistinguishable if for every circuit $D$ of size $\leq S$ we have that $|\delta_{D,\mathcal{X}} - \delta_{D,\mathcal{Y}}| \leq \epsilon$.*

**Definition 2 (Statistical Distance [8, p.131]).** *The statistical distance between two probability distributions $\mathcal{X}$ and $\mathcal{Y}$ over a set $A$ is defined to be*[1] $\Delta[\mathcal{X};\mathcal{Y}] = \frac{1}{2}\sum_{x\in A} |\Pr_{\mathcal{X}}[x] - \Pr_{\mathcal{Y}}[x]|$ .

**Lemma 1 ([3, p.500]).** *If two distributions have statistical distance of (at most) $\epsilon$, they are $\epsilon$-close. Distributions that are $\epsilon$-close cannot be distinguished with probability better than $\epsilon$ even by a computationally unbounded adversary.*

The following lemma has a proof [1] based on the triangle inequality or "hybrid argument."

**Lemma 2 ([1, p.19]).** *Let three probability distributions $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$ over a set $A$ be such that (i) $\mathcal{X}$ is $(S_1,\epsilon_1)$ indistinguishable from $\mathcal{Y}$ and (ii) $\mathcal{Y}$ is $(S_2,\epsilon_2)$ indistinguishable from $\mathcal{Z}$. Then $\mathcal{X}$ is $(S,\epsilon)$ indistinguishable from $\mathcal{Z}$ where $S = \min(S_1, S_2)$ and $\epsilon = \epsilon_1 + \epsilon_2$.*

We now focus on describing how much randomness is in a probability distribution, defining min-entropy and its computational analogue.

**Definition 3 (Min-entropy [1, p.9]).** *If $\mathcal{X}$ is a probability distribution over $A$, the min-entropy of $\mathcal{X}$ is* $\text{min-ent}(\mathcal{X}) = \min_{x\in A:\Pr_{\mathcal{X}}[x]\neq 0}(-log_2(\Pr_{\mathcal{X}}[x]))$. *(Note that if $\mathcal{X}$ has min-entropy $t$ then for all $x \in A$, $\Pr_{\mathcal{X}}[x] \leq 2^{-t}$.)*

**Definition 4 (Computational entropy $t$ [1, p.10]).** *A probability distribution $\mathcal{Y}$ has $(S,\epsilon)$ computational entropy $t$ if there exists a probability distribution $\mathcal{X}$ that is $(S,\epsilon)$ indistinguishable from $\mathcal{Y}$ and min-ent$(\mathcal{X}) \geq t$.*

**Definition 5 (Function Family [6, adapted from full paper p.7]).** *A function family $f : K \times D \to R$ (also denoted $\{f_\kappa\}_{\kappa\in K}$), where $K$ is a non-empty set of keys, is a collection of functions, $f_\kappa(\cdot) \stackrel{\text{def}}{=} f(\kappa,\cdot)$ for $\kappa \in K$, from a domain, $D$, to a range, $R$. We call $f$ a permutation family if $D = R$, and for each key $\kappa \in K$, $f_\kappa$ is a permutation on $D$.*

**Definition 6 (Truly Random Function (TRF) [5,6]).** *Denote the set of all functions from $M$ to $\{0,1\}^L$ with* $\text{Rand}^{M\to 2^L}$ *(there are $2^{L|M|}$ such functions). A function chosen at random from* $\text{Rand}^{M\to 2^L}$ *is a truly random function (TRF) with input domain $M$ and output domain $\{0,1\}^L$.*

[1] Gennaro et al.'s definition [1] is twice this value, but seems erroneous when compared with others [8,3,7] .

A TRF may be implemented by an oracle that, for each new oracle query, generates an output selected at random from $\{0,1\}^L$, and for oracle queries that are not new, replies with the same output as previously given for that input.

**Definition 7 (Cascade Construction [5]).** *The cascade construction (also known as keyed Merkle-Damgard cascade chaining) is the construction used for iterated hash functions. Let* $H : \{0,1\}^c \times \{0,1\}^* \to \{0,1\}^c$ *denote an iterated hash function, and let* $h : \{0,1\}^c \times \{0,1\}^b \to \{0,1\}^c$ *(the so-called compression function) be a family with key space* $\{0,1\}^c$*. The cascade construction of* $h$ *is the function* $h^* : \{0,1\}^c \times \left(\{0,1\}^b\right)^* \to \{0,1\}^c$ *defined by:*

$$y_0 = a, y_i = h(y_{i-1}, x_i) \text{ and } h^*(a,x) = y_n$$

*where* $x = (x_1, \ldots, x_n)$ *is a* $n \cdot b$ *bit string and* $a \in \{0,1\}^c$*. To construct* $H$*, messages must be padded to an exact multiple of* $b$ *bits. The padding, denoted* $\mathrm{pad}(|x|)$*, is a function of the input length,* $|x|$*. Let* $x_{pad} = x \parallel \mathrm{pad}(|x|)$*. Then* $H$ *is defined by* $H(a,x) = h^*(a, x_{pad})$*.*

*Let* $1 \leq c' \leq c$ *be an integer and let* $\mathrm{msb}_{c'}(\cdot)$ *denote the* $c'$ *most significant bits of a bit string. For any function* $H$ *with range* $\{0,1\}^c$*, we define for every input* $x$ *the truncated iterated hash function* $\tilde{H}(x) = \mathrm{msb}_{c'}(H(x))$*; e.g. SHA-384 has* $c' = 384$ *and* $c = 512$*.*

**Definition 8 (NMAC [5]).** $\mathrm{Nmac} : \{0,1\}^c \times \{0,1\}^c \times \{0,1\}^* \to \{0,1\}^{c'}$ *is a hash function family constructed from a (possibly truncated) iterated hash function* $\mathrm{Hash} : \{0,1\}^c \times \{0,1\}^* \to \{0,1\}^{c'}$*. If* $(k_1, k_2) \in (\{0,1\}^c)^2$ *is a couple of keys and* $x \in \{0,1\}^*$ *is the input, the definition of NMAC is* $\mathrm{Nmac}^{\mathrm{Hash}}(k_1, k_2, x) = \mathrm{Hash}(k_2, \mathrm{Hash}(k_1, x))$*.*

**Definition 9 (HMAC [5]).** *HMAC is a hash function from* $\{0,1\}^* \times \{0,1\}^*$ *to* $\{0,1\}^{c'}$*. Let ipad and opad be two b-bit strings and IV be a c-bit string. Let Hash* $: \{0,1\}^c \times \{0,1\}^* \to \{0,1\}^{c'}$ *be the (possibly truncated) iterated hash function with compression function* $h : \{0,1\}^c \times \{0,1\}^b \to \{0,1\}^c$*. If the key* $k$ *is a bit string from* $\{0,1\}^b$*, then*

$$\begin{aligned} \mathrm{Hmac}_{IV}^{\mathrm{Hash}}(ipad, opad; k, x) &= \mathrm{Hash}\left(IV, [k \oplus opad] \parallel \mathrm{Hash}\left(IV, [k \oplus ipad] \parallel x\right)\right) \\ &= \mathrm{Nmac}^{\mathrm{Hash}}(h(IV, k \oplus ipad), h(IV, k \oplus opad), x). \end{aligned}$$

*If the key* $k$ *is smaller than* $b$ *bits, then it is first padded with '0' bits to form a* $b$*-bit string, and this string is used as the key. If the key* $k$ *is larger than* $b$ *bits, it is first hashed using* Hash *to obtain a* $c'$*-bit digest, then padded with* $b - c'$ *'0' bits to obtain a* $b$*-bit string, which is then used as the key.*

## 3 Randomness Expansion

To ascertain the minimum output length required from the randomness extractor used, we begin by examining the randomness expander—also known as a pseudorandom function (PRF) family, or PRFF—to be used, since the output of the randomness extractor will be used as the key to the PRFF.

**Definition 10 (Pseudorandom Function Family [9,6]).** *A function family $f = \{f_\kappa\}_{\kappa \in K}$ is a $(S, q, \epsilon)$ pseudorandom function family (PRFF) if a circuit, $\mathcal{A}$, of size $S$ which is given oracle access to either $f_\kappa$ for $\kappa \in_R K$ or a TRF with the same domain and range as the functions in $f$, and makes at most $q$ queries to this oracle, has advantage at most $\epsilon$ in distinguishing whether it has access to a random member of $f$ or a TRF; i.e.:*

$$\epsilon \geq \mathbf{Adv}_f^{\mathrm{prf}}(q, S) \stackrel{\mathrm{def}}{=} \max_{\mathcal{A}} \left\{ \mathbf{Adv}_f^{\mathrm{prf}}(\mathcal{A}) \right\} \tag{2}$$

$$\mathbf{Adv}_f^{\mathrm{prf}}(\mathcal{A}) \stackrel{\mathrm{def}}{=} \left| \Pr[\mathcal{A}^{\mathcal{O}(\cdot)} = 1 | \mathcal{O}(\cdot) \in_{\mathrm{R}} f] - \Pr[\mathcal{A}^{\mathcal{O}(\cdot)} = 1 | \mathcal{O}(\cdot) \in_{\mathrm{R}} \mathrm{Rand}] \right| \tag{3}$$

The values $\mathbf{Adv}_f^{\mathrm{prp}}(q, S)$ and $\mathbf{Adv}_f^{\mathrm{prp}}(\mathcal{A})$, may be defined similarly for an adversary $\mathcal{A}$ against a pseudorandom permutation family, except that $\mathcal{A}$ attempts to tell the difference between the permutation family and a truly random permutation, rather than a TRF.

When one PRFF is used with various different keys (e.g. each party from a number of parties may use its own key to produce pseudorandom values from the PRFF), there is a linear decrease in security. Furthermore, the key to a PRFF may be only computationally indistinguishable from random, in which case the level of security of the PRFF and the level computational indistinguishability must be combined. This is formally stated and proven in the full version.

Function families widely believed to be pseudorandom include CBC-MAC used in conjunction with a block cipher, HMAC or the HMAC variant NMAC, and cryptographic hash functions such as SHA-1 or SHA-256 based on the cascade construction, but with the fixed IV (initialization vector) replaced with a random key. The full paper discusses the merits of each of these options in turn. Here we overview the security levels provided by each option. Some assumptions (described in the full paper) must be made on the security level of the underlying block ciphers or compression functions to arrive at the below concrete security levels.

### 3.1 CBC-MAC

Bellare et al. [6] have proved that CBC-MAC is a secure PRFF if the underlying block cipher is a secure pseudorandom permutation family and the input length is constant. The level of security provided depends on the block length, number of queries, $q$, and number of blocks of input, $l$. When $ql$ is small (e.g. 2), the security level is about $k = b - 3$ bits. Otherwise, if we have $ql \leq 2^k$ (which we are assuming when we consider a security level of $2^k$ sufficient), then we will require $k \leq (b-2)/2$. If the block cipher to be used with CBC-MAC is AES-128, AES-192, or AES-256, then the block length, $b$, will be 128 bits for each of these ciphers [10]. Therefore, the level of security provided by CBC-MAC when used in conjunction with any of these ciphers will be no greater than 125 bits, and will be less for values of $q$ and $l$ larger than 1. Hence, CBC-MAC is likely to be an acceptable choice of randomness expander for security levels of 80 bits if the number of queries to randomness expander with a single key is small and the

length of each query is also small, but inadequate for security levels of 128 bits and higher. If an unlimited number of queries or queries with a very large length are able to be made by the adversary to the randomness expander with a single key, the security level will only be $(b-2)/2 = 63$ bits when $b = 128$.

### 3.2 HMAC

Bellare [11] has proven that HMAC is a secure pseudorandom function if the compression function of the underlying hash function is a pseudorandom function. The analysis assumes that the key provided to HMAC is the same length as a block for the underlying hash function (i.e. $b$ bits). To achieve a shorter key of only $2c$ bits (where $c$ is the length of the output of the compression function), NMAC may be used, which is similar to HMAC but differs in its use of keying material. However, NMAC is generally used for analysis of HMAC only, so availability of an existing implementation is unlikely. Any implementation of NMAC will require access to the compression function underlying the hash function to be used, which may be difficult to acquire.

Hash functions likely to be used with HMAC include MD5 [12], RIPEMD-160 [13], SHA-1, SHA-256, SHA-384 and SHA-512 [14]. Table 1 shows the block size ($b$), compression function key and output length ($c$), hash function output length ($c'$) and HMAC security level for each of these algorithms, where $q$ is the number of queries using the same key and $l$ is the number of blocks per query. The traditional security level is $c/2$ bits, due to the birthday based forgery attacks against iterated MACs [15] that require $2^{c/2}$ oracle queries.

### 3.3 Cascade Construction

Bellare, Canetti and Krawczyk [9] have provided a proof of pseudorandom function family security for cryptographic hash functions such as SHA-1 or SHA-256 based on the cascade construction, but with the fixed IV (initialization vector) replaced with a random key, provided the input is prefix-free and the underlying compression function used by the hash function is a pseudorandom function family. (It is possible to remove the prefix-free requirement by using extra keying

**Table 1.** Block and key size, output length, and hash and HMAC security level

| Algorithm | $b$ | $c'$ | $c$ | Security level ($q, l \leq 2$) | | Security level ($q$ is large) | | |
|---|---|---|---|---|---|---|---|---|
| | | | | for Hash ($c-2$) | for HMAC ($c-4$) | max. for Hash ($\frac{c-20}{2}$) | conservative Hash ($\frac{c-40}{3}$) | HMAC ($\frac{c-2}{2}$) |
| MD5 | 512 | 128 | 128 | 126 | 124 | 54 | 29 | 63 |
| RIPEMD-160 | 512 | 160 | 160 | 158 | 156 | 70 | 40 | 79 |
| SHA-1 | 512 | 160 | 160 | 158 | 156 | 70 | 40 | 79 |
| SHA-224 | 512 | 256 | 224 | 254 | 252 | 118 | 72 | 127 |
| SHA-256 | 512 | 256 | 256 | 254 | 252 | 118 | 72 | 127 |
| SHA-384 | 1024 | 512 | 384 | 510 | 508 | 246 | 157 | 255 |
| SHA-512 | 1024 | 512 | 512 | 510 | 508 | 246 | 157 | 255 |

**Table 2.** Summary of required key lengths for a given security level when $q$ large

| Security level (bits) | Key Length | | | | | Security level (bits) | Key Length | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | CBC-MAC | Casc. min. | Casc. consrv. | NMAC | HMAC | | Casc. min. | Casc. consrv. | NMAC | HMAC |
| 29 | | | 128 | | | 79 | | | 320 | 512 |
| 40 | | | 160 | | | 118 | 256 | | | |
| 54 | | 128 | | | | 127 | | | 512 | 512 |
| 63 | 128 | | | 256 | 512 | 157 | | 512 | | |
| 70 | | 160 | | | | 246 | 512 | | | |
| 72 | | | 256 | | | 255 | | | 1024 | 1024 |

material, but it is unlikely to be necessary in our setting. Belare et al. provided another construction to improve security using randomization, but if the extra randomness is counted as part of the key, more keying material than HMAC is required for a similar security level.)

Table 1 shows the security level of the cascade construction using the same notation as for HMAC. Assumptions made to obtain the security levels are described in the full paper. The difference between the maximum and conservative security levels for large $q$ is due to different assumptions concerning the efficiency of the best attack against the underlying compression function.

### 3.4 Key Length Summary

In summary, when $q \leq 2$, a minimum of 128 bits will be needed to key the randomness expander, e.g. using CBC-MAC or the cascade construction, achieving a security level around 125 bits. In this case, the cascade construction allows the use of a key about two bits longer than the required security level, and requires fewer key bits than using NMAC or HMAC for the same security level.

When there is no restriction on $q$, the cascade construction provides the lowest key length for a given security level when we take the security level as being $\frac{c-20}{2}$. However, if the more conservative security level of $\frac{c-40}{3}$ bits is used, then NMAC may be better, depending on the level of security required. Table 2 summarizes the results.

## 4 Randomness Extraction

Let us consider a KA protocol that allows the participating parties to agree upon a secret value, called the pre-secret, that an adversary cannot distinguish from a value drawn uniformly at random from a particular distribution, e.g. from a group in which the DDH (Decisional Diffie-Hellman) assumption holds. Furthermore, assume a randomness extractor and expander are used to derive a final key from the pre-secret, such that the final key is indistinguishable from a uniformly random bit string. As will be seen in this section, when using the techniques of randomness extraction and expansion considered in this paper, the entropy of the pre-secret must be much larger than the security level required of the final

key. Therefore, if the pre-secret is from a suitable group, it may seem desirable to use the discrete-log short-exponent (DLSE) assumption to enable calculations required by the KA protocol to be more efficient, by using exponents shorter than the group order. In addition, if the KA protocol is Diffie-Hellman (DH) based, it may be desirable to use the $t$-DDH assumption (a relaxation of the DDH assumption) to allow the use of groups with non-prime order with the protocol. These assumptions and theorems are therefore provided in the full paper. Note that two theorems of Gennaro et al. [1] regarding use of the DLSE assumption are incorrect in their original paper and have been corrected in the full paper according to details supplied by Gennaro in a personal communication.

The most common existing randomness extractor definition is of a strong randomness extractor:

**Definition 11 (Strong randomness extractor [16]).** *A family of efficiently computable hash functions $H = \{h_\kappa : \{0,1\}^n \to \{0,1\}^c | \kappa \in \{0,1\}^d\}$ is called a $(t, \epsilon)$ strong randomness extractor, if for any random variable $X$ over $\{0,1\}^n$ that has min-entropy at least $t$, if $\kappa$ is chosen uniformly at random from $\{0,1\}^d$ and $R$ is chosen uniformly at random from $\{0,1\}^c$, the following two distributions are within statistical distance $\epsilon$ from each other: $(\kappa, h_\kappa(X)) \cong_\epsilon (\kappa, R)$.*

By Lemma 1, the above distributions are also computationally indistinguishable from each other. Notice that the definition means that the key to the randomness extractor, $\kappa$, may be made public, yet the output of the randomness extractor, given a secret input with sufficient min-entropy, is indistinguishable from a string of bits distributed uniformly at random.

Since it is likely that $X$ will only have computational entropy (not min-entropy) of a certain level, we introduce the following definition (which is similar to a recent definition of Fouque et al. [5] in an independent work).

**Definition 12 (Strong computational randomness extractor).** *A family of efficiently computable functions $H = \{h_\kappa : A \to \{0,1\}^c | \kappa \in \{0,1\}^d\}$ is a $(t, S, \epsilon, S', \epsilon')$ strong computational randomness extractor if given any probability distribution $\mathcal{X}$ over $A$ such that $\mathcal{X}$ has $(S, \epsilon)$ computational entropy at least $t$, the following two probability distributions are $(S', \epsilon')$-indistinguishable:*

$$\mathcal{H} = \{(\kappa, h_\kappa(x)) \text{ for } \kappa \in_{\mathrm{R}} \{0,1\}^d \text{ and } x \in_{\mathcal{X}} A\} \tag{4}$$

$$\mathcal{R}^{\mathrm{h}} = \{(\kappa, r) \text{ for } \kappa \in_{\mathrm{R}} \{0,1\}^d \text{ and } r \in_{\mathrm{R}} \{0,1\}^c\} \tag{5}$$

It is possible to show that a strong randomness extractor is also a strong computational randomness extractor (see the full paper). However, the converse is not necessarily true.

The above definitions assume that the key to the randomness extractor, $\kappa$, is generated afresh for each use of the randomness extractor. This may be appropriate in some protocols, where parties may have exchanged nonces with each other and can use these values to generate the key. However, it is imperative that any such nonces be authenticated (i.e. unable to be influenced by the adversary)

and not subject to replay by the adversary. Otherwise, a key derived from these nonces may not be distributed uniformly at random over $\{0,1\}^d$ as required for these extractors.

When parties are unable to generate a new key, $\kappa$, each time they use a randomness extractor, the key $\kappa$ may be fixed as part of the system parameters. However, this requires multiple uses of the randomness extractor with the one key. It turns out that the security of the randomness extractor decreases linearly with the number of queries to it using the same key. Protocols using this approach may be proven secure in one of two ways. As part of the proof of security of the protocol, one often focuses on the security of one particular session chosen at random from all sessions. In the proof, it may be possible to use the above definitions to prove the security of the protocol. The total number of sessions will appear as a factor in the security reduction (due to focusing on one session chosen at random from all sessions), and this will cater for the reduction in security due to multiple uses of the extractor with only one key. The other way to justify the use of a single key to the randomness extractor is via the theorems given in the full paper.

### 4.1 Combining Extraction and Expansion

To ascertain the security of the overall key derivation function consisting of randomness extraction and expansion, all of the relevant theorems from the full paper must be combined (e.g. extractor reuse, Diffie-Hellman assumption, short exponent theorems, expander reuse etc.). The full paper provides an example combination of theorems which is summarized here.

Let $H = \{h_\kappa : \{0,1\}^n \rightarrow \{0,1\}^c | \kappa \in \{0,1\}^d\}$ be a $(t, 2^{-e})$ strong randomness extractor, with a maximum of $q_1$ queries per (publicly known) randomness extractor key $\kappa$, and let $f = \{f_\lambda\}_{\lambda \in K}$ be a $(S_5, q_2, \epsilon_5)$ PRFF, with a maximum of $q_2$ queries per (secret) key $\lambda$. Suppose a security level of $k$ bits is desired for the final key(s) output by $f$. Let $G$ be a cyclic group of order $m$ generated by $g$, such that $m$ is odd, or $m/2$ is odd. We assume there are $q_1$ publicly known pairs $g^{a_i}, g^{b_i}$ for $1 \leq i \leq q_1$, and that the $g^{a_i b_i}$ are the inputs to $h_\kappa(\cdot)$.

We consider two cases. For the first, we require $q_1 - 1$ outputs of $H$ to be indistinguishable from random, use the other output of $H$ to key $f$, and require the $q_2$ outputs of $f$ using this key to be indistinguishable from random. The indistinguishable distributions are labelled $\mathcal{EEDH}$ and $\mathcal{EER}$.

In the second case, all $q_1$ outputs of $H$ are used to key $f$, giving a total of $q_1 q_2$ outputs of $f$, and all of these outputs must be indistinguishable from random. The indistinguishable distributions are labelled $\mathcal{EEDH}^*$ and $\mathcal{EER}^*$. Which of these cases is appropriate will depend upon the protocol in question and its proof of security. Table 3 shows the requirements in each case, where the distributions are to be indistinguishable with a security level of $k$ bits.

As an example putting it all together, suppose that a security level of $k = 80$ bits is required, we desire that the $\mathcal{EEDH}$ and $\mathcal{EER}$ distributions are indistinguishable, $q_1 = 1$ and $q_2 = 1$. Furthermore, suppose that $m$ is prime. Then we need:

**Table 3.** Requirements for the two cases to be indistinguishable from random

| For $\mathcal{EEDH}$ and $\mathcal{EER}$ indistinguishable: | For $\mathcal{EEDH}^*$ and $\mathcal{EER}^*$ indistinguishable: |
|---|---|
| – $\frac{S_5}{\epsilon_5} \geq 2^{k+1}$ | – $\frac{S_5 - S_8}{\epsilon_5} \geq 2^{k+1} q_1$ where $S_8 \approx (q_1 - 1) q_2$ |
| – $e \geq k + 2 + \log_2(q_1)$ | – $e \geq k + 2 + 2\log_2(q_1)$ |
| – $t$-DDH assumptions: $\left(2^{k+4} q_1 + q_1 + q_2, \frac{1}{2}\right)$ and $\left(q_1 + q_2 + 1, \frac{1}{2^{k+3} q_1}\right)$ | – $t$-DDH assumptions: $\left(2^{k+4} q_1^2 + q_1 q_2, \frac{1}{2}\right)$ and $\left(q_1 q_2 + 1, \frac{1}{2^{k+3} q_1^2}\right)$ |
| – $s$-DLSE assumptions: $\left(2^{i-1} s \ln(2s)(Y + 2Z), \frac{1}{2}\right)$ and $\left(Y^i s \ln(s)(Z+1), \frac{1}{Y}\right)$ where $Y \stackrel{\text{def}}{=} (\log_2(m) - s) 2^{k+5} q_1$, $Z \stackrel{\text{def}}{=} S_3 + q_1 + q_2$. | – $s$-DLSE assumptions: $\left(2^{i-1} s \ln(2s)(Y + 2Z), \frac{1}{2}\right)$ and $\left(Y^i s \ln(s)(Z+1), \frac{1}{Y}\right)$ where $Y \stackrel{\text{def}}{=} (\log_2(m) - s) 2^{k+5} q_1^2$, $Z \stackrel{\text{def}}{=} S_3 + q_1 q_2$. |

In both cases $i = 3$ unless $\log_2(m) > 2s - \log_2(\epsilon_1)$, in which case $i = 2$ and the smallest sensible value for $\epsilon_1$ is $\frac{1}{Y}$. $S_3$ is the cost of a multi-exponentiation in $G$.

- a randomness expander with an 81 bit security level, e.g. CBC-MAC with a 128 bit key for its block cipher or MD5 with a 128 bit key;
- a $(t, 2^{-82})$ strong randomness extractor for some $t$ that outputs enough bits to key the randomness expander, e.g. a universal hash function—in that case $t = 292$ (see Sect. 4.2);
- $\left(2^{84}, \frac{1}{2}\right)$ and $\left(3, \frac{1}{2^{83}}\right)$ $t$-DDH assumptions on $G$, e.g. $G$ could be of order 292 bits on an elliptic curve (292 is the maximum of $t = 292$ and $2 \cdot 85$);
- exponents of the full 292 bits since the short exponent assumption needs the short exponent to be longer than 292 bits (probably around 600 bits).

Further details of the calculations are provided in the full paper. This example contradicts the statement by Gennaro et al. [1, Sect. 6] that exponents of length $2k$ may be used to achieve a security level of $k$ bits, since in our example, we need the exponent to be of length between $5k$ and $7.4k$. It seems that Gennaro et al. have not substituted actual values into their theorem stating that short exponents may be used, and have thus come to an incorrect conclusion about how long the short exponents really need to be.

### 4.2 Available Extractors

We now compare the available randomness extractors, focusing on output lengths of 128, 160, 256 and 512 bits, as these are the possible key lengths for the randomness expanders in Sect. 3.4. The reader may make his own comparisons for other output lengths with the information provided.

We first discuss the use of the Leftover Hash Lemma (LHL) to show that a universal (or almost universal) hash function may be used as a randomness extractor. Following this, we discuss the use of a PRFF as a randomness extractor, as analysed by Chevassut et al. [2], and then summarize the results of Fouque et al. [4] on deterministic extraction of lower order bits from subgroups

of $\mathbb{Z}_p^*$. Then another work of Fouque et al. [5] is summarized with several results on using HMAC to extract randomness from the HMAC key, and a result on using the cascade construction as a randomness extractor. The full paper provides an overview and detailed comments on the problems with the first work [3] to consider the suitability of CBC-MAC, the cascade construction, and HMAC for use as randomness extractors in the standard model.

We aim for the output of the extractor to be $(S', \epsilon')$ indistinguishable from uniform with $\frac{S'}{\epsilon'} \geq 2^{81}$ as a minimum requirement (this will achieve a security level no greater than $k = 80$ bits when the randomness extractor and expander are used together). Table 3 will provide the basis for our numerical analysis of the advantages of each extractor. We will use the notation of Sect. 4.1 and assume (as was done there) that $S_4 \approx q_1$, $S_6 \approx q_2$ and $S_8 \approx (q_1 - 1)q_2$. Furthermore, we let $c$ be the key length of the expander, and hence the output length of the extractor; $t$ be the min-entropy, $b$ be the block size and $L$ be the number of blocks of the pre-secret ($ps$, e.g. the DH value) which is input to the extractor. We will examine the parameters required of each extractor to achieve various security levels in the following cases (notation is as in Sect. 4.1). In our examples, we use the cascade construction as the expander, since it is the best (see Sect. 3.4). The parameters required to achieve other security levels or in other cases can be derived by the reader.

1. Each extractor key is used only once ($q_1 = 1$; this would be the case if the key is chosen afresh in each protocol run); the expander is used only once or twice with each key ($q_2 \leq 2$); it is desired that $\mathcal{EEDH}$ and $\mathcal{EER}$ are indistinguishable (the KA protocol's security will be lower than $k$ bits, since the total number of sessions will appear as a factor in its security reduction).
2. The extractor key is a global parameter used up to $2^{30}$ times ($q_1 \leq 2^{30}$); other requirements are as for the previous case; e.g. many other applications use the extractor at a $k$-bit security level; the KA protocol proof focuses on one session; that session's two keys (output by the expander) have $k$ bits of security (again, the protocol's overall security will be lower than $k$ bits).
3. Each extractor key is used once ($q_1 = 1$); the expander is used many times with each key ($q_2 > 2$); other requirements are the same as for the first case.
4. The extractor key is the same in all KA protocol sessions (but not used in other applications), and there are up to $2^{30}$ sessions ($q_1 \leq 2^{30}$); the expander is used many times with each key ($q_2 > 2$); $\mathcal{EEDH}^*$ and $\mathcal{EER}^*$ must be indistinguishable (so the number of sessions will not be an extra factor in the protocol proof). We assume $S_8^2 \approx q_1^2 q_2^2 \leq 2^{k+1} q_1$ so that a cascade construction security level of $k + 1 + \log_2(q_1)$ bits (less conservative option) gives $\frac{S_5 - S_8}{\epsilon_5} \geq 2^{k+1} q_1$.[2]

[2] We want $(S_5 - S_8)/\epsilon_5 \geq 2^{k+1} q_1$. When using the cascade construction (less conservative option) we have $1/\epsilon_5 \geq 2^c / \left(2^{20} S_5^2\right)$ (see the comments in the full paper), so we need $\left((S_5 - S_8)\, 2^c\right) / \left(2^{20} S_5^2\right) \geq 2^{k+1} q_1$ where $c$ is the key length of the randomness extractor. When $S_5 = S_8 + 1$, we have $\left((S_5 - S_8)\, 2^c\right) / \left(2^{20} S_5^2\right) \geq 2^{k+1} q_1$ implies $2^c \geq 2^{k+21} q_1 S_8^2$. However, for a security level of $s$ bits for the randomness expander, we require $2^c \geq 2^{2s+20}$, and if $s = k + 1 + \log_2(q_1)$, this will imply the first requirement when $s \geq 2\log_2(S_8)$. For values of $S_5$ much larger than $S_8$, $S_5 - S_8 \approx S_5$ and so a security level of $k + 1 + \log_2(q_1)$ bits will be sufficient.

**Almost Universal Hash Functions.** The Leftover Hash Lemma (LHL) is well-known and allows the use of a universal (or almost universal) hash function as an extractor which is probabilistic and optimal in general [2]. There are several variations of the LHL in the literature; the one provided is mainly from Chevassut et al. [2], and similar to Dodis et al. [3, p.501].

**Definition 13 ($\delta$-AU (almost universal)).** *Let $c$ and $b$ be integers, and $\{h_\kappa\}_{\kappa\in\mathcal{K}}$ be a family of hash functions with domain $\{0,1\}^b$, range $\{0,1\}^c$ and key space $\mathcal{K}$. We say that the family $\{h_\kappa\}_{\kappa\in\mathcal{K}}$ is $\delta$-almost universal ($\delta$-AU)[3] if for every pair of different inputs $x$, $y$ from $\{0,1\}^b$ it holds that $\Pr(h_\kappa(x) = h_\kappa(y)) \leq \delta$, where the probability is taken over $\kappa \in_{\mathrm{R}} \mathcal{K}$. For a given probability distribution $\mathcal{X}$ on $\{0,1\}^b$, we say that $\{h_\kappa\}_{\kappa\in\mathcal{K}}$ is $\delta$-AU w.r.t. $\mathcal{X}$ if $\Pr(h_\kappa(x) = h_\kappa(y)) \leq \delta$ where the probability is taken over $\kappa \in_{\mathrm{R}} \mathcal{K}$ and $x, y \in_{\mathrm{R}} \mathcal{X}$ conditioned on $x \neq y$.*

An example of a universal hash function is the function that multiplies a Toeplitz matrix (one with constant diagonals) by the input to create the output [17]. The full paper gives more details and examples of universal hash functions.

**Lemma 3 (LHL with $\delta$-AU [2]).** *Let $\mathcal{X}$ be a probabilistic distribution over $\{0,1\}^b$ with min-entropy at least $t$. Let $e$ be an integer and $c \leq \alpha - 2e$ where $\alpha = \min(t, \log_2(1/\xi))$. Let $\mathcal{H} = \{h_\kappa\}_{\kappa\in\mathcal{K}}$, with $h_\kappa$ having domain $\{0,1\}^b$ and range $\{0,1\}^c$ for any $\kappa \in \mathcal{K}$, be a $\delta$-AU hash function family with $\delta = \frac{1}{2^c} + \xi$. Let $H$ be a random variable uniformly distributed on $\mathcal{H}$, $X$ denote a random variable taking values in $\{0,1\}^b$, and $H$ and $X$ be independent. Then, $(H, H(X))$ is $2^{-e}$-uniform on $\mathcal{H} \times \{0,1\}^c$.*

This lemma states that a $\delta$-almost universal hash function is a $(t, 2^{-e})$ strong randomness extractor. It was used to generate Table 4, where we must have $\xi \leq 2^{-t}$. It shows that even the most basic requirements mean a computational entropy of 292 bits in the input to the randomness extractor. More realistic requirements may mean a much higher level of computational entropy is required. Because of their significant key size requirements, and because other functions such as cryptographic hash functions are more readily available, universal hash functions are often not used for key derivation.

**PRFFs as Randomness Extractors.** Chevassut et al. [2] have shown that a PRFF may be used for randomness extraction with a publicly known key.

**Theorem 1 ([2]).** *If a family of functions, $\mathcal{F}$, is a $(S, 2, \xi)$-PRFF with domain $\{0,1\}^b$ and range $\{0,1\}^c$, $S$ is the size of a circuit that makes 2 oracle queries on an instance of $\mathcal{F}$, then it is a $(\frac{1}{2^c} + \xi)$-AU hash function family.*

By using Lemma 3, we can conclude that a PRFF can be a strong randomness extractor, although the output of the PRF will generally need to be truncated to

[3] Being $\delta$-AU in Dodis et al. [3] is the same as being $\xi$-AUH in Chevassut et al. [2] for $\delta = \frac{1}{2^c} + \xi$ where $c$ is the number of bits of output of the function. We use the notation of Dodis et al. in this paper. When $\delta = \frac{1}{2^c}$, the function is universal.

**Table 4.** Universal hash function parameter examples

| Case | $t$ | $k$ | $e$ | $c$ | Case | $t$ | $k$ | $e$ | $c$ |
|---|---|---|---|---|---|---|---|---|---|
| 1 | $c+2e$ | $k$ | $k+2$ | $\geq (k+2)+2$ | 3 | $c+2e$ | $k$ | $k+2$ | $\geq 2(k+2)+20$ |
| 1 | 292 | 80 | 82 | 128 | 3 | 420 | 80 | 82 | 256 |
| 1 | 380 | 124 | 126 | 128 | 3 | 492 | 116 | 118 | 256 |
| 1 | 476 | 156 | 158 | 160 | 3 | 900 | 192 | 194 | 512 |
| 1 | 764 | 252 | 254 | 256 | 3 | 1004 | 244 | 246 | 512 |
| 1 | 1532 | 508 | 510 | 512 | | | | | |
| 2 | $c+2e$ | $k$ | $k+32$ | $\geq (k+32)+2$ | 4 | $c+2e$ | $k$ | $k+62$ | $\geq 2(k+32)+20$ |
| 2 | 352 | 80 | 112 | 128 | 4 | 540 | 80 | 142 | 256 |
| 2 | 476 | 126 | 158 | 160 | 4 | 552 | 86 | 148 | 256 |
| 2 | 704 | 192 | 224 | 256 | 4 | 956 | 160 | 222 | 512 |
| 2 | 764 | 222 | 254 | 256 | 4 | 1020 | 192 | 254 | 512 |
| 2 | 1532 | 478 | 510 | 512 | 4 | 1064 | 214 | 276 | 512 |

a length compatible with Lemma 3. Reuse of the extractor can then be covered by one of the theorems from the full paper. For example, to achieve a security level of $k = 80$ bits in Case 1, as shown in Table 4, we will need $\xi < 2^{-292}$. This rules out the use of CBC-MAC, since the block size is only likely to be 128 bits, and so the security level will only be about 125 bits. The use of HMAC or the cascade construction seems appropriate, provided we do not need $\xi$ smaller than $2^{-508}$ or $2^{-510}$ respectively. In our example, we could use SHA-384 or better, and would need to truncate the output to 128 bits.

**Deterministic Extraction of Lower Order Bits.** The analysis of Fouque et al. [4] allows one to use the lower or higher-order bits from subgroups of $\mathbb{Z}_p^*$.

**Theorem 2.** *Let $p$ be a $b$-bit prime, that is $2^{b-1} < p < 2^b$, $G$ a subgroup of $\mathbb{Z}_p^*$ of order $q$ with $q \gg \sqrt{p}$, $l$ the integer such that $2^{l-1} \leq q \leq 2^l$ and $X$ a random variable uniformly distributed in $G$. Let $\mathrm{lsb}_c(X)$ denote the $c$ least significant bits of $X$. Let $e$ be a positive integer and let $l > t = b/2 + c + e + \log_2(b) + 1$. Then the function $\mathrm{lsb}_c(\cdot)$ is a $(t, 2^{-e})$-deterministic extractor for the $G$-group distribution. If $p^{1/2} \leq q \leq p^{2/3}$ then the requirement on $l$ may be refined to $l > t = b/4 + 3l/8 + c + e + \log_2(b) + 3$, and if $256 \leq q \leq p^{1/2}$, it may be refined to $l > t = b/8 + 5l/8 + c + e + \log_2(b) + 3$. Let $\mathrm{msb}_c(X)$ denote the $c$ most significant bits of $X$ and let $\delta = (2^n - p)/2^n$. If $3\delta < 2^{-e-1}$ and $l > t = n/2 + k + e + \log_2(n) + 1$, then $\mathrm{msb}_c(\cdot)$ is a $(t, 2^{-e})$-deterministic extractor.*

Table 5 shows some parameter examples using Theorem 2 with the four cases under consideration. Comparing it with Table 4, we can see that more computational entropy is generally required than when using a universal hash function. Fouque et al. recommended the use of the DLSE assumption to shorten the exponents required and thus improve efficiency. However, Sect. 4.1 indicates that much more than $2e$ bits will be required, contrary the indication of Fouque et al. (summarizing Gennaro et al.'s work [1]). However, one advantage of this method is that it is deterministic, and so does not require a key for the extractor.

**Table 5.** Parameter examples for least significant bits extraction

| Case | $b$ | $t$ | $k$ | $e$ | $c$ | Case | $b$ | $t$ | $k$ | $e$ | $c$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | | $k$ | $k+2$ | $\geq k+4$ | 3 | | | $k$ | $k+2$ | $\geq 2(k+2)+20$ |
| 1 | 1024 | 733 | 80 | 82 | 128 | 3 | 1024 | 861 | 80 | 82 | 256 |
| 1 | 2048 | 1178 | 80 | 82 | 128 | 3 | 1024 | 897 | 116 | 118 | 256 |
| 1 | 1024 | 777 | 124 | 126 | 128 | 3 | 2048 | 1742 | 192 | 194 | 512 |
| 1 | 1024 | 841 | 156 | 158 | 160 | 3 | 2048 | 1794 | 244 | 246 | 512 |
| 1 | 2048 | 1546 | 252 | 254 | 256 | | | | | | |
| 1 | 2048 | 2058 | 508 | 510 | 512 | | | | | | |
| 2 | | | $k$ | $k+32$ | $\geq k+34$ | 4 | | | $k$ | $k+62$ | $\geq 2(k+32)+20$ |
| 2 | 1024 | 763 | 80 | 112 | 128 | 4 | 1024 | 921 | 80 | 142 | 256 |
| 2 | 1024 | 841 | 126 | 158 | 160 | 4 | 1024 | 927 | 86 | 148 | 256 |
| 2 | 1024 | 1003 | 192 | 224 | 256 | 4 | 2048 | 1770 | 160 | 222 | 512 |
| 2 | 2048 | 1546 | 222 | 254 | 256 | 4 | 2048 | 1802 | 192 | 254 | 512 |
| 2 | 2112 | 2091 | 478 | 510 | 512 | 4 | 2048 | 1824 | 214 | 276 | 512 |

**HMAC.** Fouque et al. [5] have analysed the security of HMAC as a randomness extractor when the data from which the randomness is to be extracted (pre-secret, *ps*) is used as the key of HMAC. Because the pre-secret is used as the HMAC key, some other data (denoted *label*, of at most $l$ blocks), which is possibly adversarily generated, is used as the input to HMAC. There are two separate results, depending on whether the pre-secret is longer than one block or not.

**Theorem 3 ([5]).** *Using the notation of this section and Definition 9, let $L = 1$, let ipad and opad be chosen uniformly at random and let IV be a fixed string. Let $h'$ be the hash function defined by $h'_{IV}(pad, \cdot) = h(IV, \cdot \oplus pad)$ where the key is pad. Let $S_h$ be the circuit size for one computation of $h$. Let $h'$ be a $(S'+2S_h, q = 2, \epsilon_1)$ PRFF, and $h$ be both a $(S', q = 1, \epsilon_2)$ and $(O(l \cdot S_h), q = 2, \epsilon_3)$ PRFF. Then $\mathrm{Hmac}^{\mathrm{Hash}}_{IV}(ipad, opad; ps, label)$ is a $(t, \infty, 0, S', \epsilon')$ computational randomness extractor with $\epsilon' \leq \frac{\sqrt{2^{2c}(2^{-t}+2\epsilon_1)}}{2} + \frac{1}{2^{c'}} + \epsilon_2 + 2l\epsilon_3$.*

This is only useful if $b \gg 2c$, since $L = 1$ implies $t \leq b$ and when $t = 2c$ the term under the square root is at least one. In the case of SHA-1, we have $b = 512$ and $c = c' = 160$. To achieve a security level of $e$ bits for the output of HMAC, we want $S'/\epsilon' \geq 2^e$. If we assume $\epsilon_1 \leq (S' + 2S_h)/(S_h 2^b)$, $\epsilon_2 \leq S'/(S_h 2^c)$, $\epsilon_3 \leq lS_h/(S_h 2^c)$, and $l \ll 2^c$, and consider the case where $S' = S_h = 1$, we require $e \leq \min\left(\frac{t-2c+1}{2}, \frac{b-2c+1.6}{2}, c', c - 2\log_2(l) - 1\right)$. These conditions will also ensure that the conditions placed on $e$ when $S' = 2^{e-1}$, $S_h = 1$ and we want $\epsilon' \leq \frac{1}{2}$, are met. Hence, when $t = 512$, we achieve the maximum security level of $e = 96$ bits; for $e = 82$ bits, we need $t = 483$ bits min-entropy.

To overcome the problem of the above theorem only being useful when $b \gg 2c$, the assumptions on the compression function can be modified. That is, it is assumed that $h$ is a PRFF resistant to related key attacks (RKA) when it is keyed with a bit string of min-entropy at least $t$ (denoted $t$-RKA; $t = c$ for classical RKA). This assumption cannot be reduced to the $h$ PRFF-security against RKA, since it is possible to have a good PRFF for a uniformly distributed key that is

not a good PRFF for a high-entropy key. We omit the details of a RKA adversary used in the following theorems, but note that if the exhaustive search adversary with circuit size $S'$ is the best known $t$-RKA adversary, its advantage is smaller than $(S'/S_h)/2^t$. Fouque et al. state their revised theorem in terms of HPRF, which is constructed from several concatenations and iterations of HMAC (they do not describe HPRF in detail but refer the reader to TLS v1.2 [18]).

**Theorem 4 ([5]).** *Let $L = 1$, let ipad and opad be two fixed strings and let IV be chosen uniformly at random. Let $h$ be a function family resistant to a $t$-RKA adversary with circuit size $S'$ that makes at most 2 queries with advantage $\epsilon_0$. Let $S_h$ be the circuit size for one computation of $h$. Let HPRF be a concatenation of $v$ HMAC, and* Hash *be truncated. Let $h$ be both a $(S', q = 2v, \epsilon_1)$ and $(O(l \cdot S_h), q = 2, \epsilon_2)$ PRFF. Then* $\mathrm{Hprf}^{\mathrm{Hash}}_{ipad,opad}(IV; ps, label)$ *is a $(t, \infty, 0, S', \epsilon')$ computational randomness extractor with $\epsilon' \leq \epsilon_0 + \epsilon_1 + 4v^2 l\epsilon_2 + \frac{2v^2}{2^{c'}} + \frac{v^2}{2^c}$.*

Assuming $l = v = 1$, $\epsilon_0 \leq \frac{S'/S_h}{2^t}$, $\epsilon_1 \leq \frac{(S'/S_h)}{2^c}$, and $\epsilon_2 \leq \frac{(lS_h/S_h)}{2^c}$, we have $S'/\epsilon' \geq 2^e$ when $e \leq t-3$, $e \leq c-5$ and $e \leq c'-4$. Hence, we can extract almost all of the pre-secret's entropy when it has less entropy than the number of bits output by HPRF, and the pre-secret is only one block long.

When the pre-secret is longer than one block, it is first hashed and padded with '0' bits to obtain a $b$-bit string. The following theorem covers this case for HMAC. We omit the similar theorem for HPRF (when it is constructed from several concatenations and iterations of HMAC) due to lack of space.

**Theorem 5 ([5]).** *Let $L \geq 2$, ipad and opad be fixed strings, and IV be a variable chosen uniformly at random. Define $\hat{h} : \{0,1\}^{c'} \times \{0,1\}^c \to \{0,1\}^c$ as $\hat{h}(x, y) = h(y, x \parallel 0^{b-c'})$. Let $S_h$ be the circuit size for one computation of $h$. Let* Hash *be truncated. Let $\epsilon_2$ be the RKA advantage of an adversary against $\hat{h}$ making at most 2 related key queries with circuit size $S'$. Let $h$ be a $(S', q = 2, \epsilon_1)$, $(S', q = 1, \epsilon_3)$ and $(O(l \cdot S_h), q = 2, \epsilon_4)$ PRFF. Then* $\mathrm{Hmac}^{\mathrm{Hash}}_{IV}(ipad, opad; ps, label)$ *is a $(t, \infty, 0, S', \epsilon')$ computational randomness extractor with $\epsilon' \leq \frac{1}{2^{c'}} + \epsilon_2 + \epsilon_3 + 2l\epsilon_4 + \sqrt{2^{c'}\,(3 \cdot 2^{-t} + 2L\epsilon_1)}$.*

Assuming $\epsilon_1 \leq \frac{S'/S_h}{2^c}$, $\epsilon_2 \leq \frac{(S'/S_h)}{2^{c'}}$, $\epsilon_3 \leq \frac{(S'/S_h)}{2^c}$, and $\epsilon_4 \leq \frac{O(lS_h)/S_h}{2^c}$, we have $S'/\epsilon' \geq 2^e$ when $e \leq \frac{t-3.6-c'}{2}$, $e \leq \frac{c-c'-\log_2(L)-3}{2}$, $e \leq c'-2$ and $e \leq c-2\log_2(l)-3$. Hence, when $L = 2$, $l = 1$, and SHA-384 is used, only $e = 62$ bits of security can be achieved for the output of HMAC, and this requires $t \geq 512$. To achieve a value of $e$ close to a value of $c'$, we need $c' = e + 2 = \frac{c}{3}$. To achieve this we could further truncate the output of SHA-384 to only $c' = 170$ bits, and use this new hash function in the HMAC implementation. Then, provided $t \geq 510$, we would have $e = 168$. For $e > 168$, a new compression function $h$ with output larger than 512 bits is needed. Alternatively, to achieve our minimum requirement for Case 1 described above, of $e = 82$ and $c' = 128$, we could use SHA-384 but further truncate the output to only $c' = 128$ bits. In that case we would only need $t \geq 296$ bits. This is similar to using a universal hash function, which is not surprising, since the analysis of Fouque et al. made use of the LHL.

**Cascade Construction.** Fouque et al. [5] also analysed the use of the cascade construction as a randomness extractor when the output is truncated to contain only $c'$ bits, instead of $c$ bits. Assume the compression function $h$ of hash function $H$ (with key IV) is an $(S, q = 2, \epsilon)$ PRFF. Then $H$ is a $(t, \infty, 0, S', \epsilon')$ computational randomness extractor for prefix free distributions of at most $L$ blocks with $S = O(S')$ and $\epsilon' \leq \sqrt{2^{c'} \cdot (3 \cdot 2^{-t} + 2L\epsilon)}$. As before, assume $\epsilon \leq S/2^c$. Hence, $\epsilon' \leq \sqrt{2^{c'} \cdot (3 \cdot 2^{-t} + 2^{1-c}L \cdot O(S'))}$. To achieve a security level of $e$ bits for the output of $H$, we want $S'/\epsilon' \geq 2^e$. When $O(S') = 1$, this equates to requiring $\min\left(\frac{t-c'-3.6}{2}, \frac{c-c'-3-\log_2(L)}{2}\right) \geq e$. When the requirements for $O(S') = 1$ are met, those for when $O(S') = 2^{e-1}$ will be met also. These restrictions on $e$ are almost the same as for HMAC when the pre-secret is more than one block long, and so similar comments to those made for HMAC apply here.

## 5 Conclusion

This paper examined the use of randomness extraction and expansion in key agreement protocols to generate uniformly distributed keys. Although other works exist that provide the basic theorems necessary, they lack details or examples of what cryptographic primitives are appropriate and/or how large the parameters of those primitives must be. We have therefore summarized existing work in the area and examined the security levels achieved with the use of various extractors and expanders for particular sizes of parameters.

As noted in some existing works ([1, p.4], [2, p.2]), the large amount of min-entropy needed in the pre-secret is often overlooked in efficiency comparisons of KA protocols. In fact, using the tables presented in this paper, one may conclude that this shared secret will need a min-entropy of at least 292 bits to achieve an overall security level of 80 bits. More realistic assumptions on the number of times the randomness extractor and expander are used may require a much higher min-entropy for this security level. The tables may be used to find the min-entropy required for various security levels and assumptions on how the extractor and expander will be used. We also found that when numbers are substituted into the short exponent theorems of Gennaro et al., the exponents may need to be much longer than they suggested.

## References

1. 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), `http://eprint.iacr.org/2004/099`
2. 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), `http://eprint.iacr.org/2005/061`

3. 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)
4. 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)
5. Fouque, P.A., Pointcheval, D., Zimmer, S.: HMAC is a randomness extractor and applications to TLS. In: ASIACCS 2008: Proceedings of the, ACM symposium on Information, computer and communications security, pp. 21–32. ACM, New York (2008)
6. Bellare, M., Kilian, J., Rogaway, P.: The security of the cipher block chaining message authentication code. Journal of Computer and System Sciences 61(3), 362–399 (2000), http://www-cse.ucsd.edu/~mihir/papers/cbc.html
7. Goldreich, O.: The Foundations of Cryptography, vol. 1. Cambridge University Press, Cambridge (2001), http://wisdom.weizmann.ac.il/~oded/frag.html
8. Shoup, V.: A Computational Introduction to Number Theory and Algebra. Cambridge University Press, Cambridge (2005), http://shoup.net/ntb/
9. Bellare, M., Canetti, R., Krawczyk, H.: Pseudorandom functions revisited: The cascade construction and its concrete security. In: Proceedings of the 37th Annual Symposium on the Foundations of Computer Science, pp. 514–523. IEEE, Los Alamitos (1996)
10. NIST (National Institute for Standards and Technology): Advanced encryption standard (AES). FIPS PUB 197 (2001)
11. 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)
12. Rivest, R.: The MD5 message-digest algorithm. Internet RFC 1321, Internet Engineering Task Force (1992)
13. Dobbertin, H., Bosselaers, A., Preneel, B.: RIPEMD-160: A strengthened version of RIPEMD. In: Gollmann, D. (ed.) FSE 1996. LNCS, vol. 1039, pp. 71–82. Springer, Heidelberg (1996)
14. NIST (National Institute for Standards and Technology): Secure hash standard. FIPS PUB 180-2 (2000)
15. Preneel, B., van Oorschot, P.: On the security of iterated message authentication codes. IEEE Transactions on Information Theory 45(1), 188–199 (1999)
16. Dodis, Y.: Exposure-Resilient Cryptography. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology (2000), http://theory.lcs.mit.edu/~yevgen/academic.html
17. Mansour, Y., Nisan, N., Tiwari, P.: The computational complexity of universal hashing. In: Proceedings of the Twenty Second Annual ACM Symposium on Theory of Computing—STOC 1990, pp. 235–243. ACM Press, New York (1990)
18. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) protocol version 1.2. Internet RFC 5246, Internet Engineering Task Force (2007)