A Pseudo-Random Encryption Mode

Moni Naor *   Omer Reingold †

Block ciphers are *length-preserving* private-key encryption schemes. I.e., the private key of a block-cipher determines a permutation on strings of the length of its input. This permutation is used for encryption while the inverse permutation is used for decryption. Using a length-preserving encryption scheme saves on memory and prevents wasting communication bandwidth. Furthermore, it enables the easy incorporation of the encryption scheme into existing protocols or hardware components.

Often, block-ciphers have fixed (and relatively small) input length. This is especially true for hardware implementations. For example, the highly influential Data Encryption Standard (DES) has input length of 64 bits.[1] In such a case, the block-cipher is used in some mode of operation that enables the encryption of longer messages. The standard modes of operation (proposed in the context of DES) are ECB, CBC, CFB and OFB. Unfortunately, all these modes reveal information on their inputs or on relations between different inputs. For instance, when using the CBC-mode, the encryptions of two messages with identical prefix will also have an identical prefix. The ECB-mode reveals even more information (i.e., equalities between any pair of plain-text blocks).

For some applications it is essential to have better security than that of the standard modes of operation. Such security is formalized by the concept of a pseudo-random permutation: Let f be a pseudo-random permutation, then if the encryption of a message M is $f(M)$ then the only information this encryption leaks on M is whether or not M is equal to a previously encrypted message. For further discussion on the usage of pseudo-random permutations for encryption (and on the usage of length-preserving encryption in general) see [3, 5, 9].

This note describes a mode of operation for block-ciphers that achieves a strong notion of security: If the original block-cipher is a pseudo-random permutation then we get a pseudo-random permutation on the entire message (see a more quantitative statement below). The description is extracted from [9] where a framework for constructing and proving the security of pseudorandom permutations is introduced. In such a construction a pseudo-random permutation Π is defined to be the composition of three permutations: $\Pi \equiv h_2^{-1} \circ A \circ h_1$. In general, h_1 and h_2^{-1} are "lightweight," and A is where most of the work is done. Intuitively, there are only a few bad inputs for A and the role of h_1 and h_2^{-1} is to "filter" out these inputs.

The Mode

Let $E : \{0,1\}^\ell \mapsto \{0,1\}^\ell$ denote an encryption permutation on ℓ bits and let E^{-1} be its inverse permutation. We define an encryption permutation $\Pi : \{0,1\}^{b\cdot\ell} \mapsto \{0,1\}^{b\cdot\ell}$ on b blocks of ℓ bits. In

*Dept. of Applied Mathematics and Computer Science, Weizmann Institute of Science, Rehovot 76100, Israel. Research supported by a grant from the Israel Science Foundation administered by the Israeli Academy of Sciences. E-mail: naor@wisdom.weizmann.ac.il.

†Dept. of Applied Mathematics and Computer Science, Weizmann Institute of Science, Rehovot 76100, Israel. Research supported by a Clore Scholars award. E-mail: reingold@wisdom.weizmann.ac.il.

[1] RC5 may be considered as a counter example though there the block size is directly related to the word-size.

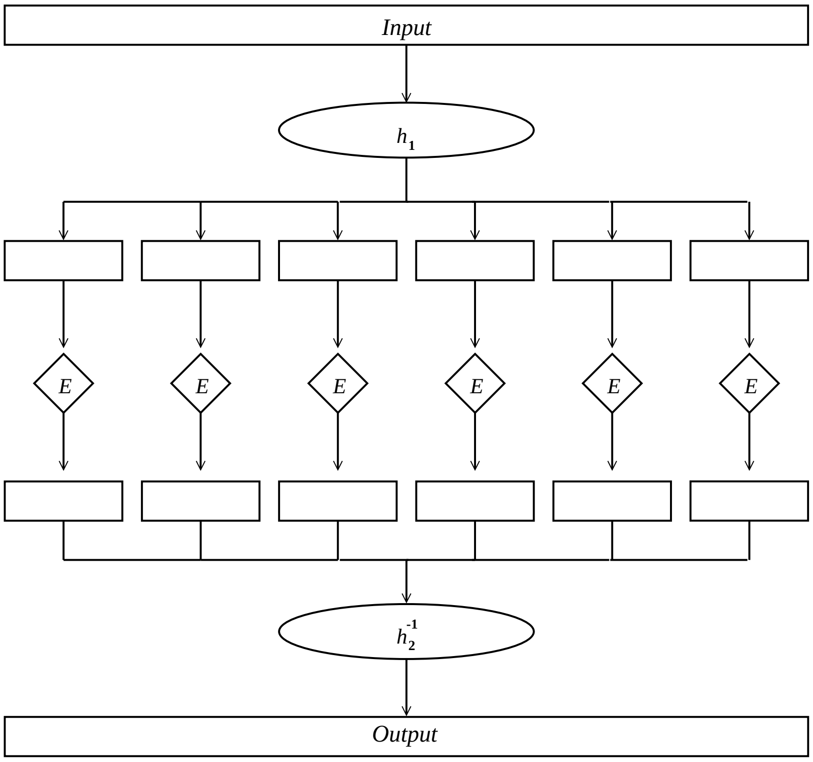

Figure 1: The Constructions of Π for $b = 6$.

fact, Π can be defined to get an input with arbitrary number of blocks. In the definition of Π we use two "non-cryptographic" (but secret) permutations, h_1 and h_2, on $b \cdot \ell$ bits that will be defined in the subsequent.

Definition of $\Pi : P_1, P_2, \ldots, P_b \mapsto C_1, C_2 \ldots, C_b$ (see Figure 1 for an illustration):

Given b blocks of ℓ bits $P_1, P_2, \ldots, P_b$ as input, Π executes the following algorithm:

1. Compute $(I_1, I_2 \ldots, I_b) = h_1(P_1, P_2, \ldots, P_b)$, where each I_j is an ℓ-bit string.
2. For $1 \leq j \leq b$ compute $O_j = E(I_j)$.
3. Output $(C_1, C_2 \ldots, C_b) = h_2^{-1}(O_1, O_2, \ldots, O_b)$.

Inversion is just as simple:

1. Compute $(O_1, O_2 \ldots, O_b) = h_2(C_1, C_2, \ldots, C_b)$, where each I_j is an ℓ-bit string.
2. For $1 \leq j \leq b$ compute $I_j = E^{-1}(O_j)$.
3. Output $(P_1, P_2 \ldots, P_b) = h_1^{-1}(I_1, I_2, \ldots, I_b)$.

Implementing h_1 and h_2

The crucial point for the application of the mode are the requirements from h_1 and h_2 (which are formally defined in [9]). We now describe one efficient way to construct h_1 and h_2. In this

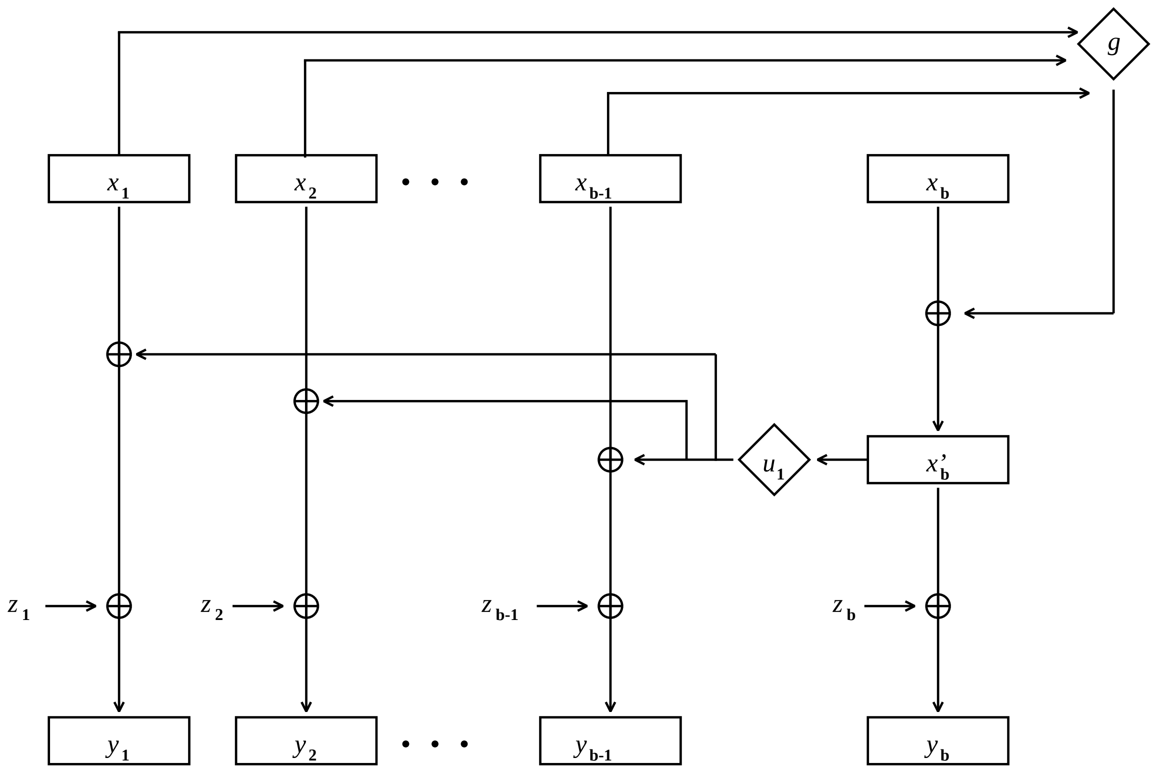

Figure 2: The Constructions of h_1. Each z_i denotes the string $u_2(i)$

construction we apply the concept of ϵ-AXU$_2$ functions originally defined by Carter and Wegman [6]; We use the terminology of Rogaway [10]:

Definition 1 *A distribution on $\{0,1\}^m \mapsto \{0,1\}^n$ functions H is ϵ-AXU$_2$ if for every pair of m-bit strings $x \neq y$ and every $z \in \{0,1\}^n$*

$$\Pr_{h \in H}[h(x) \oplus h(y) = z] \leq \epsilon$$

where $\oplus$ denotes bit-by-bit exclusive-or.

For constructions of ϵ-AXU$_2$ functions see e.g. [6, 7, 8, 10, 11, 12]. We note that some of these constructions are extremely efficient (both in computation and in key-size). We are mainly interested in ϵ-AXU$_2$ functions with domain $\{0,1\}^m$ and range $\{0,1\}^n$, where $m \gg n$. In this case, a very attractive construction can be based on the functions of Halevi and Krawczyk [7]. Their functions are based on an inner product with a random vector over a moderate size field. The rate their implementation achieves is a few machine cycles per word.

The functions h_1 and h_2 can be implemented in the same way. Therefore, it is sufficient to concentrate on the definition of h_1. In this definition we use three ϵ'-AXU$_2$ functions:

$$\begin{aligned} g &: \{0,1\}^{\ell\cdot(b-1)} \mapsto \{0,1\}^{\ell}, \\ u_1 &: \{0,1\}^{\ell} \mapsto \{0,1\}^{\ell} \textit{ and} \\ u_2 &: \{0,1\}^{\lceil \log b \rceil} \mapsto \{0,1\}^{\ell} \end{aligned}$$

Definition of $h_1 : x_1, x_2, \ldots, x_b \mapsto y_1, y_2, \ldots, y_b$ (see Figure 2 for an illustration):

Given b blocks of ℓ bits $x_1, x_2, \ldots, x_b$, the value $h_1(x_1, x_2, \ldots, x_b)$ is computed as follows:

1. Compute $x'_b = x_b \oplus g(x_1, x_2, \ldots, x_{b-1})$, where $\oplus$ denotes bit-by-bit exclusive-or.
2. Output $y_1, y_2, \ldots, y_b$, where for every $1 \le i \le b-1$ the string y_i is defined as $x_i \oplus u_1(x'_b) \oplus u_2(i)$ and y_b is defined to be $x'_b \oplus u_2(b)$.

Inversion is also simple:

1. Compute $x'_b = y_b \oplus u_2(b)$.
2. For every $1 \le i \le b-1$ compute $x_i = y_i \oplus u_1(x'_b) \oplus u_2(i)$.
3. Compute $x_b = x'_b \oplus g(x_1, x_2, \ldots, x_{b-1})$
4. Output $x_1, x_2, \ldots, x_b$.

A Few Words on Efficiency

Evaluating h_1 (or h_1^{-1}) is essentially equivalent to one computation of g (a $\{0,1\}^{\ell \cdot (b-1)} \mapsto \{0,1\}^{\ell}$ function), one computation of u_1 (a $\{0,1\}^{\ell} \mapsto \{0,1\}^{\ell}$ function) and a few additional XOR operations per block. The values $u_2(i)$ are independent of the input and can therefore be computed in advance. Furthermore, computing $u_2(i+1)$ from $u_2(i)$ can be done easily. Therefore, the dominant factor seems to be the computation of g. Using efficient constructions of ϵ-AXU$_2$ functions from large inputs to small outputs (e.g. those of [7]), we get an efficient function h_1. Since apart from evaluating h_1 and h_2^{-1}, computing Π consists of one invocation of E per block we get a very efficient mode of operation.

An important feature of this mode is that Π is highly parallelizable. The only global operation is the computation of g (and its parallelism depends on the exact choice of g). This is obviously important for implementation on parallel machines or in hardware but also has an additional advantage: It is possible that *even on a sequential machine* b parallel evaluations of E would be faster than b sequential evaluations. An interesting way to amortize several (e.g., 64) evaluation of DES was proposed by Biham [4].

A Few Words on Security

In [9] it is proven that if E is a pseudo-random permutation then so is Π. As noted above, this gives a strong notion of security. The only other place we are aware of that explicitly refers to the problem of constructing a pseudo-random permutation on the entire message is an unpublished work of Bellare and Rogaway [3]. They show how to convert the CBC-mode in order to get such a construction. The amount of work in their construction is comparable with two applications of the original CBC-mode (approximately twice the work of our construction).

It should be noted that Π in our construction is vulnerable to a birthday attack on the size of the original block. In a sense, this is the best attack on Π. Moreover, most of the known modes of operation share this vulnerability (see [2] for the CBC-mode). More quantitatively, the best advantage in distinguishing Π from a random permutation with q queries is essentially bounded by the best advantage in distinguishing E from a random permutation with $q \cdot b$ queries plus

$$(q \cdot b)^2 \cdot \left(\frac{1}{2^{\ell-1}} + \epsilon\right)$$

where $\epsilon = 2^{-\ell} - \epsilon'$ can be made as small as we wish. Therefore, in order for Π to be secure, it should be infeasible to make queries with a total number of $2^{\ell/2}$ blocks.

Even if the original block-size, ℓ, is too small (e.g., $\ell = 80$) we can still use our mode in the following way: first apply a more secure transformation (see [1] for work in this direction) on E to get a permutation E' on 2 blocks and then apply our construction with E' instead of E. Such an application makes sense since it retains most of the simplicity and efficiency of our mode.

Discussion

This note describes a simple mode of operation that is *pseudo-random.* I.e., it achieves security in a very strong sense: if the original block-cipher *hides all information on the input* (apart of equality between messages) than so does the resulting block-cipher. As described above, the standard modes of operation such as ECB and CBC leak additional information on their inputs. However, there is an inherent price to pay for such strong security: For *any* pseudo-random permutation, no part of its output can be computed before getting the entire input. This might be a problem when encrypting long data streams (since it requires memory space which is as large as the entire message and since it introduces additional latency to the system). Nevertheless, there are other scenarios where using this mode seems both feasible and desirable. For example, when encrypting data files the whole plaintext is available beforehand. Moreover, it might be important not to give an adversary that monitors the disc any information about locations of changes in files (or about the content of these changes). An additional way of using the mode of operation proposed here is by combining it with another mode such as CBC. The idea is to use the pseudo-random mode for *increasing the block-length* (e.g., from 64 bits to 1000 bytes) and then to apply the CBC-mode to the resulting block-cipher. Such a “combined-mode” is better than the pseudo-random mode in terms of latency and storage and better than the CBC-mode in terms of the information revealed on the input.

Acknowledgments

We thank Ran Canetti and Shai Halevi for many helpful comments.

References

[1] W. Aiello and R. Venkatesan, Foiling Birthday Attacks in Length-Doubling Transformations, *Advances in Cryptology - EUROCRYPT '96*, Lecture Notes in Computer Science, Springer-Verlag, 1996, pp. 307-320.

[2] M. Bellare, J. Kilian and P. Rogaway, The security of cipher block chaining, *Advances in Cryptology - CRYPTO '94*, Lecture Notes in Computer Science, vol. 839, Springer-Verlag, 1994, pp. 341-358.

[3] M. Bellare and P. Rogaway, Block cipher mode of operation for secure, length-preserving encryption, manuscript.

[4] E. Biham, A fast new DES implementation in software, *Proc. Fast Software Encryption*, Lecture Notes in Computer Science, Springer-Verlag, 1997.

[5] M. Blaze, J. Feigenbaum and M. Naor, A formal treatment of remotely keyed encryption, manuscript.

[6] L. Carter and M. Wegman, Universal hash functions, *J. of Computer and System Sciences*, vol. 18, 1979, pp. 143-154.

[7] S. Halevi and H. Krawczyk, MMH: message authentication in software in the Gbit/second rates, *Proc. Fast Software Encryption*, Lecture Notes in Computer Science, Springer-Verlag, 1997.

[8] H. Krawczyk, LFSR-based hashing and authentication, *Advances in Cryptology - CRYPTO '94*, Lecture Notes in Computer Science, vol. 839, Springer-Verlag, 1994, pp. 129-139.

[9] M. Naor and O. Reingold, On the construction of pseudo-random permutations: Luby-Rackoff revisited, To appear in: *J. of Cryptology.* Preliminary version in: *Proc. 29th Ann. ACM Symp. on Theory of Computing*, 1997. pp. 189-199.

[10] P. Rogaway, Bucket hashing and its application to fast message authentication, *Advances in Cryptology - CRYPTO '95*, Lecture Notes in Computer Science, vol. 963, Springer-Verlag, 1995, pp. 74-85.

[11] D. Stinson, Universal hashing and authentication codes, *Designs, Codes and Cryptography*, vol. 4 (1994), pp. 369-380.

[12] M. Wegman and L. Carter, New hash functions and their use in authentication and set equality, *J. of Computer and System Sciences,* vol. 22, 1981, pp. 265-279.