On the security of blockwise secure modes of operation beyond the birthday bound

Antoine Joux

Abstract—In 2002, in two independent papers, Bellare, Kohno and Namprempre and Joux, Martinet and Valette introduced the notion of blockwise security for modes of operations. This notion stems from common practice, since in all applications, modes of operation for block ciphers do not process messages as atomic entities but in an incremental manner, block after block. Soon afterward, several papers showed that many modes of operation are already blockwise secure and that others can be made secure by simple modifications. In this paper, we revisit these results, by comparing possible attacks on modes of operation after the birthday bound is reached. Amusingly, in spite having essentially identical security proofs up to this bound, modes of operation in the blockwise model behave very differently than their counterparts in the regular model, once the birthday paradox bound is crossed.

Index Terms—Modes of operation, Blockwise security, Cycle finding algorithms

I. INTRODUCTION

Blockwise security was introduced in 2002 in two independent papers. In [4], Bellare, Kohno and Namprempre showed that the practical implementation of secure shell (SSH) which relies on CBC (Cipher Block Chaining) was insecure against a variant of chosen plaintext attacks. This should not have been possible, since CBC is provably secure against chosen plaintext attacks. Independently, at Crypto 2002, Joux, Martinet and Valette [12] introduced the theoretical notion of blockwise security and showed that a gap existed between regular and blockwise security on several modes of operation such as CBC, GEM (introduced in [7]) and IACBC (see [13]). After that, a series of articles studied several aspects of blockwise security, in order to fill this practical gap.

In 2003, Fouque et al. gave in [8] formal definitions for blockwise adaptive chosen plaintext and chosen ciphertext security. They proposed a generic technique called Decrypt-then-Mask to achieve blockwise-adaptive chosen-ciphertext security with a storage limited encryption/decryption device connected to a general purpose computer. The main drawback of this scheme is that it cannot process messages on-line and thus cannot be used in some practical applications which require this feature, such as video broadcast. However, this is an intrinsic limitation of blockwise chosen ciphertext security. This limitation was further investigated by Boldyreva and Taeombok in [6], who proposed a relaxed security notion in order to allow for online schemes. As an application, they studied the security of the mode of operation HPCBC introduced in [2]. Note that in the latest version of [2], HPCBC is called HCBC2.

A theoretical treatment describing the relations between the various security models is given in [9]. This paper compares a wide variety of combination, including blockwise versus regular security, sequential versus concurrent access in several possible settings: left-or-right, real-or-random and find-then-guess security. A systematic analysis of modes of operation and their sequential blockwise security is given in [1].

Thanks to the research effort that has been dedicated to blockwise security, it is now possible to use secure modes of operation in the blockwise model in many applications including applications that require on-line processing of data. Moreover, security proofs are available for these modes of operation and the security bounds are essentially the same as the bounds of their counterparts in the regular model. As a consequence, it is widely believed that modes of operation are as secure in the blockwise model as they are in the usual model. In this paper, we revisit this belief from the point of view of the available attacks. What happens to secure schemes once we cross the threshold of the security bound? This question is important, especially when reaching the security bound is costly but not infeasible, for example, when using a 64-bit block cipher. This may occur either for legacy reasons, usually using triple-DES or when using an hardware oriented lightweight cipher, such as the block cipher from [5]. When the bound is completely unreachable, using AES for example, the question clearly becomes less important. However, the blockwise security model mostly comes from practical concerns and, in practice, block cipher with a 64-bit blocksize are still widely used.

The paper is organized as follows: in Section III we describe a few common modes of operation which are blockwise secure and recall their security properties both under the blockwise and regular security models. In Section IV, we go beyond the security bound and present the classical birthday based attacks against these modes of operation, discussing what a cryptanalyst can do using these attacks. In Section V, we present some new attacks which are applicable to these modes of operation in the blockwise security model but not applicable under the regular security model. These new attacks are less expensive and offer more leverage to a cryptanalyst than the classical birthday based attacks.

II. NOTATIONS

In this section, we briefly recall how the security of modes of operations, both in the regular and in the blockwise model are usually defined. For more details, the reader can, for example, refer to [9]. The main difference between the two models is that in the regular model, all messages are treated...
as a whole, while in the blockwise model, encryption is done block by block to avoid storing possibly long messages in the encrypting device.

All these security notions involve a game between a challenger \( C \) and an adversary \( A \). Depending on the detailed definition of this game, we obtain several flavors of the security notion. Depending on the behavior of the challenger, we may consider find-and-guess security, real-or-random security or left-or-right security. Similarly, depending on the power of the adversary, we get ciphertext only security, known plaintext security, chosen plaintext security or chosen ciphertext security. Here, for simplicity of exposition, we focus on the definitions of chosen plaintext real-or-random security and chosen plaintext left-or-right security.

At the start of the real-or-random game, the challenger \( C \) flips a random coin and obtains a bit \( b \). If \( b = 0 \), the challenger decides to play real. If \( b = 1 \), he decides to play random. Then, the challenger choose at random all the keys needed for the modes of operation. Once, the game has been initialized, in the regular security model, the adversary creates messages of his choice and sends them to the challenger. If the challenger plays real, he returns the ciphertext obtained by encrypting the message with the mode of operation under the chosen keys. If he plays random, he instead return the cyphertext obtained by encrypting a purely random message of the same length. In the blockwise model, the adversary has the additional power to send the blocks of plaintext one at a time and to receive the corresponding encrypted blocks one by one. For bookkeeping purpose, he also can send two special symbols standing for \textit{start a new message} and \textit{end of message}. In this blockwise model, the adversary may have the option of starting several encryptions at the same time, however, we only use this option once, in Section V-C.

After this interaction phase, the adversary stops the game and outputs a bit \( b' \). He wins when \( b' = b \), i.e. if he can distinguish between a challenger playing real and a challenger playing random. Assuming, without loss of generality\(^1\) that the probability \( P_A \) of success is \( \geq 1/2 \), we define the advantage of the adversary \( A \) as:

\[
\text{Adv}_A = 2P_A - 1.
\]

With this definition, the trivial adversary that just output a random bit \( b' \) has an advantage 0 and an adversary which always succeeds has an advantage 1.

This notion of advantage is also defined for classes of adversaries. For any class \( C \), the advantage \( \text{Adv}_C \) is the maximum of the advantages of all adversaries in this class. Very often, one defines the classes of adversaries by giving the amount of available resources: computational power, number of queries, length of queries, \ldots

The left-or-right game is defined very similarly. This time, the random \( b \) chosen by the challenger is interpreted as \textit{left} for 0 and \textit{right} for 1. In each interaction the adversary sends two messages of the same length (or two blocks in the blockwise model) and receives the encryption of either the left one or the right one, depending on \( b \). Once again the goal is to guess \( b \) and there is a corresponding advantage. In fact, the advantages in the real-or-random and left-or-right games are related, see [3] for the regular model and [9] for the blockwise model.

III. **Blockwise Modes of Operation and Their Security**

A. **CBC encryption**

Since cipher block chaining is probably the most frequently used mode of operation, we start by addressing this mode. We first briefly recall the process of CBC encryption. For each message, a initial value \( C_0 \) is first chosen. Then each plaintext block \( P_i \) is encrypted into:

\[
C_i = E(C_{i-1} \oplus P_i),
\]

It is well known, e.g. see [12], that CBC itself is insecure in the blockwise model. The reason is that if the adversary knows the ciphertext block \( C_{i-1} \) before submitting \( P_i \), he can directly control the input of the block cipher \( C_{i-1} \oplus P_i \). As a consequence, he can submit twice the same value to the block cipher, see the corresponding equality reflected in the encrypted values and thus derive a distinguishing attack.

For this reason, it has been proposed for blockwise implementation to use delayed CBC encryption, where \( C_i \) is not revealed before \( P_{i+1} \) has been submitted. It was shown in [10], that delayed CBC encryption is secure in the blockwise model.

To be more precise, let us first state a theorem on the security of CBC in the ordinary model adapted from [3]. The adaptation we make here consists of merging two different parameters in the theorem (number of encrypted messages and total length of these messages) into a single parameter. This simplifies the statement of the theorem, without fundamentally changing its meaning.

**Theorem 1:** Suppose \( E \) is a pseudo-random permutation family on \( n \)-bit blocks. Then for any runtime \( t \) and any bound \( q \) on the total length of processed messages (expressed in number of blocks), the advantage of any left-or-right chosen plaintext distinguisher against CBC encryption using \( E \) is bounded from above by:

\[
2 \cdot \text{Adv}_E(t, q) + \frac{3 \cdot q^2}{2^{n+1}},
\]

where \( \text{Adv}_E(t, q) \) is the advantage of the best adversary able to distinguish \( E \) from a random permutation in the class of adversaries querying at most \( q \) blocks and running in time at most \( t \).

In the blockwise model, a similar theorem is given in [10].

**Theorem 2:** Suppose \( E \) is a pseudo-random permutation family on \( n \)-bit blocks. Then for any runtime \( t \) and any bound \( q \) on the total length of processed messages (expressed in number of blocks), the advantage of any left-or-right blockwise chosen plaintext distinguisher against delayed CBC encryption using \( E \) is bounded from above by:

\[
2 \cdot \text{Adv}_E(t, q) + \frac{q^2}{2^{n-1}},
\]

where \( \text{Adv}_E(t, q) \) is the advantage of the best adversary able to distinguish \( E \) from a random permutation.
Since the two security theorems only differ by the value of the constant before $q^2/2^n$, the usual conclusion is that delayed CBC in the blockwise is essentially as secure as CBC in the ordinary model. In Sections IV and V we revisit this conclusion from the point of view of possible attacks, by comparing what happens beyond the security bound in both models.

a) Note: CFB encryption: In several places in this paper, we mention that attacks on CBC generalize to Cipher Feedback (CFB) mode. For the sake of completeness, we briefly recall how messages are encrypted in CFB. As in CBC, we first choose an initial value $C_0$, then $P_i$ is encrypted into:

$$C_i = E((C_{i-1}) \oplus P_i).$$

CFB is known to be secure both in the ordinary and blockwise models, without requiring any additional delay. The security theorems are essentially the same as in the CBC case.

B. XCBC encryption

XCBC is an authenticated encryption mode, similar to CBC encryption, where authenticity is guaranteed by adding a simple redundancy before encrypting. In order to achieve this difficult goal, the inner core of CBC encryption present in XCBC is hidden by masking the ciphertext blocks. More precisely, denoting plaintext blocks by $P_i$, inner blocks by $Z_i$ and ciphertext blocks by $C_i$, XCBC encryption proceeds as follows. First, choose a random value $r_0$, then let the initial block of ciphertext be $C_0 = E(r_0)$ and let $Z_0 = E'(r_0)$, where $E'$ is a copy of blockcipher $E$ with a different key. After that encrypt block $P_i$ using:

$$Z_i = E(P_i \oplus Z_{i-1}) \text{ and } C_i = Z_i + i \times r_0.$$ 

In the computation of $C_i$ the addition and multiplication operations are done modulo $2^n$ (for $n$-bit blocks).

XCBC is known to be secure in the ordinary model. The security theorem given for XCBC in [11] is essentially the same as the theorem for CBC given in Section III-A. As far as we know, its blockwise security status is still unknown (see [1]).

C. HCBC encryption

HCBC is a variation of CBC which is intended to remain secure without using random initial values. It has been defined in [2] as an example of an online cipher. HCBC exists in two flavors, the first flavor HCBC1 is intended to resist chosen plaintext attacks, the second flavor HCBC2 is intended to resist chosen ciphertext attacks. By definition of an online cipher, the security requirements of HCBC are different from the usual security requirements. The main modification is to say that the fact that encrypting two messages with identical prefixes yields ciphertexts with identical prefixes should no longer be considered as an attack. Indeed, this fact is intrinsic to the definition of online ciphers and cannot be avoided. Thus, we need to restrict the set of allowed adversaries to prevent them from using this single specific property.
To compensate for the lack of initial randomness, HCBC uses an extra function in addition to the block cipher \( E \), this function \( H \) takes one block of input for HCBC1 and two blocks for HCBC2, in both cases it outputs a single block. The function \( H \) is keyed (with a different key than \( E \)); it needs to be an almost-XOR universal hash function (see [2] for details). Two alternatives are suggested in [2] for the function \( H \), either use a blockcipher based construction or a specific fast construction for universal hashing. To be more precise, with a blockcipher construct we use:

\[
\begin{align*}
H(x) &= E'(x) \quad \text{with HCBC1 and} \\
H(x, y) &= E'(y \oplus E'(x)) \quad \text{with HCBC2,}
\end{align*}
\]

where \( E' \) is a copy of \( E \) with a different key. For fast universal hashing, one option is to use polynomial evaluation in \( \mathbb{F}_{2^n} \), assuming that \( hk \) denotes the key of \( H \), we can use:

\[
\begin{align*}
H(x) &= x \times hk \quad \text{for HCBC1 and} \\
H(x, y) &= x \times hk + y \times hk^2 \quad \text{for HCBC2.}
\end{align*}
\]

Once \( H \) is defined, HCBC encryption proceeds as follows. First define \( C_0 = 0^n \) and remark that since \( C_0 \) is a constant it does not need to be transmitted. To simplify notations, also define \( P_0 = 0^n \). Then to encrypt \( P_i \) use the formulas:

\[
\begin{align*}
C_i &= E(P_i \oplus H(C_{i-1})) \\
C'_i &= E(P_i \oplus H(P_{i-1}, C_{i-1})) \oplus H(P_{i-1}, C_{i-1}),
\end{align*}
\]

respectively for HCBC1 and HCBC2.

Both flavors of HCBC are secure against blockwise adaptive chosen plaintext attacks up to the birthday paradox bound. In addition, HCBC2 offers security against chosen ciphertext attacks.

IV. OVERVIEW OF SOME CLASSICAL ATTACKS IN THE REGULAR MODEL

In this section and the next, we discuss a variety of attacks against modes of operation in the regular model. We mostly consider distinguishers, either real-or-random and left-or-right chosen plaintext distinguishers.

For some modes of operation, we also describe stronger attacks, where the adversary can learn much more than a single bit of information. However, even when such stronger attacks exist, distinguishers are still interesting because they usually pinpoint the underlying weakness in a simple and precise manner.

A. CBC

Beyond the security bound of CBC encryption, the birthday paradox comes into play. As a consequence, among the ciphertext blocks, we expect to find at least one collision, say \( C_i = C_j \). Replacing each block of ciphertext by its expression, we find:

\[
\begin{align*}
E(C_{i-1} \oplus P_i) &= E(C_{j-1} \oplus P_j), \quad \text{and thus} \\
C_{i-1} \oplus P_i &= C_{j-1} \oplus P_j,
\end{align*}
\]

since \( E \) is a permutation. Rewriting this as \( P_i \oplus P_j = C_{i-1} \oplus C_{j-1} \), we learn the XOR of two plaintext blocks. As the length of the message grows, more and more collisions are found and thus more and more information is learned about the message.

For a cryptanalyst to use this attack, he must be able to efficiently detect collisions among the blocks of ciphertext. One very common technique is to sort the ciphertext blocks in order to detect these collisions. However, this requires either a large amount of main memory or a fast auxiliary storage device. For 64-bit block ciphers, the cryptanalyst needs to store and sort about \( 2^{32} \) blocks or equivalently \( 2^{35} \) bytes\(^2\) of main memory or fast storage. Despite the quick progress of computer hardware, 32 Gbytes is still a large amount of memory, only available on dedicated computers.

b) Note on CFB mode.: The above attack on CBC applies almost directly to CFB. The only difference is a small shift in the indices of plaintext and ciphertext involved in the attack. More precisely, if we detect a collision between \( C_i \) and \( C_j \), then:

\[
\begin{align*}
P_{i+1} \oplus P_{j+1} &= (C_{i+1} \oplus E(C_i)) \oplus (C_{j+1} \oplus E(C_j)) \\
&= (C_{i+1} \oplus C_{j+1}) \oplus (E(C_i) \oplus E(C_j)) \\
&= (C_{i+1} \oplus C_{j+1}).
\end{align*}
\]

Thus, instead of learning the XOR of the plaintext blocks located at the places corresponding to the collision, we learn the XOR of the next plaintext blocks.

B. XCBC

XCBC is a mode that offers authenticated encryption. Here, we consider the security of XCBC as an encryption scheme, not as a MAC. We present some straightforward birthday based attacks against XCBC.

c) Collision of randomness.: The easy distinguisher with XCBC probably is to encrypt the same short (one block) message over and over. If the same random choice it made twice, the resulting ciphertexts are clearly identical. This leads to a straightforward real or random distinguisher. To turn this into a left or right distinguisher, it suffices to feed identical one block messages on the left side and random one block messages on the right side. When a collision on the initial value is detected by looking at the first ciphertext blocks, if the next ciphertext blocks do not collide we are necessarily on the right side, otherwise, with high probability we are on the left side.

d) Inner Collisions.: The next approach to attacking XCBC is to use inner collisions. Clearly, if an internal collision between the block cipher outputs \( Z_i \) and \( Z_j \) occur within a single message, this collision, if detected, leaks a lot of information about the initial randomness \( r_0 \) because in that case \( C_j - C_i = (j - i) \times r_0 \). This information allows us to recover most bits of \( r_0 \), depending on the largest power of two dividing \( j - i \).

However, to use this fact, we need to detect the collision. Within a single message, this can be done by choosing plaintext blocks alternating between random blocks in odd positions.
and zero blocks in even positions. With this choice, any inner collision on two odd positions \( Z_{2i+1} = Z_{2j+1} \) implies an inner collision on the next inner blocks, i.e. \( Z_{2i+2} = Z_{2j+2} \). This can be used to detect the collision by remarking that:

\[
C_{2i+2} - C_{2i+1} = Z_{2i+2} + (2i + 2) \times r_0 \\
-Z_{2i+1} - (2i + 1) \times r_0 \\
= Z_{2i+2} - Z_{2i+1} + r_0 \\
= Z_{2j+2} - Z_{2j+1} + r_0 \\
= C_{2j+2} - C_{2j+1}.
\]

Once the value \( r_0 \) has been computed, the security of XCBC becomes identical to the security of CBC. Indeed, using \( r_0 \) we can remove the output masks and learn the sequence \( Z \) which is a CBC encryption of \( P \).

\section{HCBC}

The attacks that can be performed against HCBC greatly depend on the almost-XOR universal hash function used in this mode. In [2], Bellare \textit{et al.} suggest two different approaches and propose to use either a fast AXU hash or another instantiation of the block cipher. For the cryptanalyst, the main difference between these two proposals is that for most fast AXU hash function, including the function based on polynomial evaluation described in Section III-C, knowledge of a collision usually leaks the key of the hash function. On the contrary, with a block cipher construction, this does not happen.

1) HCBC using a fast almost-XOR universal hash: In order to fully describe the possible attacks beyond the birthday bound with a fast AXU hash function, we consider the case of a hash function based on polynomial evaluation, as defined in Section III-C.

In order to attack this instantiation of HCBC1, we ask for encryption of a long random message, and look for a collision \( C_i = C_j \). Replacing both ciphertexts by their expression, we find:

\[
E(P_i \oplus H(C_{i-1})) = E(P_j \oplus H(C_{j-1})) \text{ thus}
\]

\[
P_i \oplus H(C_{i-1}) = P_j \oplus H(C_{j-1}) \text{ then}
\]

\[
H(C_{i-1}) \oplus H(C_{j-1}) = P_i \oplus P_j \text{ and finally}
\]

\[
hk \times (C_{i-1} \oplus C_{j-1}) = P_i \oplus P_j.
\]

Finally, dividing by \( C_{i-1} \oplus C_{j-1} \), we recover the value of the hash function secret key \( hk \). Once this hash secret key is known, the security of HCBC becomes equivalent to the security of CBC itself. Any ciphertext collision now leaks the XOR value of two plaintext blocks.

With HCBC2, we proceed in a similar but slightly different manner. Instead of looking for collisions between ciphertext blocks, we first compute the XOR of the plaintext blocks with their corresponding ciphertext blocks and search for collisions within the resulting sequence of blocks. From \( P_i \oplus C_i = P_j \oplus C_j \), we deduce:

\[
P_i \oplus E(P_i \oplus H(P_{i-1}, C_{i-1})) \oplus H(P_{i-1}, C_{i-1}) = P_j \oplus E(P_j \oplus H(P_{j-1}, C_{j-1})) \oplus H(P_{j-1}, C_{j-1}).
\]

Thus:

\[
D_i \oplus E(D_i) = D_j \oplus E(D_j) \quad \text{where}
\]

\[
D_i = P_i \oplus H(P_{i-1}, C_{i-1}) \quad \text{and}
\]

\[
D_j = P_j \oplus H(P_{j-1}, C_{j-1}).
\]

The collision may occur for two different reasons, either we already have \( D_i = D_j \) or the collision appears when evaluating \( D_i \oplus E(D_i) \). Both cases have roughly the same probability. Thus, after collecting a small number of collision, we may assume that \( D_i = D_j \). This yields a degree two equation on \( hk \) and thus gives candidate values for this secret key. With a few additional collisions, we can learn \( hk \) exactly. Once again, given the secret key of the hash function, the security of HPCBC is reduced to the security of plain CBC.

2) HCBC using a blockcipher based hash: When the hash is based on a blockcipher construction, collisions are much harder to exploit. For HCBC1, we find as before an equation:

\[
H(C_{i-1}) \oplus H(C_{j-1}) = P_i \oplus P_j.
\]

However, recovering the key of \( H \) from this equation requires an exhaustive key search. We can also use the encryption block after the collision to obtain a condition allowing for exhaustive key search on the encryption key of \( E \). Indeed, if \( C_i = C_j \) then

\[
E^{-1}(C_{i+1}) \oplus E^{-1}(C_{j+1}) = P_{i+1} \oplus P_{j+1}.
\]

Once either of the keys is found, each new collision leaks the XOR of two plaintext blocks. As a consequence, with birthday based attacks, HCBC1 is no more resistant than the weakest of \( H \) and \( E \).

When considering HCBC2, we can perform the same attack with respect to \( H \), with a slightly different condition:

\[
H(P_{i-1}, C_{i-1}) \oplus H(P_{j-1}, C_{j-1}) = P_i \oplus P_j.
\]

This equation also suffices to obtain the key of \( H \) through exhaustive search. However, we know of no attack to directly search for the key of \( E \), without first obtaining the key of \( H \).

Since exhaustive key search on either \( E \) or \( H \) is expected to be expensive, it is interesting to also consider attacks that do not require any key search. For example, we can construct a simple distinguisher by forming a plaintext with random blocks in odd position and zero blocks in even positions. With such a plaintext, any collision between two ciphertext blocks in odd positions implies a collision on the next two ciphertext blocks (in even positions). This fact can be used to devise either a real-or-random or a left-or-right distinguisher. This distinguisher works for both version of HCBC.

\section{Attacks in the blockwise model}

\subsection{CBC}

In order to construct an attack on blockwise, delayed CBC, we are going to use the same trick as for attacking non-delayed blockwise CBC. More precisely, whenever we receive a block of ciphertext, we determine the next block of plaintext and send it back to CBC encryption. Due to the one block delay, we can only start this from the second block. For the first block of plaintext, we submit instead any value of our choice.
such as the all-zero block. Starting from the second block, we choose our next plaintext block as:
\[ P_i = P_{i-1} \oplus C_{i-2}. \]

Remark, that both values \( P_{i-1} \) and \( C_{i-2} \) are effectively known at the time we need to submit \( P_i \). Due to this specific choice the ciphertext values are thus determined by the following formula:
\[
C_i = E(C_{i-1} \oplus P_{i-1} \oplus C_{i-2})
= E(E(P_{i-1} \oplus C_{i-2}) \oplus (P_{i-1} \oplus C_{i-2})).
\]

In other words, letting \( Z_i \) denote the input to the block cipher in round \( i \), i.e. \( Z_i = P_i \oplus C_{i-1} \), we see that \( Z_i \) satisfies a recursion law:
\[ Z_i = E(Z_{i-1}) \oplus Z_{i-1}. \]

Quite interestingly, XORing the input and the output to a blockcipher is a pseudo-random function from a pseudo-random permutation, known as Davies-Meyer construction. Thus, defining the function \( f \) by \( f(x) = x \oplus E(x) \), the recursion becomes \( Z_i = f(Z_{i-1}) \).

Using classical analysis about random function, we know that the sequence \( Z \) loops back to itself after \( O(2^n/2) \) steps. Moreover, there exists efficient algorithms to detect such a loop without using too much memory. These algorithms are used for example as the basis for Pollard’s Rho factoring method. Let us briefly recall two of these algorithms due to Floyd and Brent.

\textbf{a) Floyd’s algorithm.} In order to detect a cycle in a sequence \( Z \) defined by recursion from a function \( f \), the idea of Floyd’s algorithm is to define an auxiliary sequence \( W_i = Z_{2i} \), to compute \( W \) and \( Z \) in parallel and to look for a position \( i \) where \( Z_i = W_i \). Without going further into the details, we see that Floyd’s algorithm is not suited to attack CBC encryption, because there is no way\(^3\) to compute both sequences \( Z_i \) and \( W_i \) at the same time. Of course, we could store \( Z \) and compare \( Z_i \) with \( Z_{2i} \) at each step, but this would require a large amount of memory, which we are trying to avoid. Note that this approach would still be better than sorting and could even work with slow memory.

\textbf{b) Brent’s algorithm.} Detecting cycles with Brent’s algorithm uses a different approach. The key idea is to memorize one single value and test whether we are looping back on this value. More precisely, whenever the index \( i \) is a power of \( 2 \), we store this value \( Z_{2i} \) and then compare it to the subsequent values of \( Z_i \). When we reach \( Z_{2i+1} \), we erase \( Z_{2i} \) and continue with the new value. This technique detects a cycle in time \( O(2^n/2) \) for a random function \( f \). Moreover, the sequence \( Z \) only needs to be computed once. Thus the method is applicable in our context. Note that another recent algorithm of Nivasch [14] could also be used for our purpose.

By itself, detecting a cycle is already an attack. Indeed, this can be seen as a real or random distinguisher. If the output is real, i.e comes from a blockwise CBC encryption, we find a cycle in time \( O(2^n/2) \) with high probability. If the output is random, then Brent’s algorithm almost never finds a new value equal to the stored value. However, we can strengthen the attack by doing more than this basic distinguisher. In fact, we can use the cycle to obtain the encryption of any value \( v \) of our choice. For this, when storing \( Z_{2t} \), which is a block cipher input, we also store the corresponding output \( C_{2t} \). Due to the one block delay, \( C_{2t} \) is not available immediately, but we store it whenever we receive it. When the collision is detected, we have a new input \( Z_i \) equal to \( Z_{2t} \). Of course, this equality between inputs to the block cipher implies equality between the output. So we know in advance that \( C_i = C_{2t} \). Thanks to this knowledge, we can set \( P_{i+1} = C_i \oplus v \) and thus obtain the encrypted value \( E(v) \).

As a consequence, this attack against CBC encryption in the blockwise model is more efficient than the usual attack because it does not require a large memory. It is also more powerful because we can obtain the encryption of any chosen value \( v \). This second property is extremely good for the cryptanalyst when the mode of operation is used together with a relatively weak cipher such as DES. Indeed, obtaining the encryption of a chosen value allows him to apply Hellman’s time-memory trade-off, thus recovering the key faster than by exhaustive search.

\textbf{c) Note on CFB mode.} Once again, it is possible to use the same attack with CFB. It suffices to input as plaintext block \( P_i \) the value of the previous block of ciphertext \( C_{i-1} \). This leads to a recursion:
\[ C_i = E(C_{i-1}) \oplus C_{i-1}. \]

Thus, we can proceed as above, using the sequence \( C_i \) instead of \( Z_i \).

\section*{B. XCBC}

In XCBC, we do not directly know the input values to the block cipher due to the output masking. Thus, we need to modify our plaintext feedback technique in order to cancel the randomness. To do this, we are going to rely on the following equation:
\[
Z_{i+1} - Z_i = (C_{i+1} - (i + 1) \times r_0) - (C_i - (i \times r_0))
= C_{i+1} - C_i - C_1.
\]

This equation says that feeding \( C_{i+1} - C_i - C_1 \) as input to the mode of operation is equivalent to feeding \( Z_{i+1} - Z_i - Z_1 \), whose value is based on the inner states only. To avoid mixing several consecutive computations, we alternate plaintext values, once sending the zero block and once sending the above value. In other word, we define:
\[
P_{2i+1} = 0^n \quad \text{for odd indices}
\]
\[
P_{2i+2} = C_{2i+1} - C_{2i} - C_1 \quad \text{for even indices}.
\]

This induces the following recursion formula:
\[
Z_{2i+2} = E(P_{2i+2} \oplus Z_{2i+1})
= E((Z_{2i+1} - Z_{2i} - Z_1) \oplus E(Z_{2i}))
= E(E(Z_{2i}) - Z_{2i} - Z_1) \oplus E(Z_{2i})).
\]

\footnote{Short of controlling the IV, which turn CBC into an insecure mode even against non blockwise attacks.}
To put this back into the usual framework, we define $Z'_i = Z_{2i}$ and $f(x) = E((E(x) - x - Z_1) \oplus E(x))$. As a consequence, $Z'_{i+1} = f(Z'_i)$.

We know from the random function analysis that the sequence $Z'$ loops in time $O(2^n/2)$. Since $Z'$ itself is not observed during encryption, the loop cannot be detected directly. However, it suffices to remark that when $Z'_i = Z'_j$, we have $Z_{2i} = Z_{2j}$ and also $Z_{2i+1} = Z_{2j+1}$. This implies that the computed plaintext blocks $P_{2i+2}$ and $P_{2j+2}$ collide. As a consequence, we can detect the loop on the sequence $Z'$ by observing $P$.

Once the loop is detected, we use the relation $Z_{2i} = Z_{2j}$ to deduce that:

$$C_{2j} - C_{2i} = (Z_{2j} + (2j \times r_0)) - (Z_{2i} + (2i \times r_0)) = 2(j - i)r_0.$$  

This equation allows us to learn most bits of $r_0$. The highest bit always remains unknown and a few other may also remain unknowns. If 2 divides $j - i$, the next high bit is unknown, if 4 divides $j - i$ the next two bits are unknown and so on. Guessing the few missing bits of $r_0$, we can for each output $C_k$ following the collision, compute the corresponding value of $Z_k$. Choosing the block of plaintext accordingly, we obtained an encryption of $0^n$ (or any other value of our choice). To check that our completion of $r_0$ is correct, we can repeat the approach to obtain a second encryption of $0^n$. Note that to be sure that all bits of $r_0$ are used, we need to align the queries for outputs $C_k$ with $k$ odd. Alternatively, we can obtain an encryption of zero without guessing the missing bit of $r_0$, instead aligning our query on an output $C_k$ where $k$ contains a sufficiently large power of two to cancel the contribution of the missing bits.

C. A harder example: HCBC

In this section, we consider the security of HCBC in the blockwise adaptive model and try to use our cycle finding approach. For simplicity, we only look at the easiest case of HCBC1 where a fast almost-XOR universal hash function is used. The goal here is to create a collision of the form:

$$E(P_i \oplus H(C_{i-1})) = E(P_j \oplus H(C_{j-1})),$$

with different values of $P_i$ and $P_j$. As in the ordinary model such a collision yields a linear equation on the secret key of the hash function.

The first idea would be to use the cycle finding algorithm directly. With HCBC1, we could let $P_i = C_{i-1}$ and get a recursion law:

$$C_{i+1} = E(C_i \oplus H(C_i)).$$

However, this approach is problematic. Unless we are extremely lucky, we detect the cycle somewhere in the middle. Thus, when we discover that $C_i = C_j$, we also have $C_{i-1} = C_{j-1}$ and the equation reduces to $0 = 0$, giving no information on the secret key. In order to obtain some real equation, we need to find the entrance of the cycle and get two different preimages of the same value for the function $x \oplus f(x)$. This can be done using the fact that recursion based on a function have a main cycle whose connected component covers most of the graph induced by this recursion. To exploit this fact, we proceed as follows:

- First, find the main cycle as usual and memorize a point within the cycle say $C_x$.
- Right after $C_x$, feed a random plaintext block $P_1$ and from there reapply the recursion until $C_Z$ appear again.
- With high probability, we now have a path from random point out of the cycle that goes to the main cycle. We are now going to find the distance from $P_1$ to the cycle. In order to do this, we use a dichotomy. First advance half the distance from $P_1$ to $C_x$, memorize the point, advance to $C_x$ and continue advancing until either the memorized point or $C_x$ is found again. If we find $C_x$, the distance to the cycle is longer otherwise it is shorter. Performing a number of iterations from $P_1$ to the cycle, we get finer and finer estimates for the distance from $P_1$ to the cycle. After a logarithmic number of turns, we obtain the exact distance. This allows to get two different preimages for point where the path enters the main cycle. One preimage from outside the cycle and one from inside.

With this example, the gain is much less clear. The cryptanalyst can indeed get rid of the extra memory but the cost is to make extra calls to the encryption oracle. Instead of encrypting $O(2^n/2)$ blocks, he needs to encrypt $O(n2^{n/2})$ blocks.

It is possible to avoid this extra logarithmic factor of $n$ by using the option of encrypting two different messages concurrently. The idea is to start by encrypting a first copy of the message defined above until we find a point $C_z$ in the cycle. Then, we continue to encrypt the first message until we loop back to $C_y = C_x$, this yields the length $L = y - x$ of the cycle. After this, we carry on the encryption up to $C_z$ where $z$ is the smallest multiple of $L$ greater than $y$.

After this setup, we start a second copy of the same message and compute the copies in parallel. This yields pairs $(C_1, C_{z+i})$. Assuming that $C_0$ does not lie on the main cycle, the two initial values are distinct. Moreover, at step $x \mod z$ we obtain the pair $(C_x, C_{x+z})$ with $C_{x+z} = C_x$ since $z$ is a multiple of the cycle length. Thus, somewhere between step 0 and step $x$, there is a step where two distinct values $(C_i, C_{z+i})$ are transformed into a collision $C_{i+1} = C_{z+i+1}$.

This discussion shows that HCBC1 and HCBC2 are much more resistant to this kind of blockwise attacks than CBC, especially when they are used with a hash function based on a block cipher construction. However, if the attacker pursue his effort and obtain the secret key of $H$, he can still do more in the blockwise model than in the regular model. Indeed, once $hk$ is known, he can fix the input to the blockcipher $E$ to any value $v$ by remarking that this input is equal $P_i \oplus H(C_{i-1})$ (for HCBC1) or $P_i \oplus H(P_{i-1}, C_{i-1})$ (for HCBC2). With the key, he computes the hash value then XOR it with $v$ in order to obtain the correct value needed for $P_i$. The encrypted value $E(v)$ is obtained either as $C_i$ (for HCBC1) or as $C_i \oplus H(P_{i-1}, C_{i-1})$ (for HCBC2).

D. Streaming modes of operation

Among modes of operation which are blockwise secure, we also find modes where the blockcipher is used to create
an encryption stream which is then XORed with the plaintext. The counter mode (CTR) and output feedback mode (OFB) are two specific examples of streaming modes. Due to their way of working, streaming modes are inherently immune to our attack. Indeed, the plaintext blocks are never used as inputs to the blockcipher in any way. As a consequence, the pre-existing cycles are too long to be useful. For CTR, the length of the cycle is the length of the counter (either $2^n$ for a integer based counter or $2^n - 1$ for a LFSR based counter). For OFB, the periodicity is due to cycles of the permutation $E$. The cycle is determined by the initial value used to seed the OFB mode. With high probability, we expect to be in a long cycle of length $O(2^n)$.

**VI. CONCLUSION**

In this paper, we showed that blockwise adaptive security is a subtle concept. Despite identical security statements, CBC encryption is less resistant to cryptanalysis in the blockwise model. Moreover, the efficiency of our blockwise specific attacks greatly varies from one mode to the next. Some modes such as HCBC or streaming modes seem particularly strong in this respect. As a consequence, for applications which use online encryption and which for legacy reasons have to use blockciphers with small blocks, special care should be taken to avoid this cycle finding attacks.

**REFERENCES**


