This is a very preliminary draft. Please do not distribute!

Cascaded Pseudo-Randomness and its Concrete Security (DRAFT)

Mihir Bellare* Ran Canetti† Hugo Krawczyk‡

November 10, 1995

Abstract

In this paper we study *cascaded pseudorandomness*, a notion derived from a methodolgy introduced by Merkle for the design of collision-resistant hash functions, and applied in our case for the construction of practical pseudorandom functions based on "cryptographic hash functions." By following this approach we are able to exhibit very efficient constructions based on some widely used and efficient cryptographic hash functions that follow Merkle's methodology. Examples of such hash functions are Rivest's MD5 and the US standard hash function, SHA.

It turns out that coming up with these constructions, and analyzing them, presents interesting and wider theoretical issues. In particular, it involves issues like the expansion of a finite function (defined on a particular finite domain and finite range) into one that can work over arbitrary domains, the concrete security analysis of these constructions (we are interested in the security of these functions relative to the concrete finite parameters, rather than the asymptotical behavior as traditionally approached in complexity theory), and the introduction of new analytical tools.

This work has been motivated by the search of practical primitives for pseudorandom functions and message authentication that are acceptable to the applied cryptographic world. In particular, for the increased and fast deployment of cryptography through Internet (and other) standards.

* 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.

†LCS, MIT. Supported by a post-doctoral grant from the Rothschild Foundation. Email: canetti@theory.lcs.mit.edu.

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

1 Introduction

In 1992, Rivest devised a cryptographic function which he called md5 [Ri]. It is a single, finite function which takes a 128 bit "key" a and a 512 bit input x, and returns a 128 bit output which we write $\mathsf{md5}_a(x)$.[1] This "compression" function, and others like it (cf. [SHA]), are quite intriguing. They are much more efficient to compute than familiar number-theoretic cryptographic functions, which typically require modular exponentiation. Yet they seem to have some sort of quite appreciable "cryptographic hardness." This opens up the attractive possibility of obtaining efficient but secure protocols based on compression functions.

Practitioners were quick to realize this, and have been using compression functions to design schemes for a variety of ends (e.g., message authentication and pseudorandom generation) which these functions were not initially designed to support. Unfortunately, however, these uses have not been backed by a solid security analysis. Moreover, many of the recent constructions that utilize compression functions cannot be justified based on any properties the designers suggested these functions had, or even on any natural property that we may presume these functions possess. (We will see examples later.) However, not only are the schemes proposed, but widely used and, in particular, being considered for Internet Standards. This is dangerous for the security of the protocols that utilize these schemes, and a concern that needs to be addressed.

THIS WORK. This work provides a new body of compression function based constructions which are *theoretically sound* and *practically useful.* In particular we introduce the cascade construction, an extension of Merkle's classic methodology for the design of collision-free hash functions, and show that it is a secure and efficient way of constructing a number of primitives including pseudo-random functions and message authentication codes.

Our results, on the one hand, serve to guide current uses of compression functions, and provide specific schemes to enhance and replace existing ones, in particular for message authentication. On the other hand, by increasing our understanding of the underlying functions, they guide future design of these functions. In addition, as we'll see, this research has lead us to some questions of wider theoretical appeal.

We will not be concerned with any specific compression function amongst the many examples; rather we wish to appropriately model the properties of all such functions, then suggest constructions based on this model and analyze their security.

In summary, these compression functions provide an exciting possibility for efficient, secure cryptography. But to attain it, we need to understand them better, and find the "right" ways to harness their power. The results in this paper are a step towards that end.

1.1 Finite Pseudorandom Functions

In general a compression function is a map $F(\cdot,\cdot)$ taking a key a, of some length κ, and an input x, of some length b called the block size, and returning a string $y = F(a,x)$, of some output length L. To work with such a function we first need to decide how to formally model its cryptographic properties. A standard assumption is that F is collision-free (i.e., it is infeasible to find different pairs (a_1,x_1) and (a_2,x_2) such that $F(a_1,x_1) = F(a_2,x_2)$). As shown by Merkle this is enough when one's goal is collision-free hashing, but we are interested in other goals like message authentication and encryption, which are "keyed" primitives, i.e., functions that involve the use of a secret key.

We can regard F as defining a keyed family of finite functions, as follows. Define $F_a(x) = F(a,x)$. Then the family is $F = \{F_a\}_{|a|=\kappa}$. Our suggestion is to assume that F is a finite pseudo-random function (FPRF) family [BKR, BGR] (see definition below). This assumption seems to be natural to make from what we know of compression functions and how we intend to use them.

[1] We are referring to what is called the "compression function" of Rivest's MD5 primitive. Indeed, it is actually this compression function which is the basic primitive; MD5 is built by iterating md5 in a natural way (as explained below).

FPRF FAMILIES. A FPRF family is a finite version of the pseudorandom function (PRF) families of [GGM], and were first considered in [BKR]. It consists of a collection of maps $G = \{G_a\}_{|a|=\kappa}$ in which each κ-bit *key* a specifies a particular function G_a: Dom$(G) \to$ Range(G). (The domain and range depend only on the family. We work with inputs partitioned into size b blocks, so that Dom$(G) \subseteq B^*$, where $B = \{0,1\}^b$.) Often we write $G(a,x)$ for $G_a(x)$. G is said to be (t,q,l,ϵ)-secure if a circuit of size t, given an oracle for a function g: Dom$(G) \to$ Range(G) and allowed to make at most q queries to this oracle, each of at most l blocks, has advantage at most ϵ in telling whether g is a random member of the family or a truly random function of Dom(G) to Range(G). We refer to this as the *concrete security* of the family G. Notice that there are no "asymptotics" in this definition. This difference with the traditional complexity-based definitions is crucial. FPRF families are weaker than PRF families, and harder to work with. On the other hand, they model better the objects we are interested in here as well as other objects encountered in practical cryptography.[2]

We will now assume that we have a FPRF family $F = \{F_a\}$ with Dom$(F) = B = \{0,1\}^b$ and Range$(F) = \{0,1\}^L$ which is $(t,q,1,\epsilon)$-secure, for some specific values t,q,ϵ considered to be given. (In practice these values depend on the known cryptanalytic strength of the specific function being modeled.) Furthermore, we assume that the key length and output length are the same, $\kappa = L$. (Eg. $b = 512$ and $L = \kappa = 128$ for md5.)

MESSAGE AUTHENTICATION. An application in which we are particularly interested is to message authentication (MA). A MA scheme (usually referred to as MAC – for message authentication code) has a sender and receiver who share a key a. The sender authenticates his message M by attaching a tag $\tau = MAC_a(M)$ computed as a function of the key a and M. (We stress that the message M can be of arbitrary length.) The receiver has a corresponding verification function which takes a, M, τ and says "OK" or "NOT OK." The scheme is (t,q,l,ϵ)-secure if an adversary who is a circuit of size t and is allowed to see up to q valid message-tag pairs (possibly, with messages chosen by the adversary) of messages of at most l blocks, succeeds in forgery with probability at most ϵ (where forgery means computing a valid message-tag pair for a message not included among the q queries).

1.2 Using FPRFs for authentication

Suppose we are given a FPRF family F as above and we want to design a message authentication scheme. The main problem we face is that F takes only b bit inputs, and we have a long message M.

There is, however, a well-known construction of Merkle for extending the domain of a function $F(\cdot,\cdot)$. Merkle was interested in cryptographic hashing, and suggested that if $F(\cdot,\cdot)$ is collision free, then to hash $M = x_1 x_2 \ldots x_n$, we can compute—

$$y_1 = F_{y_0}(x_1),\ \ y_2 = F_{y_1}(x_2),\ \ y_3 = F_{y_2}(x_3),\ \ \ldots,\ \ y_n = F_{y_{n-1}}(x_n)\ , \tag{1}$$

and output y_n. Here y_0 is a random but fixed string called the "starting value" which is known to everyone (i.e., part of the definition of the function). We call this function F^{hash}. [3]

So people suggested that in order to use this function for message authentication one just needs to somehow involve the (secret) MAC key in F^{hash}. In particular, it has been suggested to let $MAC_a(M) = F^{\mathsf{hash}}(M\,.\,a)$. Unraveling, this means that to compute the MAC we compute—

$$y_1 = F_{y_0}(x_1),\ \ y_2 = F_{y_1}(x_2),\ \ y_3 = F_{y_2}(x_3),\ \ \ldots,\ \ y_n = F_{y_{n-1}}(x_n),\ \ y_{n+1} = F_{y_n}(a)\ , \tag{2}$$

[2] In particular, the difference with respect to traditional pseudorandom functions[GGM] is that the latter are represented as a family of families, one for each value of a security parameter k, and the k-th family has domain and range $\{0,1\}^k$, so that we automatically have functions of any desired input length (equivalently, these functions are modeled as a single family of functions from $\{0,1\}^*$ to $\{0,1\}^*$, and the security parameterized via the input length). For us, the domain is finite, say $\{0,1\}^{512}$, and, therefore, we need to specify how to apply these functions to inputs of different lengths.

[3] With this notation one gets $\mathsf{MD5} = \mathsf{md5}^{\mathsf{hash}}$, namely, MD5 is the function obtained by iterating the md5 compression function according to Eq. 1. (We note that the full definition of MD5 specifies also a particular way to pad the input to a 512-bit block boundary, which includes appending the input length to the message.)

and output y_{n+1} as the tag.

Is this a good method? We think not, because there doesn't seem to be any way to justify it based on a reasonable assumption about the underlying F. In particular, even assuming that F is a FPRF family, this scheme cannot be proven secure. (This claim can be justified as follows. Let G be an arbitrary FPRF family. Then it is possible to construct F which is as also a FPRF family, as secure as G, but for which the MAC of y_0, under the above scheme, is exactly the key a. We omit this construction from this abstract.)

We clarify that the original Merkle construction was for the purpose of hashing, and can be proven to meet this goal (i.e., if $F(\cdot,\cdot)$ is collision-free then so is F^{hash}). But the above is not a good way to key it to get a MAC.

We want to suggest a different construction. But a key point is that we want to suggest a construction which is *not too different,* namely, a construction that is as efficient as F^{hash} and does not require any significant changes to the F^{hash} definition. Thus our question is: is there a simple way to "key" the Merkle construction so that it is a good MAC?

Actually, we shoot for something stronger, namely to "key" Merkle in such a way that the resulting construction is a pseudo-random function in its own right. (That is, we would extend F to a PRF on the domain B^*.) The reason to do this is that (F)PRFs make good MACs, but also have many other uses.

This turns out to be an interesting question. It lead us to the cascade constructions and their analysis, which we found to be of independent theoretical interest.

1.3 Previous and related work

Before we discuss our results, we should clarify some things. We have said our goal is to construct PRFs. Someone might ask why this is a question at all. After all, isn't it known that you can construct a PRF given any one-way function? Yes it is known [HILL, GGM], but that construction cannot be used here. We are starting with a FPRF family of some given concrete security, something which is actually weaker than a one-way function.[4] Also the one-way function based constructions are very in-efficient, and in particular un-usable in practice. To find alternative more efficient ways was after all the entire point of interest of FPRFs.

The problem of constructing provably good PRFs and MA schemes out of FPRF families was first considered in [BKR]. They analyzed the concrete security of the well-known cipher block chaining construction. Later another method called XOR-MACs, suited for parallel implementation, was suggested and analyzed in [BGR]. Thus it is known that there are ways to build PRFs and MA schemes out of FPRFs. But our goal is to find the "right" way to key Merkle, not do something completely different. The justification here is three-fold. First, we believe the cascade construction that follows is a natural and useful extension of Merkle's scheme and ought to be understood. Second, certain practical constraints are better met by cascades. (In particular our constructions yield the fastest known MA schemes based on FPRFs for software implementation. In contrast the methods of [BKR, BGR] are good for hardware but slower than ours in software.) Finally, the techniques involved here are very different from previous works, and interesting in their own right.

Another related work is Naor and Reingold [NaRe] who construct PRFs out of a primitive they call "synthesizers." However the candidate synthesizers they suggest are number-theoretic, so, again, our constructions will be faster. Furthermore they don't consider the concrete security.

[4] The one-way functions used in [HILL] are infinite functions which apply to inputs of any length. They can't be constructed out of the finite functions we have. One might suggest we could define a notion of finite one-way functions, convert FPRFs to this, and then try to apply [HILL, GGM], but we would find that because of the complexity of the construction, we would loose so much security that for any existing candidate FPRFs we would be left with nothing.

1.4 The deterministic cascade construction

We key the Merkle construction in a natural way— just replace the public "starting value" with the shared private key. Actually we need to do a little more: as in the scheme of Equation 2 we also need to append some key material. So the final construction has a key with two parts, $a\,.\,K$. On input $x_1 \ldots x_n$ we compute—

$$y_1 = F_a(x_1),\;\; y_2 = F_{y_1}(x_2),\;\; y_3 = F_{y_2}(x_3),\;\; \ldots,\;\; y_n = F_{y_{n-1}}(x_n),\;\; y_{n+1} = F_{y_n}(K)\,, \tag{3}$$

and output y_{n+1}. This defines a function of B^* to $\{0,1\}^L$ which we call F^{csc}_{aK}. (Recall $B = \{0,1\}^b$ is the domain of F.)

We prove that, under the assumption that F is a FPRF family, so is F^{csc}. (It follows that it is a secure MA scheme.) In other words, *we have been able to extend F to a larger domain while preserving pseudorandomness.* Let us now discuss two important aspects of this result and its proof.

A NON-STANDARD CONSTRUCTION. There is a technical feature of the cascade construction which makes it non-standard and thus harder to analyze. We now try to explain it. Roughly, the property is that the construction has no "information theoretic" analogue. By this we mean that the construction is not a direct transformation of F_a, but rather involves the underlying family F. Thus, if you are given a single random function f, there is no natural analogous way to transform f. This makes the proof harder: the typical paradigm is to study the information theoretic analogue, and this won't work. (This discussion is probably hard to understand. Perhaps it will be clearer when we present our proof.)

THE CONCRETE SECURITY. When dealing with finite objects such as the ones we are modeling, it isn't enough to say that a family is a FPRF. What is important is exactly what is the security. What we show is that if F was $(t', q, 1, \epsilon')$-secure then F^{csc} is (t, q, l, ϵ)-secure for t comparable to t' and and $\epsilon = qn\epsilon'$, where l is the maximum number of blocks in any queried message.

1.5 The randomized cascade construction

As we indicated above, the concrete security is crucial. It isn't just a question of "secure" or "not secure"— one scheme can be quantitatively "more secure" than another. In particular, we now suggest a modified cascade construction which has better security.

The idea is to randomize the cascade. It turns out that this randomization obviates the need to append key material, so now the key is just a. On input $M = x_1 \ldots x_n$, pick a random l bit string R and compute—

$$y_1 = F_a(R),\;\; y_2 = F_{y_1}(x_1),\;\; y_3 = F_{y_2}(x_2),\;\; \ldots,\;\; y_n = F_{y_{n-1}}(x_{n-1}),\;\; y_{n+1} = F_{y_n}(x_n)\,. \tag{4}$$

Output (R, y_{n+1}). We call this randomized function F^{rcsc}_a.

We can't quite consider whether F^{rcsc} is a FPRF family because, being randomized, it is not a function family. But we can consider "randomized" FPRF 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 $\epsilon = \epsilon'(q) + nq\epsilon'(1) + \sum_{i \geq 2} \frac{q^i}{2^{(i-1)l}} \epsilon'(i)$. This is a significantly better bound that the one we obtained for F^{csc} which gives (using the above notation) $\epsilon = nq\epsilon'(q)$. In particular, we obtain a MA scheme of the same security.

1.6 On the tightness of analysis

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 attack is very general, and requires no a-priori information about the underlying family. 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 MAC schemes based on the cascaded construction.

We combine this attack with a simple upper bound on the distinguishability distance ϵ of random families, to show that our analysis of the cascaded construction is 'almost' optimal for the case that no a-priori information on the underlying function family is known.

2 Definitions

We need somewhat more general definitions than those presented in Section 1.1.

Notation. Let λ denote the empty string. 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 uniformly at random according to the distribution underlying S (resp. uniformly at random from the set S). Furthermore $x_1, \ldots, x_m \stackrel{R}{\leftarrow} S$ abbreviates $x_1 \stackrel{R}{\leftarrow} S\,;\ \ldots\,;\ x_m \stackrel{R}{\leftarrow} S$.

We will work over a particular block size b. This means the inputs to our functions are strings from the set B^*, where $B = \{0,1\}^b$ is the set of possible blocks.

Finite function families. A (finite) *function family* is a set of functions F together with a probability distribution on these functions. Thus $f \stackrel{R}{\leftarrow} F$ denotes the operation of selecting a function at random from F according to the underlying distribution.

We consider "keyed" function families. This means there is some fixed convention via which each κ bit "key" a "names" a function in F, for some integer κ called the key length. Note that different keys can name the same function. We use several notations for the function denoted by the key a. Sometimes we write it at F_a; other times we view F as a two argument function and write it as $F(a, \cdot)$. The distribution on F is that given by picking a key at random from $\{0,1\}^\kappa$ and selecting the corresponding function; namely, $a \stackrel{R}{\leftarrow} \{0,1\}^\kappa\,;\ f \leftarrow F_a$. We require that given the key it is possible to compute the corresponding function. In particular we let $\mathrm{Time}(F)$ denote the time to compute $F_a(x) = F(a,x)$ given a and x.

The functions in a family F are assumed to have a common domain denoted $\mathrm{Dom}(F)$ and a common range denoted $\mathrm{Range}(F)$. In the families we consider the range will always be $\{0,1\}^L$ for some L, called the *output length.* Furthermore, the key length κ will be equal to L. The domain $\mathrm{Dom}(F)$ can vary; we will consider B, B^*, B^n, and $B^{\leq n}$ (i.e., of strings of at most n blocks).

Pseudorandomness of function families. Intuitively, a finite function family F is pseudorandom if the input-output behavior of F_a is indistinguishable from a random function of the same domain and range. This is formalized via the notion of statistical tests, or distinguishers [GGM]. A distinguisher is an oracle circuit D. Let F^1, F^2 be function families. We consider the following experiment. Choose a bit $b \stackrel{R}{\leftarrow} \{1,2\}$ at random and set g to a random member of F^b. Now provide A with an oracle for g and ask him to guess which family g came from (ie. what is the value of b.) We are interested in the probability that A is correct. Formally, we define the advantage of D in distinguishing F^1 from F^2 as

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

The probability here is over the indicated random choice of g and the coin tosses of D. In terms of the above discussed experiment, the advantage is $2(\Pr[D^g = b] - 1/2)$, the amount that the probability that D is correct is bounded away from the guessing probability $1/2$, scaled up to be between 0 and 1.

Let F be a family with domain X and range Y, and let $R = \mathrm{Maps}(X,Y)$ be the set of all functions from X to Y. We say that D $[t,q,l,\epsilon]$-breaks F if D is a circuit of size t which makes at most q oracle queries, each of which has length at most l blocks, and achieves $\mathsf{Adv}_D(F,R) \geq \epsilon$. We say that F is $[t,q,l,\epsilon]$-secure if there is no distinguisher who $[t,q,l,\epsilon]$-breaks F.

Sometimes we consider just a special class of distinguishers (eg. prefix-free distinguishers, no one of whose queries is a prefix of another), in which case we say F is $[t,q,l,\epsilon]$-secure against the class in question.

MESSAGE AUTHENTICATION CODES. See Section 1.1 for definitions.

3 The basic construction

Here we present and analyze our basic iterated construction of pseudorandom function families with an arbitrarily long domain from pseudorandom families with a fixed-length domain.

CASCADED PSEUDORANDOMNESS. Let F be a family of functions from $\mathrm{Dom}(F) = B = \{0,1\}^b$ to B^L. Moreover, the key length is the same as the output length, namely $\kappa = L$. For instance, in the case of $\mathsf{md5}$ we have $F(a,x) = \mathsf{md5}(a,x)$ where $|a| = |\mathsf{md5}(a,\cdot)| = L = 128$ and $|x| = b = 512$.

We now define a bunch of families $F^{(1)}, F^{(2)}, F^{(3)}, \ldots$. The members of $F^{(k)}$ will take inputs which are k blocks long; that is, the family $F^{(k)}$ will have domain $\mathrm{Dom}(F^{(k)}) = B^{\leq k}$. The output length and key length of each family is still L. We call $F^{(k)}$ the k-th *iteration* of F.

For $k \in \mathbf{N}$, define $F^{(k)}$ inductively:

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

where λ denoted the empty string. That is, for $n \leq k$, $F^{(k)}(x_1,\ldots,x_n)$ 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

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

PREFIX-FREENESS. We want to claim that $F^{(*)}$ is reducible to F (with parameters specified in the sequel). But there is a "technical" problem. Namely, as written, $F^{(*)}$ is certainly *not* pseudorandom. (In fact it is not even a MAC.) To see this suppose we have an oracle for $F^{(*)}$. Ask it for $F^{(*)}(\lambda)$. Now, for any x_1 we can compute $F^{(*)}(x_1)$ because the latter is just $F(F^{(*)}(\lambda),x_1)$. More generally given $F^{(*)}(x_1\ldots x_i)$ we can compute $F^{(*)}(x_1\ldots x_i x_{i+1}\ldots x_n)$ for any $x_{i+1}\ldots x_n$. This problem disappears if the inputs are encoded to be a prefix-free set. (For example, pre-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 the is true for any such encoding. 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. (We remark that in some settings prefix-free encodings are expensive. Instead, as described in Section 1.4 and Section 4, we can append key materiel.)

Theorem 3.1 Let F be a function family with $\mathrm{Dom}(F) = B = \{0,1\}^b$, range $\mathrm{Range}(F) = \{0,1\}^L$, and key length L. 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

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

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 via a hybrids argument. However, a straightforward hybrids argument does not seem to work here. Instead, we first define a bunch of 'intermediate' function families. We then show, using two different hybrid arguments, that: **(I)** if only prefix-free distinguishers, that ask at most q queries, are considered, then $F^{(*)}$ is reducible to the qth family in the bunch; and **(II)** any family in the bunch is reducible to 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 $m\kappa$. $mF_{\vec{a}}(i, x) = F_{a_i}(x)$ for all keys $\vec{a} = a_1, \ldots, a_m$ where each $a_i \in \{0,1\}^\kappa$, all $i \in [m]$ and all $x \in X$. We sometimes use $g_1, \ldots, g_m$ to denote the function in the family mG that satisfies $mG_{\vec{a}}(i, x) = g_i(x)$ for all i, x.

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

Lemma 3.2 Let F be a function family with domain $\mathrm{Dom}(F) = B = \{0,1\}^b$, range $\mathrm{Range}(F) = \{0,1\}^L$, and key length L. 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

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

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 $\mathrm{Dom}(F) = B = \{0,1\}^b$, range $\mathrm{Range}(F) = \{0,1\}^L$, and key length κ, and let $m \geq 1$. Suppose F is $(t', q, 1, \epsilon')$-secure. Then mF is $(t, q, 1, \epsilon)$-secure, where

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

We remark that the key quantity here is the advantage, which, in Lemma 3.2 decreases by a factor of l, and in Lemma 3.3 further decreases by a factor of q. Typically we think if t' as large enough that t should be thought of as being approximately t'.

Theorem 3.1 now follows from Lemmas 3.2 and 3.3. So the main problem is to prove the two lemmas. Below we provide the skeleton of the proof of the Lemma 3.2; the proofs of some missing claims are in Appendix A. The proof of Lemma 3.3 is entirely in Appendix A.

3.1 Proof of Lemma 3.2

Let $qR \stackrel{\mathrm{def}}{=} \mathrm{Maps}([q] \times B, \{0,1\}^L)$, and let $R^l \stackrel{\mathrm{def}}{=} \mathrm{Maps}(B^{\leq l}, \{0,1\}^L)$. 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 $D_2 \stackrel{\mathrm{def}}{=} U^{D_3}$ that has $\epsilon_2 \stackrel{\mathrm{def}}{=} \mathsf{Adv}_{D_2}(qF, qR) \geq \epsilon_3 / l$. 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.

THE ALGORITHM D. First, we specify an algorithm D who takes q oracles $g_1, \ldots, g_q$: $B \rightarrow \{0,1\}^L$ and also takes an input $k \in [l]$. It starts running D_3. The latter requires an oracle to a function g: $B^{\leq l} \rightarrow \{0,1\}^L$. D_2 will supply answers to the oracle queries of D_3. It begins by initializing counters j and t to 0. The answer to oracle query $x_1 \ldots x_n \in B^{\leq l}$ is now given as follows—

(1) Increment the query counter j and denote the current query by $x^j = x^j_1 \ldots x^j_n$.

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

(3) Else (namely $n \geq k$)

(3.1) If $x^j_1 \ldots x^j_{k-1} = x^p_1 \ldots x^p_{k-1}$ for some $p < j$ then let $t^j = t^p$. Else increment t and let $t^j = t$.

(3.2) Let $a^j = g_{t^j}(x^j_k)$ and return $F^{(l)}(a^j, x^j_{k+1} \ldots x^j_n)$.
Finally, D outputs whatever D_3 outputs.

THE DISTINGUISHER. Our distinguisher D_2, given oracles $g_1, \ldots, g_m$: $B^{\le l} \to \{0,1\}^L$, picks $k \stackrel{R}{\leftarrow} [l]$ at random and outputs $D^{g_1,\ldots,g_q}(k)$. We now proceed to the analysis.

THE FAMILIES H_k. We begin by defining a sequence of hybrid function families. To each $k \in \{0,1,\ldots,l\}$ and each function g: $B^{\le k} \to \{0,1\}^L$ we associate a function h^g_k: $B^{\le l} \to \{0,1\}^L$, defined for any $x_1 \ldots x_n \in B^{\le l}$ as follows—

$$h^g_k(x_1 \ldots x_n) = \begin{cases} g(x_1 \ldots x_n) & \text{if } n \le k \\ F^{(l)}(g(x_1 \ldots x_k), x_{k+1} \ldots x_n) & \text{otherwise.} \end{cases}$$

We consider the family H_k defined as the set of all h^g_k as g ranges in $\mathrm{Maps}(B^{\le k}, \{0,1\}^L)$. The distribution on these functions is that induced by picking g at random from the indicated set of functions. We now define

$$P_k \stackrel{\text{def}}{=} \Pr_{f \stackrel{R}{\leftarrow} H_k}\left[\, D^f_3 = 1 \,\right] .$$

Claim 3.4 $H_0 = F^{(l)}$ and $H_l = R'$. In particular:

$$\Pr_{f \stackrel{R}{\leftarrow} R'}\left[\, D^f_3 = 1 \,\right] = P_l \quad \text{and} \quad \Pr_{f \stackrel{R}{\leftarrow} F^{(l)}}\left[\, D^f_3 = 1 \,\right] = P_0 . \tag{5}$$

Proof: If $k = 0$ then $x_1 \ldots x_k$ is the empty string λ. Thus h^g_0 is $F^{(l)}_a$ where $a = g(\lambda)$. Since g is random, a is uniformly distributed. This means H_0 is indeed $F^{(l)}$. On the other hand if $k = l$ then h^g_l is just g itself, so $H_l = R'$. The facts about P_l and P_0 of course follow. ∎

RELATING H_k TO D. We now fix $k \in [l]$. We define and consider four experiments. Experiment 1 (resp 2) is that of running $D^{g_1,\ldots,g_q}(k)$ 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^g_3 with $g \stackrel{R}{\leftarrow} H_k$ (resp. H_{k-1}).

Recall that in all these experiments, D_3 is making oracle queries (which are replied to by D as indicated above in the first two experiments, and directly by g in the last two experiments). We want to claim that Experiments 1 and 3 are "equivalent" in the sense that the "view" of D_3 at any point is the same (ie. identically distributed) in these two experiments. Similarly for Experiments 2 and 4.

More formally, we denote by X_i the random variable whose value is the i-th oracle query. We denote by Y_i the answer that is given to X_i. We now fix particular 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 $X_i = x^i$ for $i = 1, \ldots, j$ and $\mathsf{Y}_i = y^i$ for $i = 1, \ldots, j-1$. (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 3.5 For any L bit string y—

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

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

The proof of this claim is in Appendix A.

CONCLUDING THE PROOF GIVEN CLAIM 3.5. Repeated applications of Part (1) of the above implies that the output distribution of D in Experiment 1 is that of D_3 in Experiment 3. But the latter was defined to be P_k. Similarly for Experiments 2 and 4 using Part (2) of the above. That is—

$$\Pr_{g \stackrel{R}{\leftarrow} qR}[\, D^g(k) = 1\,] = P_k \quad \text{and} \quad \Pr_{g \stackrel{R}{\leftarrow} qF}[\, D^g(k) = 1\,] = P_{k-1} .$$

Now recall that D_2 picks $k \stackrel{R}{\leftarrow} [m]$ and runs D on input k. Thus:

$$\Pr_{g \stackrel{R}{\leftarrow} qR}[\, D_2^g = 1\,] \;=\; (1/l) \cdot \textstyle\sum_{k=1}^{l} P_k$$

$$\Pr_{g \stackrel{R}{\leftarrow} qF}[\, D_2^g = 1\,] \;=\; (1/l) \cdot \textstyle\sum_{k=0}^{l-1} P_k\,.$$

So $\epsilon_2 \stackrel{\text{def}}{=} \mathsf{Adv}_{D_2}(F, R) = (1/l) \cdot (P_l - P_0)$. But now we can apply Equation (5) to see that

$$\epsilon_3 \stackrel{\text{def}}{=} \mathsf{Adv}_{D_3}(R', F^{(l)}) \;=\; \Pr_{f \stackrel{R}{\leftarrow} R'}\left[\, D_3^f = 1\,\right] - \Pr_{f \stackrel{R}{\leftarrow} F^{(l)}}\left[\, D_3^f = 1\,\right] \;=\; P_l - P_0\,.$$

Thus $\epsilon_2 \geq \epsilon_3/l$. This concludes the proof of Lemma 3.2. It remains to prove Claim 3.5. This proof is in Appendix A.

4 Getting rid of prefix-freeness

Our basic construction is valid as long as the queries are prefix-free. This can be ensured by appropriately encoding the messages (queries). For example, prepend the length $|q|$ to each query q.

However, such encodings require, of the machine evaluating the pseudorandom function, to perform at least two "passes" of reading the message (e.g., one pass to find the length, and a second pass to compute the function with the length prepended). Thus the entire message must be stored in memory (or a buffer) in between the passes. This is a drawback in the scenario where long chunks of communicated data need to be authenticated (using a pseudorandom family serving as a MAC) "on the fly".

Here we describe a construction that has practically the same effect as a prefix-free encoding, and hardly reduces the efficiency. However, we pay a little in security. The construction is what was called F^{csc} in Section 1.4. Namely given a family F with key length κ, construct the family $\alpha\text{-}\mathrm{A}F$ with key length $\kappa + \alpha$ as follows. Having key k, a where $k \in \{0,1\}^\kappa$ and $a \in \{0,1\}^\alpha$, let $\alpha\text{-}\mathrm{A}F_{k,a}(x) = F_k(xa)$ for all x, where xa denotes the concatenation of x and a.

Now, $\alpha\text{-}\mathrm{A}F$ requires only one pass; furthermore, as will be seen, α 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 $\alpha\text{-}\mathrm{A}F$, 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 a as a substring. Since a is random, this event happens with probability exponentially small in $|a| = \alpha$." However, this argument assumes that a 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 which is is (t', q, l, ϵ')-secure against prefix-free distinguishers. Then $\alpha\text{-}\mathrm{A}F$ is (t, q, l, ϵ)-secure where

$$t = t' - cq\mathrm{Time}(F)\log\alpha \quad ; \quad \epsilon = \epsilon' + lq2^{-\alpha}.$$

(Here l measures the number of *bits* in the longest query and c is a specific small constant determined by the proof.)

Let $\alpha\text{-}\mathrm{A}F^{(*)}$ 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 $\mathrm{Range}(F) = \{0,1\}^L$ and key length L. Suppose F is $(t', q, 1, \epsilon')$-secure and let $l \geq 1$. Then $\alpha\text{-}\mathrm{A}F^{(*)}$ is (t, q, l, ϵ)-secure, where

$$t = t' - cq \cdot (L + b + (l + \log\alpha)\mathrm{Time}(F) + (L + l + b)\log q) \quad ; \quad \epsilon = lq\epsilon + blq2^{-\alpha}.$$

5 The random-prepend construction

We now show how randomization can improve the concrete security of the cascade construction. The construction is to define F_a^{rcsc} on input x to choose R at random and return $(R, F^{(*)}(Rx))$. The gain is specified in the following theorem.

Theorem 5.1 Let F be a function family with $\mathrm{Dom}(F) = B = \{0,1\}^b$, range $\mathrm{Range}(F) = \{0,1\}^L$, and key length L. 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

$$\begin{aligned} t &= t' - cq(l + L + b) \cdot (\mathrm{Time}(F) + \log q) \\ \epsilon &= \epsilon'(q) + lq\epsilon'(1) + \sum_{i=2}^{q} \frac{q^i}{2^{l(i-1)}} \epsilon'(i) \, . \end{aligned}$$

Here 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, this is $\epsilon' = lq\epsilon'(q)$. But above 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 much smaller than $\epsilon'(q)$. Other terms are 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, given a query $x^j = x_1^j \ldots x_n^j$ will itself pick R^j at random, so that the "effective" query is $R^j x_1^j \ldots x_n^j$. Now if the input k had value 1, we end up invoking $g_1(R^j)$. This is a at most q query attack on the original family, and accounts for the $\epsilon'(q)$ term above. Now suppose $k \geq 2$. 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 On the tightness of the analysis

Theorem 3.1 provides bounds on the rate of increase in the advantage ϵ when applying the basic cascaded construction. The bounds are general, regardless of which underlying family is used. We investigate the tightness of these bounds. 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 $\{0,1\}^L$ and key length κ) 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 ask for the values of functions in F on points of its choice. (We call these queries $\mathsf{black\text{-}box\ queries}$, to distinguish them from the queries addressed to the tested oracle function.) In this setting, we show that there exist function families for which the analysis of Theorem 3.1 is 'almost' tight. In fact, this holds even for a *random* function family.

We present the result in more detail. Let $\epsilon_F(l, q, \tau)$ denote the highest advantage[5] in breaking family F by any algorithm that asks, of its oracle, at most q (prefix-free) queries, each consisting of at most l B-blocks; the total number of queries, including the black-box queries, is at most τ. (τ is now our notion of 'running time'. In particular, $\tau \geq q$.) In this language, Theorem 3.1 implies that

$$\epsilon_{F^{(*)}}(l, q, \tau) \leq lq \cdot \epsilon_F(1, q, l \cdot \tau) \tag{6}$$

[5] See the definition of advantage in Section 2.

for any F. We proceed in two steps as follows. Consider first a family RF of 2^κ functions chosen at random ¿from Maps$(B, \{0,1\}^L)$. We claim that, in the black-box setting, there exists a constant $c > 0$ such that

$$\epsilon_{RF}(1, q, \tau) \leq \frac{c \cdot \tau}{2^L}. \tag{7}$$

(Here the probabilities are taken also on the choices of RF.) The proof of this claim is straight-forward and is omitted here. We remark that this bound is tight; that is, there exists a simple distinguisher showing that $\epsilon_{1,1,\tau}(RF) \leq \frac{c'' \cdot \tau}{2^L}$ for some $c'' > 0$.

Next we show a simple and general distinguishing algorithm demonstrating that

$$\frac{c'(l-1)\tau^2}{2^L} \leq \epsilon_{F^{(*)}}(l, \tau, \tau) \tag{8}$$

for some $c' > 0$ and *any* F. This algorithm works even in the black-box setting described above.[6] That is, it uses an intrinsic property of the cascaded construction and requires no knowledge of the underlying family F. Inequalities (7) and (8), together with (6), imply

$$\frac{1}{\tilde{c}} \cdot (l-1)\tau \cdot \epsilon_{RF}(1, \tau, \tau) \leq \epsilon_{RF^{(*)}}(l, \tau, \tau) \leq \tilde{c} \cdot l\tau \cdot \epsilon_{RF}(1, \tau, l \cdot \tau).$$

for some $\tilde{c} > 0$.

The distinguishing algorithm is based on the difference between the probabilities of collisions in random functions and in our cascaded functions. It will ask q queries that differ only in the first block out of the l blocks. (I.e, it chooses $x_2, \ldots, x_l$ and $a_1, \ldots, a_q$ where each $x_i, a_i \in B$, and asks queries $m_1, \ldots, m_q$ where $m_j = a_j, x_2, \ldots, x_l$.) It outputs 'cascaded' iff any two queries resulted in the same answer. Now, if the oracle is a random function, then the probability of such collisions is $O(\frac{q^2}{2^L})$. 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 $\Omega(\frac{q^2}{2^L})$. Thus collisions happen with probability $\Omega(\frac{lq^2}{2^L})$.

We remark that this idea can be extended in a straightforward way to breaking MACs based on the cascaded 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 [X9.9]. A similar attack was independently reported by [PV].)

References

[BGR] 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.

[BKR] 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] Clearly, this algorithm works also in a real-life setting, where F is explicitly given.

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

[GMR] S. Goldwasser, S. Micali and R. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM Journal of Computing*, 17(2):281–308, April 1988.

[HILL] 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.

[LR1] M. Luby and C. Rackoff, "How to construct pseudorandom permutations from pseudo-random functions," *SIAM J. Computation*, Vol. 17, No. 2, April 1988.

[NaRe] 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.

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

[SHA] 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.

[X9.9] ANSI X9.9, "American National Standard for Financial Institution Message Authentication (Wholesale)," American Bankers Association, 1981. Revised 1986.

A Proofs of Lemmas, Claims and Theorems

A.1 Proof of Claim 3.5

Proof of Claim 3.5: 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 $k-1$. 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 L bit string, independent of all previous choices. But that's exactly what's returned in Experiment 1. So the claim is true in this case. We now move on to the interesting case, namely $n \geq k-1$.

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 $x_1^j, \ldots, x_k^j$, a value we denote A_j, and using this as a seed to return $F^{(l)}(\mathsf{A}_j, x_{k+1}^j \ldots x_n^j)$. 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.

Let $T = \{\, p < j \,:\, x_1^j \ldots x_k^j = x_1^p \ldots x_k^p \,\}$. In Experiment 3, $\mathsf{A}_j = g(x_1^j \ldots x_k^j)$. This means that $\mathsf{A}_j = \mathsf{A}_p$ for all $p \in T$ if $T \neq \emptyset$, and is uniformly and independently distributed otherwise. We now argue the same is true in Experiment 1. Let $S = \{\, p < j \,:\, x_1^j \ldots x_{k-1}^j = x_1^p \ldots x_{k-1}^p \,\}$. Note our construction of D guarantees that there is a single value, which we here denote v, such that $t^p = v$ for all $p \in S$. Now

- If $S \neq \emptyset$ then we return $g_v(x_k^j)$. This equals A_p for all $p \in T$ if $T \neq \emptyset$. Else it is a uniformly distributed L bit string because g_v has never before been invoked on x_k^j.
- If $S = \emptyset$ then we return the value on x_k^j of a random function g_t which has never before been invoked. So this is a uniformly distributed L bit string.

We now turn to proving Part (2). Again the case $n \le k-1$ is easy. If $n < k-1$ then to see that the value $g(x^j)$ returned in Experiment 3 is random we just need to note that x^j is not equal to any previous query. If $n = k-1$ we need to use that x^j is not a prefix of any previous query. (This is the only place we use the prefix-freeness, as opposed to distinctness, of queries.) So now assume $n \ge k-1$.

Let A_j be the same random variable as above. But now $\mathsf{A}_j = F(\mathsf{B}_j, x^j_k)$ where random variable B_j takes a value depending on the experiment. It suffices to show B_j is identically distributed in the two experiments.

Let the set S and the value v be as before. In Experiment 4, $\mathsf{B}_j = g(x^j_1 \ldots x^j_{k-1})$. This is equal to B_p for all $p \in S$ if $S \neq \emptyset$, and otherwise is uniformly distributed. We now argue the same is true in Experiment 2. Recall that $g_1, \ldots, g_m$ are now drawn randomly from F. Accordingly let $\mathsf{Seed}(i)$ be the random variable whose value is the seed of g_i; ie. $g_i = F_{\mathsf{Seed}(i)}$. Now

- If $S \neq \emptyset$ then we return $g_v(x^j_k)$. Thus $\mathsf{B}_j = \mathsf{Seed}(v)$. Thus $\mathsf{B}_j = \mathsf{B}_p$ for all $p \in S$.
- If $S = \emptyset$ then we return the value on x^j_k of a random function g_t which has never before been invoked. So $\mathsf{B}_j = \mathsf{Seed}(t)$ is a uniformly distributed L bit string.

This concludes the proof. ∎

A.2 Proof of Lemma 3.3

Proof of Lemma 3.3: Also here we use a hybrids argument. Here the argument is simpler. Let $R \stackrel{\text{def}}{=} \text{Maps}(B, \{0,1\}^L)$, and let $mR \stackrel{\text{def}}{=} \text{Maps}([m] \times B, \{0,1\}^L)$. 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 $\epsilon_1 \stackrel{\text{def}}{=} \mathsf{Adv}_{D_1}(qF, qR) \ge \epsilon_2/m$.

We first consider an algorithm D which is given oracle access to a function g and takes an input $k \in [m] = \{1, \ldots, m\}$. It works, informally, as follows. It "picks," randomly and independently, functions $g_1, \ldots, g_{k-1}$ from F. It "sets" g_k to its given function g. It also "picks" random functions $g_{k+1}, \ldots, g_m$ from R. It now runs D_2 with oracles $g_1, \ldots, g_m$— when the query is to oracle i it answers via g_i, invoking its oracle when $i = k$ and otherwise using one of its chosen functions. It outputs whatever this computation outputs.

To pick $g_1, \ldots, g_{k-1}$ from F meant to pick randomly $k-1$ seeds $a_1, \ldots, a_{k-1} \stackrel{R}{\leftarrow} \{0,1\}^\kappa$ and reply to query (i, x), for $i < k$, by $F(a_i, x)$. On the other hand, D of course can't really "pick" random functions. What it does is that each time D_2 asks a query (i, x) with $i > k$ it picks a random L bit string if (i, 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.

Now our algorithm D_1, given oracle g, is defined to pick at random an integer $k \stackrel{R}{\leftarrow} [m]$ and return $D^g(k)$. In the worst case it 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 seeds, which is $O(m\kappa)$, and to record and respond to queries, which involves computation of F_{a_i} and comes to $O(q_1 \cdot \text{Time}(F) + q_1 L \log q_1)$. This comes to $t_1 = t_2 + c \cdot (m\kappa + q_1 \cdot \text{Time}(F) + q_1 L \log q_1)$. Finally, we proceed to analyze $\mathsf{Adv}_{D_1}(F, R)$.

For $k = 0, \ldots, m$ let $G_k = F \times \cdots \times F \times R \times \cdots \times R$ where there are k copies of F and $m-k$ 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_i at random from F for $i = 1, \ldots, k$ and at random from R for $i = k+1, \ldots, m$. We let

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

be the probability that D_2 outputs one when its oracles are chosen randomly from G_k. Now note that for $k = 1, \ldots, m$—

$$\Pr_{g \xleftarrow{R} R}[\, D^g(k) = 1 \,] \;=\; P_{k-1} \quad \text{and} \quad \Pr_{g \xleftarrow{R} F}[\, D^g(k) = 1 \,] \;=\; P_k \,.$$

Thus

$$\Pr_{g \xleftarrow{R} R}[\, D_1^g = 1 \,] \;=\; (1/m) \cdot \textstyle\sum_{i=0}^{m-1} P_i \quad \text{and} \quad \Pr_{g \xleftarrow{R} F}[\, D_1^g = 1 \,] \;=\; (1/m) \cdot \textstyle\sum_{i=1}^{m} P_i \,.$$

Thus the advantage of D_1 is—

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

However now observe that

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

and

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

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

A.3 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 $\epsilon_4 \stackrel{\text{def}}{=} \mathsf{Adv}_{D_4}(\alpha\text{-A}F, R)$, U defines a distinguisher $D_3 \stackrel{\text{def}}{=} U^{D_4}$ with $\epsilon_3 \stackrel{\text{def}}{=} \mathsf{Adv}_{D_3}(F, R) \geq \epsilon_4 - lq2^{-\alpha}$.

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 $a \xleftarrow{R} \{0,1\}^\alpha$. Next, run D_4 where each query x of D_4 is answered by $g(xa)$. If any of the queries of D_4 contains a as a substring then output '1' (meaning 'g is pseudorandom'). Otherwise, output whatever D_4 outputs.

For the analysis, let

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

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 $g \xleftarrow{R} F$ then the answers given by D_3 to D_4's queries are those of a function $g \xleftarrow{R} \alpha\text{-A}F$. Thus,

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

When $g \xleftarrow{R} R$, the probability that any one of D_4's queries contains a as a substring is at most the probability that a *random* value in $\{0,1\}^\alpha$ appears as a substring in a *given* set of q strings each of length l. This probability is less than $lq2^{-\alpha}$. Thus, $P_{3,\text{R}} \leq P_{4,\text{R}} + lq2^{-\alpha}$. On the other hand, whenever D_4 outputs '1', D_3 outputs '1' as well. Thus, $P_{3,\text{F}} \geq P_{4,\text{F}}$. We conclude that

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

The other parameters of the reduction can be easily verified.