

# Fast Encryption and Authentication: XCBC Encryption and XECB Authentication Modes

Virgil D. Gligor\*      Pompiliu Donescu

VDG Inc  
6009 Brookside Drive  
Chevy Chase, Maryland 20815  
{gligor, pompiliu}@eng.umd.edu

March 30, 2001  
April 20, 2001 (revision)

## Abstract

We present the eXtended Ciphertext Block Chaining (XCBC) schemes or modes of encryption that can detect encrypted-message forgeries with high probability even when used with typical non-cryptographic Manipulation Detection Code (MDC) functions (e.g., bitwise exclusive-or and cyclic redundancy code (CRC) functions). These modes detect encrypted-message forgeries at low cost in performance, power, and implementation, and preserve both message secrecy and integrity in a single pass over the message data. Their performance and security scale directly with those of the underlying block cipher function. We also present the XECB message authentication modes. These modes have all the operational properties of the XOR-MAC modes (e.g., fully parallel and pipelined operation, incremental updates, and out-of-order verification), and have better performance. They are intended for use either stand-alone or with encryption modes that have similar properties (e.g., counter-based XOR encryption). However, the XECB-MAC modes have higher upper bounds on the probability of adversary's success in producing a forgery than the XOR-MAC modes.

## 1 Introduction

*No one said this was an easy game !*  
Paul van Oorschot, March 1999.

A long-standing goal in the design of block encryption modes has been the ability to provide message-integrity protection with simple Manipulation Detection Code (MDC) functions, such as the exclusive-or, cyclic redundancy code (CRC), or even constant functions [5, 7, 9]. Most attempts to achieve this goal in the face of chosen-plaintext attacks focused on different variations of the Cipher Block Chaining (CBC) mode of encryption, which is the most common block-encryption mode in use. To date, most attempts, including one of our own, failed [8].

---

\*This work was performed while this author was on sabbatical leave from the University of Maryland, Department of Electrical and Computer Engineering, College Park, Maryland 20742.

In this paper, we define the eXtended Ciphertext Block Chaining (XCBC) modes that can be used with an exclusive-or function to provide the authentication of encrypted messages in a single pass over the data. These modes detect integrity violations at a low cost in performance, power, and implementation, and can be executed in a parallel or pipelined manner. They provide authentication of encrypted messages in real-time, without the need for an additional processing path over the input data. The performance and security of these modes scale directly with the performance and security of the underlying block cipher function since separate cryptographic primitives, such as hash functions, are unnecessary.

We also present the XECB modes for message authentication (i.e., XECB-MAC modes) and their salient properties. These message authentication modes have all the operational properties of the XOR message authentication (XOR-MAC) modes (e.g., they can operate in a fully parallel and pipelined manner, and support incremental updates and out-of-order verification [2]), and have better performance; i.e., they use only about half the number of block-cipher invocations required by the XOR-MAC modes. However, the XECB-MAC modes have higher bounds on the adversary’s success of producing a forgery than those of the XOR-MAC modes. The XECB modes are intended for use either stand-alone to protect the integrity of plaintext messages, or with encryption modes that have similar properties (e.g., counter-based XOR encryption [1] a.k.a “counter mode”) whenever it is desired that separate keys be used for secrecy and integrity modes.

## 2 Integrity Modes for Encryption

**Preliminaries and Notation.** In defining the encryption modes we adopt the approach of Bellare *et al.* (viz., [1]), who show that an encryption mode can be viewed as the triple  $(E, D, KG)$ , where  $E$  is the encryption function,  $D$  is the decryption function, and  $KG$  is the probabilistic key-generation algorithm. (Similarly, a message authentication (MAC) mode can be viewed as the triple  $(S, V, KG)$ , where  $S$  is the message signing function,  $V$  is the message verification function, and  $KG$  is the probabilistic key-generation algorithm.) Our encryption (and authentication) modes are implemented with block ciphers, which are modeled with finite families of pseudorandom functions (PRFs) or pseudorandom permutations (PRPs).

In this context, we use the concepts of pseudorandom functions (PRFs), pseudorandom permutations (PRPs), and super-pseudorandom permutations (SPRPs) ([1], [14]). Let  $R^{l,L}$  the set of all functions  $\{0, 1\}^l \rightarrow \{0, 1\}^L$ . We use  $F$  to denote either a family of pseudorandom functions or a family of super-pseudorandom permutations, as appropriate (e.g., for the encryption schemes,  $F$  will be a family of super-pseudorandom permutations, while for our MAC schemes,  $F$  can be a family of pseudorandom functions).

Given encryption scheme  $\Pi = (E, D, KG)$  that is implemented with SPRP  $F$ , we denote the use of the key  $K \stackrel{\mathcal{R}}{\leftarrow} KG$  in the encryption of a plaintext string  $x$  by  $E^{FK}(x)$ , and in the decryption of ciphertext string  $y$  by  $D^{FK}(y)$ . The most common method used to detect modifications of encrypted messages applies a MDC function  $g$  (e.g., a non-keyed hash, cyclic redundancy code (CRC), bitwise exclusive-or function [15]) to a plaintext message and concatenates the result with the plaintext before encryption with  $E^{FK}(x)$ . A message thus encrypted can be decrypted and accepted as valid only after the integrity check is passed; i.e., after decryption with  $D^{FK}(y)$ , the concatenated value of function  $g$  is removed from the plaintext, and the check passes only if this value matches that obtained by applying the MDC function to the remaining plaintext [5, 7, 15]. If the integrity check is not passed, a special failure indicator, denoted by *Null* herein, is returned. This method<sup>1</sup> has been used in commercial systems such as Kerberos V5 [17, 21] and DCE

---

<sup>1</sup>Note that other methods for protecting the integrity of encrypted messages exist; e.g., encrypting the message with a secret key and then taking the separately keyed MAC of the ciphertext [15, 3]. These methods require two passes over the

[6, 21], among others. The encryption scheme obtained by using this method is denoted by  $\Pi\text{-g} = (\text{E-g}, \text{D-g}, \text{KG})$ , where  $\Pi$  is said to be *composed* with MDC function  $g$ . In this mode, we denote the use of the key  $K$  in the encryption of a plaintext string  $x$  by  $(E^{FK}\text{-g})(x)$ , and in the decryption of ciphertext string  $y$  by  $(D^{FK}\text{-g})(y)$ .

A design goal for  $\Pi\text{-g} = (\text{E-g}, \text{D-g}, \text{KG})$  modes is to find the simplest encryption mode  $\Pi = (\text{E}, \text{D}, \text{KG})$  (e.g., comparable to the CBC modes) such that, when this mode is composed with a simple, non-cryptographic MDC function  $g$  (e.g., as simple as a bitwise exclusive-or function), message encryption is protected against *existential forgeries*. For any key  $K$ , a forgery is any ciphertext message that is not the output of  $E^{FK}\text{-g}$ . An existential forgery (EF) is a forgery that passes the integrity check of  $D^{FK}\text{-g}$  upon decryption; i.e., for forgery  $y'$ ,  $(D^{FK}\text{-g})(y') \neq \text{Null}$ , where *Null* is a failure indicator. Note that the plaintext outcome of an existential forgery need not be known to the forger. It is sufficient that the receiver of a forged ciphertext decrypt the forgery correctly.

**Message Integrity Attack: Existential Forgery in a Chosen-Plaintext Attack.** The attack is defined by a protocol between an adversary  $A$  and an oracle  $O^2$  as follows.

1.  $A$  and  $O$  select encryption mode  $\Pi\text{-g} = (\text{E-g}, \text{D-g}, \text{KG})$ , and  $O$  selects, uniformly at random, a key  $K$  of  $KG$ .
2.  $A$  sends encryption queries (i.e., plaintext messages to be encrypted)  $x^p$ ,  $p = 1, \dots, q_e$ , to the encryption function of  $O$ . Oracle  $O$  responds to  $A$  by returning  $y^p = (E^{FK}\text{-g})(x^p)$ ,  $p = 1, \dots, q_e$ , where  $x^p$  are  $A$ 's chosen plaintext messages.  $A$  records both its encryption queries and  $O$ 's responses to them.
3. After receiving  $O$ 's encryption responses,  $A$  forges a collection of ciphertexts  $y^i$ ,  $1 \leq i \leq q_v$  where  $y^i \neq y^p, \forall p = 1, \dots, q_e$ , and sends each decryption query  $y^i$  to the decryption function of  $O$ .  $O$  returns a success or failure indicator to  $A$ , depending on whether of  $(D^{FK}\text{-g})(y^i) \neq \text{Null}$ .

Adversary  $A$  is successful if at least one decryption query  $y^i$  such that  $(D^{FK}\text{-g})(y^i) \neq \text{Null}$  for  $1 \leq i \leq q_v$ ; i.e.,  $y^i$  is an existential forgery. The mode  $\Pi\text{-g} = (\text{E-g}, \text{D-g}, \text{KG})$  is said to be secure in a message-integrity attack if the probability of an existential forgery in a chosen-plaintext attack is negligible. (We use the notion of negligible probability in the same sense as that of Naor and Reingold [16].)

**Attack Parameters.**  $A$  is allowed  $q_e$  encryption queries (i.e., queries to  $E^{FK}\text{-g}$ ), and  $q_v$  decryption queries (i.e., queries to  $D^{FK}\text{-g}$ ) totaling  $\mu_e + \mu_v$  bits, and taking time  $t_e + t_v$ .

Parameters  $q_e, \mu_e, t_e$  are bound by the parameters  $(q', \mu', t', \epsilon')$  which define the chosen-plaintext security of  $\Pi = (\text{E}, \text{D}, \text{KG})$  in a secrecy attack (e.g., in the left-or-right sense [1], for instance), and a constant  $c'$  determined by the speed of the function  $g$ . Since parameters  $(q', \mu', t', \epsilon')$  are expressed in terms of the given parameters  $(t, q, \epsilon)$  of the SPRP family  $F$ , the attack parameters can be related directly to those of the SPRP family  $F$ .

Parameters  $q_e, \mu_e, t_e, q_v, \mu_v, t_v$  are also bound by the parameters  $(t, q, \epsilon)$  of the SPRP family  $F$ , namely  $\mu_e + \mu_v \leq ql$ , and  $t_e + t_v \leq t$ . (The parameters  $q_e, q_v$  are determined by  $\mu_e, \mu_v$ .) These parameters can be set to specific values determined by the desired probability of adversary's success. Note that  $q_v > 0$  since  $A$  must be allowed verification queries. Otherwise,  $A$  cannot test whether his forgeries are correct, since  $A$  does not know key  $K$ .

---

message data, require more power, and are more complex to implement than the modes we envision for most common use. Nevertheless these methods are useful whenever key separation is desired for secrecy and integrity.

<sup>2</sup> $O$  can be viewed as two oracles, the first for the encryption function of  $O$  and the second for the decryption function of  $O$ .

The message-integrity attack defined above is not weaker than an adaptive one in the sense that the success probability of adversary  $A$  bounds from above the success probability of another adversary  $A'$  that intersperses the  $q_e$  encryption and  $q_v$  verification queries; i.e., the adversary is allowed to make his choice of forgery after seeing the result of legitimate encryptions and other forgeries. (This has been shown for chosen-message attacks against MAC functions [2], but the same argument holds here.) To date, this is the strongest of the known goal-attack combinations against the integrity (authentication) of encrypted messages [3, 10].

### 3 Definition of the XCBC and XCBC-XOR Modes

We present three XCBC modes, namely (1) stateless, (2) stateful-sender, and (3) stateful modes, and some implementation options. In general, the fewer state variables the more robust the mode is in the face of failures (or disconnections) and intrusion. This might suggest that, in practice, stateless modes are preferable. However, this may not always be the case because a good, high-performance, source of randomness that can be used for each message may be unavailable or may be hard to protect in terms of confidentiality, integrity and availability. Further, the new random number used in each message encryption by the sender must be securely transmitted to the receiver, which usually costs at least an additional block-cipher invocation. The stateful-sender mode (e.g., a counter-based mode) eliminates the need for a good source of randomness but does not always eliminate the extra block-cipher invocation and the need to protect the extra sender state variables; e.g., the source of randomness is replaced by the enciphering of a message counter but the counter must be maintained and its integrity must be protected by the sender across multiple message authentications. (The other advantage of counter-based modes, namely the ability to go beyond the “birthday barrier” when used with pseudo-random functions, does not materialize in the context of the Advanced Encryption Standard (AES) since AES is modeled as a family of pseudo-random permutations.)

Maintaining secret shared-state variables, as opposed to just sender-state, helps eliminate the extra block-cipher invocations. Extending the shared keying state with extra, per-key, random variables shared by senders and receivers is a fairly straight-forward matter; e.g., these shared variables can be generated and distributed in the same way as the shared secret key, or can be generated using the shared key (at some marginal extra cost per message) by encrypting constants with the shared key. However, maintaining the shared state in the face of failures (or disconnections), and intrusion presents an extra challenge for the mode user; e.g., enlarging the shared state beyond that of a shared secret key may increase the exposure of the mode to physical attacks. The above discussion suggests that none of the three types of operational modes is superior to the others in all environments, and hence all of them should be supported in a general mode definition.

In the encryption modes presented below, the key generation algorithm,  $KG$ , outputs a random, uniformly distributed,  $k$ -bit string or key  $K$  for the underlying  $SPRP$  family  $F$ , thereby specifying  $f = F_K$  and  $f^{-1} = F_K^{-1}$  of  $l$ -bits to  $l$ -bits. If a separate second key is needed in a mode, then a new string or key  $K'$  is generated by  $KG$  identifying  $f' = F_{K'}$  and  $f'^{-1} = F_{K'}^{-1}$ . The plaintext message to be encrypted is partitioned into a sequence of  $l$ -bit blocks (padding is done first, if necessary),  $x = x_1 \cdots x_n$ . Throughout this paper,  $\oplus$  is the *exclusive-or* operator and  $+$  represents *modulo  $2^l$  addition*.

#### Stateless XCBC Mode (XCBC\$)

The encryption and decryption functions of the stateless mode,  $\mathcal{E}\text{-XCBC}\$^{F_K}(x)$  and  $\mathcal{D}\text{-XCBC}\$^{F_K}(y)$ , are defined as follows.

```

function  $\mathcal{E}\text{-XCBC}\$^f(x)$ 
 $r_0 \leftarrow \{0, 1\}^l$ 
 $y_0 = f(r_0); z_0 = f'(r_0)$ 
for  $i = 1, \dots, n$  do {
 $z_i = f(x_i \oplus z_{i-1})$ 
 $y_i = z_i + i \times r_0$  }
return  $y = y_0 || y_1 y_2 \dots y_n$ 

```

```

function  $\mathcal{D}\text{-XCBC}\$^f(y)$ 
Parse  $y$  as  $y_0 || y_1 \dots y_n$ 
 $r_0 = f^{-1}(y_0); z_0 = f'(r_0)$ 
for  $i = 1, \dots, n$  do {
 $z_i = y_i - i \times r_0$ 
 $x_i = f^{-1}(z_i) \oplus z_{i-1}$  }
return  $x = x_1 x_2 \dots x_n$ 

```

### Stateful-Sender XCBC Mode (XCBCS)

The encryption and decryption functions of the stateful-sender mode,  $\mathcal{E}\text{-XCBCS}\$^{F_K}(x, ctr)$  and  $\mathcal{D}\text{-XCBCS}\$^{F_K}(y)$ , are defined as follows.

```

function  $\mathcal{E}\text{-XCBCS}\$^f(x, ctr)$ 
 $r_0 = f(ctr); z_0 = f'(r_0)$ 
for  $i = 1, \dots, n$  do {
 $z_i = f(x_i \oplus z_{i-1})$ 
 $y_i = z_i + i \times r_0$  }
 $ctr' \leftarrow ctr + 1$ 
 $y = ctr || y_1 y_2 \dots y_n$ 
return  $y$ 

```

```

function  $\mathcal{D}\text{-XCBCS}\$^f(y)$ 
Parse  $y$  as  $ctr || y_1 \dots y_n$ 
 $r_0 = f(ctr); z_0 = f'(r_0)$ 
for  $i = 1, \dots, n$  do {
 $z_i = y_i - i \times r_0$ 
 $x_i = f^{-1}(z_i) \oplus z_{i-1}$  }
return  $x = x_1 x_2 \dots x_n$ 

```

Note that in the XCBCS mode the counter  $ctr$  can be initialized to a known constant such as  $-1$  by the sender.  $ctr'$  represents the updated  $ctr$  value. In both of the above modes the complexity is  $n + 2$  block-cipher invocations, where  $n$  is the length of input string  $x$  in blocks.

### Stateful XCBC Mode (XCBCS)

Let  $IV$  be a random and uniformly distributed variable that is part of the keying state shared by the sender and receiver.

$\mathcal{E}\text{-XCBCS}\$^{F_K}(x)$  and  $\mathcal{D}\text{-XCBCS}\$^{F_K}(y)$ , are defined as follows.

```

function  $\mathcal{E}\text{-XCBCS}\$^f(x)$ 
 $r_0 \leftarrow \{0, 1\}^l$ 
 $y_0 = f(r_0); z_0 = IV + r_0$ 
for  $i = 1, \dots, n$  do {
 $z_i = f(x_i \oplus z_{i-1})$ 
 $y_i = z_i + i \times r_0$  }
return  $y = y_0 || y_1 y_2 \dots y_n$ 

```

```

function  $\mathcal{D}\text{-XCBCS}\$^f(y)$ 
Parse  $y$  as  $y_0 || y_1 \dots y_n$ 
 $r_0 = f^{-1}(y_0); z_0 = IV + r_0$ 
for  $i = 1, \dots, n$  do {
 $z_i = y_i - i \times r_0$ 
 $x_i = f^{-1}(z_i) \oplus z_{i-1}$  }
return  $x = x_1 x_2 \dots x_n$ 

```

Note that in the XCBCS mode the shared  $IV$  value can be generated randomly by  $KG$  and distributed to the sender and receiver along with key  $K$  thereby saving one block cipher invocation, or can be generated using key  $K$  by standard key-separation techniques thereby requiring an additional block encryption operation per key. In the former case, the complexity of the mode is exactly  $n + 1$  block-cipher invocations and, in the latter, is *asymptotically*  $n + 1$  block-cipher invocations.

*Chaining Sequence.* The *block chaining sequence* is that used for the traditional CBC mode, namely  $z_i = f(x_i \oplus z_{i-1})$ , where  $z_0$  is the initialization vector,  $x_i$  is the plaintext and  $z_i$  is the ciphertext of

block  $i, i = 1, \dots, n$ . In contrast with the traditional CBC mode, the value of  $z_i$  is not revealed outside the encryption modes, and, for this reason,  $z_i$  is called a *hidden* ciphertext block. The actual ciphertext output,  $y_i$ , of the XCBC modes is defined using extra randomization, namely  $y_i = z_i + i \times r_0$ , where  $i \times r_0$  is the *modulo*  $2^l$  addition of the random, uniformly distributed, variable  $r_0$ ,  $i$  times to itself; i.e.,  $i \times r_0 \stackrel{\text{def}}{=} \underbrace{r_0 + \dots + r_0}_{i \text{ times}}$ .

Examples for why the randomization is necessary include those which show that, without randomization, the swapping of two  $z_i$  blocks of a ciphertext message, or the insertion of two arbitrary but identical blocks into two adjacent positions of a ciphertext message, would cause the decryption of the resulting forgery with probability one whenever an bitwise exclusive-or function is used as the MDC (which is what we intend to use, since these functions are among the fastest known). Correct randomization sequences, such as  $i \times r_0$ , ensure that, among other things, collisions between any two  $z_i$  values is negligible regardless of whether these values are obtained during message encryption, forgery decryption, or both. Note that this probability is negligible even though the randomization sequence  $i \times r_0$  allows low-order bits of some  $z_i$ 's to become known. (A detailed account of why such collisions contribute to an adversary's success in breaking message integrity is provided in the proof of the XCBC\$ mode; viz., Appendix A.) Examples of incorrect randomization sequences can be readily found; e.g., the sequence whereby each element  $i$  is computed as an bitwise exclusive-or of  $i$  instances of  $r_0$ .

*Initialization.* In *stateless* implementations of the XCBC modes  $r_0 \leftarrow \{0, 1\}^l$ ; i.e.,  $r_0$  is initialized to a random, uniformly distributed,  $l$ -bit value for every message. The value of  $r_0$  is sent by the sender to the receiver as  $y_0 = f(r_0)$ . In contrast, in *stateful-sender* implementations, which avoid the use of a random number generator, a counter,  $ctr$ , is initialized to a new  $l$ -bit constant (e.g., -1) for every key  $K$ , and incremented on every message encryption. In *stateful* implementations, a random initialization-vector value  $IV$  that is shared by the sender and receiver is generated for every key  $K$ , and used to create a per-message random initialization vector  $z_0$ .

In all XCBC modes, the initialization vector  $z_0$  is independent of  $r_0$ . While non-independent  $z_0$  and  $r_0$  values might yield secure initialization, simple relationships between these values can lead to the discovery of  $r_0$  with non-negligible probability, and integrity can be easily broken.<sup>3</sup> Since we use  $z_0$  in the definition of function  $g(x)$  (see below),  $z_0$  should also be unpredictable so that  $g(x)$  has a per-message unpredictable value.

The choice of encrypting  $r_0$  with a second key  $K'$  to obtain  $z_0$  (i.e.,  $z_0 = f'(r_0)$ ) is made exclusively to simplify the both the secrecy [1] and the integrity proofs; e.g., such a  $z_0$  is independent of  $r_0$  and is unpredictable. To eliminate the use of the second key and still satisfy the requirements for  $z_0$  suggested above, we can compute  $z_0 = f(r_0 + 1)$  in stateless and stateful sender implementations, whereas in stateful implementations we compute  $z_0 = IV + r_0$ , where the per-message  $r_0$  can be generated as a random value, or as an encryption of  $ctr$  in the XORC mode. This eliminates the additional block-cipher invocations necessary in the stateless and stateful-sender modes at the cost of maintaining an extra shared state variable (IV). This choice still satisfies the requirements for  $z_0$ .

*Generalization.* The above method for protecting message integrity against existential forgeries in chosen-plaintext attacks can be generalized as follows. Let the output ciphertext  $y_i$  be computed as  $y_i = z_i \text{ op } E_i$ , where *op* is the randomization operation,  $E_i$  are the elements of the randomization sequence, and  $z_i$  the hidden ciphertext. The encryption mode  $\Pi$  (1) must be secure in adaptive chosen-plaintext attacks with respect to secrecy, and (2) must use the input plaintext blocks  $x_i$  to generate the input to  $f$ . The PCBC

---

<sup>3</sup>As a simple example illustrating why this is the case, let  $z_0 = r_0 + 1$ , choose  $x_1$  such that  $z_0 \oplus x_1 = r_0$  with non-negligible probability, and then compute  $y_1 - y_0 = r_0$ . With a known  $r_0$ , one can cause collisions in the values of  $z_i$  and break integrity.

[12, 15], and the “infinite garble extension” [5] modes are suitable, but counter-mode/XORC and XOR\$ are not (since they fail condition (2)). Operation *op* must be invertible, so  $\oplus$ , modular  $2^l$  addition and subtraction are appropriate. Elements  $E_i$  must be unpredictable such that collisions among  $z_i$ ’s (discussed above) could only occur with negligible probability. Other sequences can be used. For example  $E_i = a^i \times r_0$  can be used, where  $E_i$  is a linear congruence sequence with multiplier  $a$ , where  $a$  can be chosen so that the sequence passes spectral tests to whatever degree of accuracy is deemed necessary. (Examples of good multipliers are readily available in the literature [11].)

**XCBC-XOR Modes.** To illustrate the properties of the XCBC modes in integrity attacks, we choose  $g(x) = z_0 \oplus x_1 \oplus \dots \oplus x_n$  for plaintext  $x = x_1 \dots x_n$ , where  $z_0$  is defined as the initialization vector of the mode. In this example, block  $g(x)$  is appended to the *end* of a  $n$ -block message plaintext  $x$ , and hence block  $x_{n+1} = z_0 \oplus x_1 \oplus \dots \oplus x_n$ . For this choice of  $g(x)$ , the integrity check performed at decryption becomes  $z_0 \oplus x_1 \oplus \dots \oplus x_n = f^{-1}(z_{n+1}) \oplus z_n$ , where  $z_{n+1} = y_{n+1} - (n+1) \times r_0$ , and  $z_n = y_n - n \times r_0$ .

*Message Padding.* Standard padding methods (e.g., ASN.1), which typically require that a bit pattern and its length be added to the last block of a message to obtain an integer number of (padded) plaintext blocks, have the undesirable consequence that an additional block cipher invocation is required for the extra block of padding added for plaintexts of an integer number of blocks. Alternatives that avoid standard padding are known [4], but they require use of an extra (shared secret) key – a somewhat less desirable alternative when maintaining the unpredictability of the redundant padding information added by a mode is not an operational goal.

*Known Padding Pattern.* The goal of the first padding option for the XCBC modes is two-fold: (1) avoid extra block-cipher invocations, and (2) avoid the use of extra keying material. Padding with a known pattern is performed as follows: (1) use a pattern that always starts with a “1” bit followed by the minimum number of “0” bits necessary to fill the last block of plaintext [4]; (2) if the last block of a message is unpadded, use block  $g'(x) = \overline{z_0} \oplus x_1 \oplus \dots \oplus x_n$  as the  $x_{n+1}$  plaintext block, where  $\overline{z_0}$  is the bitwise complement of  $z_0$ ; otherwise, use  $g(x) = z_0 \oplus x_1 \oplus \dots \oplus x_n$ .

At decryption, the integrity check performs the exclusive-or of  $f^{-1}(z_{n+1}) \oplus z_n$  with  $x'_1 \oplus \dots \oplus x'_n$ , where  $x'_1, \dots, x'_n$  are the plaintext blocks obtained at decryption, and then compares the result with the  $z_0$  computed during decryption; if this check fails, the result is compared with  $\overline{z_0}$ , the complement of  $z_0$  computed at decryption, and only if this second comparison for equality fails the ciphertext-message decryption returns failure. If the the comparison check with  $z_0$  passes, meaning that the message was padded at encryption, the padding pattern is checked, extracted (providing some extra confidence, if found) and removed. It follows that the decryption of unpadded (but unforged) messages would fail first the first equality check but not the second. Of course, the extra check would be required only for unpadded messages and forgeries. This padding scheme satisfies our goals at a modest cost; i.e., that of including padding bits in the ciphertext and an extra check for equality.

*Unpredictable Padding Pattern.* The goal of the second padding option for the XCBC modes, in addition to (1) above, is to retain the unpredictability of the redundant information added by these modes to user input. This goal is set for pragmatic reasons, since these modes are secure with respect to chosen-plaintext attacks. It stems from the long-standing belief that a mode of encryption should avoid adding redundant information that provides an adversary additional conditions to verify the success of his attacks (e.g., key guessing) beyond those already available to him from knowledge of user input; e.g., in a ciphertext-only attack, the adversary who knows nothing about the plaintext would benefit from added predictable redundancy by padding and integrity checks.

In the XCBC modes, padding with an unpredictable pattern is performed as follows. Let Mask be a

random and uniformly distributed block that is part of the keying state shared by the sender and receiver. The Mask can be generated and distributed along with the key or it can be generated by any of the available standard methods (e.g., encrypt a constant with the shared secret key to initialize Mask). For each plaintext input whose last block is incomplete, fill the last block with the known bit pattern used in the Known-Padding-Pattern option above (i.e., the pattern that always starts with a “1” bit followed by the minimum number of “0” bits necessary to fill the last block of plaintext). Perform the bitwise exclusive-or operation between the Mask and the filled last plaintext block. Use the result as the plaintext block  $x_n$  in the computation of the  $x_{n+1} = g(x)$  block. Use  $z_0$  to compute  $g(x)$  for padded messages and  $\overline{z_0}$  for unpadded ones as in the Known-Padding-Pattern option above. At decryption, use the same integrity check as that used in the Known-Padding-Pattern option (defined above), and if the check for padded messages passes, perform the bitwise exclusive-or of the Mask and the recovered block  $x'_n$ , and check and extract the known padding bit pattern from  $x'_n$  before returning the plaintext to the user.

The stateless and stateful encryption modes  $\Pi$ -g obtained by the use of schemes  $\Pi = \text{XCBC\$}$ ,  $\Pi = \text{XCBC\$\$}$ , or  $\Pi = \text{XCBCS}$  with function  $g(x) = z_0 \oplus x_1 \oplus \dots \oplus x_n$  are denoted by  $\text{XCBC\$-XOR}$ ,  $\text{XCBC\$\$-XOR}$ , and  $\text{XCBCS-XOR}$  respectively.

### **Examples of Other Encryption Modes that Preserve Message Integrity.**

Recently, C.S. Jutla [13] proposed an interesting scheme in which the output blocks  $z_i$  of CBC encryption are modified by (i.e., bitwise exclusive-or operations) with a sequence  $E_i$  of pairwise independent elements. In this model,  $E_i = (i \times IV_1 + IV_2) \bmod p$ , where  $IV_1, IV_2$  are random values generated from an initial random value  $r$ , and  $p$  is prime, and the complexity is  $n + 3$ , where  $n$  is the length of the plaintext input in blocks. In contrast with C.S. Jutla’s scheme, the elements of the XCBC sequence,  $E_i = (i \times r_0) \bmod 2^l$ , are not pairwise independent, and the complexity is  $n + 2$  for the stateless and stateful-sender cases, and  $n + 1$  for the stateful case. Also, the performance of the required modular  $2^l$  additions is slightly better than that of  $\bmod p$  additions, where  $p$  is prime. However, the pairwise independence of C.S. Jutla’s  $E_i$  sequence should yield a slightly tighter bound on the probability of successful forgery illustrating, yet again, a fundamental tradeoff between performance and security. (The bound is tighter by a fraction of a  $\log_2$  factor depending on the value of  $p$ , which would mean that the attack complexity is within the same order of magnitude of the XCBC bound – viz., Section 5).

More recently, P. Rogaway [19] has proposed other schemes that use interesting variations of non-independent and pairwise-independent elements for the