Efficient Constructions of Variable-Input-Length Block Ciphers

Sarvar Patel[1], Zulfikar Ramzan[2] and Ganapathy S. Sundaram[1]

[1] Lucent Technologies
{sarvar, ganeshs}@bell-labs.com
[2] DoCoMo Communications Laboratories, USA**
ramzan@docomolabs-usa.com

Abstract. Existing block ciphers operate on a fixed-input-length (FIL) block size (e.g., 64-bits for DES). Often, one needs a variable-input-length (VIL) primitive that can operate on a different size input; it is, however, undesirable to construct this primitive from "scratch." This paper contains two constructions that start with a fixed-input-length block cipher and show how to securely convert it to a variable-input-length block cipher without making any additional cryptographic assumptions. Both constructions model the FIL block cipher as a pseudorandom permutation (PRP) – that is, indistinguishable from a random permutation against adaptive chosen plaintext attack. The first construction converts it to a VIL PRP and is an efficiency improvement over the scheme of Bellare and Rogaway [4]. The second construction converts it to a VIL super pseudorandom permutation (SPRP) – that is, the resulting VIL block cipher is indistinguishable from a random permutation against adaptive chosen *plaintext and ciphertext* attack.

1 Introduction

A cryptographic primitive which operates on an input of fixed size is called a fixed-input-length (FIL) primitive. For example, block ciphers typically operate on messages of fixed size (64 bits in the case of DES [18]). But often in practice, one is faced with the situation of applying a cryptographic primitive on data of varying lengths. A striking example is the need for an encryption algorithm which deals with messages of varying sizes but at the same time preserves the property that the length of ciphertext equals the length of the plaintext. This situation is very common in Internet applications where traffic consists of "packets" of varying sizes. If a block cipher is being used for encryption, then the blocks that need to be encrypted could be of varying lengths. Differential packet sizes are also prevalent in wireless applications: this is due to the fact that the frames of data that are sent to each user may be different from user to user because of the difference in the so called path-loss of the users relative to the base station.

** Work done while the author was a visiting member at Lucent Technologies.

H. Handschuh and A. Hasan (Eds.): SAC 2004, LNCS 3357, pp. 326–340, 2005.

Moreover, channel conditions are a function of time, thereby forcing a change in the transmission rates (and hence block sizes). This calls for the development of variable-input-length (VIL) cryptographic primitives.

In general, one wants to avoid having to construct new primitives "from scratch" to deal with specific applications, since this approach might be prone to error. Instead, one can attempt to utilize a FIL primitive as a building block in order to build a VIL primitive. In this paper, we take this approach and provide *efficient constructions* for the conversion of a FIL block cipher to a VIL block cipher. We also provide proofs of security relating the security of the VIL primitive to the security of the underlying FIL primitive.

1.1 Related Work

FIL to VIL for Block Ciphers. The first formal treatment for converting a fixed-input-length block cipher to a variable-input-length block cipher is due to Bellare and Rogaway [4]. They formalized the problem and gave a generic technique for constructing a block cipher which operates on any arbitrary length input from a block cipher that works on a fixed length input; they used the idea of a parsimonious pseudorandom function and parsimonious encryption scheme. In addition, their cipher possesses the customary requirement, initially due to Luby and Rackoff [15], of being a secure "pseudorandom permutation (PRP)" as long as the original cipher is.

A number of other papers could also be used as partial solutions towards converting FIL block ciphers into VIL block ciphers. For example, the celebrated paper of Luby and Rackoff [15] showed how to convert a n-bit to n-bit pseudorandom function (PRF) into a block cipher operating on $2n$-bits. The subsequent work by Naor and Reingold [17], provided constructions for converting block ciphers operating on n-bits to block ciphers on cn-bits for a constant $c \geq 1$. Bleichenbacher and Desai [7] have a construction that potentially converts a FIL SPRP to a VIL SPRP, though they do not provide a formal security proof. More recently, Halevi and Rogaway [12, 13] have provided constructions of tweakable enciphering schemes that operate on mn bits where m can be any positive integer and n is the size of the underlying block cipher. It was initially unclear how to use the techniques of [15] and [17] to attain provably-secure ciphers that operate on lengths which are not a multiple of n. One of the contributions of the present paper is to utilize these previous results to achieve such a construction.

FIL to VIL for Other Primitives. The FIL to VIL problem has been addressed for Message Authentication Codes (MACs) and for PRFs. The elegant FIL-MAC to VIL-MAC work of An and Bellare [1] is a Damgård-like [10] nested iteration construction. Numerous works implicitly address the issue of converting a fixed-input-length PRF into a variable-input-length one; to name a few: Bellare, Kilian and Rogaway's CBC-MAC analysis [3] (which assumes messages are fixed length, but arbitrarily large), Petrank and Rackoff's [21] extension to variable-length messages, Bellare, Canetti, and Krawzyck's [16] cascade construction, and Bernstein's [5] protected counter sum construction.

1.2 Meaning of FIL to VIL

There is some ambiguity in the meaning of a FIL to VIL construction which we would like to clarify. A VIL primitive operates on messages $x \in \{0,1\}^*$ or some large set containing strings of various lengths. The VIL construction can only use the given FIL primitive in conjunction with other non-cryptographic operations, but it cannot use other types of FIL cryptographic primitives. For example, if a VIL MAC is being constructed then only the given FIL MAC primitive can be used, but other cryptographic primitives like a PRG or PRF cannot be used. There are two possible meanings of using the FIL cryptographic primitive:

- Oracle Model: In this model, we only have oracle access to the FIL cryptographic primitive. We can query the FIL primitive with an input and get back an output. But, we cannot look inside the primitive for a key or run many instances of the primitive. This is a restrictive model which is useful to model certain scenarios; e.g., a smartcard or an assistant server which can answer our queries without giving internal access.
- Keyed Model: Unlike the Oracle model, here we are given a FIL cryptographic primitive which takes a single fixed key and message as input. Next, we can run various instances of the FIL primitive each keyed with its own key. However, any other key material must be derived from the single given key and using the FIL primitive itself without making any additional cryptographic assumption other than what is implied by the FIL primitive. The restriction of not having extra key material is appropriate because it would not be an apples to apples comparison of VIL constructions which take large keys with those that do not. Next, it is interesting and important to see if an efficient construction can be so achieved. Practically speaking, in existing systems, layer and functionality separation may mean, for example, that after a session key agreement, a key of fixed size may be handed to the encryption layer to encrypt the messages. As designers of the encryption layer we may be able to use a VIL construction, but we cannot request more key material from the session key agreement protocol because that may be a standardized protocol over which we have no control. So, if we need more keys for the VIL construction we have to create them using the given key and the FIL primitive. Bellare-Rogaway [4] used this keyed model and both of our constructions do as well.

1.3 Our Results

In this work, our goal is to provide very efficient VIL constructions for block ciphers. We utilize various classes of universal hash functions together with the existing cryptographic primitives to attain very efficient constructions; using such hash functions in conjunction with cryptographic primitives is a well-studied idea, but the novelty of this paper is their use in constructing block ciphers in the variable-input-length setting. Moreover, for our constructions we provide an exact security analysis. In some cases we utilize a technique / framework due to Naor and Reingold [17] that enables us to provide clean proofs of security in the presence of adaptive adversaries. We obtain the following results:

– Sections 3 and 4 give a FIL to VIL Block Cipher construction that is almost twice as fast as the Bellare-Rogaway construction [4]. Here we model the block cipher as a pseudorandom permutation.
– Section 5 gives a FIL to VIL Block Cipher that is *super pseudorandom*. We provably achieve an open goal suggested by Bellare-Rogaway [4].

In both cases, the concrete security of our schemes are limited by birthday bounds, so $2^{n/2}$ should be sufficiently large, where n is the starting block size.

2 Definitions

We introduce the notions of pseudorandom functions (PRFs) and pseudorandom permutations (PRPs). Although these primitives are often treated asymptotically, we model them in the concrete security framework. This is necessary since we deal with *fixed-input-length* primitives; as a result, meaningful security results are not captured by an asymptotic treatment. The exposition borrows freely from [3], [16].

Notation. For a bit string x, we denote its length by $|x|$. If $a, b > 0$ are integers, and $a \leq b$, then the substring of x starting at bit position a and ending at bit position b (counting from the left) is denoted $x[a, \ldots, b]$. Let S be a probability space, then the process of picking an element from S according to the underlying probability distribution is denoted $x \xleftarrow{R} S$. We use I_n to denote $\{0,1\}^n$ (the set of bit strings of length n). The set of all functions mapping I_n to I_m is denoted $\mathcal{F}_{n,m}$, and set of permutations on I_n is denoted $\mathcal{P}_n$.

Computational Model. We follow the convention in [3] and model our adversary $\mathcal{A}$ as a program for a Random Access Machine. This adversary will have access to an oracle for computing a specified function f; it can make black-box queries to this oracle, and we assume that it will receive a correct response in unit time. We denote by $\mathcal{A}^f$ an adversary with access to an oracle for computing function f. Following the convention of [3], we define the running time of the adversary to be its execution time plus the length of its description.[1] The query complexity of $\mathcal{A}$ is defined as the number of queries it makes to its oracle.

Finite Function Families. A finite function family $\mathcal{F}$, is a collection of functions, all of which have domain $Dom(\mathcal{F})$ and range $Range(\mathcal{F})$. Our focus is on function families in which each function in the family can be formally specified by (by at least one) "key." Typically, the key for a function family will be a pre-defined fixed-length bit string. And for a function family $\mathcal{F}$, and a key k, we let $\mathcal{F}_k$ denote the function associated with the given key, and we assume that computing $\mathcal{F}_k$ at any given point of $Dom(\mathcal{F})$ is easy given the key k.

Examples. Perhaps the simplest example is the set of all functions with domain I_k and range I_ℓ, under the uniform distribution. We denote this family by

[1] By defining the running time as such, we prevent anomalies that may arise from embedding arbitrarily large lookup tables in $\mathcal{A}$'s description.

$\mathsf{Rand}^{k\to\ell}$. A function in this family can be represented by $k2^\ell$ bits – hence an appropriate key space is I_{k2^ℓ}. Another simple example is the set of all *permutations* on I_ℓ. We denote this family by Perm^ℓ. Any block cipher constitutes a keyed family of permutations. For example, DES [18] has key space I_{56}, with domain and range I_{64}, and the AES algorithm (Rijndael [9]) is typically instantiated with a key space, a domain, and a range of I_{128} (though the specification accommodates alternate lengths).

Distinguishability. The concept of distinguishability, due to Goldreich, Goldwasser, and Micali [11], helps capture the idea of a "computational distance" between two function families. This notion will be useful when we discuss pseudorandom functions and permutations. Suppose that $\mathcal{F}^0$ and $\mathcal{F}^1$ are two function families that have both identical domains and identical ranges. An adversary $\mathcal{A}$ will get oracle access to either a function sampled from $\mathcal{F}^0$, or a function sampled from $\mathcal{F}^1$. The adversary will not, however, be told whether the oracle really sampled from $\mathcal{F}^0$ or $\mathcal{F}^1$. The adversary's goal is to determine which function family was actually sampled. Informally, distinguishability corresponds directly to the adversary's success rate in making this determination. In particular, let $\mathsf{Adv}_{\mathcal{A}}(\mathcal{F}^0,\mathcal{F}^1) \triangleq \Pr[f \xleftarrow{R} \mathcal{F}^0 : \mathcal{A}^f = 1] - \Pr[f \xleftarrow{R} \mathcal{F}^1 : \mathcal{A}^f = 1]$, where the probabilities are taken over the choice of f and $\mathcal{A}$'s internal coin tosses. Now, we say that $\mathcal{A}$ (t, q, n, ϵ)-distinguishes $\mathcal{F}^0$ from $\mathcal{F}^1$ if $\mathcal{A}$ runs for time at most t, makes at most q queries to its oracle each length at most n, and $\mathsf{Adv}_{\mathcal{A}}(\mathcal{F}^0,\mathcal{F}^1) \geq \epsilon$.

Pseudorandom Functions and Permutations. Pseudorandomness captures the computational distance between $\mathsf{Rand}^{k\to\ell}$, and another function family $\mathcal{F}$ with domain I_k and range I_ℓ.

Definition 1. *Let $\mathcal{F}$ be a keyed function family with domain I_k and range I_ℓ. Let $\mathcal{A}$ be an adversary that is equipped with an oracle. Then,* $\mathsf{Adv}^{\mathsf{prf}}_{\mathcal{F}}(\mathcal{A}) \triangleq \Pr[f \xleftarrow{R} \mathcal{F} : \mathcal{A}^f = 1] - \Pr[f \xleftarrow{R} \mathsf{Rand}^{k\to\ell} : \mathcal{A}^f = 1]$. *For any integers $q, t \geq 0$, we define an insecurity function* $\mathsf{Adv}^{\mathsf{prf}}_{\mathcal{F}}(q,t) \triangleq \max_{\mathcal{A}}\{\mathsf{Adv}^{\mathsf{prf}}_{\mathcal{F}}(\mathcal{A})\}$. *Where the maximum is taken over choices of adversary $\mathcal{A}$ that are restricted to running time at most t, and q oracle queries.*

We employ the convention due to [3] and incorporate the amount of time it takes to sample f from $\mathcal{F}$ into the running time of $\mathcal{A}$.

We now consider the concept of a pseudorandom permutation family, which was originally defined by Luby and Rackoff [15]. The original notion considered the computational indistinguishability between a given family of permutations and the family of all *functions*. Following the treatment of Bellare et al. [3], we measure the pseudorandomness of a permutation family on I_ℓ in terms of its indistinguishability from Perm^ℓ.

Definition 2. *Let $\mathcal{F}$ be a keyed permutation function family with domain and range I_ℓ. Let $\mathcal{A}$ be an adversary that is equipped with an oracle. Then,*

$$\mathsf{Adv}^{\mathsf{prp}}_{\mathcal{F}}(\mathcal{A}) \triangleq \Pr[f \xleftarrow{R} \mathcal{F} : \mathcal{A}^f = 1] - \Pr[f \xleftarrow{R} \mathsf{Perm}^\ell : \mathcal{A}^f = 1].$$

We define an insecurity function $\mathsf{Adv}^{\mathsf{prp}}_{\mathcal{F}}(q,t) \triangleq \max_{\mathcal{A}}\{\mathsf{Adv}^{\mathsf{prp}}_{\mathcal{F}}(\mathcal{A})\}$*, for any integers* $q, t \geq 0$*. The maximum is taken over choices of adversary* $\mathcal{A}$ *that are restricted to running time at most* t*, and* q *oracle queries.*

Luby and Rackoff [15] also considered the notion of a super pseudorandom permutation (SPRP). In this setting, the adversary is given access to both an oracle that computes the permutation for a given element, and an oracle that computes the inverse of the permutation.

Definition 3. *Let* $\mathcal{F}$ *be a keyed permutation function family with domain and range* I_ℓ*. Let* $\mathcal{A}$ *be an adversary that is given access to two oracles. Then,*

$$\mathsf{Adv}^{\mathsf{sprp}}_{\mathcal{F}}(\mathcal{A}) \triangleq \Pr[f \xleftarrow{R} \mathcal{F} : \mathcal{A}^{f,f^{-1}} = 1] - \Pr[f \xleftarrow{R} \mathsf{Perm}^{\ell} : \mathcal{A}^{f,f^{-1}} = 1].$$

We define an insecurity function $\mathsf{Adv}^{\mathsf{sprp}}_{F}(q,t) \triangleq \max_{\mathcal{A}}\{\mathsf{Adv}^{\mathsf{prp}}_{\mathcal{F}}(\mathcal{A})\}$*, for any integers* $q, t \geq 0$*. The maximum is taken over choices of adversary* $\mathcal{A}$ *that are restricted to running time at most* t*, and* q *oracle queries.*

The security of a Block Cipher against chosen plaintext attacks can be understood by examining it as a pseudorandom permutation, whereas the security against chosen plaintext and ciphertext attacks can be understood by examining it as a super pseudorandom permutation.

Universal Hash Functions. Let H be a family of functions with domain D and range S [2] that comes with an induced distribution (e.g., uniform); functions can be sampled from H according to this distribution. Let ϵ be a "small" constant such that $1/|S| \leq \epsilon \leq 1$.

- We call H an ϵ-almost universal family of hash functions if, for all $x \neq y \in D$, $\Pr_{h \in H}[h(x) = h(y)] \leq \epsilon$.
- We call H ϵ-almost-Δ-universal family of hash functions if, for all $x \neq y \in D$, $\Pr_{h \in H}[h(x) - h(y) = \delta] \leq \epsilon$.
- We call H an ϵ-almost-strongly-universal family of hash functions if, for all $x \neq y \in D$, $\Pr_{h \in H}[h(x) = a, h(y) = b] \leq \epsilon/|S|$. If H consists only of permutations, we say that H is a strongly universal family of permutations.

The above definitions are due to [8] [22]. As an example, the linear congruential hash $h(x) = ax + b \bmod p$ where a is non-zero and p is a prime, meets the above criteria. For simplicity, we often say that h is a certain type of universal hash function to mean that h was drawn from the family H according to its equipped distribution. We will later need universal and Δ-universal hash functions to operate on variable-length domains. Standard techniques of padding and length appending to create variable-input-length universal hash functions can be used (e.g., UMAC [6]) and we do not discuss them further.

[2] S is a usually a finite group with '+' and '−' as the addition and subtraction operators respectively.

3 FIL to VIL PRP: An Example Construction

Before presenting our general construction, we provide a concrete instantiation which takes the DES block cipher [18] with key K and creates a variable-input-length block cipher for block sizes larger than 64 bits. This example is primarily pedagogical – in practice, one should apply our construction on a longer starting block length to avoid birthday-type attacks. As depicted in Figure 1, we need several keys. We use key K_1 in DES in the second round, we use K_2 in the last round where DES is called in counter mode, and we need a key for the universal hash function h in round 1. To generate these keys, we run DES with key K and inputs $1, 2, \ldots, i$ and label the outputs $K_1, K_2, \ldots, K_i$. We can then use key K_1 to key DES in round 2 and we can use key K_2 to key DES in round 3. The rest of the keys can be used as the hash keys. This key expansion step will take place only once. The exact hash key size we need in order to deal with large inputs depends upon the exact nature of the hash function. For concreteness, we will use the Δ-universal hash function utilized in UMAC [6] which can work with a limited size hash key and specifies methods (e.g., padding, length appending, and a well-known Toeplitz key-scheduling trick) to deal with variable-length inputs.

Encryption

1. The message M is divided into two parts M_{pref} of size $|M| - n$ bits and M_{suff} of size $n = 64$ bits.
2. In round 1, the universal hash function h is applied on M_{pref} and the result is added to M_{suff} to create S. M_{pref} is also carried forward to round 2.
3. In round 2, S is encrypted using the DES block cipher keyed with key K_1 resulting in output T. M_{pref} is carried forward.
4. In round 3, T is carried forward. T is also used as the initial counter value used to encrypt M_{pref} using DES in counter mode; i.e., DES is keyed with K_2 and called with as many inputs $T, T + 1, \ldots$ as needed to create enough stream bits to XOR with M_{pref} to create C_{pref}. The output is (T, C_{pref}).

To see that the above procedure yields a variable-input-length block cipher it suffices to note that each round yields an invertible permutation. Round 1 is a Feistel permutation where the universal hash function is the underlying round function, so it is invertible. Round 2 block encrypts S and is invertible by the nature of block ciphers. Finally, round 3 uses T as the initial value for counter mode encryption, and so is also invertible. The details for decryption follow.

Decryption

1. The ciphertext C is divided into two parts, T of size n bits and C_{pref} of size $|M| - n$ bits.
2. $T, T + 1, \ldots$ are fed through DES block cipher keyed with K_2 to create a stream of output bits which are XORed with C_{pref} to recover M_{pref}. We have now inverted round 3 to recover (T, M_{pref}).
3. T is decrypted using DES block cipher keyed with K_1 to recover S. We have now inverted round 2 to recover (S, M_{pref}).

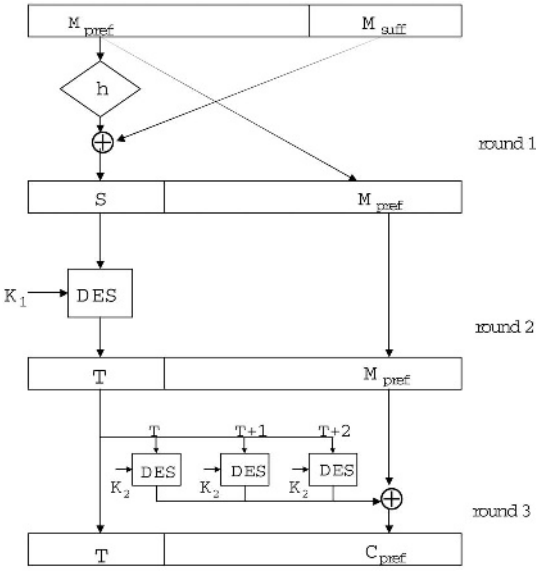

Fig. 1. An example of our construction for FIL to VIL conversion of DES

4. M_{pref} is fed to the universal hash function h and the result is XORed with S to recover M_{suff}. We have now recovered the plaintext $M = (M_{pref}, M_{suff})$.

As we describe next, we can instantiate the above example with any block cipher (not just one with a 64-bit block length) and any Δ-universal hash function. Furthermore, round 3 can employ other encryption schemes besides counter mode.

4 FIL to VIL PRP: Generalization and Security

The problem of constructing a variable-input-length encryption mode for block ciphers was considered by Bellare and Rogaway [4]. They give a generic approach for solving this problem, and then instantiate it with a specific construction. The generic approach involves utilizing a *parsimonious pseudorandom function* together with a *parsimonious encryption scheme*. It turns out that the CBC-MAC is a parsimonious PRF. In addition, both CBC-mode encryption and counter-mode encryption (with a random initial counter) serve as examples of parsimonious encryption schemes. In this section, we give an efficient construction for taking an existing fixed-input-length pseudorandom permutation, and building a variable-input-length parsimonious PRF (this is equal to round 1 and round 2 in figure 1). Our construction is more efficient than the CBC-MAC. Overall the construction in [4] requires two cryptographic passes on the entire input, whereas our construction requires one cryptographic pass and one non-cryptographic pass using a computationally lightweight universal hash function.

We now describe the Bellare-Rogaway framework [4] which we use to generalize our construction and analyze its security.

PARSIMONIOUS PRF. Let $\mathcal{F}$ be a keyed function family with domain I_k and range I_n, where $k \geq n$. We call $\mathcal{F}$ a parsimonious family if, for any key $a \in Keys(\mathcal{F})$, and any input $x \in I_k$, the last n bits of x are uniquely determined by: the remaining bits of x, the key a, and $\mathcal{F}_a(x)$.

PARSIMONIOUS ENCRYPTION. Following Bellare-Rogaway [4] we define a parsimonious encryption scheme via three algorithms $S = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. The algorithm $\mathcal{K}$ is a key-generation algorithm, and returns a random key κ to be used for the encryption. The algorithm $\mathcal{E}$ takes this key κ and the message M, picks a random, fixed-length IV, and then encrypts M to get a ciphertext $C = (IV; C^*)$, where C^* and M have the same length.

GENERAL SCHEME FOR VIL BLOCK CIPHERS. Given a parsimonious PRF and encryption scheme, we can construct a general VIL scheme $\mathcal{F}$ as follows. We let G be the parsimonious PRF whose domain is the message space and whose range is I_n. Let $\mathsf{Recover}$ denote G's corresponding recovery algorithm that obtains the last n bits of the message M given the key to G, the first $|M| - n$ bits of M, and the output of G. Let $S = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a parsimonious encryption scheme. Let K_{prf} and K_{enc} be the secret keys for the parsimonious PRF and encryption schemes respectively. Let M_{pref} be the first $|M| - n$ bits of M.

Algorithm Encrypt K_{prf}, K_{enc} (M)
$T = G_{K_{prf}}(M)$
$C_{pref} = \mathcal{E}_{K_{enc}}(M_{pref}; T)$
return $C = (T; C_{pref})$

Algorithm Decrypt K_{prf}, K_{enc} (C)
Let T be the first n bits of C.
$M_{pref} = \mathcal{D}_{K_{enc}}(C)$
$M_{suff} = \mathsf{Recover}_{K_{prf}}(M_{pref}, T)$
return $M = (M_{pref}; M_{suff})$

SECURITY FOR VIL MODE ENCRYPTION. Before giving any security analysis for general VIL Mode block cipher encryption, we discuss security for parsimonious encryption. The security for a parsimonious encryption scheme is defined by the adversary's inability to distinguish the encryption of a message from the encryption of a randomly chosen string of equal length. This definition was given in [4], but follows a definition given by [2]. More formally, if $S = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ is a parsimonious encryption scheme, and $\mathcal{A}$ is a distinguishing adversary, then

$$\mathsf{Adv}^{\mathsf{priv}}_{\mathcal{A}}(S) \triangleq \Pr[K \leftarrow \mathcal{K} : \mathcal{A}^{\mathcal{E}_K(\cdot)} = 1] - \Pr[K \leftarrow \mathcal{K} : \mathcal{A}^{\mathcal{E}_K(\$^{|\cdot|})} = 1].$$

In the first experiment, the oracle returns a random encryption of the message under the given key K, and in the second, a random encryption of a random string of the same length as the message (under the key K) is returned. We define $\mathsf{Adv}^{\mathsf{priv}}_{S}(t, q, \mu)$ as $\max_{\mathcal{A}}\{\mathsf{Adv}^{\mathsf{priv}}_{S}(\mathcal{A})\}$. Here the maximum is taken over all adversaries $\mathcal{A}$ who are restricted to time t, and make at most q oracle queries whose total length is no more than μ bits. Now, Bellare-Rogaway [4] proved

the following theorem relating the security of their general VIL block cipher construct in terms of its constituent parsimonious PRF and encryption scheme.

Theorem 1 (Bellare-Rogaway [4]). *Let $\mathcal{B}$ denote the VIL block cipher constructed from the parsimonious PRF family $\mathcal{F}$ and the parsimonious encryption scheme S. Moreover, suppose that the functions in $\mathcal{F}$ have range I_n. Then*

$$\mathsf{Adv}^{\mathsf{prp}}_{\mathcal{B}}(t,q,\mu) \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathcal{F}}(t',q,\mu) + \mathsf{Adv}^{\mathsf{priv}}_{S}(t',q,\mu) + \frac{q^2}{2^n},$$

where $t' = t + O(qn + \mu)$.

VIL Parsimonious PRF. We now show how to efficiently construct a parsimonious PRF that can handle variable input lengths. As pointed out above, our construction is the most efficient known parsimonious PRF. Combining our parsimonious PRF with an existing parsimonious encryption scheme (see [4] for examples) we get a very efficient scheme for VIL block cipher encryption. For now, we assume that we have a PRP over I_n (any block cipher will work). We show how to construct a *parsimonious* PRF family with domain I_{n+b} and range I_n, where $n \leq b$. Referring to figure 1, $b = |M| - n = |M_{pref}|$.

Construction 1. *Let $\mathcal{P}$ be any pseudorandom permutation family on I_n, and let H be an ϵ-almost Δ-universal family of hash functions with domain I_b and range I_n. We construct a parsimonious PRF $ParG$ with domain I_{n+b} and range I_n as follows: A key of a function sampled from $ParG$ is a pair $\langle h, g\rangle$ where h is sampled from H and g is sampled from $\mathcal{P}$. For every input $x \in I_{n+b}$, we define the value of $ParG_{h,g}$ as $ParG_{h,g}(x) = g(h(x[1 \ldots b]) \oplus x[b+1 \ldots n+b])$.*

Remark. We observe that $ParG$ is parsimonious: given the output T and $x[1 \ldots b]$, it is easy to see $x[b+1 \ldots n+b] = g^{-1}(T) \oplus h(x[1 \ldots b])$.

Note that if $b < n$, we can simply append a fixed padding to the input x, to achieve total length $2n$; the security bounds we prove remain the same, and almost the exact same security proof will go through. Our construction is more efficient than [4] because, the CBC pass on the input in the first round in [4] has been replaced by a non-cryptographic Δ-universal hash applied to the input. We note that one can use this idea of applying a Δ-universal hash function to all but the last block to speed-up some MAC constructions in the Wegman-Carter paradigm [23] (e.g., UMAC [6]), especially for shorter messages.

At this point, the reader may feel that something is amiss because the task of dealing with variable input has been passed to the Δ-universal hash function without adequately dealing with all the issues. We deal with them individually:

- Variable Input Length: As previously mentioned, Δ-universal hash functions can be made to handle variable-length inputs by using standard techniques of padding and length appending; e.g., see UMAC [6].
- Large Universal Hash Keys: There are some universal hash functions whose key size grows linearly with the input, but not all suffer from this problem. Tree hashing makes the key size grow much slower (about logarithmic in the

input size). There are other universal hash function constructions whose key sizes are not dependent on the input size, but are rather dependent on the output size [14]. Again UMAC [6] is an example of how one can limit the key size of a universal hash function without compromising efficiency.

– Single Key: In the keyed model that we are working in, we are only given a single key K, yet we need keys for the universal hash function, a key for the block cipher g in $ParG_{h,g}$ and a key for the block cipher in the parsimonious encryption scheme S. To generate the needed keys, we run the FIL PRP or block cipher with key K and inputs $1, 2, \ldots, i$ and label the outputs $K_1, K_2, \ldots, K_i$. We can then use K_1 to key the block cipher g, and use K_2 to key the block cipher in the parsimonious encryption S. The rest of the keys can be used as the hash keys. We note that this key expansion step takes place once, so the amortized cost is minimal.

We state the security theorem, but leave the detailed proof for the full version of this paper.

Theorem 2. *Define ParG as in construction 1. Let ϵ_1 be the parameter associated with the Δ-universal family of hash functions in the construction, and suppose that the underlying pseudorandom permutation family $\mathcal{P}$ utilized by ParG is (t, q, n, ϵ_2)-secure. Then, for any adversary $\mathcal{A}$ restricted to t time steps, and q oracle queries of length at most $n + b$:*

$$\mathsf{Adv}^{\mathsf{prf}}_{ParG}(\mathcal{A}) = \Pr[g \xleftarrow{R} ParG : \mathcal{A}^g = 1] - \Pr[g \xleftarrow{R} \mathsf{Rand}^{n+b\to n} : \mathcal{A}^g = 1] \leq \binom{q}{2} \cdot \epsilon_1 + \epsilon_2.$$

Proof Sketch. We use the standard argument of demonstrating that the transcripts resulting from interacting with an idealized $ParG$ oracle are distributed identically to those from interacting with a truly random function so long as certain "bad" conditions do not occur. These bad conditions are related to the likelihood that the g component does not see the same input from two distinct queries. By the Δ-universal property of H, this happens with low probability.

5 FIL PRP to VIL SPRP

We now show how to convert a fixed-input-length block cipher that is secure against chosen plaintext attacks to a variable-input-length block cipher that is secure against both chosen plaintext and ciphertext attacks. This construction achieves an open goal stated in [4]. The VIL SPRP construction requires about 5 cryptographic passes over the input, thus it should be considered a first step in constructing more efficient VIL SPRPs. The idea is to first treat the original PRP as a PRF and create two different variable-input-length PRFs of specific lengths from it. Finally, we use these PRFs in an *unbalanced* Feistel network together with universal hash functions in the right places to yield the desired result (see figure 2). The construction we outline works when one needs to convert a block

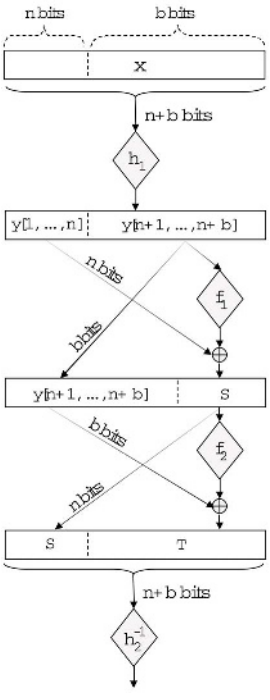

Fig. 2. Constructing a VIL SPRP from a PRP

cipher on I_n to a cipher on I_{n+b} where $b \geq n$. We can extend the ideas to work for the case when $b < n$, but there is a loss in security.

Construction 2. *Let $\mathcal{P}$ be any pseudorandom permutation family on I_n, and let H be a family of pairwise independent permutations on I_{n+b}, and let H' be a universal family of hash functions with domain I_b and range I_n. Define f_1 and f_2 as follows:*

$$f_1(x) \triangleq p_{k_0}(h'_1(x)), \tag{1}$$
$$f_2(x) \triangleq (p_{k_1}(x), p_{k_2}(x), \ldots, p_{k_r}(x))[1, \ldots, b] \tag{2}$$

where $r = \lceil \frac{b}{n} \rceil$ and $p_{k_0}, p_{k_1}, \ldots, p_{k_r}$ are independently keyed permutations drawn from $\mathcal{P}$, and h'_1 is drawn from H'. Now, we define a new permutation family $\mathcal{P}'$ which maps input $x \in I_{n+b}$ to $h_2^{-1}(S(x), T(x))$, where

$$\begin{aligned} y &\triangleq h_1(x), \\ S(x) &\triangleq y[1, \ldots, n] \oplus f_1(y[n+1, \ldots, n+b]), \text{ and} \\ T(x) &\triangleq y[n+1, \ldots, n+b] \oplus f_2(S(x)), \end{aligned}$$

where h_1, h_2 are drawn from H.

We state the security theorem. Due to space constraints, we sketch the proof.

Theorem 3. *Let $\mathcal{P}$ be a $(t, (1+\lceil \frac{b}{n} \rceil)q, n, \epsilon_1)$-secure pseudorandom permutation family on I_n, let H be a family of pairwise independent permutations on I_{n+b}, let H' be an ϵ_2 universal family of hash functions with domain I_b and range I_n, and let $\mathcal{P}'$ be the permutation family defined above. Then $\mathcal{P}'$ is $(t, q, n+b, \epsilon')$ secure where: $\epsilon' = \binom{q}{2}(2/2^n + 1/2^b + 1/2^{n+b-1} + \epsilon_2) + \epsilon_1$.*

Proof Sketch. The proof easily follows by first utilizing theorem 4.1 from the paper of Naor and Reingold [17]. We first assume that the underlying round functions are truly random (from which the final advantage can be bounded by several applications of the triangle inequality in a series of hybrid arguments in which we eventually replace the truly random functions with f_1 and f_2 as above). In order to invoke theorem 4.1 of [17], we need to identify the "BAD" conditions (as a function of h_1 and h_2 on the transcript of the adversary's interaction with the block cipher). Letting the input-output pairs be denoted (x_k, c_k) for $1 \leq k \leq q$ (where the adversary makes q queries), the condition $BAD(h_1, h_2)$ occurs whenever $h_1(x_i)[n+1, \ldots, n+b] = h_1(x_j)[n+1, \ldots, n+b]$ or $h_2(c_i)[1, \ldots, n] = h_2(c_j)[1, \ldots, n]$ for $1 \leq i < j \leq q$. By the strongly-universal property of the h_1, the first condition occurs with probability 2^{-b} and the second occurs with probability 2^{-n}. To complete the proof, one merely has to form the hybrid argument by showing that f_1 and f_2 are pseudorandom functions. To do so, one should first replace the PRP p with a PRF. Then, the proof that f_1 is pseudorandom is very similar to the proof that our parsimonious function from the previous construction is pseudorandom. It is also clear that f_2 is pseudorandom since it is the concatenation of invocations of a pseudorandom function on random and independently chosen keys. Now, the final hybrid step involves showing that PRPs are statistically close to PRFs, which is well known.

A few remarks are in place:

- Single Key Model: Since we only have a single key K for a block cipher, we need to specify how the rest of the keys are created. There are 4 rounds in our construction and keys are needed in each round. The first and fourth rounds need keys for the pairwise independent permutation h_1 and h_2. The second round needs keys for the universal hash h_1' and k_0. The third round needs keys $k_1, \ldots, k_r$. We create the other keys from the permutation, $p_K()$ by having arguments $p_K(roundnumber, index)$; for round 2, we set $k_0 = p_K(2, 0)$ and use index values 1 and higher to create the keys for the universal hash h_1'. For round 3, $k_1 = p_K(3, 1) \ldots k_i = p_K(3, i)$. For rounds 1 and 4, we cannot reuse keys created for a specific length input message as part of the keys for another larger length input message. The keys have to be independent. We achieve this by including a length parameter in the argument. To create keys for h_1 we would run $p_K(1, length, 1) \ldots p_K(1, length, i)$. Similarly for h_2 we would run $p_K(4, length, 1) \ldots p_K(4, length, i)$.
- Efficiency: In rounds two and three we have to basically do a cryptographic pass over the entire input. The keys $k_0, \ldots, k_r$ and the keys for h_1' are generated once and can be reused, hence their cost is not dominant when amortized over multiple runs. However, the keys for h_1 and h_2 cannot be reused

and they have to be created again for each separate message length. Since a pairwise independent permutation takes a key whose size is twice the message length, we need to do effectively two cryptographic passes for round one and another two cryptographic passes for round 4. Thus, for the Strong VIL construction we effectively need five cryptographic passes.

– Optimizations: Note that when a message length has been previously used, then the keys of h_1 and h_2 previously calculated for that length can be reused. Thus, a table of keys can be kept for each length. This makes sense in applications that only involve a few specific message lengths. We also note that the above construction can be optimized in several ways using some standard tricks from [17, 19, 20]. First, the pairwise independent functions can be replaced by Δ-universal hash functions; the security proof is very similar to the ones in [17] and the full version of [19] (but we can no longer simply use theorem 4.1 from [17]). We may further use the same key material in the PRF in rounds 2 and 3 by strengthening the condition on the hash function as was done in [19]. Finally, we can in some cases recycle the key material used in the outer round hash functions by considering Feistel group operations other than XOR as was done in [20]. If we were to replace the pairwise independent permutation in the first round with a Feistel-permutation with a pseudorandom round function (as we did in the third round), and use a Feistel-permutation with a Δ-universal round function for the last round, we can eliminate the need for any additional key generation phase, thereby allowing us to handle dynamic block lengths efficiently (i.e., we generate keys once and they can be used for any block length). These types of tricks are fairly standard, so we omit a full discussion due to space constraints.

6 Conclusion and Open Problems

The constructions in this paper have been motivated by one dominant thought: push the application of universal hash functions in all directions to create VIL primitives from FIL primitives. The harder part has been to know exactly which cryptographic operations can be replaced by universal hashes, what kind of universal hashes should be used (e.g., universal vs. Δ-universal hashes), and providing security proofs. Specifically:

1. We show how to construct a VIL PRP from a FIL PRP which is almost twice as fast as the previous construction [4].
2. We show how to construct a VIL SPRP from a FIL PRP which solves an open problem in [4].

There are many open problems remaining in constructing VIL block ciphers including the construction of a VIL PRP and a VIL SPRP from a FIL PRP in the oracle model and a more efficient VIL SPRP construction in either model. The problem of creating efficient and secure VIL PRPs and VIL SPRPs for messages smaller than two block lengths remains open.

References

[1] J. An and M. Bellare. Constructing VIL-MACs from FIL-MACs: Message authentication under weakened assumptions. In *Proc. CRYPTO 99*.

[2] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption: Analysis of the DES modes of operation. *FOCS 1997*.

[3] M. Bellare, J. Kilian, and P. Rogaway. The security of cipher block chaining. In *Proc. CRYPTO 94*.

[4] M. Bellare and P. Rogaway. On the construction of Variable-Input-Length ciphers. In *Proc. Fast Software Encryption*, 1999.

[5] D. J. Bernstein. How to stretch random functions: The security of protected counter sums. *J. Cryptology*, 12(3):185–192, Summer 1999.

[6] J. Black, S. Halevi, H. Krawczyk, T. Krovetz, and P. Rogaway. UMAC: fast and secure message authentication. In *Proc. CRYPTO 99*.

[7] D. Bleichenbacher and A. Desai. A construction of a super-pseudorandom cipher. Manuscript, February 1999.

[8] J. L. Carter and M. N. Wegman. Universal classes of hash functions. *JCSS*, 18(2):143–154, April 1979.

[9] J. Daemen and V. Rijmen. AES proposal Rijndael. NIST AES Proposal, 6/98.

[10] I. Damgård. A design principle for hash functions. In *Proc. CRYPTO 89*.

[11] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, October 1986.

[12] S. Halevi and P. Rogaway. A tweakable enciphering mode. In *Proc. CRYPTO '03*.

[13] S. Halevi and P. Rogaway. A parallelizable enciphering mode. In *Proc. RSA Conference, Cryptographer's Track '04*.

[14] H. Krawczyk. LFSR-based hashing and authentication. In *Proc. CRYPTO 94*.

[15] M. Luby and C. Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM J. Computing*, 17(2):373–386, April 1988.

[16] M.Bellare, R. Canetti, and H. Krawczyk. Pseudorandom functions revisited: The cascade construction and its concrete security. In *Proc. FOCS 1996*.

[17] M. Naor and O. Reingold. On the construction of pseudo-random permutations: Luby-Rackoff revisited. *J. of Cryptology*, 12:29–66, 1999. Previously in STOC 97.

[18] National Bureau of Standards. FIPS publication 46: Data Encryption Standard, 1977. Federal Information Processing Standards Publication 46.

[19] S. Patel, Z. Ramzan, and G. Sundaram. Towards making Luby-Rackoff ciphers optimal and practical. In *Proc. Fast Software Encryption*, 1999. Full version available from `http://theory.lcs.mit.edu/~zulfikar/MyResearch/homepage.html`.

[20] S. Patel, Z. Ramzan, and G. Sundaram. Luby-Rackoff ciphers: XOR is not so exclusive. In *Proc. Selected Areas of Cryptography*, 2002.

[21] E. Petrank and C. Rackoff. CBC MAC for Real Time Data Sources. Technical Report 97-26, Dimacs, 1997.

[22] D. R. Stinson. Universal Hashing and Authentication Codes. *Design, Codes, and Cryptography,* 4:369–380, 1994. Preliminary version appeared at CRYPTO 1991.

[23] Mark N. Wegman and J. Lawrence Carter. New hash functions and their use in authentication and set equality. *JCSS*, 22(3):265–279, June 1981.