Side Channel Attacks on CBC Encrypted Messages in the PKCS#7 Format

Vlastimil Klíma [1] and Tomáš Rosa [1,2]

{vlastimil.klima, tomas.rosa}@i.cz

[1] ICZ a.s., V Olšinách 75, 100 97 Prague 10, Czech Republic
[2] Department of Computer Science and Engineering, Faculty of Electrical Engineering,
Czech Technical University in Prague, Karlovo náměstí 13, 121 35 Prague 2, Czech Republic

Abstract

Vaudenay has shown in [5] that a CBC encryption mode ([2], [9]) combined with the PKCS#5 padding [3] scheme allows an attacker to invert the underlying block cipher, provided she has access to a valid-padding oracle which for each input ciphertext tells her whether the corresponding plaintext has a valid padding or not. Having on mind the countermeasures against this attack, different padding schemes have been studied in [1]. The best one is referred to as the ABYT-PAD. It is designed for byte-oriented messages. It removes the valid-padding oracle, thereby defeating Vaudenay's attack, since all deciphered plaintexts are valid in this padding scheme. In this paper, we try to combine the well-known cryptographic message syntax standard PKCS#7 [8] with the use of ABYT-PAD instead of PKCS#5. Let us assume that we have access to a $\text{PKCS\#7}_{\text{CONF}}$ oracle that tells us for a given ciphertext (encapsulated in the PKCS#7 structure) whether the deciphered plaintext is correct or not according to the PKCS#7 (v1.6) syntax. This is probably a very natural assumption, because applications usually have to reflect this situation in its behavior. It could be a message for the user, an API error message, an entry in the log file, different timing behavior, etc. We show that access to such an oracle again enables an attacker to invert the underlying block cipher. The attack requires single captured ciphertext and approximately 128 oracle calls per one ciphertext byte. It shows that we cannot hope to fully solve problems with side channel attacks on the CBC encryption mode by using a “magic” padding method or an obscure message-encoding format. Strong cryptographic integrity checks of ciphertexts should be incorporated instead.

Keywords: CBC, symmetrical encryption, padding, ABYT-PAD, ABIT-PAD, PKCS#7, cryptanalysis, side channel attack, confirmation oracle.

1 Introduction

Vaudenay's attack [5] has been further studied in [1], where several extensions of and countermeasures were proposed. The only effective padding types defined there were referred to as the ABYT-PAD (arbitrary-tail padding) for byte-oriented messages and the ABIT-PAD for bit-oriented messages. The ABYT-PAD was defined in the following way; let the last byte (bit) of the message be X; pick an arbitrary distinct byte (bit) Y and add one or more bytes (bits) Y as needed to the end of the message. The receiver reads the last byte (bit) of the plaintext and removes all successive bytes (bits) which are the same as Y from the end of the plaintext. The main benefit of this padding is that there is no incorrectly padded plaintext. Therefore, it is no longer possible to use Vaudenay's attack based on a valid-padding oracle, because such an oracle doesn't tell us any new information (its output has a zero entropy overall). However, even when using these methods, there are a lot of other vulnerabilities and possible attacks. In [1] it was underlined that such attacks are pervasive when the integrity of ciphertexts is not guaranteed. As an example, the authors of [1] designed a so-called "cryptographic relay" (a device), which consists of two cryptographic schemes. The first one uses the substantial padding scheme described above, while the second uses a length-preserving scheme (e.g. CTR mode). The device decrypts the ciphertext coming from the first scheme and then re-encrypts it using the second scheme. Because the second scheme does not hide the original plaintext length, it is possible to use this information for an attack on the first scheme. In this paper, we show that it is not necessary to design such an abstract scheme to carry out a successful attack. We simply combine the well-known cryptographic message syntax standard PKCS#7 [8] with the use of the ABYT-PAD padding scheme instead of the PKCS#5 padding. Let us assume that we have access to an oracle $\text{PKCS\#7}_{\text{CONF}}$ which tells us for a given ciphertext (encapsulated in the PKCS#7 structure) whether the decrypted plaintext is correct or not according to the PKCS#7 syntax. This is probably a very natural assumption, because applications usually have to reflect this situation in its behavior. It could be a message for the user, an API error message, an

entry in the log file, different timing behavior, etc. We show that by having access to such an oracle an attacker can invert the underlying block cipher for a particular arbitrary key, thereby deciphering the secret encrypted message. This attack requires a single captured ciphertext belonging to the key and approximately 128 oracle calls per one ciphertext byte. This kind of attack can be extended to other padding schemes (i.e. ABIT-PAD, etc.). Surprisingly, our attack is allowed by those PKCS#7 (v1.6) properties that are planned to improve version v1.5. According to the new version (v1.6) there are not only data-octets (bytes) encrypted (as in the previous version), but it also encrypts the length-octets and type-octets. The main idea of the attack is to carefully combine the changes at the beginning and the end of the encrypted message. Then we use the PKCS#7-confirmation oracle, which tells us whether the change was correct or not in the sense of the PKCS#7 v1.6 format. This information thwarts the original good property of the ABYT-PAD scheme that all deciphered plaintexts are valid. The "improvement" of version 1.5 of the standard thus brought a new kind of attack. It follows that an improvement that is good under a local estimation may turn out to be a bad choice under a broader context evaluation. On the other hand, we do not express the opinion that our attack is a problem of the step of moving from the PKCS#7 v1.5 to the PKCS#7-v1.6 standard. The conclusion of our paper is that, just like the area of asymmetrical cryptography, we cannot hope to fully solve these problems with side channel attacks just by using a "magic" padding method or an obscure message-encoding format.

The rest of the paper is organized as follows; firstly we introduce the necessary notation and description of the PKCS#7 format (§2) and the confirmation oracle $PKCS\#7_{CONF}$ (§3). An attack is then presented in §4. In §5 we summarize the complexity of the attack and its extensions. Countermeasures are presented in section §6 and a conclusion is made in §7.

2 Preliminaries

2.1 Notation

We will denote CT the ciphertext C without an initializing value (IV), thus C = (IV, CT). We will assume the block cipher, which works over n-bytes blocks, where n is a positive integer. We will denote $E_K(B)$ and $D_K(B)$ enciphering and deciphering of a data block B under a secret (symmetric) key K. We will denote $ENC\text{-}CBC_K$ and $DEC\text{-}CBC_K$ enciphering and deciphering of the whole plaintext and ciphertext in the CBC mode, respectively. To be consistent with the ASN.1 notation, we will talk about octets with the assumption that the term octet means the same as byte in this paper. We will use "BIG ENDIAN" ordering of bytes inside the data block, i.e. b_1 will be the most significant byte in the n-byte block $B = (b_1, ..., b_n)$. Hexadecimal numbers are denoted using the prefix 0x, the exclusive OR operation is denoted $\oplus$. We will use it for bits, bytes and blocks of bytes. If the blocks are indexed, we use the second index to pick the byte from it, for instance $CT_{s,n}$ is the last byte of the block CT_s. Note that $C_1 = IV$ and $CT_1 = C_2$ is the first “payload” block corresponding to the plaintext P_1 in our notation.

2.2 Description of PKCS#7 data structures, padding scheme and our assumptions

2.2.1 PKCS#7

As stated in [4], the standard PKCS#7 describes the general syntax for cryptographically protected data, e.g. data which is encrypted, digitally signed, etc. Data syntax is described using the notation ASN.1 [6]. It admits recursion, so that one envelope (c.f. [4]) can be nested inside another. The values produced according to this standard are intended to be BER-encoded [7], which means that the values would typically be represented as octet strings. The syntax is general enough to support many different content types. PKCS#7 defines the six following ones: data, signed data, enveloped data, signed-and-enveloped data, digested data, and encrypted data. These content types are defined using the notation ASN.1 and they are used in a number of applications, programs, protocols etc. For instance, we may take the banking protocol SET or the standard for a secure electronic mail S/MIME. We will concentrate on the content type "enveloped data". It contains the data (a binary content) encrypted by the symmetric encryption key, where this key is transmitted using a public-key algorithm. We will assume that the data is as usually encrypted by a block cipher in the CBC mode.

2.2.2 ASN.1 encoding

Before encryption, the data being encrypted is ASN.1 encoded first, usually by using the BER/DER encoding [7]. In most cases encoding consists of adding some type-octets together with some length-octets before the data

itself. The type-octets define the type of data (type of data structures) and the length-octets define the length of the data. This length means the length of the original data, which follows after the length-octets. The triplet (type-octets, length-octets, data-octets) is then padded and encrypted in the CBC mode.

2.2.3 Data types

There are a lot of data types that can be used in various applications for data being encrypted. These data types are usually publicly defined and their octet codes are thus well known for a concrete application. Probably the most often used one is the data type OCTET STRING, encoded as one octet with the value 0x04. Without loss of generality, we may assume that the data type is OCTET STRING for the purpose of our side channel attack description.

2.2.4 Data length

Similarly, the length of the original data being encrypted is encoded into one or more length-octets. If the data length is less than 128, then there is only one length-octet the value of which states exactly the data length. If the data length is higher than or equal to 128, then it is encoded as a 2 to 128 octets long string of the length-octets. In this case the first octet has the most significant bit set to 1 and its remaining bits express the number of the following length-octets. The remaining length-octets then express the data length in the integer base 256. For instance, if the length of the original data is less than 64 KBytes, then the first length-octet is 0x82 and the following successive two octets give the particular length in the base 256. The number of length-octets doesn't play any important role in our side channel attack. For the sake of simplicity, we will assume that the data being encrypted has only one length-octet.

2.3 Encryption in the PKCS#7 version 1.5 and 1.6

Let us have L bytes of data. We will assume their ASN.1 encoding as (0x04, L, data). Version 1.5 of PKCS#7 [4] was designed to enable PEM compatible formats, but this brought some inconveniences for applications. It required applications to "dip under" the ASN.1 and deal directly with the BER/DER encoding of data (signing, encryption). Such BER/DER "hacking" made it difficult for users of ASN.1 compilers to generate encoding/decoding subroutines, because the head of the data and the data itself were processed separately. Having accepted the ascendancy of S/MIME over PEM, and the desirability of avoiding low-level "hacking" of the BER/DER encoding, version 1.6 of PKCS#7 [8] modified the processing rules to operate on the entire BER/DER encoding of the data. In particular, this means that data-octets are encrypted together with the type-octets and length-octets as one binary stream (type-octets, length-octets, data-octets). It is anticipated that version 2 of PKCS#7 will also incorporate this change.

2.4 Using ABYT-PAD padding scheme

Before the encryption, we need to append an appropriate padding. Finally, we encrypt the quadruple (0x04, L, data, padding) in the CBC mode. Here we assume that the padding scheme used is ABYT-PAD [1], however the attack can be easily extended to some other schemes (for instance ABIT-PAD). According to [1], ABYT-PAD is defined in the following way.

Let the last byte of the plaintext be X and pick an arbitrary distinct byte Y. We add one or more bytes of Y as needed to the end of the plaintext in such a way that the new plaintext length is an integer multiple of n. Emphasize that at least one byte of Y must be appended. In the case of an empty plaintext, Y can be an arbitrary value. If the padding is not more than n octets long, we talk about a "short" ABYT-PAD padding. In the case of an unlimited padding length, we will talk about a "long" ABYT-PAD padding. The receiver reads the last byte of the plaintext and removes all the successive identical bytes from the end of the plaintext. In the following text, we will assume that the padding is the "short" one, i.e. the number of padding bytes has to be from 1 to n. However, it will become clear that it is easily possible to extend the attack to the long ABYT-PAD padding. We also assume that the symmetric key and the underlying block cipher are always the same during our attack.

3 Confirmation oracle PKCS#7$_{\mathbf{CONF}}$

Assume that a sender encrypts messages using: the PKCS#7 data type "enveloped data", a block cipher in the CBC mode, and the ABYT-PAD padding scheme. The encrypted data is ASN.1 encoded, padded, and encrypted

using a random symmetric key, and then the ciphertext CT is put in a specific place in the highly structured data block "enveloped data". The symmetric key is then encrypted using a public-key scheme and is also put in its specific place in the "enveloped data". The initialization value (IV) is saved in this structure, outside the ciphertext CT (in data type "Content Encryption Algorithm Identifier"), as well. In particular, we note that it is possible to change IV without disturbing any other content of the "enveloped data". Furthermore, we assume that when the attacker changes the length and the content of the ciphertext CT later on, she will also eventually change the appropriate length octets of all "higher" structures containing it. Thus, the changed ciphertext CT will be correctly encapsulated in the PKCS#7 structure. In the following, we will focus only on the receiver's dealing with the IV and CT items in the block "enveloped data".

Definition (PKCS#7 confirmation oracle PKCS#7$_{CONF}$(C)).

Let us have the ciphertext C = (IV, CT).

We conjecture a PKCS#7$_{CONF}$ confirmation oracle, PKCS#7$_{CONF}$(C): C → (ANSWER = "OK/BAD"), to encapsulate the following procedure:

1. *P = DEC-CBC$_K$(C); the plaintext P is obtained by deciphering the ciphertext C in the CBC mode under the symmetric key K*
2. *Remove the padding from the plaintext P; the resulting message is denoted as M.*
3. *Parse M according to PKCS#7:*
   - *Check the type-octets of M; according to the assumptions in §2 we expect one concrete value to be here - 0x04. If it is not here, an error has occurred.*
   - *Check the length-octets of M; we expect one length octet to be here (L), furthermore, L must be equal to the length of M, obtained in step 2. If it is not, an error has occurred.*
4. *If the two previous checks in step 3 are successful, the answer of PKCS#7$_{CONF}$(C) is "OK"; otherwise it is "BAD".*

We note that no error messages are expected to occur in steps 1 and 2. For the sake of convenience, we will use the symbols O and PKCS#7$_{CONF}$ when referring to the PKCS#7$_{CONF}$ confirmation oracle interchangeably.

4 Attack description

Let the attacker intercept a valid ciphertext C = (IV, CT_1, CT_2, ... CT_s), s ≥ 1, and let (P_1, P_2, ... P_s) denote the corresponding plaintext. We will show that she can then compute X = D_K(Y) for any arbitrary chosen Y obviously implying that she can decipher the whole intercepted ciphertext C.

Recall that we are working with short messages (one length-octet) with the short ABYT-PAD padding. However, we will show in section 5 how to modify the attack for longer messages and longer ABYT-PAD padding. The attack has several steps.

The first step is to be carried out only once, in the preparation phase. For simplicity, we assume that we have only one intercepted ciphertext C. If we had more ciphertexts, the preparation phase could be easier.

4.1 Preparation phase: Finding the valid length L of the message

Since C is a valid ciphertext, the corresponding message M (plaintext without padding) conforms to PKCS#7. Thus we have $P_{1,1}$ = 0x04 and $P_{1,2}$ = L, where L is the length of M. In this step we determine the value of L using a PKCS#7$_{CONF}$ oracle.

Let s ≥ 3. This condition guarantees that changes made in C_{s-1} will not affect the first plaintext block containing the length octet. Let us denote LPAD the length of padding and LDATA the length of remaining data bytes in the last plaintext block, i.e. LDATA + LPAD = n. At first, we will successively test every byte from the end of the last plaintext block whether it is a padding byte or a data byte. When we find the first data byte from the end, we have the value LDATA and we stop testing. According to our assumptions, 0 ≤ LDATA ≤ n -1, because at least one byte ($P_{s,n}$) is a padding byte according to ABYT-PAD padding. Therefore we begin our test with $P_{s,n-1}$ following the pseudocode written below. We will denote C' our changes in the original ciphertext C.

CTemp = CT_{s-1}

LDATA = 0

For j = (n -1) downto 1

{

$CT_{s-1,j}$ = $CTemp_j$ ⊕ 1

If O(C') = "OK" then LDATA = j, break

/* Note.

The change will result in the corruption of the whole block P_{s-1} and of the byte $P_{s,j}$.

If O(C') returns "OK", then the change of the original plaintext byte $P_{s,j}$ to the value $P_{s,j} \oplus 1$ didn't affect the length L. Therefore $P_{s,j}$ is a data byte and we have LDATA = j.

If O(C') returns "BAD", the length L doesn't conform to the padding. There are two possibilities.

(i) $P_{s,j}$ was the last data byte, but it has been artificially changed to the padding byte

(ii) $P_{s,j}$ was the padding byte, which has been changed to a non-padding byte

We can decide between these two possibilities by setting $CT_{s-1,j}$ to the value $CTemp_j \oplus 2$ and calling the oracle O again.

*/

$CT_{s-1,j}$ = $CTemp_j$ ⊕ 2 and call O(C')

If O(C') = "OK" then LDATA = j, break

/* Note.

If it returns "OK", $P_{s,j}$ was the data byte. If it returns "BAD", $P_{s,j}$ was a padding byte. In this case we continue to test the next bytes on the left.

*/

}

L = (n - 2) + (s - 2)*n + LDATA

/* Note that (n-2) bytes are counted from the first block and LDATA bytes come from the last block. The resting (s - 2) blocks have the full length n.*/

Remarks

- There is the possibility of further optimising this process by various methods, for instance by the interval halving method, similarly as in [1]. In this case, we would need $O(\log_2 n)$ oracle calls. On the other hand, it is only a marginal improvement, because this step is carried out only once in the preparation phase.
- When the intercepted ciphertext has only one block CT_1, i.e. $s = 1$, we can use the same process as in case $s \geq 3$, because we will have full control over changes in the first plaintext block.
- In the case $s = 2$, we can artificially lengthen the original ciphertext (IV, CT_1, CT_2), for instance, as (IV, CT_1, CT_2, IV, IV, CT_1, CT_2). It is only necessary to change the second byte of the primary IV appropriately, because we added 4*n bytes to the message M. For instance if n = 8, we artificially added 32 bytes to it and we can fix the length easily by xoring the byte $C_{1,2}$ with 0x20, because the length of the original message M was less than 14. Now we can follow the process above for $s \geq 3$.

Now we have the plaintext byte $P_{1,2}$ = L of the ciphertext C. The complexity of this step is maximally 2*(n - 1) oracle calls.

4.2 Computing $X = D_K(Y)$ leaving one byte of uncertainty

Now we use the first two blocks (IV, CT_1) of the intercepted ciphertext C and create a new one C' = (IV, CT_1, S, T, Y), where S and T are arbitrarily chosen and Y is the block which is to be decrypted. Let us denote P = (P_1, P_2, P_3, P_4) as the plaintext corresponding to the ciphertext C'. We have $P_{1,1}$ = 0x04 and $P_{1,2}$ = L, where L is known from the previous step. Using the following pseudocode, we determine $X = D_K(Y)$ leaving one byte of uncertainty.

ITemp =IV , TTemp = T, A = $X_n \oplus T_n$

For i = (n - 1) downto 1

{

/* In this loop we derive the i-th byte $P_{4,i}$, where $P_{4,i+1} = ... = P_{4,n}$ are padding bytes, all equal to A.*/

N = (n - 2) + n + n + (i - 1)

$IV_2 = ITemp_2 \oplus P_{1,2} \oplus N$

/* After deciphering C', the oracle O gets the number N in the place of $P_{1,2}$. Thus it will expect i - 1 data bytes and n - (i - 1) padding bytes in the last plaintext block P_4. */

(*) For j = 0 to 255 do

{

$T_i = TTemp_i \oplus j$

If O(C') = "OK" go to (**)

/* If O(C') = "OK", the plaintext is PKCS#7 conforming and $X_i \oplus T_i$ is the padding byte A. Thus we have $X_i \oplus T_i = X_{i+1} \oplus T_{i+1} = ... = X_n \oplus T_n = A$ and we can continue to derive the next byte. */

}

If (i >1) then $T_{i-1} = TTemp_{i-1} \oplus 1$ else $S_n = S_n \oplus 1$

Go to (*)

/* If the oracle O has always responded "BAD" in the preceding cycle, it means that when the correct value (A) occurred on the i-th byte, it accidentally also occurred on its left side. Therefore, we change the left byte and go back to (*). Now, the oracle must once respond "OK". */

(**) /* Continue to the next loop. */

}

At the end of the procedure we have

(4.2) $X_1 \oplus T_1 = X_2 \oplus T_2 = ... = X_n \oplus T_n = A$,

where the values of T_i (and eventually also S_n) have been adjusted above. Note that for n > 32 we will need more length-octets, so it would be necessary to slightly modify this procedure.

In this step, we need 128*(n - 1) oracle calls on average. According to the procedure written above, the maximum number of oracle calls is clearly limited by the number 512*(n - 1).

4.3 Determining the remaining byte of uncertainty

Now we use the blocks T and Y created in the previous step. The ciphertext C = (T, Y) gives the one plaintext block consisting of n bytes having the same value A. We will now change T_1, T_2 and T_n to obtain a PKCS#7 conforming message. We construct the message in the way to have the length of n - 3 octets and one padding byte. We then determine the value of A in the following way.

TTemp = T

For j = 0 to 255 do

{

T_1 = $TTemp_1$ ⊕ 0x04 ⊕ j, T_2 = $TTemp_2$ ⊕ (n - 3) ⊕ j, T_n = $TTemp_n$ ⊕ 1

C' = (T, Y)

/* Note that C' corresponds to the plaintext (A ⊕ 0x04 ⊕ j, A ⊕ (n-3) ⊕ j, A,..., A, A ⊕ 1). */

If O(C') = "OK" then A = j, break

/* If O(C') = "OK", the plaintext is PKCS#7 conforming. Thus, we have A ⊕ 0x04 ⊕ j = 0x04. We then easily obtain the unknown A as A = j. */

}

Now, we restore the value T from TTemp and we substitute it with A into the system of equations (4.2), thereby deriving the value of X.

This step requires 128 oracle calls on average. It takes maximally 256 oracle calls.

5 Complexity of the attack and its extensions

Recall that the complexity of the attack is at most 2*(n-1) calls in the preparation phase. This phase is carried out only once for a particular symmetric key. To decipher each ciphertext block, we then need 128*(n-1) + 128 = 128*n oracle calls on average. The maximum number of oracle calls per ciphertext block is bounded above by 512*(n - 1) + 256.

Now we summarize our remarks on possible extensions and modifications of the attack.

- In most cases of longer messages with more than one length-octet, we can easily derive the plaintext byte $P_{1,2}$ = L in the preparation phase. In these cases $P_{1,2}$ is the first length-octet and thus $P_{1,2}$ is equal to the number of remaining length-octets + 0x80 (c.f. §2). We can estimate the number of remaining length-octets directly from the length of the ciphertext.
- Generally speaking, when longer messages (with any kind of ABYT-PAD padding) with more than one length-octet are used, we can successively change the third, fourth, etc. byte of the IV and send the ciphertext to the oracle O. If we change the length octet, the oracle returns "BAD". If it returns "OK", we hit the first data octet. Now we have the number of length octets (W) and we get $P_{1,2}$ = 0x80 + W immediately.
- If we may assume that the ciphertext contains at least n bytes of ABYT-PAD padding (one full block or more), we can artificially shorten the ciphertext (IV, CT_1, CT_2, ..., CT_{s-1}, CT_s) to the last two blocks only (CT_{s-1}, CT_s). We then use the procedure from §4.3 to determine the value of padding, i.e. we compute the value of $D_K(CT_s) \oplus CT_{s-1}$. Basing on this knowledge, we can continue directly to step 4.2, thereby bypassing step 4.1.
- A variant of this attack can be also derived for the ABIT-PAD padding. Since the only difference between ABYT-PAD and ABIT-PAD is the size of the elementary block unit, the derivation is a matter of changing the byte-oriented approach for a bit-oriented one. Note that the PKCS#7 format discussed here is byte-oriented in its nature, therefore even when using ABIT-PAD, we pad bytes. However, we may expect that the $PKCS\#7_{CONF}$ based on ABIT-PAD would, in a certain way, allow an attacker to do the inversion of $E_K(B)$ bit-by-bit instead of byte-by-byte. Such an approach generally helps the attacker

to improve the effectiveness of her attack. Such an improvement would be useful in case of extremely long padding string.

6 Countermeasures

The attack presented here is based on the behaviour of a typical transport-layer application, which routinely receives a ciphertext, deciphers it, and decodes its data payload, which is then passed to the upper layers. If an error occurs during this processing (e.g. ASN.1 parser fails, the padding is incorrect, etc.), it is natural that the application informs its communicating peer. Ignoring these errors is theoretically possible, but in practice it wouldn't make a lot of sense. Moreover, such a failure would be probably detectable from the behaviour of the upper-layer application.

We emphasize that the attack discussed here is not only a particular problem for padding methods. Generally speaking, such an attack can be expected whenever the following conditions are fulfilled:

(i) there are some formatting rules set for plaintexts which must be checked,

(ii) an attacker can freely modify captured ciphertexts and re-send them to the communicating application,

(iii) the changes made in (ii) induce predictable changes of the corresponding plaintext.

The combination of the CBC mode with the PKCS#5 padding scheme was perhaps the most obvious way in which the conditions given above were fulfilled. The only thing that is still surprising is the amount of time it took for the cryptanalysts to disclose this weakness. To avoid other possible "surprises", we should constantly verify these conditions when designing new encryption schemes. In this paper we addressed the situation where the first condition seems to be thwarted, since the padding method ABYT-PAD does not impose any checkable rules. However, the condition is easily restored if we incorporate the formatting rules given by the PKCS#7 standard. It clearly follows that despite being very tempting, we cannot hope to solve problems of attacks addressed here and in [1] and [5] by thwarting only the first condition written above.

A better way seems to be to focus on conditions (ii) and (iii). We can use the paper of Krawczyk [12] to conjecture that the best way is generally to thwart the second condition by using the authentication of ciphertexts. This countermeasure was also generally recommended in [1]. However, in some older applications it might not always be easy to introduce such a modification. Therefore, we looked at thwarting the last conditions. In [10] we designed a method that effectively prevents an attacker from making predictable changes of the plaintext by changing the ciphertext. Our method then enables any padding method that is limited to the last block to be used. The main idea of our approach is encrypting the last block in a different way and under a different key. This so-called strengthened encryption changes the definition of the encryption and decryption process, but it is still compatible with the original CBC encryption from the point of view of data structures and their length. From a practical viewpoint, we conjecture that for existing applications and protocols it is better to change the program codes or data semantic rather than the data structure itself. We must emphasize that our method works well, unless there are other structural checks of plaintexts. Therefore, generally we strongly recommend adding a cryptographic check of ciphertexts in the sense of [12].

7 Conclusions

In this paper we have shown that we cannot hope to fully solve problems with side channel attacks on the CBC encryption mode by using a magic padding method or an obscure message-encoding format. Vaudenay showed in [5] that the CBC encryption mode ([2], [9]) combined with the PKCS#5 padding scheme ([3]) allows an attacker to invert the underlying block cipher, provided she has access to an oracle which for each input ciphertext states whether the corresponding plaintext has a valid padding or not. Countermeasures against this attack using different padding schemes were studied in [1] and the best method was referred to as the ABYT-PAD.

In this paper, we combine the well-known cryptographic message syntax standard PKCS#7 [8] with the use of ABYT-PAD in the place of the PKCS#5 padding scheme. We assume that the attacker has access to an oracle $\text{PKCS\#7}_{\text{CONF}}$ which tells her for a given ciphertext (encapsulated in the PKCS#7 structure) whether the deciphered plaintext is correct or not according to the PKCS#7 (v1.6) syntax. This is a very natural and straightforward assumption, because applications usually have to reflect this situation in their behaviour. It could be a message for the user, an API error message, an entry in the log file, different timing behaviour, etc. We have shown that having access to such an oracle enables the attacker to invert the underlying block cipher and decipher

the encrypted message. It requires a single ciphertext for a particular key and approximately 128 oracle calls per ciphertext byte.

Surprisingly, the attack is allowed by those PKCS#7 (v1.6) properties that are designed to improve version v1.5. They are also planned for version 2. However, the improvement of the standard brought a new kind of attack. It follows that an improvement beneficial according to a local estimation may turn out to be a bad choice under a broader context evaluation. On the other hand, we do not express the opinion that our attack is a problem of the step of moving from the PKCS#7 v1.5 to the PKCS#7 v1.6 standard.

The attack described here can be also easily extended on other TLV-like schemes. The TLV stands for tag-length-value, which is a common nickname of many data protocols and formats used nowadays. Since TLV involves also many standards used in the banking sector, it indicates that existing systems in such areas deserve certain amount of attention according to the attack presented here.

The discussed problems with side channel attacks on the CBC encryption mode should be solved using strong cryptographic integrity checks of ciphertexts. Our contribution should be regarded as further evidence that these checks must be included in the new cryptographic standards and protocols.

Acknowledgements

The second author is grateful to his postgraduate supervisor Dr. Petr Zemánek for his continuous support in research projects.

References

[1] Black, J., and Urtubia, H.: Side-Channel Attacks on Symmetric Encryption Schemes: The Case for Authenticated Encryption, In Proc. of *11th USENIX Security Symposium*, San Francisco 2002, pp. 327-338.

[2] NIST Special Publication SP 800-38A 2001 ED, Recommendation for Block Cipher Modes of Operation, December 2001.

[3] PKCS#5 v. 2.0: Password-Based Cryptography Standard, RSA Laboratories, March 25, 1999.

[4] PKCS #7 v1.5: Cryptographic Message Syntax Standard, RSA Laboratories, November 1, 1993.

[5] Vaudenay, S.: Security Flaws Induced By CBC Padding - Application to SSL, IPSEC, WTLS..., *EUROCRYPT '02*, pp. 534-545, Springer-Verlag, 2002.

[6] ITU-T Recommendation X.680 (1997), ISO/IEC 8824-1:1998, Information Technology - Abstract Syntax Notation One (ASN.1): Specification of Basic Notation.

[7] ITU-T Recommendation X.690 (1997), ISO/IEC 8825-1:1998, Information Technology - ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).

[8] Extensions and Revisions to PKCS #7 (Draft PKCS #7 v1.6), An RSA Laboratories Technical Note, May 13, 1997.

[9] RFC 2268: Baldwin, R., and Rivest, R.: The RC5, RC5-CBC, RC5-CBC-Pad, and RC5-CTS Algorithms, October 1996.

[10] Klíma, V., and Rosa, T.: Strengthened encryption in the CBC mode, *Cryptology ePrint Archive: Report 2002/061*, http://eprint.iacr.org/2002/061.pdf.

[11] Rosa, T.: Future Cryptography: Standards are not Enough, in Proc. of *Security and Protection of Information, NATO-IDET, 2001*, Military Academy in Brno, pp. 237 - 245, Brno, 2001.

[12] Krawczyk, H.: The Order of Encryption and Authentication for Protecting Communications (or: How Secure Is SSL?), *CRYPTO' 01*, pp. 310 - 331, Springer-Verlag, 2001.