An abridged version of this paper appears in *Proceedings of the* 37th *Symposium on Foundations of Computer Science*, IEEE, 1996.

Pseudorandom Functions Revisited: The Cascade Construction and its Concrete Security

Mihir Bellare* Ran Canetti† Hugo Krawczyk‡

August 7, 1996

Abstract

Pseudorandom function families are a powerful cryptographic primitive, yielding, in particular, simple solutions for the main problems in private key cryptography. Their existence based on general assumptions (namely, the existence of one-way functions) has been established.

In this work we investigate new ways of designing pseudorandom function families. The goal is to find constructions that are both efficient and secure, and thus eventually to bring the benefits of pseudorandom functions to practice.

The basic building blocks in our design are certain limited versions of pseudorandom function families, called finite-length input pseudorandom function families, for which very efficient realizations exist in practical cryptography. Thus rather than starting from one-way functions, we propose constructions of "full-fledged" pseudorandom function families from these limited ones. In particular we propose the cascade construction, and provide a concrete security analysis which relates the strength of the cascade to that of the underlying finite pseudorandom function family in a precise and quantitative way.

*Department of Computer Science & Engineering, Mail Code 0114, University of California at San Diego, 9500 Gilman Drive, La Jolla, CA 92093. Email: `mihir@cs.ucsd.edu`.

†IBM T.J. Watson Research Center, PO Box 704, Yorktown Heights, New York 10598. Email: `canetti@watson.ibm.com`. Work done while author was at MIT.

‡IBM T.J. Watson Research Center, PO Box 704, Yorktown Heights, New York 10598. Email: `hugo@watson.ibm.com`.

1 Introduction

The notion of a pseudorandom function family was proposed by Goldreich, Goldwasser and Micali [9]. In such a family, each function is specified by a short, random key, and can be easily computed given the key. Yet the function behaves like a random one, in the sense that if you are not given the key, and are computationally bounded, then the input-output behavior of the function looks like that of a random function.

Pseudorandom functions (PRFs) enable the following very general paradigm of cryptographic protocol design, most importantly in the setting of private key cryptography (in this setting, a number of parties share a short, random key a which the adversary does not know): First design and prove secure a protocol assuming the parties share a truly random function. Then substitute the random function with a pseudorandom function indexed by the shared key a. The properties of pseudorandom function families guarantee that security is preserved.

In particular, pseudorandom functions provide, via this paradigm, simple and convenient solutions to the two most basic problems of private key cryptography, namely message authentication and encryption. Data D can be authenticated by appending the tag $f(D)$, where $f = F_a$ is the function indexed by key a in familiy F. Similarly, data D can be encrypted by picking r at random and transmitting $(r, f(r) \oplus D)$, where $\oplus$ is bitwise exclusive-or. It is not hard to see that these methods are secure if f is truly random, and hence also if f is chosen from F.

Therefore, the study of practical candidates for pseudorandom functions is essential in providing solutions to the increasing use of cryptography in real world applications. This search forms the basic motivation for our work.

It is important to note that the above applications impose requirements on the *sizes of inputs and outputs* of the pseudorandom functions in the family. The data to be authenticated or encrypted can be mega-bytes or more in length (e.g., a movie). Thus, to be useful for authentication, the function F_a must be able to take long inputs. (This means inputs of any pre-specified length, or, even better, inputs of variable length.) Encryption done in the above way poses similar requirements on the output length. Since this will be a crucial point, let's introduce a piece of terminology straight away. We call a pseudorandom function a VI-PRF (the "VI" standing for "Variable-length Input") if it can take inputs of any pre-specified length, or of variable length. These are the kinds of pseudorandom functions we want.

So how can one construct (variable-length input) pseudorandom function families? A traditional approach constructs pseudorandom function families from pseudorandom generators [9]. Pseudo-random generators are in turn constructed from any one-way function [11]. (Recently a different construction of pseudorandom function families from trapdoor functions was proposed [15].) In these works pseudorandom function families are regarded as a compound primitive, whereas one-way functions (or trapdoor functions) are the building blocks. In all cases, the suggested candidates for the building blocks come from number-theoretic problems or combinatorial problems.

We follow a different approach to constructing variable-length input pseudorandom function families which has the potential of yielding more efficient constructions. The idea is to move to different building blocks.

1.1 From FI-PRFs to VI-PRFs

We assume the existence of *fixed-length input* pseudorandom function (FI-PRF) families. This is a primitive first introduced by Bellare, Kilian and Rogaway [5] in order to model the popular Data Encryption Algorithm. A FI-PRF family is a family of functions which already have the pseudorandom function property. But they have the following serious drawback: unlike VI-PRF functions, the members of this family only take fixed-length inputs. For example the members of the family might map 512 bits to 128 bits. [1]

So, for us the basic "building blocks" are FI-PRF families rather than one-way functions or pseudorandom generators. We then show how to construct VI-PRF families from FI-PRF families in an effective way. Let us now explain the background and motivation for this assumption and approach.

THE BASIS FOR FI-PRFS. There is a body of cryptographic primitives, extensively used in practice, which are more efficient than number theoretic primitives yet seem to possess appreciable cryptographic strengths. Examples of such primitives include DES [2], and the compression functions of SHA [18] and of MD5 [16]. In particular, it might be reasonable to assume that such primitives, or simple enhancements or extensions of them, behave like pseudorandom functions in the first place. Such a suggestion was first made in the context of DES [13, 5] and has also been suggested for the compression function of MD5 [4], the last two both in the context of constructing message authentication codes. However, these primitives have finite-length inputs and outputs. (For example the compression function of MD5 provides a family each member of which is specified by a 128 bit key and maps a 512 bit input to a 128 bit output.) So we are talking about FI-PRF families; whereas applications need VI-PRFs. This motivates our study of constructing VI-PRFs from FI-PRFs.

THE MERITS OF A GENERAL CONSTRUCTION. It is important to find general constructions that work assuming the existence of *any* FI-PRF family. The reason is the same as that driving the search for constructions based on any one-way function. Namely, some particular candidate might fail or new (possibly better) candidates may emerge. In particular, we caution that at this time, the assumption that DES, or some other specific function, behaves like a FI-PRF has not been extensively investigated or validated, and for some of these functions, the assumption may turn to be false.

General techniques for extending FI-PRFs to VI-PRFs also provide for modular design of pseudorandom function families: First engage in the easier task of designing a FI-PRF. Next construct a VI-PRF using the general technique.

1.2 The cascade construction

BACKGROUND. Our construction stems from a popular construction of collision-resistant hash functions. A collision-resistant cryptographic hash function (CRH) H is a function that hashes arbitrarily long inputs to outputs of fixed length k (say, $k = 128$ bits), while guaranteeing that it is computationally infeasible to find $x \neq y$ such that $H(x) = H(y)$. The construction of CRHs, due to Merkle [14] and Damgård [8], is as follows. First design a "compression function" h that takes two inputs of fixed length – one of k bits and the other of b bits — and outputs a k-bit string. Make sure that h is collision-resistant (i.e., it should be infeasible to find $(a, x) \neq (a', x')$ such that $h(a, x) = h(a', x')$). Next, define H as follows. First fix some arbitrary k-bit initial value IV. Next, partition each input x into b-bit blocks $x = x_1, \ldots, x_n$ and compute:

$$
\begin{aligned}
y_0 &= \text{IV} \\
y_i &= h(y_{i-1}, x_i) \qquad \text{for } i = 1, \ldots, n.
\end{aligned}
$$

The output is $H(x) = y_n$. It is easy to see that if h is collision-resistant then so is H.

[1] The notion of security is appropriately quantified to discuss finite objects [5]. We discuss this later.

Known CRHs (e.g., Rivest's MDi series, SHA) follow this paradigm. These CRH are in wide use. Their strength lies in their efficiency and versatility: in particular, they can be run, in software, faster than most other cryptographic transforms.

THE BASIC CASCADE CONSTRUCTION. We assume we are given a FI-PRF family F. Each k bit key a specifies a function $F_a\colon \{0,1\}^b \to \{0,1\}^k$. We write $F_a(z) = F(a,z)$. Now, we propose to construct a family $F^{(*)}$ which we call the *basic cascade construction.* Each function $F_a^{(*)}$ is still indexed by a k-bit key and returns a k-bit output, but takes input of arbitrary length. The construction will be the same as above, with F playing the role of h, and one other (important) difference: the key a to $F_a^{(*)}$ plays the role of IV. Namely we compute:

$$\begin{aligned} y_0 &= a \\ y_i &= F(y_{i-1}, x_i) \qquad \text{for } i = 1, \ldots, n, \end{aligned}$$

where n is the number of input blocks. The output is $F(a,x) = y_n$. (In short, $F_a^{(*)}(x_1, \ldots, x_n) = F_{F_a^{(*)}(x_1,\ldots,x_{n-1})}(x_n)$.) Our claim is that, given the above assumption on the basic family F, the resultant family $F^{(*)}$ is a pseudorandom function family on any set of inputs that is prefix-free (that is, as long as no input is a prefix of another input). In particular, this holds when all inputs are of the same length.

Since what we have done is replace the fixed IV in the Merkle construction with the key to our PRF, we view the cascade construction as a "keyed" version of Merkle's construction.

Now, we might view the compression function h of SHA or MD5 as specifying a FI-PRF family, via $F_a(z) = h(a,z)$. Then we have an efficient construction of a pseudorandom function family taking inputs of any desired (but fixed) length nb.

GENERAL, VARIABLE LENGTH INPUTS. If inputs are allowed to be prefixes of other inputs then the pseudorandomness property vanishes. [2] Fortunately, the construction can be modified to work. We show that if some prefix-free encoding is applied before invoking $F^{(*)}$ then a good VI-PRF is obtained. An alternative that may be more attractive in practice is to append some "key-material" to the message before doing the construction. More precisely, let the key consist of $k+\delta$ bits, where the first, k-bit part is called a and the second, δ-bit part is called d. Let $F^{\mathsf{acsc}}_{a.d}(x) \stackrel{\text{def}}{=} F_{F_a^{(*)}(x)}(d)$. We call this the *append cascade* construction. Below we focus on this one, since in practice variable length is required.

1.3 Security

We want to say that if F is a FI-PRF family then F^{acsc} is a VI-PRF family. But there is a small problem. Traditionally, pseudorandom function families, like similar primitives (such as one way functions and pseudorandom generators), are presented in an asymptotic way: there is an infinite sequence of *instances* of a primitive, where each instance corresponds to a specific value of a *security parameter.* As the security parameter tends to infinity, the probability of breaking the corresponding instance becomes negligible. (In the case of pseudorandom function families the security parameter is related to the key-length.) Here, however, we deal with primitives that consist of a *single instance*: there is a single MD5, a single SHA and a single DES. Asymptotic behavior in the security parameter has no meaning here. We thus resort to explicitly specifying the parameters determining the security.

[2] For $F^{(*)}$, note that if x_1, x_2 are each b bits long then $F_a^{(*)}(x_1x_2) = F_{F_a^{(*)}(x_1)}(x_2)$, so $F_a^{(*)}(x_1x_2)$ can be computed given $F_a^{(*)}(x_1)$. But for a random function R one has very low probability of computing $R(x_1x_2)$ given $R(x_1)$.

Another benefit of explicitly specifying these parameters is that it highlights the quality of a transformation from one primitive to another. This is part of a general program in which one investigates not only the existence of polynomial time reductions between primitives, but the actual efficiency they achieve. The quality of a security reduction is important because it tells us exactly what strength we can expect from a scheme, for example how much time we can allow the adversary and still know that the scheme is not breakable except with a specific probability.

The above two ingredients, explicit security parameters and carefully quantified security reductions, form the basis for what we call *concrete security analysis*. In the case of pseudorandom functions this study was initiated in [5, 4]. Security preserving reductions are the subject of other works as well, e.g. [10, 12, 1].

Following [5], we say that a function family G is (t, q, l, ϵ)-secure if a program that runs in time t (more precisely, the running time plus size of the description of the program, in some fixed RAM model of computation, must be bounded by t), given an oracle for a function E and allowed to make at most q queries to this oracle, each of at most l blocks, has advantage at most ϵ in distinguishing whether E is a random member of G or a truly random function with the appropriate domain and range.

Now we are finally able to state our main result: We show that if F was $(t', q, 1, \epsilon')$-secure then F^{acsc} is (t, q, l, ϵ)-secure, for t comparable to t' and $\epsilon = ql\epsilon'$. Thus we establish a concrete lower bound on the hardness to break the cascaded family in terms of the strength of the given family.

TIGHTNESS. We complement the analysis of the cascade construction by demonstrating its optimality. We present a simple algorithm, based on the Birthday Paradox, for distinguishing between a function generated via our cascaded construction and a truly random function. The resultant attack achieves a distinguishing probability equal to the one proven in our lower bound. The attack is very general, and requires no a-priori information about the underlying FI-PRF. In particular, it applies even if the underlying function family is chosen at random from all function families with a given size. (We also show how to translate this distinguisher into an attack on message authentication schemes based on the cascaded construction. See Section 6). This attack, combined with an additional simple observation, demonstrates the optimality of the analysis.

1.4 The randomized cascade construction

In the above cascade construction the probability of successful attack against the constructed family, F^{acsc}, is larger than the probability of successful attack against the underlying function F by a factor of ql. We modify the cascade construction to obtain more moderate increase in the probability of successful attack. The idea is to randomize the cascade. It also turns out that this randomization obviates the need to append key material, so now the key is just a. The output for input x is computed by picking a random b bit string R and outputting $(R, F_a^{(*)}(R \,.\, x))$. Note that this is now a probabilistic function, since to compute the output on a given input, a probabilistic choice is made. We call this randomized function F_a^{rcsc}.

Technically F^{rcsc}, being randomized, cannot be a pseudorandom function family. But we can consider "randomized" families in a natural way. We show that if F was $(t', i, 1, \epsilon'(i))$-secure for all $i = 1, \ldots, q$ then F^{rcsc} is (t, q, l, ϵ)-secure for t comparable to t' and $\epsilon = \epsilon'(q) + lq\epsilon'(1) + \delta$ where $\delta = \sum_{i \geq 2} \frac{q^i}{2^{(i-1)l}} \epsilon'(i)$ is negligible. In contrast, F^{acsc} obtains, using this notation, $\epsilon = lq\epsilon'(q)$. In the typical cases where $\epsilon'(1)$ is much smaller than $\epsilon'(q)$, F^{rcsc} does significantly better than F^{acsc}.

1.5 Related work

In comparison to constructions based on one-way or trapdoor functions [11, 9, 15], ours are more efficient in practice. Let us now turn to constructions based on FI-PRFs.

The first construction of VI-PRFs from FI-PRFs is in [5]. They analyze another popular construction (namely, Cipher Block Chaining, or CBC) and show that it yields a VI-PRF if the underlying functions are FI-PRFs. That work differs from ours in the practical cryptographic primitive considered (it builds on block ciphers as opposed to compression functions as in our case).

FI-PRFs are used to construct message authentication codes in [4]. Underlying their constructions is a construction of a *randomized* family of VI-PRFs. But they don't provide a (deterministic) VI-PRF family.

If F is a FI-PRF family and H is a collision-resistant function then the family G defined by $G_a(x) = F_a(H(x))$ is a VI-PRF family. (This simple observation was made in [6].) A variant is to let $\mathcal{H}$ be a collection of almost universal$_2$ hash functions [7, 19], meaning that for any fixed but distinct x, x', the probability that $h(x) = h(x')$ is small, when h is drawn randomly from $\mathcal{H}$. Then $G_{a,h}(x) = F_a(h(x))$ is also a VI-PRF family. (Note that h is part of the key).

We note that, in general, no one of these constructions is superior to the other in all the aspects. Choosing between one approach or the other in practice may depend on a variety of considerations like efficiency and availability of software and hardware implementations, security of the underlying primitive, quality of the reduction proving security, key size, and even non-technical aspects as the regulation of cryptography by local governments. It is therefore important to back these different approaches with sound and efficient constructions. Furthermore, it is important to investigate constructions that exist in practice and figure out their quality, even when alternatives are known, because in many cases the practical constructions have advantages in the settings in which they are used.

In a companion work, we build message authentication functions from collision-resistant hash functions [3]. That work uses a different assumption than the one here. It assumes the underlying compression function to be a secure message authentication function, and the iterations (à la Merkle) be collision resistant. Using our work, one can show that if the compression function is a good pseudorandom function then both of the above assumptions in [3] are fulfilled.

Finally, we remark that the construction, and our proof methodology, are reminiscent of the [9] construction and proof. They construct VI-PRFs from length doubling pseudorandom bit generators, whereas we use FI-PRFs. However the basic cascade construction (which, as we have said, is a keyed version of Merkle's construction) may be viewed as a generalization of the binary tree construction of [9]. In our case the tree has arity 2^b. (Recall b is the input block-size. The tree is as follows. The root stores the original key a. The children of a node labeled y are the 2^b strings $F_y(x)$ for $x \in \{0,1\}^b$.) In particular, the constructions coincide if $b = 1$. (That is, we have a FI-PRF family which takes just one bit of input. Such a family can be seen as a length doubling pseudorandom bit generator defined by $G(y) = F_y(0) \,.\, F_y(1)$.) Our proof is influenced by that of [9], but new features emerge because we are dealing with pseudorandom functions, not generators, making the reducibility question a different one. Comparing efficiency, given the generator, the [9] construction takes m applications of it to process an m-bit message, while given the FI-PRF, we take m/b applications of it.

2 Definitions

We define FI-PRF families and their concrete security, slightly extending the definitions of [5].

NOTATION. Denote by $|x|$ the length of string x. Let $[m] = \{1, \ldots, m\}$ for any integer $m \geq 0$. If S is a probability space (resp. a set) then $x \stackrel{R}{\leftarrow} S$ denotes the operation of selecting an element from S (resp. uniformly at random from S). Furthermore $x_1, \ldots, x_m \stackrel{R}{\leftarrow} S$ abbreviates $x_1 \stackrel{R}{\leftarrow} S\,; \ldots ; x_m \stackrel{R}{\leftarrow} S$. Let $\mathrm{Maps}(X, Y)$ be the set of all functions mapping from set X to set Y.

FINITE FUNCTION FAMILIES. A (finite) *function family* F is a finite collection of functions together with a probability distribution on them. Thus $f \stackrel{R}{\leftarrow} F$ denotes the operation of selecting a function at random from F according to the underlying distribution. All functions in the collection are assumed to have the same domain, denoted $\mathrm{Dom}(F)$, and the same range, denoted $\mathrm{Range}(F)$.

We consider keyed families. Here there is a set of "keys" and each key names a function in F according to some fixed convention. (Note that different keys can name the same function.) Usually the set of keys is $\{0,1\}^k$ for some integer k called the key length. We use several notations for the function denoted by a key a. Sometimes we write it F_a; other times we view F as a two argument function and write F_a as $F(a, \cdot)$. When some key length, and some association of keys to functions have been fixed and agreed upon, we adopt the convention that the distribution on the function family is that given by picking a key at random from $\{0,1\}^k$ and selecting the corresponding function; namely, $f \stackrel{R}{\leftarrow} F$ is equivalent to $a \stackrel{R}{\leftarrow} \{0,1\}^\kappa\,;\, f \leftarrow F_a$. We require that given the key it is possible to compute the corresponding function efficiently. Measuring the computation time, let $\mathrm{Time}(F, n)$ denote the time of a program to compute $F_a(x) = F(a, x)$ given a and x with $|x| \leq n$.

In the families we consider the range $\mathrm{Range}(F)$ will be fixed. Furthermore, it will equal the domain of the keys, namely $\{0,1\}^k$. The domain $\mathrm{Dom}(F)$ can vary; however, we will work over a particular block size b (eg. $b = 512$) and denote $B = \{0,1\}^b$. This means we will consider domains of the form B, B^*, B^n, and $B^{\leq n}$ (i.e., strings of at most n blocks).

PSEUDORANDOMNESS OF FUNCTION FAMILIES. Intuitively, a finite function family F is pseudorandom if the input-output behavior of a random member of the family is indistinguishable from the behavior of a random function of the same domain and range. This is formalized via the notion of statistical tests, or distinguishes [9]. A distinguisher is an oracle algorithm; it is given a random member either of a family F^1 or of a family F^2, and tries to decide which is the case. For a distinguisher D let

$$\mathsf{Adv}_D(F^1, F^2) \;=\; \Pr_{g \stackrel{R}{\leftarrow} F^1}[\, D^g = 1\,] - \Pr_{g \stackrel{R}{\leftarrow} F^2}[\, D^g = 1\,] \ ,$$

where the probabilities are over the choices of g and the coin tosses of D.

We are interested in the resources used by the distinguisher to make its decision. Salient resources are computing time, memory requirements, and number and length of oracle queries. Even though time and memory limitations may be very different,[3] for this extended abstract we unify the two notions (and slightly abuse notation) by letting time equal the running time *plus* the memory required. (Without loss of generality the memory includes the size of the code.) Say that D (t, q, l, ϵ) distinguishes F^1 from F^2 if it runs for time t, makes q oracle queries each of length at most l blocks, and $\mathsf{Adv}_D(F^1, F^2) \geq \epsilon$. (Here ϵ is called the *distinguishing probability.*)

Let F be a family with domain X and range Y, ie. $F \subseteq \mathrm{Maps}(X, Y)$. View $\mathrm{Maps}(X, Y)$ as a function family with the distribution being uniform; that is, drawing a function at random from this family just means picking a random function of X to Y.[4] We say that D (t, q, l, ϵ)-breaks F

[3] While 2^{50} running time may be considered reasonable, memory size of 2^{50} is unreasonable.

[4] Here we slightly abuse the notation: if the domain X is infinite then choosing a function uniformly from $\mathrm{Maps}(X, Y)$ has no meaning. Instead, one can think of a "random function" as a process that answers each new query with a random element in the range Y, and answers queries that have already been made in the past in a consistent way.

if D (t, q, l, ϵ) distinguishes F from Maps(X, Y). We say that F is (t, q, l, ϵ)-secure if there is no distinguisher who (t, q, l, ϵ)-breaks F.

3 The basic cascade construction

Here we present and analyze the basic cascade construction. Let F be a family of functions from $\mathrm{Dom}(F) = B = \{0,1\}^b$ to $\{0,1\}^k$. The key length is the same as the output length, namely k. For instance, in the case of the compression function of md5 we have $k = 128$ and $b = 512$. Define the families $F^{(1)}, F^{(2)}, F^{(3)}, \ldots$ as follows. The members of $F^{(l)}$ will take inputs which are at most l blocks long; that is, the family $F^{(l)}$ will have domain $\mathrm{Dom}(F^{(l)}) = B^{\leq l}$. We call $F^{(l)}$ the l-th *iteration* of F. For $l \in \mathbf{N}$, define $F^{(l)}$ inductively:

$$
\begin{aligned}
F^{(l)}(a, \lambda) &= a \\
F^{(l)}(a, x_1 \ldots x_n) &= F(F^{(l)}(a, x_1 \ldots x_{n-1}), x_n) \quad \text{if } n \geq 1
\end{aligned}
$$

where λ denotes the empty string. That is, for $n \leq l$, $F^{(l)}(x_1, \ldots, x_n)$ is computed as follows.

$a_0 \leftarrow a$
for $i = 1, \ldots, n$ **do:** $a_i \leftarrow F(a_{i-1}, x_i)$
Output a_n

Let $F^{(*)}$ be $F^{(*)}(a, x_1, \ldots, x_n) \stackrel{\text{def}}{=} F_a^{(n)}(a, x_1, \ldots, x_n)$. We call $F^{(*)}$ the *iteration* of F.

Prefix-freeness. We want to claim that the pseudorandomness of F translates to the pseudorandomness of $F^{(*)}$ (with parameters specified in the sequel). But there is the "technical" problem described in Section 1.2. This problem disappears if the inputs are encoded to be a prefix-free set. (For example, prepend-pend to every input its length, using some appropriate encoding, and then apply $F^{(*)}$.) With this, $F^{(*)}$ is indeed pseudorandom, as we will show below.

We don't want to fix any particular prefix-free encoding since any such encoding will work. So instead we introduce the notion of a *prefix-free distinguisher.* This is a distinguisher D with the property that the set of queries it asks always forms a prefix-free set. We'll prove $F^{(*)}$ is pseudorandom with respect to prefix-free distinguishers. This implies that using any prefix-free encoding will suffice against arbitrary distinguishers.

Prefix-free encodings have some drawbacks that makes them unattractive in some settings. We elaborate on these drawbacks and our solution in the next section. Here we show:

Theorem 3.1 *Let F be a function family with $\mathrm{Dom}(F) = B$, range $\mathrm{Range}(F) = \{0,1\}^k$, and key length k. Suppose F is $(t', q, 1, \epsilon')$-secure and let $l \geq 1$. Then $F^{(*)}$ is (t, q, l, ϵ)-secure against prefix-free distinguishers, where*

$$t = t' - cq(l + k + b) \cdot (\mathrm{Time}(F) + \log q) \ \ ; \ \ \epsilon = l\epsilon' \ .$$

Here c is a specific, small constant whose value can be determined from the proof.

The rest of this section is devoted to proving Theorem 3.1. A natural approach to such a proof is to reduce the security of $F^{(*)}$ to the security of F. However, we could not find a straightforward reduction. Instead, we first define a bunch of "intermediate" function families. We then show, using two different reductions, that: **(I)** if only prefix-free distinguishers, that ask at most q queries, are considered, then the security of $F^{(*)}$ reduces to the security of the qth family in the bunch; and **(II)** the security of any family in the bunch reduces to the security of F. The theorem will follow.

The multi-oracle families. Let F be a family of functions from X to Y. The intermediate function families, called the *multi-oracle families*, are defined as follows. Informally, in the mth multi-oracle family the distinguisher is given m functions, $g_1, \ldots, g_m$, such that either all the $g_i' s$ are drawn independently from F or all are drawn independently from $\mathrm{Maps}(X, Y)$. (To break this family, the distinguisher has to tell which is the case.) More formally, define mF as the following family of functions from $[m] \times X$ to Y, with key length mk. The key is interpreted as a vector $\vec{a} = a_1, \ldots, a_m$; a function $mF_{\vec{a}}$ in the family mF takes two inputs, $i \in [m]$ and $x \in X$, and returns $F_{a_i}(x)$. We sometimes use $f_1, \ldots, f_m$ to denote the function $mF_{\vec{a}}$ that satisfies $mF_{\vec{a}}(i, x) = f_i(x)$ for all i and x.

Now that the multi-oracle families are defined, we can state two lemmas that together prove Theorem 3.1:

Lemma 3.2 *Let F be a function family with domain $\mathrm{Dom}(F) = B$, range $\mathrm{Range}(F) = \{0,1\}^k$, and key length k. Assume qF is $(t', q, 1, \epsilon')$-secure and let $l \geq 1$. Then $F^{(*)}$ is (t, q, l, ϵ)-secure against prefix-free distinguishers, where*

$$t = t' - cql \cdot (b + k + \log q + \mathrm{Time}(F)) \quad ; \quad \epsilon = l\epsilon' \ .$$

Here c is a specific, small constant whose value can be determined from the proof.

Lemma 3.3 *Let F be a function family with domain $\mathrm{Dom}(F) = B$, range $\mathrm{Range}(F) = \{0,1\}^k$ and key length k, and let $m \geq 1$. Suppose F is $(t', q, 1, \epsilon')$-secure. Then, mF is $(t, q, 1, \epsilon)$-secure, where*

$$t = t' - c \cdot (mk + q \cdot \mathrm{Time}(F) + q(k+b)\log q) \quad ; \quad \epsilon = m\epsilon' \ .$$

We remark that the key quantity here is the distinguishing probability ϵ, which, in Lemma 3.2 increases by a factor of l, and in Lemma 3.3 further increases by a factor of q. Typically t' is large enough that t should be thought of as being approximately t'.

Theorem 3.1 now follows from Lemmas 3.2 and 3.3. The proof of the former is in Appendix A. The proof of the latter is in Appendix B.

4 Getting rid of prefix-freeness

Our basic construction is valid as long as the queries are prefix-free. Prefix-freeness can be ensured by appropriately encoding the queries. For example, prepend the length $|q|$ to each query q in an appropriate way.

However, such encodings require, of the machine evaluating the pseudorandom function, to perform at least two "passes" of reading the input (e.g., one pass to find the length, and a second pass to compute the function with the length prepended). Thus the entire input must be stored in memory (or a buffer) in between the passes. This is a serious drawback when long chunks of data need to be processed "on the fly". (An important such scenario is the use of pseudorandom function families to generate message authentication codes for communicated data).

Here we describe a construction that has practically the same security as prefix-free encoding, and hardly reduces the efficiency. In particular only one pass is needed. The construction is what was called F^{acsc} in Section 1.2. Namely given a family F with key length k, construct the family $\delta\text{-}\mathrm{A}F$ with key length $k + \delta$ as follows. Having key (a, d) where $a \in \{0,1\}^k$ and $d \in \{0,1\}^\delta$, let $\delta\text{-}\mathrm{A}F_{a,d}(x) = F_a(xd)$ for all x, where xd denotes the concatenation of x and d.

Now, $\delta\text{-}\mathrm{A}F$ requires only one pass; furthermore, as will be seen, δ can be pretty small. Thus the efficiency constraint is satisfied. Why is it secure? an informal argument may proceed as follows: "in order to break $\delta\text{-}\mathrm{A}F$, one has to generate queries such that, when translated to F, one query will be a prefix of another. In order for this to happen, the latter query has to contain d as a substring.

Since d is random, this event happens with probability exponentially small in $|d| = \delta$." However, this argument assumes that d remains unpredictable by the distinguisher. This unpredictability, in turn, depends on the security of the construction... thus a more rigorous proof is needed. We show:

Theorem 4.1 *Let F be a function family with*$\mathrm{Dom}(F) = B = \{0,1\}^b$*, which is is (t', q, l, ϵ')-secure against prefix-free distinguishers. Then δ-$\mathrm{A}F$ is (t, q, l, ϵ)-secure where*

$$t = t' - cq\mathrm{Time}(F)\log\delta \;\; ; \;\; \epsilon = \epsilon' + blq2^{-\delta}.$$

Here c is a specific small constant determined by the proof.

A proof of Theorem 4.1 appears in Appendix C. Let δ-$\mathrm{A}F^{(*)}$ denote the family constructed from $F^{(*)}$ via the above construction. Combining Theorems 3.1 and 4.1 we get:

Corollary 4.2 *Let F be a function family with domain* $\mathrm{Dom}(F) = B = \{0,1\}^b$*, range* $\mathrm{Range}(F) = \{0,1\}^k$ *and key length k. Suppose F is $(t', q, 1, \epsilon')$-secure and let $l \geq 1$. Then δ-$\mathrm{A}F^{(*)}$ is (t, q, l, ϵ)-secure, where*

$$\begin{aligned} t &= t' - cq \cdot (k + b + (l + \log\delta)\mathrm{Time}(F) + (k + l + b)\log q) \\ \epsilon &= lq\epsilon' + blq2^{-\alpha}. \end{aligned}$$

5 The random-prepend construction

We now show how randomization can improve the security of the cascade construction. Given family F, define F_a^{rcsc} as follows: on input x choose $r \stackrel{R}{\leftarrow} B$ and return $(r, F^{(*)}(rx))$. The gain is specified in the following theorem. (Here $\epsilon(i)$ denotes the distinguishing probability when i queries are made.)

Theorem 5.1 *Let F be a function family with* $\mathrm{Dom}(F) = B = \{0,1\}^b$*, range* $\mathrm{Range}(F) = \{0,1\}^k$*, and key length k. Suppose F is $(t', i, 1, \epsilon'(i))$-secure for all $i = 1, \ldots, q$, and let $l \geq 1$. Then F^{rcsc} is (t, q, l, ϵ)-secure, where*

$$\begin{aligned} t &= t' - cq(l + k + b) \cdot (\mathrm{Time}(F) + \log q) \\ \epsilon &= \epsilon'(q) + lq\epsilon'(1) + \delta \end{aligned}$$

Where $\delta = \sum_{i=2}^{q} \frac{q^i}{2^{l(i-1)}}\epsilon'(i)$, and c is a specific, small constant whose value can be determined from the proof.

Compare the ϵ here to the $\epsilon = lq\epsilon'$ of Theorem 3.1. In the current notation, Theorem 3.1 obtains $\epsilon' = lq\epsilon'(q)$. But here the first term is a significant improvement; it is only $\epsilon'(q)$. Now the second term has the lq product, but it is multiplied only by $\epsilon'(1)$, the advantage under a single query, which is presumably much smaller than $\epsilon'(q)$. The δ term is insignificant. Thus, overall, there is a significant reduction in ϵ.

Let us briefly sketch the proof of this theorem. We return to the proof of Lemma 3.2. The algorithm D_2, given a query $x^j = x_1^j \ldots x_n^j$ made by D_3 will itself pick r^j at random, so that the "effective" query is $r^j x_1^j \ldots x_n^j$. Now if $i = 1$ then we end up invoking $g_1(r^j)$. Thus at most q queries are asked of the original family and accounts for the $\epsilon'(q)$ term above. Suppose $i \geq 2$. Then, the maximum number of times any particular g_{t^p} is invoked is the maximum number of times any particular value r turns up in the set $\{r^1, \ldots, r^q\}$.

6 Optimality of the analysis

Theorem 3.1 provides a general upper bound on the rate of increase in the advantage ϵ when applying the basic cascaded construction. We investigate the optimality of this bound. In other words, how much can our analysis be improved, without resorting to the particularities of specific function families?

We formalize this question as follows. We introduce a setting where the function family F (with domain B, range $\{0,1\}^k$ and key length k) can be accessed only as a black box. That is, consider an algorithm for deciding whether a given function is random or chosen from F. Instead of being given an explicit description of how to compute functions in F, the algorithm can now only query for the values of functions in F at points of its choice. That is, it can ask two types of queries: *oracle queries,* addressed at the function tested for being chosen from the family, and *family-queries,* addressed at the functions in the family in question. (Family-queries should specify a key a and a value x, and are answered by $F_a(x)$.) In this setting, called the *black-box-family setting*, we show that there exist function families for which the analysis of Theorem 3.1 is (almost) optimal. In fact, optimality holds even for a *random* function family. (Still, we note that other general bounds may exist that, while not improving our bound in the case of the black-box family setting, do better with other pseudorandom families.)

We present the result in more detail. Let $\epsilon_F(l,q,\tau)$ denote the highest distinguishing probability in breaking family F by any algorithm that asks a total of at most τ oracle and family queries, out of which at most q are (prefix-free) oracle queries, and each oracle query consists of at most l B-blocks. In this language, Theorem 3.1 implies that

$$\epsilon_{F^{(*)}}(l,q,\tau) \leq lq \cdot \epsilon_F(1,q,\tau + O(l \cdot q)) \tag{1}$$

for any F. We proceed in two steps as follows. First we show a simple and general distinguishing algorithm demonstrating that

$$\frac{c(l-1)q^2}{2^k} \leq \epsilon_{F^{(*)}}(l,q,q) \tag{2}$$

for some $c > 0$ and *any* F. This algorithm works even in the black-box-family setting described above. That is, it uses an intrinsic property of the cascade construction and requires no knowledge of the underlying family F.

Inequality (2) by itself does not yet demonstrate optimality of (1). The following additional observation is needed. Consider the family RF of 2^k functions chosen at random from $\mathrm{Maps}(B,\{0,1\}^k)$. It can be seen, in a straightforward way, that in the black-box-family setting

$$\epsilon_{RF}(1,q,\tau) \leq \frac{c' \cdot \tau}{2^k} \tag{3}$$

for some $c' > 0$. (Here the probabilities are taken also on the choices of the functions in RF.) Altogether we now have

$$\frac{1}{\tilde{c}} \cdot (l-1)q \cdot \epsilon_{RF}(1,q,q) \quad \leq \quad \epsilon_{RF^{(*)}}(l,q,q) \tag{4}$$

and

$$\epsilon_{RF^{(*)}}(l,q,q) \quad \leq \quad \tilde{c} \cdot lq \cdot \epsilon_{RF}(1,q,O(lq)). \tag{5}$$

for some $\tilde{c} > 0$. (Inequalities (3) and (2) imply (4). Inequality (1) implies (5)).

THE DISTINGUISHING ALGORITHM. The idea is to capitalize on the difference between the probabilities of collisions in random functions and in functions obtained via the cascade construction. We will ask q queries that differ only in the first block out of the l blocks. (I.e, we choose $x_2, \ldots, x_l$ and $a_1, \ldots, a_q$ where each $x_i, a_i \in B$, and ask queries $m_1, \ldots, m_q$ where $m_j = a_j, x_2, \ldots, x_l$.) Output 'cascaded' iff any two queries resulted in the same answer. For the analysis, note that if the oracle is a random function, then the probability of such collisions is $O(\frac{q^2}{2^k})$. In a cascaded construction, however, the probability of collisions is much higher: consider two queries $m = a, x_2, \ldots, x_l$ and $m' = a', x_2, \ldots, x_l$. If $F^{(*)}(a, x_2, \ldots, x_k) = F^{(*)}(a', x_2 \ldots, x_k)$ for *any* $k \leq l$, then also $F^{(*)}(m) = F^{(*)}(m')$. The probability of collision after k blocks, given that no collision occurred yet, is $\Omega(\frac{q^2}{2^k})$. Thus collisions happen with probability $\Omega(\frac{lq^2}{2^k})$. Inequality (1) follows.

We remark that this idea can be extended in a straightforward way to breaking message authentication codes (see [5] for definitions) based on the cascade construction of pseudorandom functions. We demonstrate the technique on messages of two blocks: first choose an arbitrary block $b \in B$. Next, keep asking MACS for messages of the form $a_i b$ where the a_i's are random blocks, until $a_i \neq a_j$ are found such that MAC$(a_i b)$ =MAC$(a_j b)$. Now, with significant probability MAC(a_i) =MAC(a_j) as well. This is exploited as follows: Choose $c \in B$, ask for $m \stackrel{\text{def}}{=}MAC(a_i c)$, and output the message-tag pair $(a_j c, m)$. If indeed MAC(a_i) =MAC(a_j) then also MAC$(a_j c) = m$. (We note that this attack applies to a variety of well known MAC schemes, including DES-CBC-MAC. A similar attack was independently reported by [PV].)

Acknowledgments

We thank Moni Naor for helpful comments.

References

[1] W. AIELLO AND R. VENKATESAN, "Foiling birthday attacks in length doubling transformations," *Advances in Cryptology – Eurocrypt 96 Proceedings*, Lecture Notes in Computer Science Vol. 1070, U. Maurer ed., Springer-Verlag, 1996.

[2] ANSI X3.106, "American National Standard for Information Systems — Data Encryption Algorithm — Modes of Operation," American National Standards Institute, 1983.

[3] M. BELLARE, R. CANETTI AND H. KRAWCZYK, "Keying hash functions for message authentication", *Advances in Cryptology – Crypto 96 Proceedings*, Lecture Notes in Computer Science Vol. 1109, N. Koblitz ed., Springer-Verlag, 1996.

[4] M. BELLARE, R. GUÉRIN AND P. ROGAWAY, "XOR MACs: New methods for message authentication using finite pseudorandom functions," *Advances in Cryptology – Crypto 95 Proceedings*, Lecture Notes in Computer Science Vol. 963, D. Coppersmith ed., Springer-Verlag, 1995.

[5] M. BELLARE, J. KILIAN AND P. ROGAWAY, "The security of cipher block chaining." *Advances in Cryptology – Crypto 94 Proceedings*, Lecture Notes in Computer Science Vol. 839, Y. Desmedt ed., Springer-Verlag, 1994.

[6] M. BELLARE AND P. ROGAWAY, "Entity authentication and key distribution," *Advances in Cryptology – Crypto 93 Proceedings*, Lecture Notes in Computer Science Vol. 773, D. Stinson ed., Springer-Verlag, 1993.

[7] L. Carter and M. Wegman, "Universal Hash Functions". J. of Computer and System Science 18, 1979, pp. 143–154.

[8] I. Damgård, "A design principle for hash functions," *Advances in Cryptology – Crypto 89 Proceedings*, Lecture Notes in Computer Science Vol. 435, G. Brassard ed., Springer-Verlag, 1989.

[9] O. Goldreich, S. Goldwasser and S. Micali, "How to construct random functions," *Journal of the ACM,* Vol. 33, No. 4, 210–217, (1986).

[10] O. Goldreich, R. Impagliazzo, L. Levin, R. Venkatesan, and R. Zuckerman, D., "Security Preserving Amplification of Hardness." *Proceedings of the 31st Symposium on Foundations of Computer Science*, IEEE, 1990.

[11] J. Håstad, R. Impagliazzo, L. Levin and M. Luby, "Construction of a pseudo-random generator from any one-way function," Manuscript. Earlier versions in STOC 89 and STOC 90.

[12] A. Herzberg and M. Luby, "Public Randomness in Cryptography." *Advances in Cryptology – Crypto 92 Proceedings*, Lecture Notes in Computer Science Vol. 740, E. Brickell ed., Springer-Verlag, 1992.

[13] M. Luby and C. Rackoff, "How to construct pseudorandom permutations from pseudorandom functions," *SIAM J. Computation,* Vol. 17, No. 2, April 1988.

[14] R. Merkle, "One way hash functions and DES," *Advances in Cryptology – Crypto 89 Proceedings*, Lecture Notes in Computer Science Vol. 435, G. Brassard ed., Springer-Verlag, 1989. (Based on unpublished paper from 1979 and his Ph. D thesis, Stanford, 1979).

[15] M. Naor and O. Reingold, "Synthesizers and their application to the parallel construction of pseudo-random functions," *Proceedings of the 36th Symposium on Foundations of Computer Science*, IEEE, 1995.

[PV] B. Preneel and P. van Oorschot, "MD-x MAC and building fast MACs from hash functions," *Advances in Cryptology – Crypto 95 Proceedings*, Lecture Notes in Computer Science Vol. 963, D. Coppersmith ed., Springer-Verlag, 1995.

[16] R. Rivest, "The MD5 message-digest algorithm," IETF Network Working Group, RFC 1321, April 1992.

[17] P. Rogaway and D. Coppersmith, "A software optimized encryption algorithm," Workshop on software encryption, Cambridge, 1993.

[18] FIPS 180, "Secure Hash Standard", Federal Information Processing Standard (FIPS), Publication 180, National Institute of Standards and Technology, US Department of Commerce, Washington D.C., May 1993.

[19] M. Wegman and L. Carter, "New Hash Functions and Their Use in Authentication and Set Equality", *JCSS* Vol. 22, 1981, pp. 265–279.

A Proof of Lemma 3.2

Let $qR \stackrel{\text{def}}{=} \text{Maps}([q] \times B, \{0,1\}^k)$, and let $R^{(l)} \stackrel{\text{def}}{=} \text{Maps}(B^{\leq l}, \{0,1\}^k)$. We construct an algorithm U: given black-box access to a prefix-free distinguisher D_3 that asks at most q queries each of at most l blocks and has $\mathsf{Adv}_{D_3}(F^{(l)}, R^l) = \epsilon_3$, U defines a distinguisher $D_2 \stackrel{\text{def}}{=} U^{D_3}$ that has $\epsilon_2 \stackrel{\text{def}}{=} \mathsf{Adv}_{D_2}(qF, qR) \geq \epsilon_3/l$. To avoid confusion, we refer to D_3 as the *black-box* of U, while the term *oracle* is reserved to the input function to be decided upon.

Construction of $D_2 = U^{D_3}$. Given oracles $g_1, \ldots, g_m\colon B^{\leq l} \to \{0,1\}^k$, pick $i \stackrel{R}{\leftarrow} [l]$ at random and start running D_3. The latter requires an oracle to a function $g\colon B^{\leq l} \to \{0,1\}^k$. D_2 will supply answers to the oracle queries of D_3. It begins by initializing a counters t to 0. The answer to the jth oracle query $x_1^j \ldots x_n^j \in B^{\leq l}$ is computed follows.

(1) If $n \leq i-1$ then pick at random an i-bit string a^j and return a^j.

(2) Else (namely $n \geq i$)

- (2.1) If $x_1^j \ldots x_{i-1}^j \neq x_1^p \ldots x_{i-1}^p$ for all $p < j$ then increment t and let $c^j = t$. Else let $c^j = c^p$ where p is such that $x_1^j \ldots x_{i-1}^j = x_1^p \ldots x_{i-1}^p$.
- (2.2) Using the oracle for $g_1, \ldots, g_q$ compute $a^j = g_{c^j}(x_i^j)$. Answer $F^{(l)}(a^j, x_{i+1}^j \ldots x_n^j)$ to D_3.

Finally, output whatever D_3 outputs.

Analysis of $D_2 = U^{D_3}$. We relate ϵ_2 to ϵ_3 via a hybrids argument, as follows. Define a sequence of hybrid function families. With each $i \in \{0,1,\ldots,l\}$ and each function $g\colon B^{\leq i} \to \{0,1\}^k$ we associate a function $h_i^g\colon B^{\leq l} \to \{0,1\}^k$, defined as follows for any $x_1 \ldots x_n \in B^{\leq l}$.

$$h_i^g(x_1 \ldots x_n) \stackrel{\text{def}}{=} \begin{cases} g(x_1 \ldots x_n) & \text{if } n \leq i \\ F^{(*)}(g(x_1 \ldots x_i), x_{i+1} \ldots x_n) & \text{otherwise.} \end{cases}$$

Define the family $H_i \stackrel{\text{def}}{=} \{h_i^g \;:\; g \in \text{Maps}(B^{\leq i}, \{0,1\}^k)\}$. We now define

$$P_i \stackrel{\text{def}}{=} \Pr_{f \stackrel{R}{\leftarrow} H_i}\left[D_3^f = 1\right] .$$

Claim A.1 $H_0 = F^{(l)}$ and $H_l = R'$. In particular:

$$\begin{aligned} \epsilon_3 &= \mathsf{Adv}_{D_3}(R^{(l)}, F^{(*)}) \\ &= \Pr_{f \stackrel{R}{\leftarrow} R'}\left[D_3^f = 1\right] - \Pr_{f \stackrel{R}{\leftarrow} F^{(l)}}\left[D_3^f = 1\right] \\ &= P_l - P_0 . \end{aligned} \tag{6}$$

Proof: If $i = 0$ then $x_1 \ldots x_i$ is the empty string λ. Thus h_0^g is $F_a^{(l)}$ where $a = g(\lambda)$. Since g is random, a is uniformly distributed. This means H_0 is indeed $F^{(l)}$. If $i = l$ then h_l^g is just g itself, so $H_l = R'$. Equation 6 of course follows. ∎

Next we show that if the oracle of D_2 is a random element of qR then D_3 "sees" an oracle chosen from H_i. Similarly if the oracle of D_2 is a random element of qF then D_3 "sees" an oracle chosen from H_{i-1}. More precisely, let $D_2(i)$ denote algorithm D_2 with a fixed $i \in [l]$. Consider four experiments. Experiment 1 (resp 2) is that of running $D^{g_1,\ldots,g_q}(i)$ with $g_1, \ldots, g_q \stackrel{R}{\leftarrow} R$ (resp. $g_1, \ldots, g_q \stackrel{R}{\leftarrow} F$). Experiment 3 (resp. 4) consists of running D_3^h with $h \stackrel{R}{\leftarrow} H_i$ (resp. H_{i-1}). We want to claim that

Experiments 1 and 3 are "equivalent" in the sense that what D_3 "sees" at any point is the same (ie. identically distributed) in these two experiments. Similarly for Experiments 2 and 4.

For this purpose, denote by X_s the random variable whose value is the s-th oracle query. Let Y_s denote the answer that is given to X_s. We now fix a particular sequence of queries $x^1, \ldots, x^j$ and answers $y^1, \ldots, y^{j-1}$. For $t = 1, 2, 3, 4$ we let Distribution t refer to the distribution given by Experiment t conditioned on the events $X_s = x^s$ for $s = 1, \ldots, j$ and $\mathsf{Y}_s = y^s$ for $s = 1, \ldots, j-1$. (Namely, D_3 has posed questions $x^1, \ldots, x^{j-1}$ and obtained answers $y^1, \ldots, y^{j-1}$; it has also posed question x^j but as yet got back no answer.) We write $\Pr_t[\cdot]$ for the probability under this (conditional) distribution. Now the above "equivalence" claims amount to the following.

Claim A.2 For any k bit string y—

(1) $\Pr_1[\mathsf{Y}_s = y] = \Pr_3[\mathsf{Y}_s = y]$

(2) $\Pr_2[\mathsf{Y}_s = y] = \Pr_4[\mathsf{Y}_s = y]$.

Proof: First we prove Part (1). Take the case that the number of blocks n in the current query $x^j = x_1^j \ldots x_n^j$ is at most $i-1$. The distinguisher D_3 is prefix-free so in particular x^j does not occur as a previous query. So the value $g(x^j)$ returned in Experiment 3 is a new random k bit string, independent of all previous choices. But that is exactly what is returned in Experiment 1. So the claim is true in this case. We now move on to the more interesting case, namely $n \geq i$.

In this case, in both Experiments 1 and 3, the answer returned to query $x^j = x_1^j \ldots x_n^j$ is computed by first computing, as a certain function of $x_1^j, \ldots, x_i^j$, a value we denote A_j, and using A_j as a key to return $F^{(*)}(\mathsf{A}_j, x_{i+1}^j \ldots x_n^j)$. It suffices to show that the random variable A_j is identically distributed in the two experiments. So now let us look at how it is computed in the two experiments.

In Experiment 3, $\mathsf{A}_j = g(x_1^j \ldots x_i^j)$. This means that if the set $T = \{ p < j \,:\, x_1^j \ldots x_i^j = x_1^p \ldots x_i^p \}$ is non-empty then $\mathsf{A}_j = \mathsf{A}_p$ for all $p \in T$. Otherwise A_j is uniformly and independently distributed. We now argue the same is true in Experiment 1. Let $S = \{ p < j \,:\, x_1^j \ldots x_{i-1}^j = x_1^p \ldots x_{i-1}^p \}$. First, our construction of D_2 guarantees that if $S \neq \emptyset$ then there is a single value, which we here denote v, such that $c^p = v$ for all $p \in S$. Now,

- If $S \neq \emptyset$ then we return $g_v(x_i^j)$. If $T \neq \emptyset$ then $g_v(x_i^j) = \mathsf{A}_p$ for all $p \in T$. Else ($T = \emptyset$), $g_v(x_i^j)$ is a uniformly distributed k bit string because g_v has never before been invoked on x_i^j.
- If $S = \emptyset$ then we return the value on x_i^j of a random function g_t which has never before been invoked. So this is a uniformly distributed k bit string.

We now turn to proving Part (2). The case $n \leq i-1$ is identical to Part (1). Assume $n \geq i$, and consider the j-th query $x_1^j \ldots x_n^j$. Let A_j be the same random variable as above. But now in both experiments A_j is computed via the family F. Let B_j be the random variable such that $\mathsf{A}_j = F(\mathsf{B}_j, x_i^j)$. We show that, as long as the query $x_1^j \ldots x_{i-1}^j$ is not made, B_j is identically distributed in the two experiments. (If $x_1^j \ldots x_{i-1}^j$ is queried then the two experiments differ: Let y denote the answer D_3 gets to query $x_1^j \ldots x_{i-1}^j$. In Experiment 4 $\mathsf{B}_j = y$, whereas in Experiment 2 B_j and y are independently (and uniformly) distributed values in $\{0,1\}^k$.)

Let the set S and the value v be as before. In Experiment 4, $\mathsf{B}_j = g(x_1^j \ldots x_{i-1}^j)$. This is equal to B_p for all $p \in S$ if $S \neq \emptyset$, and otherwise is uniformly distributed. We argue that the same it true for Experiment 2. Recall that $g_1, \ldots, g_m$ are now drawn randomly from F. Accordingly let $\mathsf{Key}(i)$ be the random variable whose value is the key of g_i; ie. $g_i = F_{\mathsf{Key}(i)}$. Now

- If $S \neq \emptyset$ then we return $g_v(x_i^j)$. Thus $\mathsf{B}_j = \mathsf{Key}(v)$. Thus $\mathsf{B}_j = \mathsf{B}_p$ for all $p \in S$.

– If $S = \emptyset$ then we return the value on x_i^j of a random function g_t which has never before been invoked. So $\mathsf{B}_j = \mathsf{Key}(t)$ is a uniformly distributed k bit string.

This concludes the proof. ▮

Concluding the proof given Claim A.2. Repeated application of Part (1) of the above implies that the output distribution of D_2 in Experiment 1 is equal to the output distribution of D_3 in Experiment 3, that is to P_i. Similarly for Experiments 2 and 4 using Part (2) of the above. That is

$$\begin{aligned} \Pr_{h \stackrel{R}{\leftarrow} qR}\left[D_2^h(i) = 1\right] &= P_i \\ \Pr_{h \stackrel{R}{\leftarrow} qF}\left[D_2^h(i) = 1\right] &= P_{i-1} \ . \end{aligned}$$

Now recall that D_2 picks $i \stackrel{R}{\leftarrow} [m]$. Thus:

$$\Pr_{h \stackrel{R}{\leftarrow} qR}\left[D_2^h = 1\right] = (1/l) \cdot \textstyle\sum_{i=1}^l P_i \tag{7}$$

$$\Pr_{h \stackrel{R}{\leftarrow} qF}\left[D_2^h = 1\right] = (1/l) \cdot \textstyle\sum_{i=1}^l P_{i-1} \ . \tag{8}$$

So $\epsilon_2 = \mathsf{Adv}_{D_2}(qF, qR) = (1/l) \cdot (P_l - P_0) = (1/l) \cdot \epsilon_3$.

B Proof of Lemma 3.3

Also here we use a hybrids argument. Let $R \stackrel{\text{def}}{=} \text{Maps}(B, \{0,1\}^k)$, and let $mR \stackrel{\text{def}}{=} \text{Maps}([m] \times B, \{0,1\}^k)$. We construct an algorithm U: given black-box access to a prefix-free distinguisher D_2 with $\epsilon_2 \stackrel{\text{def}}{=} \mathsf{Adv}_{D_2}(mF, mR)$, U defines a distinguisher $D_1 \stackrel{\text{def}}{=} U^{D_2}$ with $\epsilon_1 \stackrel{\text{def}}{=} \mathsf{Adv}_{D_1}(qF, qR) \geq \epsilon_2/m$.

Algorithm D_1, given oracle g, first picks at random an integer $i \stackrel{R}{\leftarrow} [m]$. Next it proceeds, informally, as follows. It "picks," randomly and independently, functions $g_1, \ldots, g_{i-1}$ from F. It "sets" g_i to its oracle function g. It also "picks random functions" $g_{i+1}, \ldots, g_m$ from R. It now runs D_2 with oracles $g_1, \ldots, g_m$: when the query is to oracle j it answers via g_j, invoking its oracle when $j = i$ and otherwise using one of its chosen functions. It outputs whatever this computation outputs.

To pick $g_1, \ldots, g_{i-1}$ from F meant to pick randomly $i-1$ keys $a_1, \ldots, a_{i-1} \stackrel{R}{\leftarrow} \{0,1\}^k$ and reply to query (j, x), for $j < i$, by $F(a_j, x)$. Furthermore, D_1 of course can't really "pick random functions". What it does is that each time D_2 asks a query (j, x) with $j > i$ it picks a random k bit string if (j, x) was not asked before, returns it, and also records it; else it has already picked a reply so it looks up its record and returns this reply.

In the worst case D_1 still has to make $q_1 = q_2$ queries to its oracle g (although on the average it is q_2/m). Its running time t_1 is that of D_2 plus the time to pick the extra keys, which is $O(mk)$, and to record and respond to queries, which involves computation of F_{a_j} and comes to $O(q_1 \cdot \text{Time}(F) + q_1 k \log q_1)$. Altogether this comes to $t_1 = t_2 + c \cdot (mk + q_1 \cdot \text{Time}(F) + q_1 k \log q_1)$.

For $i = 0, \ldots, m$ let $G_i = F \times \cdots \times F \times R \times \cdots \times R$ where there are i copies of F and $m - i$ copies of R. To pick a vector $(g_1, \ldots, g_m)$ of functions at random from this space means the obvious thing, namely to pick g_j at random from F for $j = 1, \ldots, i$ and at random from R for $j = i+1, \ldots, m$. We let

$$P_i = \Pr_{(g_1, \ldots, g_m) \stackrel{R}{\leftarrow} G_i}\left[D_2^{g_1, \ldots, g_m} = 1\right]$$

be the probability that D_2 outputs one when its oracles are chosen randomly from G_i. Let $D_1(i)$ denote the operation of D_1 with fixed i. Now note that for $i = 1, \ldots, m$:

$$\begin{aligned} \Pr_{g \xleftarrow{R} R}[\, D^g(i) = 1\,] &= P_{i-1} \\ \Pr_{g \xleftarrow{R} F}[\, D^g(i) = 1\,] &= P_i \, . \end{aligned}$$

Thus

$$\begin{aligned} \Pr_{g \xleftarrow{R} R}[\, D_1^g = 1\,] &= (1/m) \cdot \textstyle\sum_{j=i}^{m} P_{j-1} \\ \Pr_{g \xleftarrow{R} F}[\, D_1^g = 1\,] &= (1/m) \cdot \textstyle\sum_{j=1}^{m} P_j \, . \end{aligned}$$

Thus the advantage of D_1 is:

$$\begin{aligned} \mathsf{Adv}_{D_1}(F, R) &= \Pr_{g \xleftarrow{R} F}[\, D_1^g = 1\,] - \Pr_{g \xleftarrow{R} R}[\, D_1^g = 1\,] \\ &= (1/m) \cdot (P_m - P_0) \, . \end{aligned}$$

However now observe that

$$\begin{aligned} P_m &= \Pr_{g_1, \ldots, g_m \xleftarrow{R} F}[\, D_2^{g_1, \ldots, g_m} = 1\,] \\ &= \Pr_{g \xleftarrow{R} mF}[\, D_2^g = 1\,] \end{aligned}$$

and

$$\begin{aligned} P_0 &= \Pr_{g_1, \ldots, g_m \xleftarrow{R} R}[\, D_2^{g_1, \ldots, g_m} = 1\,] \\ &= \Pr_{g \xleftarrow{R} mR}[\, D_2^g = 1\,] \, . \end{aligned}$$

Thus $\mathsf{Adv}_{D_1}(F, R) \geq (1/m) \cdot \mathsf{Adv}_{D_2}(mF, mR) = \epsilon_2 / m$ as required.

C Proof of Theorem 4.1

Let X and Y be the domain and range of the family F, and let $R \stackrel{\text{def}}{=} \text{Maps}(X, Y)$. We construct an algorithm U: given black-box access to a prefix-free distinguisher D_4 with $\epsilon_4 \stackrel{\text{def}}{=} \mathsf{Adv}_{D_4}(\delta\text{-A}F, R)$, U defines a distinguisher $D_3 \stackrel{\text{def}}{=} U^{D_4}$ with $\epsilon_3 \stackrel{\text{def}}{=} \mathsf{Adv}_{D_3}(F, R) \geq \epsilon_4 - blq2^{-\delta}$.

Distinguisher D_3, with oracle access to a function g (where g is chosen at random either from F or from R), will proceed as follows. First pick $d \xleftarrow{R} \{0,1\}^\delta$. Next, run D_4 where each query x of D_4 is answered by $g(xd)$. If any of the queries of D_4 contains d as a substring then output '1' (meaning 'g is pseudorandom'). Otherwise, output whatever D_4 outputs.

For the analysis, let

$$\begin{aligned} P_{4,\text{R}} &= \Pr_{g \xleftarrow{R} R}[\, D_4^g = 1\,] \\ P_{4,\text{F}} &= \Pr_{g \xleftarrow{R} \delta\text{-A}F}[\, D_4^g = 1\,] \\ P_{3,\text{R}} &= \Pr_{g \xleftarrow{R} R}[\, D_3^g = 1\,] \\ P_{3,\text{F}} &= \Pr_{g \xleftarrow{R} F}[\, D_3^g = 1\,] \, . \end{aligned}$$

Now, if g, the oracle of D_3, is taken at random from R, then the answers given by D_3 to D_4's queries are those of a random function. That is, any two different queries are answered by independently chosen values from Y. Furthermore, if $g \stackrel{R}{\leftarrow} F$ then the answers given by D_3 to D_4's queries are those of a function $g \stackrel{R}{\leftarrow} \delta\text{-A}F$. Thus,

$$P_{4,\mathrm{F}} - P_{4,\mathrm{R}} \geq \epsilon_4.$$

When $g \stackrel{R}{\leftarrow} R$, the probability that any one of D_4's queries contains d as a substring is at most the probability that a *random* value in $\{0,1\}^\delta$ appears as a substring in a *given* set of q strings each of length bl. This probability is less than $blq2^{-\delta}$. Thus, $P_{3,\mathrm{R}} \leq P_{4,\mathrm{R}} + blq2^{-\delta}$. On the other hand, whenever D_4 outputs '1', D_3 outputs '1' as well. Thus, $P_{3,\mathrm{F}} \geq P_{4,\mathrm{F}}$. We conclude that

$$\epsilon_3 \geq P_{3,\mathrm{F}} - P_{3,\mathrm{R}} \geq P_{4,\mathrm{F}} - P_{4,\mathrm{R}} + lq2^{-\alpha} \geq \epsilon_4 - blq2^{-\delta}.$$

The other parameters of the reduction can be easily verified.