Comments to NIST concerning AES Modes of Operation:

OCB Mode: Parallelizable Authenticated Encryption

Phillip Rogaway
University of California at Davis (USA) and
Chiang Mai University (Thailand)
rogaway@cs.ucdavis.edu
www.cs.ucdavis.edu/∼rogaway

Preliminary Draft: October 16, 2000

Abstract

This note describes a parallelizable block-cipher mode of operation that simultaneously provides privacy and authenticity. It does this using only $\lceil |M|/n \rceil + 2$ block cipher invocations. Here M is the plaintext (an arbitrary bit string) and n is the block length. The scheme refines one recently suggested by Jutla [Ju00].

1 Introduction

Background When message privacy is a goal, message authenticity often is, too. The correct approach for achieving privacy-plus-authenticity has been to encrypt the plaintext and, separately, compute a message authentication code (MAC). Done in this way, the cost for privacy-plus-authenticity is about the cost to encrypt plus the cost to MAC.

A recent paper by **Jutla** gives a computationally cheaper alternative [Ju00]. Jutla's modes of operation, IACBC and IAPM, provide privacy-plus-authenticity at a cost lower than the cost to encrypt plus the cost to MAC. In fact, the cost of IACBC and IAPM—at least for long messages—isn't much more than the cost of CBC mode or CTR mode, respectively. Jutla's design is a lovely and timely development.

A related paper by **Gligor** and **Donescu** offers a different privacy-plus-authenticity mode of operation [GD00]. Their XCBC mode is CBC-like, like Jutla's IACBC. As in Jutla's scheme, blocks are offset after they are enciphered. But [GD00] simplifies the way that offsets are computed.

OCB Mode This note builds on [Ju00]. We describe a new mode of operation, OCB mode (Offset CodeBook), which refines Jutla's IAPM scheme in some significant, though rather low-level, ways. Like Jutla's IAPM, the new mode is parallelizable: the work for computing the different ciphertext blocks can be done at the same time. We believe this to be an important attribute in support of good hardware and software speed. Some further properties of OCB include:

- *Arbitrary domain.* Any string $M \in \{0,1\}^*$ can be encrypted; in particular, $|M|$ need not be a multiple of the block length n.
- *Short ciphertexts.* The way we extend the domain to $\{0,1\}^*$ is *not* to pad to a multiple of n. That would lead to a lengthening of the ciphertexts. Our ciphertexts are, instead, as short as possible.
- *Fewer block-cipher calls.* Our mode uses only $\lceil |M|/n \rceil + 2$ block-cipher invocations. Minimizing the number of block-cipher calls is especially important when messages are short. In many domains, short messages are quite common.
- *Nonces.* Our mode requires a nonce (often called an IV in this context). The nonce must be non-repeating, but it does *not* have to be unpredictable. Requiring of a nonce only that it be non-repeating is less error prone for the user, and it is often more efficient as well, since constructing an unpredictable value would usually require making an additional block-cipher call.

- *Stride associated to key, not to message.* We generate our sequence of offset values in a computationally cheaper way than [Ju00]. Namely, the nonce $Nonce$ is mapped to an unpredictable value R by a single application of the underlying block cipher. This value R forms the initial offset. The ith additional offset is obtained by adding to R an amount iL, where the "stride" L does *not* depend on $Nonce$—it depends only on the key. As such, the stride L need be computed only once.
- *Refinements to multiplication and addition semantics.* For forming the offsets $+iL + R$, we describe three possible instantiations for addition and multiplication. Each refines those suggested before.
- *Single underlying key.* The key K used for the encryption mode is a single block-cipher key, and all block-cipher invocations are keyed using this one key. (However, it is still convenient to store the stride L, and failing to do so will increase the cost by one block-cipher call.)

Comparisons Neither [Ju00] nor [GD00] worry about type of low-level concerns which drove the work here, namely: (1) aggressively minimizing the number of block-cipher calls; (2) what to do when $|M|$ is not a positive multiple of n; (3) avoiding multiple encryption keys; and (4) making sure that non-repeating (non-random) nonces work fine. We maintain that if such goals are eventually to be sought, they have to be addressed from the beginning. The reason is that these are very "fragile" schemes—tweak them a little and they usually break—making it surprisingly difficult to achieve the listed goals. Similarly, small algorithmic changes completely invalidate any proofs.

As we have indicated, OCB mode resembles Jutla's IAPM [Ju00]. The main differences are: (1) factoring the offset-calculations so that much of the work is done only once; (2) further tricks for faster offset calculations; (3) dealing with "short" messages in a correct and optimal way; and (4) a type of "lazy" key separation;

No parallelizable encryption scheme is given in [GD00]. But [GD00] includes the idea of offsetting ciphertext blocks by multiples of a hidden value L, modulo 2^n. A related idea, offsetting the ith block by $(a + bi) \bmod p$ (where a and b are associated to the random value r used to encrypt the message) is mentioned, albeit briefly, in [Ju00].

Security The security claims about OCB encryption are semantic security under adaptive chosen-plaintext attack (CPA) [BDJR97, GM84], and integrity of ciphertexts, in the sense of [BN00, BR00, KY00]. Proving security is ongoing work, being done jointly with Mihir Bellare and John Black. As the proofs are technical and not yet complete, it is possible that some unforeseen issue will arise; the current algorithms should be considered provisional.

We point out that, by a result of [BN00], semantic security under CPA, coupled with authenticity of ciphertexts, implies semantic security under chosen-ciphertext attack (CCA). This, in turn, implies non-malleability. We believe that non-cryptographers implicitly assume properties like non-malleability when designing their higher level-protocols, and so a scheme which is CCA-secure is less likely to be misused.

2 Notation

Fix a block cipher E which enciphers an n-bit string X using a k-bit key K, obtaining ciphertext block $Y = E_K(X)$. For $E = \text{AES}$ we have $n = 128$ and $k \in \{128, 192, 256\}$.

The authentication tag which each ciphertext includes can have any number of bits, $tagLen$, from 1 to n; one uses the $tagLen$-bit prefix of an n-bit string. A standard should allow such tag-truncation since tags in excess of 80 bits, say, utilize extra bits but provide no meaningful increment to security. A default value of $tagLen = 64$ is probably good.

By 0^i and 1^i we means strings of i 0's and 1's, respectively. For A a string of length less than n, by $pad_n(A)$ we mean the string $0^{n-|A|-1}1\,A$: that is, prepend 0-bits and then a 1-bit so as to get to length n. (Appending a 1-bit and then 0-bits would also be fine.)

If A is a binary string then $|A|$ denotes its length, in bits. If A and B are strings then AB denotes their concatenation. If A and B are strings of equal length then $A \oplus B$ is their bitwise XOR and $A \vee B$ is their bitwise OR and $A \wedge B$ is their bitwise AND. By $A[\text{bit } i]$ we mean the i-th bit of A (regarded, where necessary, as the number 0 or the number 1), where characters are numbered left-to-right, starting at 1. By $A[\text{bits } \ell \text{ to } r]$ we mean $A[\text{bit } \ell]A[\text{bit } \ell + 1] \cdots A[\text{bit } r]$.

3 OCB Encryption (in general, and OCB/add)

Addition and multiplication We assume two operations: an addition operator $+ : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ and a multiplication operator denoted $\cdot : \mathsf{N} \times \{0,1\}^n \to \{0,1\}^n$, where $\mathsf{N} = \{1,2,3,\cdots\}$. Henceforth we omit the multiplication symbol. For concreteness, we now give these two operators a particular instantiation. Later we will revise this meaning to demonstrate a couple of further possibilities.

OCB/add For the addition modulo 2^n version of OCB, instantiate $+$ by computer addition of n-bit words (ignoring any carry) and instantiate iL, for $i \geq 1$, by repeated addition.

(A more formal definition follows. Let $A, B \in \{0,1\}^n$. By $\mathsf{str2num}(A)$ we mean the nonnegative integer that is represented by A, that is, $\sum_{i=1}^{n} 2^{n-i} A[\text{bit } i]$. If a is an integer then $\mathsf{num2str}_n(a)$ is the unique n-bit string A such that $\mathsf{str2num}(A) = a \bmod 2^n$. By $A + B$ we denote $\mathsf{num2str}_n(\mathsf{str2num}(A) + \mathsf{str2num}(B))$. By iA, where i is an integer, we mean the string $\mathsf{num2str}_n(i \cdot \mathsf{str2num}(A))$. The $\cdot$ symbol in the last expression means multiplication in the integers.)

Given a k-bit key K, derive from it a key L by way of $L = E_K(1^n) \vee 0^{n-1}1$. This forces L to be odd.

Nonces Encryption under OCB mode requires a nonce, *Nonce*. The nonce would typically be a counter (maintained by the sender) or a random value. One particular nonce value, $\mathit{Nonce} = 1^n$, is prohibited. Security is maintained even if the adversary can control the nonce (subject to the constraint that, during the adversary's chosen-plaintext attack, no nonce may be repeating and no nonce may be 1^n).

Definition of OCB We now define OCB. When addition and multiplication are as just given, we are defining OCB/add. Let M be the message we wish to encrypt using OCB mode. Let *Nonce* be the nonce, a string of length n. The stride L is defined from K in the manner we have specified. OCB encryption is given by the following algorithm. See Figure 1 as well.

```
Algorithm OCB-Enc          // Encrypt $M$ using $(K, L)$ and Nonce, and block cipher $E$
$R = E_K(\mathit{Nonce})$
Let $m = \max\{1, \lceil |M|/n \rceil\}$
Let $M[1], \ldots, M[m]$ be strings s.t. $M[1] \cdots M[m] = M$ and $|M[i]| = n$ for $1 \leq i < m$
for $i = 1$ to $m - 1$ do
        $C[i] = E_K(M[i] \ + \ (iL + R)) \ + \ (iL + R)$
if $|M[m]| = n$ then $\mathit{Mask} = E_K(mL + R) \ + \ (mL + R)$
        $C[m] = M[m] \oplus \mathit{Mask}$
        $\mathit{PreTag} = (M[1] \oplus \cdots \oplus M[m-1] \oplus M[m]) \ + \ ((m+1)L + R)$
        $\mathit{Tag} = E_K(\mathit{PreTag})$
    else $W = \mathit{pad}_n(M[m])$
        $\mathit{Mask} = E_K(mL + R) \ + \ (mL + R)$
        $C[m] = M[m] \oplus \mathit{Mask}[\text{bits } n - |M[m]| + 1 \text{ to } n]$
        $\mathit{PreTag} = (M[1] \oplus \cdots \oplus M[m-1] \oplus W) \ + \ ((m+1)L + R)$
        $\mathit{Tag} = E_K(\mathit{PreTag}) \ + \ ((m+2)L + R)$
$C = C[1] \cdots C[m]$
$T = \mathit{Tag}[1..\mathit{tagLen}]$
return ( $\mathit{Nonce}, \ C, \ T$ )
```

The description above is a functional one; in an implementation, no multiplications would be performed—repeated additions would be used instead, as in:

```
$\mathit{Offset} = R$
for $i = 1$ to $m - 1$ do
        $\mathit{Offset} = \mathit{Offset} + L$
        $C[i] = E_K(M[i] \ + \ \mathit{Offset}) \ + \ \mathit{Offset}$
```

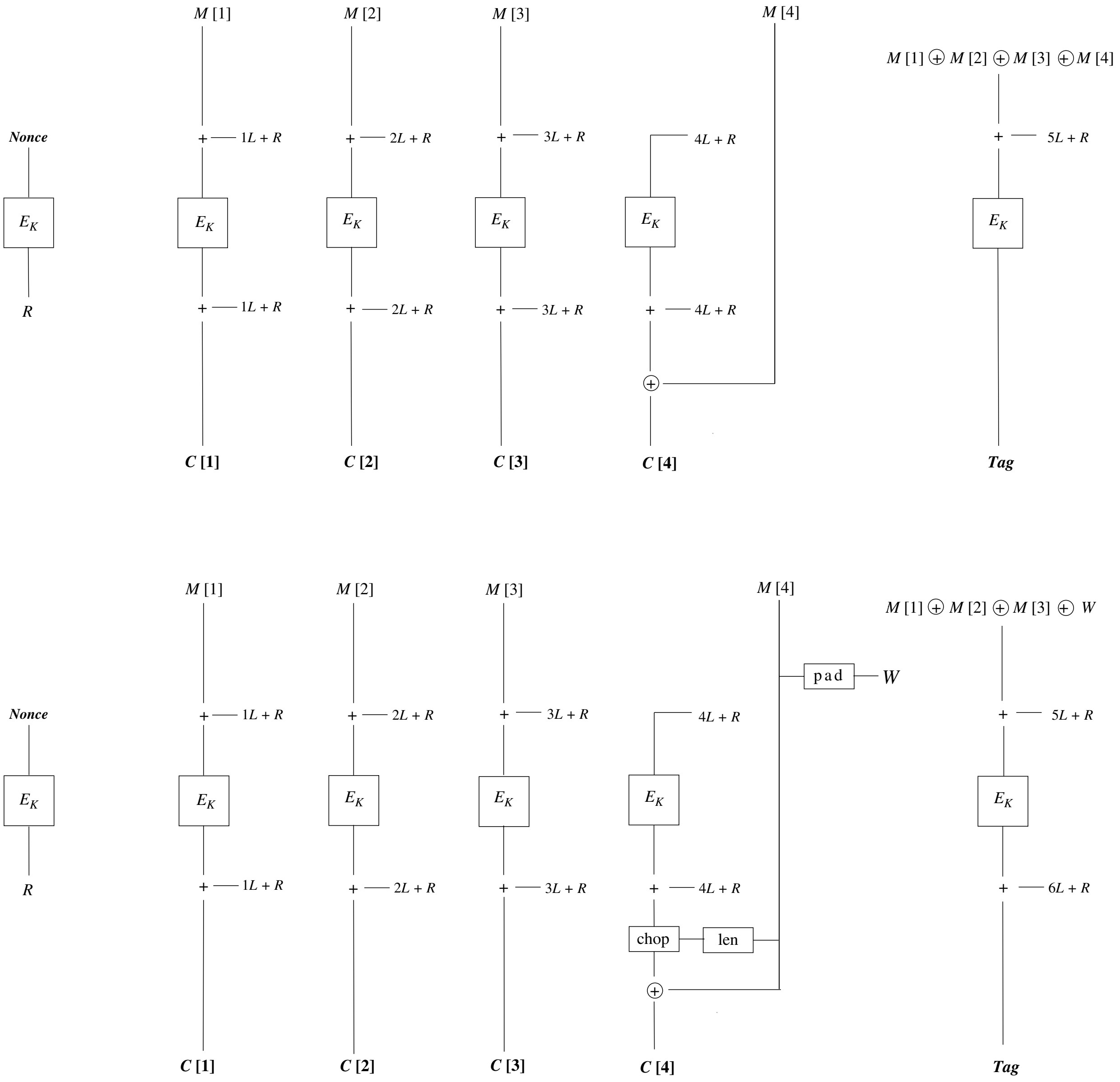

Figure 1: *OCB Encryption of a four-block message $M = M[1]M[2]M[3]M[4]$. The top half shows what happens when all four blocks are full n-bit blocks. The bottom half shows what happens when the final block has length less than n. In either case, Nonce as a non-repeating value. The stride L is determined from the underlying key K. Calculate $C = C[1]C[2]C[3]C[4]$ and Tag as shown, and transmit Nonce, C, and a prefix T of Tag. Addition and mulitplication can be given several different meanings, as discussed in the text.*

Scheme	Meaning of $A + B$	Meaning of iL	Definition of L
OCB/add	Add 128-bit numbers. Ignore any carry	Repeated addition (as defined in the prior column)	$E_K(1^{128}) \vee 1$
OCB/mod	Add 128-bit numbers mod p.	Repeated addition (as defined in the prior column)	$E_K(1^{128})$
OCB/xor	XOR	Multiply $\gamma(i)$ by L in GF(2^{128}), where $\gamma(i)$ is the ith word in canonical Gray-code ordering	$E_K(1^{128}) \wedge \text{Const}$

Figure 2: Three instantiation possiblities for OCB. Here $A, B \in \{0,1\}^{128}$ and $i \in \{1, 2, 3, \ldots\}$. The underlying key is K and L is derived from K as specified in the rightmost column.

Because of the chain of additions used to make the *Offset*-values, the description above might seem to imply that OCB (without multiplies) is sequential. This is not correct. To illustrate what goes on in a parallel implementation, suppose one has two processors, P_1 and P_2, and one wants to OCB-encrypt $M = M[1] \cdots M[m]$. Start processor P_1 with $\mathit{Offset} = \mathit{Nonce}$, and start processor P_2 with $\mathit{Offset} = \mathit{Nonce} + L$. Processor P_1 will be responsible for odd-indexed words while P_2 will handle even-indexed ones. Each increments its own *Offset* by $2L$, not by L. While enciphering its blocks processor P_1 computes its contribution to the authentication tag, as does P_2. One of the processors will compute the final tag.

Decryption (with authentication check) of a ciphertext $(\mathit{Nonce},\ C[1] \cdots C[m],\ T)$ is the obvious algorithm: compute $M[1] \cdots M[m]$ from $C[1] \cdots C[M]$ and *Nonce*, recompute the T, and see if it matches the tag received. If the full tag is available then process can be defined by a "depth 1" circuit in E: compute E_K^{-1} for the tag.

4 OCB/mod: Trading the Ring $\mathsf{Z}/2^n\mathsf{Z}$ for the Field Z_p

In this section we expand upon a suggestion made by [Ju00] and compute the offsets modulo a prime p. We are not suggesting that multiple authenticated encryption schemes should be standardized—rather, we are admitting that a bit more work (experimental and theoretical) is needed in order to make a well-informed choice.

Fix a prime $p = 2^n - \delta$ just smaller than 2^n (e.g., choose the largest prime less than 2^n). Jutla suggests [Ju00, p. 4] that, when it is time to encrypt a message, a random value r is selected and r is then mapped, using the underlying block cipher, into IV_1 and IV_2. One presumes these to be numbers in $[0..p-1]$ or $[0..2^n - 1]$. The ith offset is then calculated as $(IV_1 + iIV_2) \bmod p$. This can be implemented with repeated additions, each modulo p.

As an optimization, the value IV_2 need *not* vary with each message. It plays the same role as the stride L, and can be computed in a way that depends only on the underlying key K. Doing this saves one block-cipher invocation with every message encrypted. The stride L should no longer be chosen to be an odd number; set $L = E_K(1^n)$ instead. See Figure 2.

This OCB/mod approach approach is still efficient, but it is less efficient and more involved than OCB/add. What has been gained? The security bound will be a little better, nothing more. Details will be in the full paper.

A second trick can be used, but at some small cost, it would appear, to the security bound. Instead of reducing all sums modulo p, redefine the semantics of addition by saying that $A+B$ is the n-bit sum where, whenever you generate a carry, you must increment the sum by δ, where $p = 2^n - \delta$. A few points in the field now have two representations.

5 OCB/xor: A Gray-Code Trick and the Field $\mathrm{GF}(2^n)$

In this section we describe yet another method of offsetting the blocks $M[1], M[2], \ldots, M[m-1]$: we will change the semantics of $+$ to be XOR (that is, addition in GF(2^n)) and we will change the semantics of iL as well. When $\bmod\ 2^{128}$ additions are inconvenient, this approach may be preferred. We assume in this section that $n = 128$.

Notation If i is a positive integer then $\mathsf{ntz}(i)$ is the number of trailing 0's in the binary representation of i. So, for example, $\mathsf{ntz}(1) = \mathsf{ntz}(3) = 0$, $\mathsf{ntz}(2) = 1$, and $\mathsf{ntz}(24) = 3$. If L is an n-bit string, then $L\ll 1$ means a left shift of L by one bit (msb disappearing, and a zero coming into the lsb). Similarly, $L\gg 1$ means a right shift of L by one bit (lsb disappearing, and a zero coming into the msb).

Algorithm Given a key K for E derive from it an n-bit key L by way of $L = E_K(1^n) \wedge 0^2 1^{30} 0^2 1^{30} 0^2 1^{30} 0^2 1^{30}$. This ensures that the top two bits of every 32-bit word are zero, allowing for some pleasant implementation optimizations. Now define $L(0) = L$ and, for $i \geq 0$, define

$$L(i+1) \quad = \quad \begin{cases} L(i)\ll 1 & \text{if } \mathsf{msb}(L(i)) = 0 \\ (L(i)\ll 1) \oplus 0^{120}10^4 1^3 & \text{if } \mathsf{msb}(L(i)) = 1 \end{cases}$$

Now given a string M, the OCB algorithm proceeds as we have defined already, but with addition being defined as bitwise XOR, and iL being defined by

$$iL \quad = \quad \begin{cases} 0^n & \text{if } i = 0 \\ (i-1)L \oplus L(\mathsf{ntz}(i)) & \text{if } i \geq 1 \end{cases}$$

Note that each offset is obtained from the previous one by XORing it with the appropriate $L(i)$. The $L(i)$ values can be computed once, in advance, or they can be computed on the fly with the specified bit twiddling.

Explanation The following explanation assumes more mathematical background than the rest of this document. Understanding this explanation is not needed for understanding the algorithm's definition.

The algorithm just given is identical to the earlier ones except that (1) addition is done in the field GF(2^{128}); and (2) the ith offset is $\gamma(i) \times L$, where γ is a particular (convenient) permutation on $\{1, 2, 3, \ldots, 2^n - 1\}$ and $j \times L$ is the number j, treated as a field element, multiplied (in this field) by L. Let us elaborate.

We have constructed the $L(i)$ values in such a manner that $L(i)$ is the string that represents $2^i \times L$, where 2^i and L are regarded as points in the field GF(2^{128}) and $\times$ refers to multiplication in the field. Here we are are representing points using the irreducible polynomial $p(x) = x^{128} + x^7 + x^2 + x + 1$. A string $a_{127} \cdots a_1 a_0$ corresponds to the formal polynomial $a_{127}x^{127} + \cdots + a_1 x + a_0$.

A *Gray code* on $\{0,1\}^n$ is a permutation of $\{0,1\}^n$, say $(g_0, g_1, \ldots, g_{2^n-1})$, such that g_i and g_{i+1} differ (in the Hamming sense) by just one bit. Also, g_0 and g_{2^n-1} differ in just one bit. We implicitly make use of the Gray code $\mathcal{G}(n)$ constructed as follows: $\mathcal{G}(1) = (0, 1)$, and, for $i \geq 0$, if $\mathcal{G}(i) = (g_0, \ldots, g_{2^i-1})$ then $\mathcal{G}_{i+1} = (0g_0, 0g_1, \ldots, 0g_{2^i-2}, 0g_{2^i-1}, 1g_{2^i-1}, 1g_{2^i-2}, \ldots, 1g_1, 1g_0)$. This is easily seen to be a Gray code, and it is not hard to prove that, in this code, $g_{i+1} = g_i \oplus 1\ll\mathsf{ntz}(i)$. Thus it is easy to compute the successive words of this code.

Moving from strings to numbers, the Gray code that we are using is $\gamma(1) = 1$, $\gamma(2) = 3$, $\gamma(3) = 2$, $\gamma(4) = 6$, $\gamma(5) = 7$, $\gamma(6) = 5$, $\gamma(7) = 4$, $\gamma(8) = 12$, and so forth. The ith offset has been defined as $iL = \gamma(i) \times L$.

Comments The Gray-code trick can also be used, all by itself, in Jutla's construction, where one wants to XOR different subsets of vectors $L(1), L(2), \ldots, L(t)$. In [Ju00] the $L(i)$-values would be obtained afresh with each message encrypted. What we have suggested is better in two ways. First, the $L(i)$-values are fixed—they don't have to be recomputed with each message. And second, they don't have to be computed by using the block cipher lots of different times: they can be computed by applying the block cipher *once*, and then doing some shifting and XORing to get successive values. The shifting and XORing is minimal; the key-setup cost would be much lower than invoking the block cipher for each $L(i)$.

One further trick was built into the definition of L. We defined L in a way that ensures that the top two bits of every 32-bit word are 0-bits. This means that one can change L to $2L$, or change L to $4L$, or change $4L$ to $2L$, and so forth, using either two or four shift operations (on a 64-bit machine or a 32-bit machine, respectively). This means that only one time in eight does one have to obtain a new $L(i)$ value by going to memory or doing bit twiddling; the rest of the time one shifts the current αL-value to get the $\alpha' L$ value that you want. The more zero-bits one sets aside at the beginning of each word the fewer times one has to go to memory or do bit-twiddling. But one quickly gets a diminishing return, and the security bound degrades with the number of forced zero-bits. So two or three 0-bits on the top of each word is probably a good choice.

Acknowledgments

Virgil Gligor described [GD00] to me at CRYPTO ’00, and **Charanjit Jutla** gave a rump-session talk on [Ju00] at the same conference. These events inspired this work.

Thanks to **Mihir Bellare** (UC San Diego) and **John Black** (University of Nevada, Reno) for extensive discussions. As indicated, Mihir and John are currently working with me on the proofs.

This work was carried out while the author was at Chiang Mai University (on leave of absence from the University of California, Davis).

References

[BDJR97] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption: Analysis of the DES modes of operation. *Proceedings of 38th Annual Symposium on Foundations of Computer Science* (FOCS 97), IEEE, 1997. Available from `http://www.cs.ucdavis.edu/~rogaway/`

[BN00] M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *Advances in Cryptology – ASIACRYPT ’00.* Lecture Notes in Computer Science, T. Okamoto., ed., Springer-Verlag, 2000. Available from `http://www-cse.ucsd.edu/users/mihir/`

[BR00] M. Bellare and P. Rogaway. Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient encryption.. *Advances in Cryptology – ASIACRYPT ’00.* Lecture Notes in Computer Science, T. Okamoto., ed., Springer-Verlag, 2000. Available from `http://www.cs.ucdavis.edu/~rogaway/`

[GD00] V. Gligor and P. Donescu. Fast encryption and authentication: XCBC encryptions and XECB authentication modes. Manuscript, August 18, 2000. Available from `http://www.eng.umd.edu/~gligor/`

[GM84] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, vol. 28, April 1984, pp. 270–299.

[Ju00] C. Jutla. Encryption modes with almost free message integrity. Manuscript, August 1, 2000. Available from `http://eprint.iacr.org/` (reference number 2000/039).

[KY00] J. Katz and M. Yung. Unforgeable encryption and adaptively secure modes of operation. *Fast Software Encryption ’00.* Lecture Notes in Computer Science, B. Schneier, ed., 2000.