Padding Oracle Attacks on the ISO CBC Mode Encryption Standard

Kenneth G. Paterson* and Arnold Yau**
Information Security Group,
Royal Holloway, University of London,
Egham, Surrey, TW20 0EX, UK
{kenny.paterson, a.yau}@rhul.ac.uk

Abstract. In [8] Vaudenay presented an attack on block cipher CBC-mode encryption when a particular padding method is used. In this paper, we employ a similar approach to analyse the padding methods of the ISO CBC-mode encryption standard. We show that, for several of the padding methods referred to by this standard, we can exploit an oracle returning padding correctness information to efficiently extract plaintext bits. In particular, for one padding scheme, we can extract all plaintext bits with a near-optimal number of oracle queries. For a second scheme, we can efficiently extract plaintext bits from the last (or last-but-one) ciphertext block, and obtain plaintext bits from other blocks faster than exhaustive search.

Keywords
padding oracle attack, CBC-mode encryption, ISO standard

1 Introduction

1.1 Background

In [8] Vaudenay presented an attack on block cipher CBC-mode encryption when a particular padding method is used. The attack requires an oracle which on receipt of a ciphertext, decrypts it and replies to the sender whether the padding is valid or not. The attack model assumes the attacker to have intercepted some such padded then CBC-mode encrypted ciphertext under some key K, and have access to the aforementioned padding validity oracle (operating using the same key K). The result is that the attacker can recover the plaintext corresponding

* This author supported by the Nuffield Foundation, NUF-NAL02
** This author supported by EPSRC and Hewlett-Packard Laboratories Bristol through CASE award 01301027. Also supported by EU Fifth Framework Project IST-2001-324467 "CASENET".

to any block of ciphertext using an average of $128b$ oracle calls, where b is the number of bytes in a block and a byte is eight bits.

Further research has been done by Black and Urtubia [1], who generalised Vaudenay's attack to other padding schemes and modes of operations, and presented a padding method which prevents the attack. In [2], Canvel *et al* demonstrated the practicality of padding oracle attacks and showed how subtleties in security protocol implementation can lead to flaws. First of all they realised an SSL/TLS padding oracle by exploiting timing information that is available upon submission of correctly and incorrectly padded ciphertexts. Secondly an attack against the IMAP protocol when used over SSL/TLS was implemented. In a typical setting, the attack recovers the IMAP password within one hour. Klíma and Rosa [7] applied idea of a "format correctness oracle" (of which padding is a special case) to construct a PKCS#7 validity oracle and were able to decrypt one PKCS#7 formatted ciphertext byte with on average 128 oracle calls.

1.2 ISO standards

The current ISO standard for modes of operation of a block cipher is the second edition of ISO/IEC 10116 [4] (the third edition [5] is under development at the time of writing). It does not, however, specify any padding methods for the modes of operation (including CBC) that require one. In Section 5: Requirements it indicates that padding methods are beyond its scope and instead refers to ISO/IEC 9797-1 [3] (MACs using a block cipher) and 10118-1 [6] (general hash functions) where a few such methods are defined. Using a similar approach to [8], we have found attacks of various severity against some of those methods when used with CBC-mode encryption. Thess attacks do not, however, entail any security implications for those padding methods when they are used within their proper contexts (i.e. MACs and hash functions).

Note that in Annex B.2.3 of ISO/IEC 10116, ciphertext-stealing and another method are described for the special treatment of the last two blocks when encrypting under CBC-mode, when padding the plaintext is *not* acceptable. The standard does not prescribe that these methods be used, only that they *can* be used instead of padding. We emphasise that we are not attacking these two methods, but rather the padding methods in ISO/IEC 9797-1 and 10118-1 that are recommended for use in ISO/IEC 10116.

1.3 Our contribution

We assume that an attacker has access to a padding oracle operating under the fixed key K and has intercepted a ciphertext encrypted in CBC-mode under that key. The attacker's aim is to recover the plaintext for that ciphertext. We further assume that the attacker is able to choose the initialisation vector when submitting ciphertexts to the oracle. This assumption prevents our attack from working when secret IVs are used; this is permitted in [4]. Some or all of these assumptions may be unwarranted when one is attacking a real system.

Under the above assumptions, our main results are as follows:

1. Attacking against padding method 3 of [3], the attacker can recover the plaintext for every ciphertext block with $n + O(\log_2 n)$ oracle calls for each block, where n is the block size.
2. There are two attacks against padding method 3 of [6], though they are to some degree interdependent. The padding method requires a parameter r to be chosen where $1 \leq r \leq n$. In the first of our two attacks, the attacker can recover all plaintext bits to all ciphertext blocks with a complexity of $O(2^{r-1})$ oracle calls per block when $r < n$. When $r = n$ the complexity increases to $O(2^n)$. In our second attack, depending on which of two possible states the padding is in, the attacker either recovers the whole of the last plaintext block with $n + O(\log_2 n)$ oracle calls, or recovers some u bits of the last-but-one plaintext block which then speeds up the first attack by a factor of 2^{u-1} in recovering the remaining $n - u$ bits of the block.

We will first introduce some notation used throughout the paper, followed by a review of CBC-mode encryption. Then we present in turn each padding method in [3] and [6] and, if applicable, our attack against it. We conclude with a few remarks about the need for careful cryptographic design to prevent side-channel attacks.

2 Symbols, Notation and CBC-Mode Review

2.1 Symbols and notation

Each symbol and notation will be introduced on their first use, but we find it convenient to gather them here for reference purposes.

C : ciphertext output after CBC-mode encryption and ciphertext the attacker is trying to decrypt
C' : ciphertext to be submitted to an oracle during an attack
$d_K(Y)$: decryption of ciphertext block Y under key K
$e_K(X)$: encryption of plaintext block X under key K
D : unpadded data string to be CBC-mode encrypted
I_j : the j^{th} intermediate block during CBC-mode encryption, i.e. $D_j \oplus C_{j-1}$, or in the case $j = 1$, it is $D_1 \oplus IV$
I'_j : the j^{th} intermediate block during the attack, i.e. $d_k(C'_j)$
IV : the initialisation vector used in CBC-mode
L_D : the length (in bits) of the data string D
n : the block size (in bits) of the block cipher
P : the result of applying a given padding method to D
P' : data string computed by the padding oracle in the course of verifying padding
q : the number of blocks in data string P after padding
VALID and INVALID: oracle responses to, respectively, correct and incorrect padding after receipt and decryption of some ciphertext
$X||Y$: the result of concatenation of strings X and Y

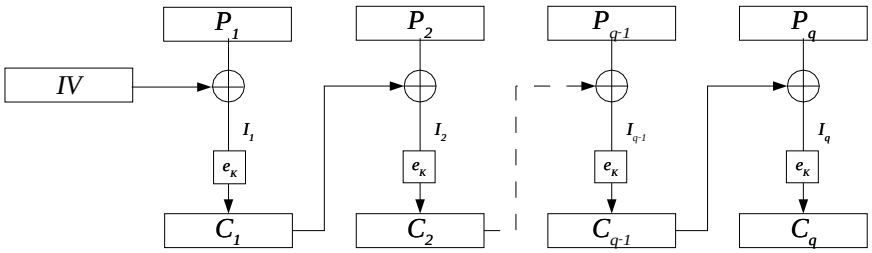

Fig. 1. CBC-mode encryption

$X \oplus Y$: the result of exclusive-or (XOR) of strings X and Y
X_2 : the binary representation of the value X
X_j : the j^{th} block of the plaintext or ciphertext X
$X_{j,k}$: the k^{th} bit of the plaintext or ciphertext block X_j

2.2 Review of CBC-mode encryption

Cipher Block Chaining (CBC) is a mode of operation for an n-bit block cipher for encrypting data of arbitrary length. It has been standardised in second edition of ISO/IEC 10116 [4] and it is, quite naturally, included in the lastest draft of the third edition of that standard [5].

Let the encryption operation of the block cipher under key K be e_K, and the data we wish to encrypt be D. CBC-mode encryption (Figure 1) operates as follows:

1. A *padding method* is applied to D to make a padded message P of bitlength a multiple of n.
2. P is divided into n-bit blocks $P_1, P_2 \ldots P_q$.
3. An n bit number is chosen, at random or in a specified way, as the initialisation vector IV.
4. Compute ciphertext block $C_1 = e_K(IV \oplus P_1)$ and then
5. $C_i = e_K(C_{i-1} \oplus P_i)$, for $2 \leq i \leq q$
6. The resulting $C = IV||C_1||C_2|| \ldots ||C_q$ is the CBC-encrypted ciphertext.

We assume that IV is always prepended to the ciphertext. This allows a more concise notation for our attacks to follow and means that IV effectively plays the role of the "zeroth" ciphertext block; we write $C_0 = IV$.

Let d_K denote the inverse operation to e_K. To decrypt a block C_i (Figure 2) we simply have to compute $D_i = d_k(C_i) \oplus C_{i-1}$ for $2 \leq i \leq q$, and $D_1 = d_k(C_1) \oplus IV$.

Some security properties of CBC-mode are outlined in Section 2 of [8].

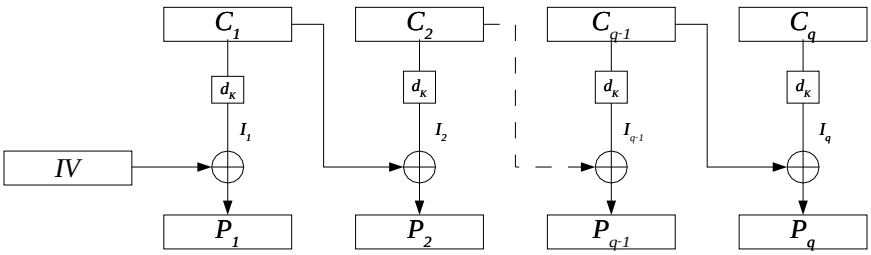

Fig. 2. CBC-mode decryption

3 Attacking the Padding Methods of ISO/IEC 9797-1

3.1 The standard

The standard [3] specifies six algorithms to compute an m-bit MAC using an n-bit block cipher with a secret key. The algorithms themselves are essentially instances of the CBC-MAC method or variants of it. Padding is applied, as with CBC-mode encryption, when the plaintext is not of length (in bits) a multiple of n, the block cipher size. For some methods it is always applied, regardless of the plaintext length.

3.2 Padding method 1

The method is described as follows:

> "The data string D to be input to the [...] algorithm shall be right-padded with as few (possibly none) '0' bits as necessary to obtain a data string whose length (in bits) is a positive integer multiple of n."

Notice that this method is many-to-one: different data strings may be padded to yield the same result, which means that padding cannot be removed unambiguously if the length of the plaintext is not known. Consequently, given a padded data string, one cannot even tell where the data/padding boundary is, let alone check for padding validity. In fact, without data length information, every plaintext P is a validly padded version of at least one data string D. This of course limits the applicability of the padding technique to cases where the plaintext is of a fixed length, or where the proper length is somehow otherwise conveyed to the recipient.

No attack can be based on information returned from a padding oracle because any ciphetext submitted to such an oracle will decrypt to give a correctly padded plaintext.

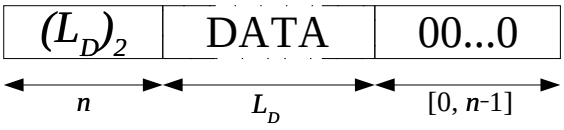

Fig. 3. ISO/IEC 9797-1 padding method 3

3.3 Padding method 2

The method:

> "The data string D to be input to the [...] algorithm shall be right-padded with a single '1' bit. The resulting string shall then be right-padded with as few (possibly none) '0' bits as necessary to obtain a data string whose length (in bits) is a positive integer multiple of n."

This method has been analysed in [1] (it is called OZ-PAD in that paper). The key result of [1] is that the method appears to resist padding oracle attacks. This is because practically all data strings are correctly padded, with the only exception being when a block contains all '0' bits. However this padding mechanism still lacks what is known as "semantic security" — an INVALID reply from the padding oracle would tell the attacker that the decrypted plaintext block is not a particular bit string. See [1] for details.

3.4 Padding method 3

The method (Figure 3):

> "The data string D to be input to the [...] algorithm shall be right-padded with as few (possibly none) '0' bits as necessary to obtain a data string whose length (in bits) is a positive integer multiple of n. The resulting string shall then be left-padded with a block L. The block L consists of the binary representation of the length (in bits) L_D of the unpadded data string D, left-padded with as few (possibly none) '0' bits as necessary to obtain an n-bit block. The right-most bit of the block L corresponds to the least significant bit of the binary representation of L_D."

We have an attack against this padding scheme that decrypts, a block at a time, arbitrary ciphertexts $C_1||C_2||\ldots||C_q$. This attack takes $n + O(\log_2 n)$ oracle calls per block. There are two phases to this attack: determining L_D and the actual decryption.

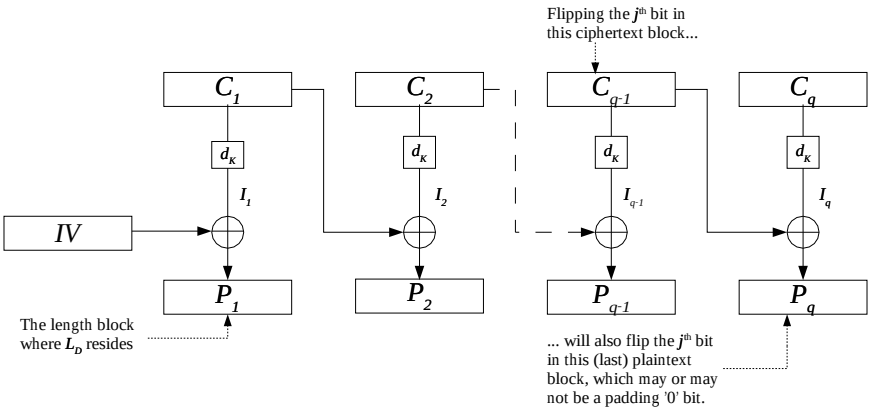

Fig. 4. Attack phase 1 — obtaining L_D

Phase 1: Determining L_D We want to find L_D, the content of the first block, which indicates the length of the unpadded data. To do that we use the padding oracle to determine the number of '0' bits that have been appended to the last block, if any. This is performed as follows (Figure 4).

Firstly notice that in CBC-mode decryption, flipping (complementing) any single bit at position i in block C_j would flip also the decrypted plaintext bit at position i in block P_{j+1} (whilst corrupting the whole of plaintext block P_i). This allows us to flip arbitrary bits within a block in the decrypted plaintext by appropriately altering the ciphertext. This observation is in fact the basis of all of our attacks.

A padded data string consists of $q \geq 2$ blocks. Here we consider the case $q \geq 3$; the case $q = 2$ is handled separately below. The string is right-padded with some '0' bits and left-padded with the length block containing the binary representation of L_D. L_D is effectively a pointer to the last bit of the unpadded data, all the bits after which should be '0.' Let's now see what happens if we flip a single bit in P_q, the last plaintext block of the data string (by flipping a bit in C_{q-1}, the last-but-one ciphertext block). This change does not affect the decryption of C_1 (since $q \geq 3$) so the length block is left intact. So one of two things might happen:

1. The bit flipped is part of the original unpadded data. The padding is therefore still intact and correct and the oracle returns VALID.
2. The bit flipped is one of those '0' bits padded. The oracle therefore detects a '1' bit where it should have been '0,' and thus returns INVALID.

This means that after flipping a single bit in D_q, a VALID oracle response implies the padding boundary is to the right of the current position, and to the left otherwise. So we now can work out the exact location of the boundary by

flipping the last plaintext block one bit at a time, say from right to left. The transition point of oracle response from INVALID to VALID tells us the location of the boundary we are after. This can be made more efficient by using a binary search similar to that presented in Section 3 of [1]. Once we have the boundary it is trivial to compute the value L_D from the number of blocks in the ciphertext and the position of the boundary within the last block.

This phase is presented in Algorithm 9797-1-m3-get-L_D-general below. The notation $X_{a,b}$ denotes the bit at position b of the ciphertext or plaintext block X_a. We number the positions in a block from 0 to $n-1$, going from left to right.

This method of obtaining L_D does not work when the unpadded data string consists only of a single block (this includes the case of the data being the null string). Here, the padded data string consists of two blocks $P_1||P_2$. Now flipping bits in the last (second) plaintext block would require changes in the first ciphertext block C_1, which in turn would corrupt the first plaintext block where L_D is supposed to reside.

Fortunately, there is a way to circumvent this problem at least for block sizes $n = 2^m$, $m \geq 1$, the most common situation in practice. Let $C = IV||C_1||C_2$ be the ciphertext for which we wish to determine L_D. It is not hard to see that if $IV' = IV \oplus \underbrace{0\ldots0}_{n-m-1}\,1\,\underbrace{0\ldots0}_{m}$, then $C' = IV'||C_1||C_1||C_2$ is also a valid ciphertext unless $L_D = 0$ or $L_D = n$ (in which cases the padding oracle will return INVALID on submission of C'). In the situation where C' is valid then we can simply apply the method described above to C' to obtain $L'_D = L_D + 2^m$, and hence L_D.

We need to apply a further trick to distinguish the remaining cases, i.e. when $L_D = 0$ or $L_D = n$. Now we set $IV'' = IV \oplus \underbrace{0\ldots0}_{n-m-2}\,11\,\underbrace{0\ldots0}_{m}$ and submit $C'' = IV''||C_1||C_1||C_2$ to the padding oracle. If $L_D = 0$, then C'' will, on decryption, contain a length field L''_D with $L''_D = 3n$. Since the unpadded data in C'' is of length at most $2n$, the padding oracle will output INVALID. On the other hand, if $L_D = n$, then C'' will yield $L''_D = 2n$ and C'' accepted as VALID. Hence one futher oracle query on a carefully chosen C'' is sufficient to decide whether $L_D = 0$ or $L_D = n$.

The special case $q = 2$ is presented in Algorithm 9797-1-m3-get-L_D-special below.

Phase 2: Decrypting We now have L_D, the binary encoding of which is the content of the first plaintext block. We can deduce that I_1, the first intermediate block, is equal to $L_D \oplus IV$. Note that by manipulating IV, we can change the content of the first block to indicate a data length of any desired value. If L'_D is the desired value, we can take $IV' = (L'_D)_2 \oplus I_1 = (L'_D)_2 \oplus (L_D)_2 \oplus IV$.

We are now ready to decrypt an arbitrary ciphertext block C_k from the ciphertext $IV||C_1||C_2||\ldots||C_q$, where $2 \leq k \leq q$ (Figure 5). Note that there is no need to decrypt C_1 as it just encrypts the value L_D. The decryption is done in a bit-by-bit fashion, starting from the rightmost bit. So to begin with we submit

Algorithm
9797-1-m3-get-L_D-general

Input: $IV||C_1||C_2||\dots||C_q$
Output: L_D

Ensure: $q \geq 3$
$C := IV||C_1||C_2||\dots||C_q$
$l := 0$
$u := n - 1$
repeat
$h := \lceil (l+u)/2 \rceil$
$C_{q-1,h} := C_{q-1,h} \oplus 1$
if oracle(C) = VALID **then**
l := h
else if oracle(C) = INVALID **then**
u := h-1
end if
$C_{q-1,h} := C_{q-1,h} \oplus 1$
until $l = u$
return $L_D := (q-1)n + l + 1$

Algorithm
9797-1-m3-get-L_D-special

Input: $IV||C_1||C_2$
Output: L_D

Ensure: $n = 2^m, m \geq 1, q = 2$
$IV' := IV \oplus \underbrace{0\dots0}_{n-m-1}\,1\,\underbrace{0\dots0}_{m}$
$C' := IV'||C_1||C_1||C_2$
if oracle(C') = VALID **then**
L'_D = 9797-1-m3-get-L_D-general(C')
return $L_D := L'_D - 2^m$
else
$IV'' := IV \oplus \underbrace{0\dots0}_{n-m-2}\,11\,\underbrace{0\dots0}_{m}$
$C'' := IV''||C_1||C_1||C_2$
if oracle(C'' = VALID) **then**
return $L_D := n$
else
return $L_D := 0$
end if
end if

to the oracle the ciphertext $C' = IV'||C_1||R||C_k$ where

$$IV' = (2n-1)_2 \oplus (L_D)_2 \oplus IV,$$

and R is a random n-bit block.

After decryption, L'_D, the length field in the resulting plaintext $P'_1||P'_2||P'_3$ points to the last-but-one bit of P'_3, the last block. Now the padding oracle outputs VALID for C' if the last bit of P'_3 is equal to '0,' and INVALID if $P'_{3,n-1}$ is equal to '1.' We then have $I'_{3,n-1} = P'_{3,n-1} \oplus R_{n-1}$ and this block I'_3 is equal to the original intermediate block I_k. So we can obtain $P_{k,n-1} = I'_{3,n-1} \oplus C_{k-1,n-1}$.

To decrypt the next bit, we construct a new ciphertext for the oracle. We want, after decryption, the value in L'_D to decrement by one, and to ensure that $P'_{3,n-1}$ is '0'. We can achieve the former by altering IV appropriately and the latter by keeping/flipping last bit of C'_2 if the previous response was VALID/INVALID. Submitting the resulting ciphertext to the oracle, a VALID response indicates $P'_{3,n-2}$ equals '0,' and '1' otherwise. We can then compute $I'_{3,n-2} = P'_{3,n-2} \oplus R_{n-2}$. Note that the random block R at this iteration which may (or may not) have changed from the last iteration. We can now obtain $P_{k,n-2} = I'_{3,n-2} \oplus C_{k-1,n-2}$.

The process is repeated, decrementing L'_D by one per iteration while making sure the bit positions in P'_3 corresponding to those we have obtained stay at '0.' One bit of I'_3 and one bit of P_k are obtained at each iteration and we stop after $n-1$ iterations when the $n-1$ rightmost bits of those blocks are determined.

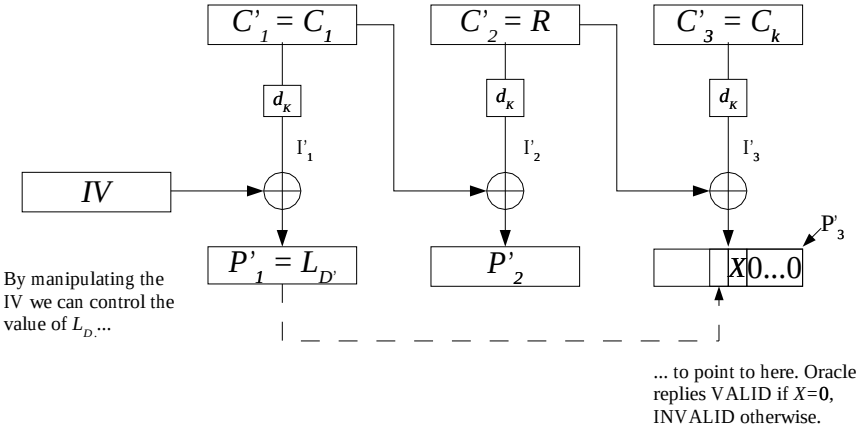

Fig. 5. Attack phase 2 — decrypting

We cannot get the leftmost bit of the block I'_3 (hence $P_{k,0}$) using this approach because at the next step L'_D would indicate a length $2n$, a multiple of the block size, and according to the standard, we would never append a new block in such cases.

Instead, we extract this leftmost bit $P_{k,0}$ by using a different approach. We assume that standard binary encoding is used for length information, with least significant bit in the rightmost position. (A similar attack can be mounted in the opposite situation too, but we omit the details.) Consider the ciphertext $C' = IV'||C'_1||R$ where $IV' = C_{k-1} \oplus 0P_{k,1}P_{k,2}\dots P_{k,n} \oplus (n)_2$, $C'_1 = C_k$ and R is a random n-bit block. This ciphertext is constructed in such a way that the length field is equal to $P_{k,0}0\dots0 \oplus (n)_2$, indicating a length of either n or $n+2^{n-1}$ depending on the value of $P_{k,0}$. So if C' is submitted to the oracle, then an output of VALID(INVALID) tells us that $P_{k,0} = 0$ ($P_{k,0} = 1$, respectively.)

We summarise the decryption phase as the pair of algorithms 9797-1-m3-decrypt and 9797-1-m3-decrypt-last-bit below. In these algorithms, Ω is the function which takes as input a ciphertext C and is defined as:

$$\Omega(C) = \begin{cases} 0 & \text{if the padding oracle returns VALID for input } C, \\ 1 & \text{if the padding oracle returns INVALID for input } C. \end{cases}$$

Complexity Phase 1, in the general case ($q \geq 3$), should take no more than $\log_2 n$ oracle calls using binary search. To decrypt many messages encrypted under a fixed key K, this phase only needs to be performed once. Phase 2 takes one oracle call per plaintext bit, thus n calls per plaintext block. For the special case $q = 2$, one further oracle call is required in situations where $L_D = 0$

Algorithm
9797-1-m3-decrypt

Input: L_D, IV, C_1, C_k
Output: $P_{k,1}P_{k,2}\dots P_{k,n-1}$, the rightmost $n-1$ bits of P_k

$R :=$ a random n-bit block
for $j := n-1$ to 1 **do**
  $IV' := IV \oplus L_D \oplus (n+j)_2$
  $b := \Omega(C')$
  $C' := IV'||C_1||R||C_k$
  $P_{k,j} := b \oplus R_j \oplus C_{k-1,j}$
  $R := R \oplus \underbrace{0\dots0}_{j}\, b\, \underbrace{0\dots0}_{n-j-1}$
end for
return $P_{k,1}P_{k,2}\dots P_{k,n-1}$

Algorithm
9797-1-m3-decrypt-last-bit

Input: $C_{k-1}, C_k, P_{k,1}P_{k,2}\dots P_{k,n}$
Output: $P_{k,0}$, the leftmost bit of P_k

$R :=$ a random n-bit block
$IV' := C_{k-1} \oplus 0P_{k,1}P_{k,2}\dots P_{k,n} \oplus (n)_2$
$C' := IV'||C^k||R$
$P_{k,0} := \Omega(C')$
return $P_{k,0}$

or $L_D = n$ to distinguish between them (no further oracle calls are needed otherwise).

Fewer than $\log_2(n) + 1 + (q-1)n$ oracle calls are needed to recover all the bits of plaintext from a q block ciphertext (remember that the first block contains L_D which is not part of the unpadded data string, and its value is already known after phase 1 anyway).

Optimality The oracle returns one bit of information per use, so $(q-1)n$ is information theoretically the smallest number of oracle calls needed to recover $(q-1)n$ bits of plaintext entropy. Hence our attack makes nearly optimal use of the padding oracle, especially when many ciphertexts are decrypted for the same key K.

4 Attacking the Padding Methods of ISO/IEC 10118-1

4.1 The standard

ISO/IEC 10118 is a standard for hash functions, Part 1 [6] of which describes the general construction of a hash function.

Padding methods 1 and 2 in this standard are identical to the respective methods in ISO/IEC 9797-1 which were already discussed in the previous section. We focus instead on padding method 3 of [6].

4.2 Padding method 3

In the standard, L_1 is used to denote the block size. It will henceforth be replaced by our usual notation n to be consistent with the rest of this paper. The method is as follows (Figure 6):

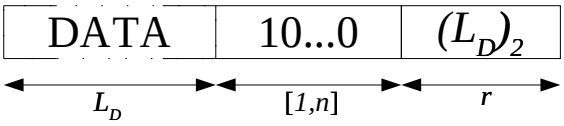

Fig. 6. ISO/IEC 10118-1 padding method 3

"This padding method requires the selection of a parameter r (where $r \leq n$), e.g. $r = 64$, and a method of encoding the bit length of the data D, i.e. L_D as a bit string of length r. The choice for r will limit the length of D, in that $L_D < 2^r$.

"The data D [...] is padded using the following procedure.

1. D is concatenated with a single '1' bit.
2. The result of the previous step is concatenated with between zero and $n - 1$ '0' bits, such that the length of the resultant string is congruent to $n - r$ modulo n. The result will be a bit string whose length will be r bits short of an integer multiple of n bits (in the case $r = n$, the result will be a bit string whose length is an exact multiple of n bits).
3. Append an r-bit encoding of L_D using the selected encoding method, yielding the padded version of D."

The above description can be summarised as "pad a '1' followed by the smallest number of '0's needed to push the r bits of L_D right to the end of a block." Using this method, the padded bits for data string D are appended in one of two ways:

Same-block $(L_D \bmod n) \leq (n - r - 1)$ The last block has enough space after the last plaintext bit to contain at least a single '1' bit and the r bits of L, the length block that holds L_D. The number of padded bits is between $r + 1$ and $n - 1$.

Cross-block $(L_D \bmod n) \geq (n - r)$ The last block does not have enough space to contain a '1' bit and the r bits of L. The number of padded bits is between n and $n + r$ and the padding extends over two blocks. Note that this will always be the case when $r = n$.

We have identified two attacks against this method, though they are to some degree dependent on each other. Note that no encoding method (for L_D) is specified in the standard. Our attacks work no matter which encoding method is used, though the attacker needs to know this method. We expect that base 2 encoding will be used in most cases and it will be used for illustrative purposes henceforth.

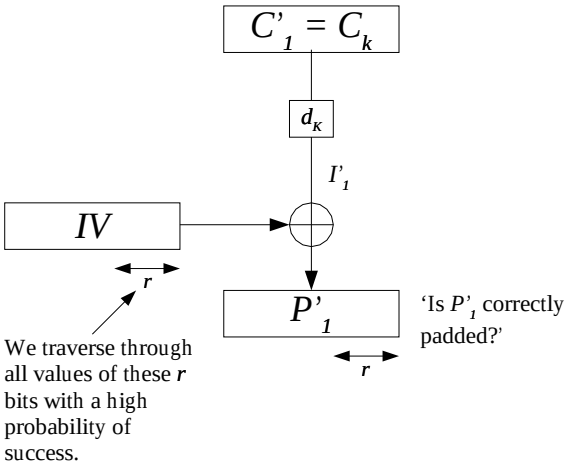

Fig. 7. Directed IV search

Attack 1: Directed IV search This attack works against any block C_k of the ciphertext $IV||C_1||C_2||\ldots||C_q$ and recovers the corresponding plaintext block using on average $2^{r-1} + 2^{2r-n+1}$ and at most $2^r + 3 \cdot 2^{2r-n-1}$ padding oracle queries, provided $r \leq n-1$. When $r = n$, the attack requires on average 2^n and at most 2^{n+1} oracle queries.

We first consider the case $r \leq n-1$. We submit to the oracle strings of the form

$$IV'||C_1'$$

where IV' is a specially selected initialisation vector and $C_1' = C_k$, hoping that the oracle returns VALID. This situation is depicted in (Figure 7). If it does, then the plaintext block P_k can be extracted using Attack 2 below on the ciphertext $IV'||C_1'$. The overall complexity will be the sum of the two attacks' complexities, and will be dominated by the complexity of this first phase.

How then should IV' be selected? Notice that there is a probability of $1-2^{r-n}$ that there is a '1' somewhere in the leftmost $n-r$ bits of P_k. Thus, if we traverse through all 2^r possible settings of the rightmost r bits of IV', then with probability $1-2^{r-n}$ we will obtain (at least) one VALID reply from the padding oracle. The expected number of oracle queries in this situation is therefore 2^{r-1}. But with probability 2^{r-n}, all replies will be INVALID. Now if we flip the bit in position $n-r-1$ of IV' and repeat the above process, it is easy to see that we are guaranteed to obtain at least one VALID response. A simple analysis shows that the number of oracle queries needed is equal to $2^{r-1} + 2^{2r-n+1}$ on average and is always at most $2^r + 3 \cdot 2^{2r-n-1}$.

Algorithm
10118-1-m3-a1-general

Input: C_k, n, r
Output: IV' s.t. $IV'||C_k$ is a valid ciphertext

Ensure: $1 \leq r < n$
  $IV_0 :=$ a random n-bit block
  $IV' := \underbrace{0 \ldots 0}_{n}$
  $i := 0$
  repeat
    $IV' := IV_0 \oplus \underbrace{0 \ldots 0}_{n-r-1} \underbrace{i_2}_{r+1}$
    $C := IV'||C_k$
    $i := i + 1$
  until oracle(C) = VALID
  return IV'

Algorithm
10118-1-m3-a1-special

Input: C_k, n, r
Output: IV', R s.t. $IV'||R||C_k$ is a valid ciphertext

Ensure: $r = n$
  $IV_0 :=$ a random n-bit block
  $R_0 :=$ a random n-bit block
  for $i := 0$ to $2^n - 1$ **do**
    $R := R_0 \oplus \underbrace{i_2}_{n}$
    for $j := 0$ to 1 **do**
      $IV' := IV_0 \oplus \underbrace{0 \ldots 0j}_{n}$
      $C := IV'||R||C_k$
      if oracle(C) = VALID **then**
        return IV', R
      end if
    end for
  end for

An algorithm for Attack 1 in the case $r \leq n-1$ is given in Algorithm 10118-1-m3-a1-general above.

Next we consider the case $r = n$. Here a valid plaintext must be at least two blocks in length and a three-block ciphertext $IV'||R||C_k$ is required to perform the attack. Instead of modifying only the initialisation vector as before, we now also change the random block R at each iteration. The most likely valid two-block plaintext to obtain at random is

$$\underbrace{x_0 x_1 \ldots x_{n-2} 1}_{n} || \underbrace{(L_D = n-1)_2}_{n}$$

where each x_i can be either '0' or '1.' A valid two-block plaintext is guaranteed to occur if we traverse through all 2^{n+1} possible settings of the second plaintext block along with rightmost bit of the first plaintext block (by, respectively, changing R and the rightmost bit of IV'), so on average this strategy has a complexity of 2^n oracle calls.

This special case is illustrated in Algorithm 10118-1-m3-a1-special above.

Decrypting. Once we have a valid padding we can employ Attack 2 below with input a valid ciphertext of the form $IV'||C_k$ (when $r \leq n-1$) or $IV'||R||C_k$ (when $r = n$).

We consider first the case $r \leq n-1$. Here the plaintext corresponding to the ciphertext submitted to Attack 2 will always be same-block padded (because it only contains one block). Then Attack 2 will efficiently recover the entire last

plaintext block for this ciphertext, which we denote by P_1'. P_1' will in general consist of data bits, padding bits and length information. From P_1', it is trivial to recover P_k (the plaintext block that we are actually after). We have:

$$P_k = P_1' \oplus IV' \oplus C_{k-1}.$$

For the case $r = n$, the first phase of Attack 2 below efficiently recovers the length of the unpadded data for the valid ciphertext $IV'||R||C_k$. This information is contained in the length field which occupies all of P_2', the second plaintext block for this ciphertext. Thus after the first phase of Attack 2, P_2' is known. Now P_k can be recovered from:

$$P_k = P_2' \oplus R \oplus C_{k-1}.$$

COMPLEXITY. Obtaining a valid plaintext block takes on average $2^{r-1} + 2^{2r-n+1}$ oracle calls when $r \leq n-1$ and on average 2^n oracle calls in the case $r = n$. Our use of Attack 2 below has a complexity of $n + O(\log_2 n)$ oracle calls for all values of r (recall that for $r = n$, only the first phase of Attack 2 is needed, while for $r \leq n-1$, the plaintext is same-block padded in which case Attack 2 is efficient). Thus our use of Attack 2 does not contribute significantly to the overall complexity to decrypt a single block.

IMPACT. This attack applies to any ciphertext block and all n bits within the block are recovered. For many choices of r this attack is many orders faster than an exhaustive key search, and for a small enough r this attack will be practical whenever a padding oracle is available. When $r = n$, our attack is still better than an exhaustive key search for block ciphers whose key size is greater than the block length. It is interesting to note that the parameter r seemingly innocent of any security implications turns out not to be so at all.

Attack 2: Attacking the last block(s) This attack is conceptually similar to the one against padding method 3 of ISO/IEC 9797-1 given above: there are two phases, the first of which determines L_D and the second of which recovers any plaintext that is found in a "mixed" block – that is, a block that consists of both data and padding bits. There is obviously at most one such block in any plaintext padded using this padding method, which is either the last block or the one that immediately precedes it. If the padding ends exactly on a block boundary, then our attack does not recover any (unpadded) plaintext.

OBTAINING L_D. We want to know L_D, the data length. For ease of presentation we first examine the case $r \leq n-2$, but our algorithm to follow handles all values of r. Here, in the same-block padded case, the last plaintext block P_q corresponding to the last ciphertext block C_q in the ciphertext $IV||C_1||C_2||\ldots||C_q$ has a format as follows:

$$\underbrace{[DATA]}_{t}\underbrace{10\ldots0}_{p}\underbrace{(L_D)_2}_{r}$$

where $t+p+r=n$ and $p \geq 1$. In the cross-block padded case, the above format spans the last two blocks P_{q-1} and P_q and we put $t+p+r=2n$. We note that the attacker does not at first know which of the cases he is faced with.

Given a q-block ciphertext, we want to flip the plaintext bit $P_{q,n-r-2}$, the rightmost position at which a data bit could ever reside, given q and our assumption on r. We submit to the padding oracle the ciphertext

$$IV||C_1||C_2||\ldots||C_{q-1} \oplus \underbrace{0\ldots0}_{n-r-2}10\underbrace{0\ldots0}_{r}||C_q.$$

(Recall that $C_0 = IV$, so the case where $q = 1$ is included here.)

Upon submission of the above ciphertext, the oracle will return:

- VALID meaning the padding has not been disturbed so the bit flipped is a data bit. Since this bit is at the rightmost possible data bit position, we can deduce the data length $L_D = (q-1)n + n - r - 1$. Or else,
- INVALID meaning a padding bit has been flipped so the padding is no longer valid. Therefore the padding boundary is somewhere to the left of this bit, so we continue by resetting this bit and flipping the bit immediately to the left, and test the resulting ciphertext for padding correctness. We repeat this, flipping bits further and further to the left (and into the previous block if necesssary) until the first time the oracle returns VALID. This indicates that the tested bit is the last data bit, and L_D is determined accordingly.

One might worry about instances when cross-block padding arises, where flipping bits in the last plaintext block (by flipping bits in the last-but-one ciphertext block) would turn the last-but-one plaintext block into "garbage" and along with it, potentially, any padding bits within it, so the oracle might report INVALID for the wrong bits. On closer inspection, however, this turns out not to be an issue because all we want to know is whether the padding boundary is to the left or right of the bit in question. Even if the oracle does report INVALID for the wrong bits, it does still imply the boundary is to the left, and VALID would just mean that unpadded data bits have been corrupted so the boundary is still to the right.

A binary search can also be applied here: for any single flipped bit, a VALID response means the start of the padding is to the right of this bit, whereas INVALID means it is to the left. This speed-up is made in Algorithm10118-1-m3-a2-get-L_D below.

We are now ready for the decryption stage. Same-block and cross-block padded messages are treated differently; recall that knowledge of L_D indicates which case the attacker is faced with.

Decrypting: Same-block. Recall the structure of the last plaintext block: t data bits, followed by p padding bits in the form $10\ldots0$ and finally r bits of an encoding of data length L_D. We can recover the remaining t bits of the plaintext in the last block, again using a similar method to decryption phase of the attack

Algorithm 10118-1-m3-a2-get-L_D

Input: $IV||C_1||C_2||\ldots||C_q, n, r$
Output: L_D

$C := IV||C_1||C_2||\ldots||C_q$
$l := (q-2)n + n - r$
$u := (q-1)n + n - r - 1$
repeat
  $h := \lfloor (l+u)/2 \rfloor$
  $C_{\lfloor h/n \rfloor, h \bmod n} := C_{\lfloor h/n \rfloor, h \bmod n} \oplus 1$
  if oracle(C) = VALID **then**
    l := h+1
  else if oracle(C) = INVALID **then**
    u := h
  end if
  $C_{\lfloor h/n \rfloor, h \bmod n} := C_{\lfloor h/n \rfloor, h \bmod n} \oplus 1$
until $l = u$
return $L_D := l$

on ISO/IEC 9797-1 method 3. We submit to the oracle $IV'||C'_1$ where $C'_1 = C_q$ and

$$IV' = C_{q-1} \oplus \underbrace{0\ldots0}_{n-r}\underbrace{(L_D)_2}_{r} \oplus \underbrace{0\ldots0}_{t}\underbrace{10\ldots0}_{p}\underbrace{(t-1)_2}_{r}.$$

After decryption the length block in the plaintext block P'_1 should have the value $t-1$ which points to the last-but-one bit of the original data sub-block, with the middle padding sub-block being all '0's. A VALID response means the last (t^{th}) data bit in P'_1 is a '1,' and '0' otherwise.

By decrementing the length field sub-block in P'_1 one by one whilst keeping all recovered bit positions '0,' a single bit is revealed at each iteration until the whole block is recovered. We can now compute the intermediate block I'_1 by XORing the final IV with D'_1, and then by XORing I'_1 with C_{q-1} we get the original last plaintext block.

This decryption procedure is presented in Algorithm 10118-1-m3-decrypt-same-block below.

Decrypting: Cross-block. For cross-block padded plaintexts, P_q is determined completely by L_D and the padding. However, the padding extends into the penultimate plaintext block P_{q-1}. Suppose u bits of padding are present in P_{q-1}. Then we show how to decrypt C_{q-1} using Attack 1 above, but with a speed-up factor of 2^{u-1}.

Let $v = L_D \bmod n$, then the number of known plaintext bits u is equal to $n-v$ and those bits are of the form $\underbrace{10\ldots0}_{u}$. If we submit the ciphertext $IV'||C'_1$

Algorithm 10118-1-m3-decrypt-same-block

Input: $L_D, IV, C_{q-1}, C_q, r, n$
Output: $P_q := P_{q,0}P_{q,1}\dots P_{k,t-1}\underbrace{10\dots0}_{p}\underbrace{(L_D)_2}_{r}$

Ensure: L_D indicates that the plaintext is same-block padded
$C'_1 = C_q$
$t := L_D \bmod n$
for $j := t-1$ to 0 **do**
$IV' := C_{q-1} \oplus \underbrace{0\dots0}_{n-r}\underbrace{(L_D)_2}_{r} \oplus \underbrace{0\dots0}_{t}\underbrace{10\dots0}_{p}\underbrace{(j)_2}_{r}$
$C' := IV'||C'_1$
$b := \Omega(C') \oplus 1$
$P_{q,j} := b \oplus IV'_j \oplus C_{q-1,j}$
$IV'_j := IV_j \oplus b$
end for
return $P_q := P_{q,1}P_{q,2}\dots P_{k,t}\underbrace{10\dots0}_{p}\underbrace{(L_D)_2}_{r}$

to the oracle where

$$IV' = C_{q-2} \oplus \underbrace{0\dots0}_{n-u}\underbrace{10\dots0}_{u} \oplus \underbrace{0\dots0}_{n-r}\underbrace{(n-r-1)_2}_{n-r}$$

and $C'_1 = C_{q-1}$, then we only need to go through all 2^{r-u+1} settings of the $r-u+1$ bits to the left of the u known bits (by changing IV') to guarantee a valid plaintext. This strategy takes on average 2^{r-u} oracle calls which is a fraction $2^{-(u-1)}$ of the original 2^{r-1} oracle calls for Attack 1 without the knowledge of the u padding bits.

Complexity. It takes $\log_2 n$ oracle calls to find L_D. For same-block padded plaintexts, it takes one call per bit for decrypting. So to recover the t data bits of the last block, $t + \log_2 n$ oracle calls are required.

For cross-block padded plaintexts, on average 2^{r-u} oracle calls are needed to recover the whole of the penultimate plaintext block P_{q-1}, where u is the number of known bits from finding L_D.

Impact. The attack is highly efficient in terms of oracle queries at extracting plaintext bits from the last plaintext block P_q. A maximum of $n-r-1$ bits of data can be recovered in this way and the attack is therefore significant for short messages, especially in combination with a small r. One might argue that $r = n$ is a natural choice for the implementor. In this case, the padding is always cross-block and the attacker must resort to the speeded-up version of Attack 1.

5 Conclusions

We argue that, at least for the CBC-mode of operation for a block cipher standard, it is not good enough just to standardise the mode; an entire specification handling bit-level computations is needed, which necessarily includes padding issues. Padding methods devised for hashing or MACs, as we have shown, may not be suited to encryption operations where a different adversarial model may be applicable.

We also make the point that there is a need for careful consideration of the potential for side-channel cryptanalysis for cryptographic primitives and security protocols in their design phase. Designs should be fully specified so as to allow as little room as possible for the implementor to take potentially weak approaches during implementation.

We agree with the argument in Section 7 of [1] for the practice of the encryption being accompanied by strong integrity checks when possible and appropriate. Such "authenticated encryption" would, within the context of this paper, prevent any practical attempts at constructing a valid ciphertext which in turn precludes the existence of a padding oracle, and hence all the associated attacks that we have discovered.

Acknowledgement

We thank Alain Hiltgen for useful comments on the paper and for showing us how to extract the leftmost bits of plaintext in Section 3.4.

References

1. J. Black and H. Urtubia. Side-Channel Attacks on Symmetric Encryption Schemes: The Case for Authenticated Encryption. *Proceedings of the 11th USENIX Security Symposium, San Francisco, CA, USA, August 5-9, 2002*, pp. 327–338, 2002.
2. B. Canvel, A. Hiltgen, S. Vaudenay, and M. Vuagnoux. Password Interception in a SSL/TLS Channel. In *Proc. CRYPTO 2003*, D. Boneh (ed.), LNCS Vol. 2729, pp. 583–599, 2003.
3. ISO/IEC 9797-1: Information technology — Security tehniques — Message Auhentication Codes (MACs) — Part 1: Mechanisms using a block cipher. 1999.
4. ISO/IEC 10116 (2nd edition): Information technology — Security techniques — Modes of operation for an n-bit block cipher. 1997.
5. ISO/IEC 3rd CD 10116 (3rd edition): Information technology — Security techniques — Modes of operation for an n-bit block cipher (Commitee Draft). 2002.
6. ISO/IEC FDIS 10118-1: Information technology — Security techniques — Hash-functions — Part 1: General (Final Draft). 2000
7. V. Klima and T. Rosa. Side Channel Attacks on CBC Encrypted Messages in the PKCS#7 Format. Cryptology ePrint Archive, Report 2003/098, 2003.
8. S. Vaudenay. Security Flaws Induced by CBC Padding — Applications to SSL, IPSEC, WTLS. . . . In *Proc. EUROCRYPT'02*, LNCS Vol. 2332, pp. 534–545, 2002.