Padding Oracle Attacks on CBC-mode Encryption with Secret and Random IVs

Arnold K. L. Yau*, Kenneth G. Paterson and Chris J. Mitchell

Information Security Group,
Royal Holloway, University of London,
Egham, Surrey, TW20 0EX, UK
{a.yau,kenny.paterson,c.mitchell}@rhul.ac.uk

Abstract. In [8], Paterson and Yau presented padding oracle attacks against a committee draft version of a revision of the ISO CBC-mode encryption standard [3]. Some of the attacks in [8] require knowledge and manipulation of the initialisation vector (IV). The latest draft of the revision of the standard [4] recommends the use of IVs that are secret and random. This obviates most of the attacks of [8]. In this paper we consider the security of CBC-mode encryption against padding oracle attacks in this secret, random IV setting. We present new attacks showing that several ISO padding methods are still weak in this situation.

Keywords: padding oracle; CBC-mode; ISO standards; side channel

1 Introduction

Vaudenay [9] introduced the notion of padding oracle attacks on CBC-mode encryption. His work showed that several uses of CBC-mode encryption in well-known products and standards are potentially vulnerable to attack whenever the attacker can submit ciphertexts for decryption and has access to a side-channel which tells him only whether or not the corresponding plaintext is correctly padded. Canvel *et al.* [7] applied and extended the ideas of [9] to show that a particular implementation of SSL used to protect e-mail passwords could be attacked and the passwords extracted. Further padding methods were examined in [6].

In [8], Paterson and Yau examined the security of the ISO standard for CBC-mode encryption with respect to padding oracle attacks. The draft revision of the standard [3] analyzed in [8] proposes the use of padding methods from ISO/IEC 9797-1 [1] and ISO/IEC 10118-1 [2]. Paterson and Yau showed that several of these padding methods, when used with CBC-mode encryption, are vulnerable to padding oracle attacks[1]. The work of [8] highlights the dangers of "cutting-and-pasting" methods from one set of standards into another.

Partly as a consequence of the work of [8], a later draft of the revised ISO standard [4] omits all mention of padding methods. Additionally, it recommends that "integrity-protected secret" and "randomly chosen statistically unique" IVs be used. The motivation for using secret IVs given in [4] is "to prevent information leakage". The recommendation for random IVs is in-line with the formal security analysis of [5] which shows (in a sense that can be made precise) that CBC-mode is secure provided that the underlying block cipher is strong and that the IV is random. We also note that [4] allows the use of multiple IVs (called starting variables, or SVs in [4]) and interleaving of multiple cipher block chains; this allows for parallelism in encryption. We expect that in most applications, a single IV will be used, and this is the situation we focus on here.

The attack model in [8] assumes that the IV can be chosen by the attacker and is submitted to the padding oracle along with the ciphertext. To be successful, most of the attacks in [8] do in fact

* Supported by EPSRC and Hewlett-Packard Laboratories Bristol through CASE award 01301027.

[1] In fact, [8] claims padding oracle attacks against the second edition of ISO/IEC 10116, though this edition of the standard makes no mention of padding methods. Padding methods did not appear in draft revisions of the standard until the committee draft stage in the proposed 3rd edition of ISO/IEC 10116.

require the attacker to have knowledge of the IV and the ability to manipulate it. For this reason, the attacks in [8] would not apply to CBC-mode as defined in [4] if the padding methods of [1] and [2] were used and if the new recommendations to use secret, random IVs were followed. More specifically, the only attack in [8] that remains practicable is Attack 2 against padding method 3 of [2]. This attack on its own arguably has a small impact on the confidentiality of data because it works only against the last one or two blocks of a target ciphertext and recovers relatively few useful data bits.

Despite their omission from the draft ISO standard [4], padding methods are needed in order to fully specify the CBC-mode of operation. It is not unreasonable to assume that, in the absence of any other guidance, an implementer of CBC-mode according to [4] might borrow techniques from other ISO standards, as was indeed proposed in [3]. Here, we demonstrate that padding oracle attacks can still be effective against CBC-mode encryption even when IVs are secret and random. In particular, we show that several padding methods from [1, 2] are still weak even in this situation.

1.1 Attack Models

Before giving details of our attacks, we clarify the attack models under which these attacks will take place.

When IVs are secret and random, a variety of practical methods could be used to ensure the IVs are available to both encrypting and decrypting parties. For example, the IV could be encrypted using ECB-mode and prefixed to the ciphertext. Alternatively, a value V could be prefixed to the ciphertext and the IV generated by encrypting V using ECB mode. Or, as a third possibility, a pre-agreed list of IVs could be used and an index sent with ciphertexts to indicate which entry in the list was used as the IV. Because these approaches include information determining IVs along with ciphertexts, they allow the adversary to influence which IV is used by the padding oracle when decrypting, without the adversary necessarily knowing the *actual* value of the IV. In particular, they allow the adversary to force the oracle to re-use an old IV. We can model this kind of attack by assuming that, when submitting a ciphertext to the padding oracle, the attacker specifies an additional string I which in some way determines the IV used by the padding oracle. The contents of I will depend on the particular method used for establishing IVs: for example, in the case of encrypted IVs, I will simply be the encrypted IV, while in the case of a pre-established list, I would be an index in the list.

We expect that the above kind of approach for establishing secret, random IVs is most likely to be used in practice. But it is also conceivable that a second approach, in which no information at all about the IV is transmitted as part of ciphertexts, might be used. For example, the communicating parties may be able to maintain a synchronised counter and then obtain IVs by applying a keyed pseudo-random function to the counter. We also want to model attacks in this scenario, which presents a tougher attack environment to the adversary. We can do this by assuming that the padding oracle simply selects a fresh, random IV before every decryption and that no IV-related information is included in ciphertexts.

Thus in this paper, we will consider two slightly different attack models. In the first model, IVs are secret and random but are determined by additional information I available to the attacker and submitted to the oracle. In the second model, IVs are secret and random and the attacker has no control over the IV used by the padding oracle. Obviously, attacks in the second model are more powerful, but attacks in the first model already capture many likely practical situations.

1.2 Our Results

In Section 3.2, we introduce a new padding oracle attack against CBC-mode when used with padding method 3 of [1]. Our new attack applies for secret, random IVs in the first attack model. The new attack uses a set of auxiliary ciphertexts corresponding to plaintexts of different lengths as an aid to recovering the plaintext corresponding to a target ciphertext block. The complexity of the attack depends on the spread of lengths of the auxiliary ciphertexts; it can be as low as n queries to the padding oracle, where n is the block size.

We have been able to adapt the attacks of [8] against CBC-mode when used with padding method 3 of [2] to the secret and random IV setting without significant penalties on complexity or generality. These attacks are applicable in our second, tougher attack scenario. An attack applicable to any ciphertext block is presented in Section 4.2. This attack first constructs a valid ciphertext with the target block as the final block and then uses the attack of Section 4.3 to decrypt that block. The first phase requires, on average, roughly 2^{r-1} calls to the padding oracle. Here r is a parameter associated with the padding method. The attack of Section 4.3 is applicable to the final block of any ciphertext and is always efficient, requiring only $O(n)$ oracle queries to recover all the plaintext bits in the last block.

We note that our results do not contradict the results of [5], since the security model of [5] does not cater for the kind of side-channel information that a padding oracle provides to an attacker. We also note that all of our attacks are independent of the particular block cipher used.

Our attacks can be further developed to handle the situation where multiple IVs are in use. Again, we can obtain attacks against method 3 of [1] for multiple secret, random IVs in the first attack model. We can also find attacks against method 3 of [2] for multiple secret, random IVs in the second attack model. Since the modifications to our existing attacks are quite straightforward, we do not include the details in this paper. Nor have we analyzed the other padding methods from [1, 2] in the secret and random IV setting. Padding method 1 in both standards does not de-pad uniquely and is only useful when plaintexts have fixed or known lengths. We expect that padding oracle attacks may be possible against this method. As was noted in [6, 8], padding method 2 in the two standards seems to be largely immune to such a side-channel analysis and indeed makes a good candidate for recommendation as a padding method in the ISO standard for CBC-mode encryption.

2 Symbols and Notation

We largely use the same notation as in [8], with only one major difference. In [8], the first block of the ciphertext C_0 submitted to the padding oracle was taken to be the IV. Here, the attacker no longer submits the IV (since he does not know it), but he may or may not submit additional information I, depending on whether the attack is in the first or second attack model. Therefore in our new notation, the first block of the ciphertext will be the first encrypted block C_1, and, in making padding oracle queries, we will prepend additional information I to ciphertexts whenever appropriate. The context will make clear when this is being done.

For a detailed description of CBC-mode encryption, see [8, Section 2.2]. We summarise our other frequently used notation here for ease of reference.

C : ciphertext output after CBC-mode encryption; target ciphertext the attacker is trying to decrypt.
C' : ciphertext to be submitted to the padding oracle during an attack.
$d_K(Y)$: decryption of ciphertext block Y under key K.
D : unpadded data string to be CBC-mode encrypted.
$e_K(X)$: encryption of plaintext block X under key K.
I : information determining the IV in our first attack model.
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.
q : the number of blocks in data string P after padding.
VALID and INVALID: padding oracle responses to, respectively, correct and incorrect padding after receipt and decryption of ciphertext.
$X||Y$: the result of concatenation of strings X and Y.
$X \oplus Y$: 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 $(1 \leq j \leq q)$.
$X_{j,k}$: the k^{th} bit of the plaintext or ciphertext block X_j, $0 \leq k < n$.

3 Analysis of Padding Method 3 of ISO/IEC 9797-1

3.1 Review of Padding Method and Previous Attack

We reproduce the original text of the padding method from [1]:

> "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."

The attack in [8, Section 3.4] decrypts, one block at a time, arbitrary ciphertexts $C_1||C_2||\ldots||C_q$ that are padded using the above method. The attack makes repeated use of a padding oracle and has two phases.

The general case of the first phase applies to ciphertexts consisting of three or more blocks and was presented as Algorithm 9797-1-m3-get-L_D-general in [8]. The algorithm, when given a q-block valid ciphertext as input, finds L_D by manipulating the padding bits. The procedure requires the re-use of old IVs. Since we will use it in our new attack, we reproduce this algorithm here as Algorithm 1, with notation modified to reflect the use of additional information I to determine IVs. In the algorithm (which, in common with all the algorithms presented here, can be found in the Appendix), I denotes the IV-determining information that accompanied the target ciphertext.

The special case of the first phase applies to two-block ciphertexts and was presented as Algorithm 9797-1-m3-get-L_D-special in [8]. This algorithm does require the ability to directly manipulate bits in the IV and so does not apply in either of our attack models.

The second phase of the attack on Method 3 of ISO/IEC 9797-1 in [8, Section 3.4] is the actual decryption. Algorithm 9797-1-m3-decrypt in [8] returns the rightmost $n-1$ bits of a plaintext block but in so doing makes repeated updates to the IV. It is therefore unusable in our attack models. Algorithm 9797-1-m3-decrypt-last-bit in [8] returns the leftmost bit of a plaintext block. It is also unusable, since it requires a customised setting of the IV and a successful run of Algorithm 9797-1-m3-decrypt.

3.2 An Attack with Secret and Random IVs

We require some further mild assumptions in order to obtain an attack against padding method 3 of [1] with secret and random IVs. The attack is in our first attack model. We assume that, in addition to having a target ciphertext C which he wishes to decrypt, the attacker has also gathered a set of m auxiliary ciphertexts labelled C^1, C^2, ..., C^m, and associated IV-determining information $I^1, \ldots, I^m$. We write q_j for the number of blocks in ciphertext C^j and require that $q_j \geq 3$ for each j. The attacker can immediately use Algorithm 1 and the padding oracle to find the length L_j of each ciphertext C^j. We write $F_j = L_j \bmod n$. We require that the F_j be distinct and that no F_j is equal to zero. Without loss of generality, we can then write $1 \leq F_1 < F_2 < \ldots < F_m \leq n-1$. We also set $F_{m+1} = n$.

Notice that auxiliary ciphertexts with the required properties can easily be selected from a larger pool of ciphertexts. The auxiliary ciphertexts are not themselves decrypted in the course of the attack (though they can individually be used as target ciphertexts if their decryption is desired).

Our attack is presented in Algorithm 2 and described in words below.

The attack attempts to recover the plaintext block P_k matching the block C_k of the q-block ciphertext C. In fact, we are only able to extract the rightmost $n - F_1$ bits of P_k for each $k \geq 2$. The attack attempts to construct, for decreasing values of j, a valid q_j-block ciphertext whose last block is the target block C_k and whose first block is C_1^j. Because of the padding rule, such a ciphertext must correspond to a plaintext in which the last block P'_{q_j} consists entirely of '0's in the rightmost $n - F_j$ positions. By carefully controlling the values in the penultimate ciphertext block, we can ensure that only a relatively small number of trials is needed in order to achieve this for each successive value of j. Eventually, when $j = 1$, we have a ciphertext with last block C_k where the matching plaintext block P'_{q_1} has '0's in the rightmost $n - F_1$ positions. From this information and C_{k-1} it is easy to recover the rightmost $n - F_1$ positions of the original plaintext block P_k.

We now explain in more detail the operation of the attack. We begin by considering the rightmost $n - F_m$ positions. Consider submitting to the padding oracle a ciphertext of the form:

$$I^m, C_1^m || \underbrace{00\ldots0||\ldots||00\ldots0}_{q_m-3 \text{ blocks}} ||S||C_k$$

where S is a block taking on a random value in the rightmost $n - F_m$ positions. Because I^m determines the original IV used in obtaining C^m, block C_1^m indicates that $n - F_m$ '0' padding bits should be found in the last plaintext block, and hence the oracle will return VALID with a probability of $2^{F_m - n}$. An INVALID response indicates that another value of S should be tested. In the algorithm we simply use an increasing $(n - F_m)$-bit counter for this purpose. After an average of around 2^{n-F_m-1} and at most 2^{n-F_m} trials, we will obtain a VALID response. In this case, we learn that $S \oplus d_K(C_k)$ is equal to '0' in the rightmost $n - F_m$ positions.

Notice that from this information and knowledge of C_{k-1}, we could immediately recover the rightmost $n - F_m$ bits of P_k. However, we now preserve the successful value of S by setting $R = S$, and proceed to examine the rightmost $n - F_{m-1}$ bits. Now consider submitting to the padding oracle a ciphertext of the form:

$$I^{m-1}, C_1^{m-1} || \underbrace{00\ldots0||\ldots||00\ldots0}_{q_{m-1}-3 \text{ blocks}} ||S||C_k$$

where now S is a block taking on a random $(F_m - F_{m-1})$-bit value in positions $F_{m-1}, F_{m-1} + 1, \ldots, F_m - 1$, and equalling R in the rightmost $n - F_m$ positions. Now block C_1^{m-1} indicates that $n - F_{m-1}$ '0' padding bits should be found in the last plaintext block. By using R to set the rightmost $n - F_m$ bits of S, we have already arranged '0' bits in the rightmost $n - F_m$ positions of the last plaintext block. So the oracle returns a VALID response with probability $2^{-(F_m - F_{m-1})}$. Again, we use a counter to test the $2^{F_m - F_{m-1}}$ values in positions $F_{m-1}, F_{m-1}+1, \ldots, F_m - 1$. After an average of about $2^{F_m - F_{m-1} - 1}$ and at most $2^{F_m - F_{m-1}}$ trials, we will obtain a VALID response. In this case, we learn that $S \oplus d_K(C_k)$ is equal to '0' in the rightmost $n - F_{m-1}$ positions.

It is now straightforward to see how Algorithm 2 proceeds in this manner to eventually construct a valid ciphertext of the form:

$$I^1, C_1^1 || \underbrace{00\ldots0||\ldots||00\ldots0}_{q_1-3 \text{ blocks}} ||R||C_k$$

so that the corresponding last plaintext block contains '0' padding bits in the rightmost $n - F_1$ positions. Then a simple calculation shows that the rightmost $n - F_1$ bits of P_k are equal to the rightmost $n - F_1$ bits of the block $R \oplus C_{k-1}$.

3.3 Complexity and Impact

It takes an average of just over $2^{F_{j+1} - F_j - 1}$ oracle queries to obtain a VALID response and recover the bits at positions F_j to $F_{j+1} - 1$ of P_k. So the average number of oracle queries needed to recover

$n - F_1$ bits of plaintext is $\sum_{j=1}^{m} 2^{F_{j+1}-F_j-1}$. The worst-case complexity is twice this. Notice that when $F_1 = 1$ and $F_{j+1} - F_j = 1$ for each j, the average number of oracle queries needed to decrypt all but the leftmost bit of an n-bit block is just $n - 1$. In this case, at most two oracle queries are made for each j. In fact, since the outcome of the second oracle query is determined by the first, it is trivial to modify the attack so that $n - 1$ queries also represents the worst-case performance.

As an example, suppose the block size $n = 64$ and the data is byte-oriented. Suppose we can obtain 7 auxiliary ciphertexts whose lengths modulo 64 are 8, 16, 24, ..., 56. Then we have $m = 7$ and the average number of oracle queries needed to obtain 56 out of 64 plaintext bits is roughly 900. If the plaintext has some sort of predictability (e.g. ASCII characters making up an English text, or certain positions in a message within some known protocol), then the remaining byte might be easily guessed.

3.4 Limitations

Unfortunately, we have not succeeded in finding a method to extract the leftmost $F_1 \geq 1$ bits of the plaintext block P_k. The underlying reason is that, when the original data fits exactly within blocks, the default padding rule is to add no padding bits at all. This makes it difficult to set up a padding oracle test giving plaintext information.

Algorithm 1 can only find the contents of the length block for ciphertexts with at least 3 blocks. Whilst we are usually more interested in plaintext bits than length information, it would be convenient if Algorithm 2 could be applied to block C_1 of a two-block target ciphertext to extract the length information L_D. However, this would require knowledge of the IV (since block C_{k-1} is used at the last stage of our attack to recover the original plaintext bits). A lower bound on this length can be found by running Algorithm 2 on target block C_2 and finding the position of the rightmost one in P_2.

3.5 Comparison

The secret and random conditions on IVs have forced us to develop a completely new attack strategy against padding method 3 of [1]. The corresponding attack in [8] makes near-optimal use of the padding oracle and extracts all plaintext bits. To be efficient, our new attack requires the collection of auxiliary ciphertexts with a good spread of data lengths. There might be scenarios where this is unrealistic. Our new attack can never extract the leftmost data bits in each block. In the best case, it can recover all but the leftmost bit of plaintext using an optimal number of oracle queries (if we ignore the cost of finding the lengths of the auxiliary ciphertexts). Our attack cannot be extended to yield efficient attacks in the second attack scenario in which the adversary has no information about IVs at all. The reason is that the length information is placed in the first plaintext block – as a result, a random setting of the IV is almost certain to produce an INVALID response from the padding oracle.

In summary, in comparison to [8], the secret IV restriction has succeeded in increasing the complexity and decreasing the effectiveness of an attack. However, the attack is still feasible in many circumstances.

4 Analysis of Padding Method 3 of ISO/IEC 10118-1

4.1 Review of Padding Method and Previous Attacks

We reproduce below the original description of the padding method from [2], except that here, and throughout, we use n in place of L_1 to denote the block size:

> "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."

No encoding method (for L_D) is specified in the standard. We assume that base 2 encoding is used. Our attacks here work no matter which encoding method is used, though the attacker needs to know this method.

Using this padding method, the padding bits for data string D are appended in one of two ways:

Same-block Here $(L_D \bmod n) \leq (n-r-1)$. The last block of D has enough space after the last data bit to contain at least a single '1' bit and the r bits encoding L_D. The number of padding bits (including the length information) is between $r+1$ and $n-1$.

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

In [8], the authors presented two inter-dependent attacks against this padding method. The first attack creates a valid ciphertext with the target ciphertext block as the last block, while the second attack decrypts the last block of any ciphertext.

In more detail, Attack 1 of [8] (named "directed IV search") takes a ciphertext block C_k as input, and outputs a valid ciphertext of the form $IV'||C_k$. It operates by searching for an IV setting that produces a valid ciphertext. This ciphertext is then fed into Attack 2 for decryption. The need to vary the IV in a controlled manner means that the attack does not work when IVs are secret.

Attack 2 of [8] (named "attacking the last block(s)") takes as input a whole ciphertext and operates in two phases. In the first phase, it finds L_D; in some cases (including those resulting from Attack 1 of [8]) this involves changing bits in the IV. So this phase does not work in general for secret IVs. In the second phase plaintext bits are extracted. In the case of a same-block padded ciphertext, this second phase does not require any control over the IV. So it will continue to function with only minor modifications in the new setting. In the case of a cross-block padded ciphertext, the second phase can be used to speed up Attack 1 of [8]. This will fail with secret IVs, since Attack 1 of [8] requires their controlled modification.

Despite the failure of Attacks 1 and 2 of [8], a similar strategy can be followed and the original attacks can be modified to work in the tougher of our two attack scenarios. Analogues of Attacks 1 and 2 of [8] are presented in Sections 4.2 and 4.3.

4.2 Attacking an Arbitrary Ciphertext Block

The attack we present in this section attempts to decrypt an arbitrary block C_k of a ciphertext $C_1||C_2||\ldots||C_q$. In fact, our attacks only work for $k \geq 2$. It proceeds in two phases. In the first phase, a valid ciphertext is constructed having C_k as the final block. In the second phase, the attack of Section 4.3 is used to decrypt that final block. From this, P_k is easily found. Note that if C_q is the target block, then one should proceed directly to the attack of Section 4.3.

Phase 1: Constructing a valid ciphertext In this phase, we construct a valid three-block or four-block ciphertext having target block C_k as the last block. We aim for ciphertexts of these lengths because they simplify the second phase of the attack: we will see in Section 4.3 that ciphertexts containing $q \geq 3$ blocks are the easiest ones to deal with.

This phase splits into two cases, dependent on the value of r.

In the first case, we have $r < n$. The algorithm for this case is given in Algorithm 3 and is next described in words. The algorithm essentially submits three-block ciphertexts of the form:

$$\underbrace{00\ldots0}_{n}||R_2||C_k$$

to the padding oracle, for various values of R_2 chosen in such a way that at least one choice is guaranteed to produce a valid ciphertext. Our algorithm works no matter what IVs are used by the padding oracle. Note that we suppress any information I in submissions to the padding oracle here, and throughout this section, because we are operating in the second attack model.

In more detail, a counter i is used to determine the rightmost $r+1$ bits of R_2, while the leftmost $n-r-1$ bits are set to '0'. This effectively means that ciphertexts with all possible values of the length field in plaintext block P'_3 are submitted to the oracle as i runs between 0 and $2^r - 1$, the first half of the search space. At least one choice of i in this range is guaranteed to result in a VALID response from the oracle unless C_k and the selection of R_2 mean that the leftmost $n-r$ bits of P'_3 are all '0'. If this last case occurs, then considering all i between 2^r and $2^{r+1}-1$ ensures that one of the leftmost $n-r$ bits of P'_3 is a '1' and that at least one choice of i results in a VALID response. We will evaluate the average and worst-case complexity of this case of Phase 1 below.

In the second case, where $r = n$, a similar attack applies. We now submit four-block ciphertexts of the form:

$$\underbrace{00\ldots0}_{n}||R_1||R_2||C_k$$

to the padding oracle, where we try all possible settings of R_2 and the rightmost bit of R_1. We are then guaranteed to encounter a valid ciphertext after a maximum of 2^{n+1} oracle calls. The algorithm for this case is given in Algorithm 4; we analyse its complexity in detail below.

Phase 2: Decrypting C_k Once we have a valid three or four-block ciphertext, the attack of Section 4.3 can be applied to obtain the plaintext block P'_3 (or P'_4 in the four-block case) corresponding to the final block of C'. From P'_3, the original plaintext block P_k can be recovered using the relation $P_k = P'_3 \oplus R_2 \oplus C_{k-1}$. (A similar procedure applies for the four-block case.) As we shall see below, the attack of Section 4.3 is always efficient when attacking the last block of a three-block (or four-block) ciphertext. So this approach allows efficient extraction of P_k.

A little more detail is appropriate at this stage. We focus on the three-block case. The first phase of the attack in Section 4.3 finds the length L_D of the data encrypted in C'. If $L_D > 2n$, then the data is same-block padded, while if $L_D \leq 2n$ it is cross-block padded. If it happens that the data is cross-block padded, then all the bits in P'_3 (or P'_4 in the four-block case) are already determined and are of the form:

$$\underbrace{00\ldots0}_{n-r}\underbrace{(L_D)_2}_{r} \quad \text{or} \quad \underbrace{10\ldots0}_{n-r}\underbrace{(L_D)_2}_{r}.$$

So in this case no actual decryption step is needed to recover P_k. Notice that this case will always apply when $r = n$ or $r = n-1$. When the data is same-block padded, we must proceed to the second phase of the attack in Section 4.3. In the three-block case, this phase will efficiently recover the entire plaintext block P'_3 consisting of (in general) data bits, padding bits and length information. From P'_3, we can recover P_k using the relation $P_k = P'_3 \oplus R_2 \oplus C_{k-1}$. A similar procedure applies for the four-block case.

Complexity We begin by analyzing Phase 1 of the attack in the case where $r < n$. The analysis is complicated by the fact that Algorithm 3 might output a valid three-block ciphertext C' for which the corresponding plaintext $P' = P'_1||P'_2||P'_3$ is cross-block padded. This will have the effect of slightly lowering the average-case complexity when compared to the corresponding attack in [8]. Such a cross-block padded plaintext requires that blocks $P'_2||P'_3$ take the form:

$$P'_{2,0}P'_{2,1}\ldots P'_{2,L_D-n-1}\underbrace{10\ldots0}_{2n-L_D}||\underbrace{00\ldots0}_{n-r}\underbrace{(L_D)_2}_{r}$$

where each $P'_{2,i}$ can be either a '0' or '1' bit and $(2n-r) \leq L_D \leq (2n-1)$. There are r n-bit patterns (corresponding to the r possible values of L_D) for P'_3 that have the correct form. So the probability that Phase 1 produces cross-block padding is at most $r2^{r-n}$ as we vary the rightmost r bits of R_2 in Algorithm 3. Of course, such cross-block padding may never occur during the execution of Algorithm 3: given that R_1 and the decryption key K are fixed, there may be no choice of R_2 that produces the required bit pattern in $P'_2 = d_K(R_2) \oplus R_1$.

In any case, we see that there is a probability of at least $1 - 2^{r-n}$ that either there is a '1' somewhere in the leftmost $n-r$ bits of P'_3, or we obtain a cross-block padded ciphertext. In these cases, Algorithm 3 takes on average 2^{r-1} oracle calls. On the other hand, there is a probability of at most 2^{r-n} that the leftmost $n-r$ bits of P'_3 are all '0' and Algorithm 3 tries all 2^r possible settings for the rightmost bits of P'_3 without a VALID response. Algorithm 3 will then take on average a further 2^{r-1} oracle calls before obtaining a VALID response. A simple calculation now shows that the average number of oracle calls needed by Algorithm 3 is at most $2^{r-1} + 2^{2r-n}$, while in the worst-case it is 2^{r+1}. When r is small relative to n, the average-case complexity is dominated by the term 2^{r-1}.

Phase 1 of the attack in the case $r = n$ uses Algorithm 4. This algorithm uses on average 2^n oracle calls to obtain a VALID response and 2^{n+1} in the worst case.

Phase 2 uses the attack in Section 4.3 for the same-block padded case, which has a complexity of $O(n)$ oracle calls. So Phase 2 does not contribute significantly to the overall complexity required to decrypt a single block (unless r is very small).

Impact This attack applies to any ciphertext block C_k of a ciphertext $C_1||C_2||\ldots||C_q$, except for the first block C_1. It is not possible to decrypt C_1 because of the use of the relation $P_k = P'_3 \oplus C_{k-1} \oplus R_2$ at the end of the attack: this would necessitate an XOR with the secret IV. The attack recovers all n bits within the block and does so many orders faster than exhaustive search for many choices or r. When $r = n$ our attack is still better than exhaustive key search for block ciphers whose key size is greater than the block length. We restate the observation from [8] that the seemingly innocuous parameter r has unexpected implications for security.

Comparison This attack is an adaptation of Attack 1 in [8] to the second of our attack models, where IVs are secret, random and completely hidden from the adversary. These extra restrictions do not seem to be a major hindrance to the effectiveness of the attack. Specifically, the complexity of the attack has remained practically the same as the corresponding attack in [8], and, except for the first ciphertext block, the impact remains unchanged. The attack uses three-block or four-block ciphertexts instead of two-block ones when $r < n$; this is not expected to be of any practical significance.

4.3 Attacking the Last Block(s)

The attack we present in this section attempts to decrypt the last block C_q of a ciphertext $C_1||C_2||\ldots||C_q$. It is an adaptation of Attack 2 in Section 4.3 of [8] to the secret and random IV setting, and, like that attack, proceeds in two phases. Phase 1 determines the length L_D of the ciphertext, while Phase 2 will recover plaintext bits in the mixed block containing both padding and data bits. (If there is such a block, then it is unique.) Recall that, as well as being directly applicable to the last block C_q, our attack can also be used in conjunction with the attack in Section 4.2 to decrypt arbitrary ciphertext blocks.

Phase 1: Finding L_D This phase of our attack is derived from the corresponding phase in [8]. The case $q = 2$ requires special treatment and our methods fail completely when $q = 1$. We first examine the general case $q \geq 3$.

For ease of presentation we take $r \leq n - 2$, but Algorithm 5 handles all values of r. Here, in the same-block padded case, the last plaintext block P_q has the following format:

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

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 our q-block ciphertext, the rightmost position at which a data bit could ever reside is at $P_{q,n-r-2}$. Consider then submitting to the padding oracle the ciphertext:

$$C_1||C_2||\ldots||C_{q-1} \oplus \underbrace{00\ldots0}_{n-r-2}1\underbrace{00\ldots0}_{r+1}||C_q.$$

The oracle will return either:

- VALID, meaning the padding has not been disturbed so the bit flipped in P'_q by modifying C_{q-1} is a data bit. Since this bit is at the rightmost possible data bit position, we can deduce that the data length L_D equals $(q-1)n + n - r - 1 = qn - r - 1$.
- or 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.

We can generalise the above observation about $P_{q,n-r-2}$ to produce Algorithm 5, a binary search algorithm to find L_D. In this algorithm, we initialise two pointers l and u at the extremities of the possible padding range and modify the ciphertext so as to invert the plaintext bit that lies in the middle position $h := \lfloor (l + u)/2 \rfloor$ of the range. We then submit the ciphertext to the oracle. A VALID response means the start of the padding is to the right of this test bit so we set the lower pointer l to the position $h + 1$, whereas INVALID indicates it is to the left and we set the upper pointer u to h. We must then reset the test bit before proceeding to the next test. This process is repeated until the upper and lower pointers coincide, at which point they indicate the rightmost data bit. It is then easy to determine L_D. Clearly, the algorithm makes roughly $\log_2 n$ calls to the padding oracle and so is efficient.

This completes our discussion of the general case where $q \geq 3$. Next we focus on the case $q = 2$. This case requires special treatment because setting up a binary search as above requires the ability to modifiy plaintext bits in the whole range of padding positions, which in this case includes those in the rightmost r positions of the plaintext block P_1. This in turns necessitates the ability to modify bits in the corresponding positions in the IV, which is not possible in the setting of secret and random IVs.

Our solution, presented in Algorithm 6, is to perform a binary search over the restricted range of those padding positions in the second (and last) plaintext block P_2. This is done by initializing the lower and upper pointers to n and $2n + r - 1$ respectively. If the search finishes pointing to any position between $P_{2,1}$ and $P_{2,n-r-1}$ then this indicates the actual leftmost padding position from which L_D can be determined. On the other hand, if the search ends pointing at $P_{2,0}$, then we can deduce that the bit at that position is a padding bit and hence the boundary is somewhere to the left of that position. From this we can deduce that the plaintext block P_2 consists only of padding bits and encoded length information, and that $L_d \leq n$. We could go further and deduce most of the contents of block P_2, but these bits are not usually of much interest to the attacker. In this case, we cannot continue with the attack.

We note that this $q = 2$ version of the length-finding algorithm is never invoked by the attack in Section 4.2 (unless C_2 is the last block and happens to be the initial target).

Finally we consider the case $q = 1$. Here we are not able to find L_D by performing any kind of search for the data/padding boundary since this would require manipulating the IV. Thus our methods fail in this case.

Phase 2: Decrypting We assume that $q \geq 2$ and that L_D has been successfully obtained from Phase 1. This will always be the case for $q \geq 3$ and often the case for $q = 2$. Same-block and cross-block padded messages are treated differently; recall that knowledge of L_D indicates with which case the attacker is faced.

Decrypting: Same-block Recall the structure of the last plaintext block P_q: t unknown data bits, followed by p padding bits in the form $10\ldots0$ and finally r bits encoding the data length L_D. The only bits remaining to be found are the t data bits. We can assume that $t \geq 1$ and recover these as follows. Consider submitting to the oracle the ciphertext $C' = R||C_q$ where:

$$R = C_{q-1} \oplus \underbrace{00\ldots0}_{n-r}\underbrace{(L_D)_2}_{r} \oplus \underbrace{00\ldots0}_{t}\underbrace{10\ldots0}_{p}\underbrace{(n+t-1)_2}_{r}.$$

This ciphertext is constructed in such a way that, after decryption to obtain plaintext $P'_1||P'_2$, the length block in P'_2 encodes the length $n+t-1$, while the p padding bits are modified to be all '0's. Moreover, data bits are copied intact from P_q to P'_2, so that $P_{q,i} = P'_{2,i}$ for $0 \leq i < t$. From the construction of C', we see that the oracle will output VALID if and only if $P'_{2,t-1} = 1$. Since we have $P_{q,t-1} = P'_{2,t-1}$, we can obtain the last data bit of block P_q.

This idea can be extended to recover all t data bits in P_q in a similar manner: we reduce the length field in P'_2 one step at a time whilst fixing the data in all recovered bit positions to be '0' so that they become part of a valid padding. A single bit of P'_2 and hence of P_q is revealed at each iteration, until all the data bits in P_q are recovered. This procedure is given in detail in Algorithm 7. Note that the algorithm makes use of the function $\bar{\Omega}$ defined by:

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

Note that $\bar{\Omega}$ is the complement of the function Ω in [8].

Decrypting: Cross-block For cross-block padded plaintexts with $q \geq 3$ blocks, P_q is determined completely by L_D and the padding. However, the padding often extends into the penultimate plaintext block P_{q-1} and we can exploit this fact when decrypting block C_{q-1}.

Suppose $t = L_D \bmod n$ and $t \neq 0$. Then $u = n - t$ bits of padding of the form $\underbrace{10\ldots0}_{u}$ are present in P_{q-1}. We show how to decrypt C_{q-1} using the attack in Section 4.2, but with a speed-up factor of 2^{u-1}. Consider ciphertexts of the form $C' = 00\ldots0||R_2||C_{q-1}$ where:

$$R_2 = C_{q-2} \oplus \underbrace{00\ldots0}_{t}\underbrace{10\ldots0}_{u} \oplus \underbrace{00\ldots0}_{n-r}\underbrace{(3n-r-1)_2}_{r}.$$

Upon decryption, this ciphertext will produce a plaintext block P'_3 of the form:

$$P'_{3,0}P'_{3,1}\ldots P'_{3,t-1}y_0y_1\ldots y_{u-1}$$

where $y_0y_1\ldots y_{u-1}$ are the u least significant bits of the binary encoding of the length field $3n-r-1$. Now it is straightforward to see that running through all 2^{r-u+1} settings of the $r-u+1$ bits immediately to the left of the rightmost u bits (by varying the relevant bits of R_2) will ensure that at least one valid three-block ciphertext C' is obtained. Naturally, after obtaining such a valid C', we can apply the attack of this section again, now using C' as the input ciphertext. Eventually, that attack will output a candidate P'_3 for the decryption of block C_{q-1} in ciphertext C'; from this we can deduce the decryption P_{q-1} of C_{q-1} in the original ciphertext C using the relation $P_{q-1} = P'_3 \oplus R_2 \oplus C_{q-2}$.

This strategy takes on average about 2^{r-u} oracle calls which is roughly a fraction $2^{-(u-1)}$ of the number of oracle calls needed on average for the corresponding attack in Algorithm 3 without the knowledge of the u padding bits.

Unfortunately this strategy does not work for two-block cross-block padded ciphertexts in our attack model, because the very last step would need to use IV in place of C_{q-2}.

Complexity For $q \geq 3$, Phase 1 of the attack takes roughly $\log_2 n$ oracle calls to find the data length L_D. For same-block padded plaintexts, Phase 2 then takes one call per bit for decrypting. So to recover the t data bits in the last block, $t + \log_2 n$ oracle calls are required. For cross-block padded plaintexts, the block P_q is completely determined by L_D. Then Phase 2 needs on average around 2^{r-u} oracle calls to recover the whole of the penultimate plaintext block P_{q-1}. Here u is the number of known padding bits in P_{q-1} and we have ignored the comparatively small cost of running the length-finding and last-block decryption algorithms of this section.

For two-block ciphertexts, Phase 1 will take on average $\log_2(n - r)$ oracle calls to find either the actual value of L_D or to find that $L_D \leq n$. In the former case, the complexity of Phase 2 is exactly as above. In the latter case, the data is cross-block padded but we are not able to recover the penultimate plaintext block. Phase 1 of the attack is not successful for single-block ciphertexts and no data bits can be extracted using our attack in this case.

It is important to note that, even though the two attacks presented here and in Section 4.2 are inter-dependent, there is no possibility of the attack entering an infinite loop. This is not difficult to show.

Impact The attack is highly efficient (in terms of oracle access) at extracting plaintext bits in 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 the attack in Section 4.2.

Comparison One impact of assuming that IVs are secret and random on the attack in this section is that Phase 1 of the attack is prevented from determining the exact data length of single-block ciphertexts, and two-block ones when the plaintext is cross-block padded. This, in turn, stops us from extracting any data bits in these cases. This is in contrast to the corresponding cases in [8], where the ability to manipulate the IV can be used to advantage.

The complexity of the two phases remains unchanged when compared to the corresponding attack in [8] ($\log_2 n$ oracle calls to find L_D and one oracle call per data bit extracted for same-block padding). Short ciphertexts, typically two or three blocks long, are used throughout, so there is little or no message expansion.

5 Conclusions

We have shown that the use of IVs that are secret and random does not prevent padding oracle attacks on CBC-mode encryption. We have shown this to be the case in the context of two padding methods previously analyzed in [8]. The use of secret, random IVs required us to develop new ideas and to extend the analysis of [8]. The new attacks are, at best, of roughly equal complexity to those of [8] and the assumptions we have made to obtain attacks seem reasonable. The attacks recover most, if not all, plaintext bits many orders of magnitude faster than exhaustive key search.

The 2004 FCD text for the 3rd edition of ISO/IEC 10116 [4], which supersedes [3], contains new text regarding padding methods in Clause 5 (Requirements). It now reads

> *...Padding techniques...are not within the scope of this International Standard, and throughout this standard it is assumed that any padding, as necessary, has already been applied.*

This effectively off-loads the responsibility of choosing a padding method to the implementor of this standard (if it is published with the text as it stands). In our view, not specifying a padding method at all has the potential to be even more dangerous than specifying a method that is known to be weak against certain attack types. After all, there is no guarantee that an implementor will not choose a method that falls to some even more realistic form of attack. Methods that appear to resist padding oracle attacks have been analysed [6]. For example, padding method 2 of [1], in which the plaintext is padded with a single '1' and as many '0's as are necessary to complete a block, seems like a good candidate. We currently know of no reason not to recommend it for use. We argue that the more complete and unambiguous a specification is, the smaller the chance for insecure approaches to be taken by an implementor.

Finally, we wish to repeat the point made in [6, 8] that padding oracle attacks can be easily thwarted by the proper use of strong integrity checks. It is now widely held that encryption should be accompanied by a data integrity mechanism whenever feasible and appropriate. Of course there are situations (for example, constrained environments) where the use of a MAC algorithm in addition to encryption is not possible. In these scenarios, the careful selection of a padding method and the avoidance of padding oracles in implementations is of paramount importance.

References

1. *ISO/IEC 9797-1: Information technology — Security techniques — Message Authentication Codes (MACs) — Part 1: Mechanisms using a block cipher*, 1999.
2. *ISO/IEC 10118-1 (2nd edition): Information technology — Security techniques — Hash-functions — Part 1: General*, 2000.
3. *ISO/IEC 2nd CD 10116 (revision): Information technology — Security techniques — Modes of operation for an n-bit block cipher*, 2002. (Second committee draft of proposed 3rd edition of the standard).
4. *ISO/IEC FCD 10116 (2nd edition): Information technology — Security techniques — Modes of operation for an n-bit block cipher*, 2004. (Final committee draft of proposed 3rd edition of the standard).
5. M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A Concrete Analysis of Symmetric Encryption: Analysis of the DES Modes of Operations. In *38th IEEE Symposium on Foundations of Computer Science*, pages 394–409. IEEE, 1997.
6. J. Black and H. Urtubia. Side-Channel Attacks on Symmetric Encryption Schemes: The Case for Authenticated Encryption. In *Proceedings of the 11th USENIX Security Symposium, San Francisco, CA, USA, August 5-9, 2002*, pages 327–338. USENIX, 2002.
7. B. Canvel, A. Hiltgen, S. Vaudenay, and M. Vuagnoux. Password Interception in a SSL/TLS Channel. In D. Boneh, editor, *Advances in Cryptology — CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 583–599. Springer-Verlag, 2003.
8. K.G. Paterson and A. Yau. Padding Oracle Attacks on the ISO CBC Mode Padding Standard. In T. Okamoto, editor, *Topics in Cryptology — CT-RSA 2004*, volume 2964 of *Lecture Notes in Computer Science*, pages 305–323. Springer-Verlag, 2004.
9. S. Vaudenay. Security Flaws Induced by CBC Padding — Applications to SSL, IPSEC, WTLS .... In L. Knudsen, editor, *Advances in Cryptology — EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 534–545. Springer-Verlag, 2002.

Appendix

We present here pseudo-code for the various algorithms developed in the text.

Algorithm 1

Input: $I, C_1||C_2||\dots||C_q$
Output: L_D

function 9797-1-M3-GET-L_D-GENERAL
    $l := 0$
    $u := n - 1$
    repeat
        $h := \lceil (l + u)/2 \rceil$
        $C_{q-1,h} := C_{q-1,h} \oplus 1$
        if ORACLE$(I, C_1||C_2||\dots||C_q) = \mathsf{VALID}$ **then**
            $l := h$
        else
            $u := h - 1$
        end if
        $C_{q-1,h} := C_{q-1,h} \oplus 1$
    until $l = u$
    return $L_D := (q-1)n + l + 1$
end function

Algorithm 2

Input: auxiliary ciphertexts $C^1, C^2, \dots, C^m$, IV-determining information $I^1, I^2, \dots, I^m$, length information $q_1, \dots, q_m$ and $F_1, \dots, F_m$, target ciphertext blocks C_{k-1}, C_k
Output: rightmost $n - F_1$ bits of P_k

function 9797-1-M3-DECRYPT
    $R := \underbrace{00\dots0}_{n}$
    $F_{m+1} := n$
    for $j := m$ to 1 **do**
        $i := -1$
        repeat
            $i := i + 1$
            $S := R \oplus \underbrace{00\dots0}_{F_j} \ \underbrace{(i)_2}_{F_{j+1}-F_j} \ \underbrace{00\dots0}_{n-F_{j+1}}$
        until ORACLE$(I^j, C^j_1||\underbrace{00\dots0||\dots||00\dots0}_{q_j-3 \text{ blocks}}||S||C_k) = \mathsf{VALID}$
        $R := R \oplus \underbrace{00\dots0}_{F_j} \ \underbrace{(i)_2}_{F_{j+1}-F_j} \ \underbrace{00\dots0}_{n-F_{j+1}}$
    end for
    return rightmost $n - F_1$ bits of $R \oplus C_{k-1}$
end function

Algorithm 3

Input: C_k, r, n
Output: A valid three-block ciphertext, the last block of which is C_k

Require: $1 \le r < n$

function 10118-1-M3-GENERAL(C_k, r, n)
  $R_1 := \underbrace{00\ldots0}_{n}$
  $R_2 := \underbrace{00\ldots0}_{n}$
  $i := 0$
  while ORACLE($R_1||R_2||C_k$) = INVALID **do**
    $i := i + 1$
    $R_2 := \underbrace{00\ldots0}_{n-r-1}\underbrace{(i)_2}_{r+1}$
  end while
  return $R_1||R_2||C_k$
end function

Algorithm 4

Input: C_k, r, n
Output: A valid four-block ciphertext, the last block of which is C_k

Require: $r = n$

function 10118-1-M3-SPECIAL(C_k, r, n)
  $R_1 := \underbrace{00\ldots0}_{n}$
  $R_2 := \underbrace{00\ldots0}_{n}$
  $i := 0$
  while ORACLE($\underbrace{00\ldots0}_{n}||R_1||R_2||C_k$) = INVALID **do**
    $i := i + 1$
    if $i = 2^r$ **then**
      $i := 0$
      $R_1 := \underbrace{00\ldots01}_{n}$
    end if
    $R_2 := \underbrace{(i)_2}_{n}$
  end while
  return $\underbrace{00\ldots0}_{n}||R_1||R_2||C_k$
end function

Algorithm 5

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

Require: $q \geq 3$

function 10118-1-M3-FIND-L_D-GENERAL($C_1||C_2||\dots||C_q$, n, r)
  $C := C_1||C_2||\dots||C_q$
  $l := (q-2)n + n - r$
  $u := (q-1)n + n - r - 1$
  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
      $u := h$
    end if
    $C_{\lfloor h/n \rfloor, h \bmod n} := C_{\lfloor h/n \rfloor, h \bmod n} \oplus 1$
  until $l = u$
  return $L_D := l$
end function

Algorithm 6

Input: $C_1||C_2, n, r$
Output: L_D or "Plaintext length at most n"

function 10118-1-M3-FIND-L_D-SPECIAL($C_1||C_2$, n, r)
  $C := C_1||C_2$
  $l := n$
  $u := 2n - r - 1$
  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
      $u := h$
    end if
    $C_{\lfloor h/n \rfloor, h \bmod n} := C_{\lfloor h/n \rfloor, h \bmod n} \oplus 1$
  until $l = u$
  if $l > n$ **then**
    return $L_D := l$
  else
    return "Plaintext length at most n"
  end if
end function

Algorithm 7

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

Require: L_D indicates that the plaintext is same-block padded

function 10118-1-M3-DECRYPT(L_D, C_{q-1}, C_q, r, n)
  $t := L_D \bmod n$
  $p := n - r - t$
  $R := C_{q-1} \oplus \underbrace{00\dots0}_{t}\underbrace{10\dots0}_{p}\underbrace{(L_D)_2}_{r} \oplus \underbrace{00\dots0}_{n-r}\underbrace{(n+t)_2}_{r}$
  for $j := t-1$ to 0 **do**
    $R := R \oplus \underbrace{00\dots0}_{n-r}\underbrace{(n+j+1)_2}_{r} \oplus \underbrace{00\dots0}_{n-r}\underbrace{(n+j)_2}_{r}$
    $P_{q,j} := \bar{\Omega}(R||C_q)$
    $R_j := R_j \oplus P_{q,j}$
  end for
  return $P_q := P_{q,0}P_{q,1}\dots P_{q,t-1}\underbrace{10\dots0}_{p}\underbrace{(L_D)_2}_{r}$
end function