CBC Padding: Security Flaws in SSL, IPSEC, WTLS, ...

Serge Vaudenay

Swiss Federal Institute of Technology (EPFL)
Serge.Vaudenay@epfl.ch

Abstract In many standards, e.g. SSL/TLS, IPSEC, WTLS, messages are first pre-formatted, then encrypted in CBC mode with a block cipher. Decryption needs to check if the format is valid. Validity of the format is easily leaked out from communication protocols because the receiver usually sends an error message when the format is not valid. This is a side channel.
In this paper we show that the validity of the format of the decryption is actually a hard core bit predicate. We demonstrate this by implementing an efficient and practical side channel attack which enables the decryption of any ciphertext. The attack complexity is $O(NbW)$ where N is the message length in blocks, b is the block length in words, and W is the number of possible words (typically 256).
We also discuss about extensions to other padding schemes and various ways to fix the problem.

Variable input length encryption is traditionally constructed from a fixed input length encryption (namely a block cipher) in a special mode of operation. In RFC2040 [2], the RC5-CBC-PAD algorithm is proposed, based on RC5 which enables the encryption of blocks of $b = 8$ words where words are bytes. Encryption of any word sequence with an RC5 secret key K is performed as follows.

1. Pad the word sequence with n words, all being equal to n, such that $1 \leq n \leq b$ and the padded sequence has a length which is a multiple of b.
2. Write the padded word sequence as a block sequence $x_1, \ldots, x_N$ in which each block x_i consists of b words.
3. Encrypt the block sequence in CBC mode with a (fixed or random or secret, whatsoever) IV with a permutation C defined by RC5 with key K: get

$$y_1 = C(\mathrm{IV} \oplus x_1), \quad y_i = C(y_{i-1} \oplus x_i); i = 2, \ldots, N \qquad (1)$$

where $\oplus$ denotes the XOR operation.

The encryption of the message is the block sequence $y_1, \ldots, y_N$.

Although decryption is not clearly defined in RFC2040 [2], it makes sense to assume that the receiver of an encrypted message first decrypts in CBC mode, then checks if the padding is correct and finally removes it. The question is: how must the receiver behave if the padding is not correct? Although the receiver should not tell the sender that the padding is not correct, it is meaningful that non-procession of a decrypted message ultimatly leaks this bit of information. This leads to an attack which uses an oracle which for any block sequence tells if the padding of the corresponding CBC-decrypted sequence is correct according to the above algorithm.

This attack model is similar to the attack of Bleichenbacher against PKCS#1 v1.5 [4] and of Manger against PKCS#1 v2.0 [8]. This paper shows that similar attacks are feasible in the symmetric key world.

The paper is organized as follows. We first recall the well known properties and security issues for the CBC mode. We describe our attack against RC5-CBC-PAD. Then we discuss about extensions to other schemes: ESP, random padding, ... We also discuss about applications in real life: for SSL, IPSEC, WTLS. Next we present some possible fixes which actually do not work like replacing the CBC mode by a double CBC mode, the HCBC mode or other modes which are in a standard process run by NIST. We further propose a fix which does work: a new padding scheme. We finally conclude.

1 CBC Folklore

Several security properties of the CBC mode are already well known. We think it is useful to recall them.

1.1 Integrity Limits

Fault tolerance is quite high but integrity is not guaranteed: one can replace a ciphertext block without altering most of the plaintext blocks (it only modifies two plaintext blocks), one can remove one block, insert a sequence of blocks by copy and paste, ... Manipulation of ciphertext blocks only induces modification in the corresponding side plaintext blocks.

1.2 Confidentiality Limits

Confidentiality also has security flaws. Obviously, when using a fixed IV, one can easily see when two different messages have a common prefix block sequence by just looking at the two ciphertexts.

More generally, when two ciphertext blocks y_i and y_j are equal, one can deduce from Eq. (1) that $y_{i-1} \oplus y_{j-1} = x_i \oplus x_j$. We can then exploit the redundancy in the plaintext in order to recover x_i and x_j from $y_{i-1} \oplus y_{j-1}$. This flaw is however quite negligible: since the ciphertext blocks get a distribution which is usually undistinguishable from a uniform distribution, the probability that two b-words blocks out of N are equal is given by the birthday paradox theorem

$$p \approx 1 - e^{-\frac{1}{2}N^2.W^{-b}}$$

where W is the number of possible words. The attack is efficient when N reachs the order of magnitude of $\sqrt{W^b}$. Therefore, for $b = 8$ and $W = 256$, we need about 32GigaBytes in order to get 39% chances of success for this attack which leaks information on 16 Bytes only.

1.3 Authentication Limits

CBC mode is also used for message authentication code (MAC). Raw CBC-MAC (i.e. taking the last encrypted block as a MAC) is well known to have security flaws: with the MAC of three messages m_1, m_2, m_3 where m_2 consits of m_1 augmented with an extra block, we can forge the MAC of a fourth message which consists of m_3 augmented with an extra block. This is fixed by re-encrypting the raw CBC-MAC, but this new scheme have still attacks of complexity essentially $\sqrt{W^b}$. (See [9,11].)

2 The Attack

Let b be the block length in words, and W be the number of possible words. (We assume that $W \geq b$ and that all integers between 1 and b can non ambiguously be encoded into words in order to make the CBC-PAD scheme feasible.)

We say that a block sequence $x_1, x_2, \ldots, x_N$ has a correct padding if the last block x_N ends with a word string of n words equal to n with

$n > 0$: 1, or 22, or 333, ... Given a block sequence $y_1, y_2, \ldots, y_N$, we define an oracle $\mathcal{O}$ which yields 1 if the decryption in CBC mode has a correct padding. Decryption is totally defined by a block encryption function C and IV. Oracle $\mathcal{O}$ is thus defined by C and IV.

2.1 Last Word Oracle

For any y, we want to compute the last word of $C^{-1}(y)$. We call it the "last word oracle".

An attacker which accesses to $\mathcal{O}$ can easily implement this oracle with $W/2$ 2-block oracle calls on average. (Or a single one, but with a probability of succes of W^{-1}.)

Let $r_1, \ldots, r_b$ be random words, and let $r = r_1 \ldots r_b$. We forge a fake ciphertext $r|y$ by concatenating the two blocks r and y. If $\mathcal{O}(r|y) = 1$, then $C^{-1}(y) \oplus r$ ends with a valid padding. In this case, the most likely valid padding is the one which ends with 1. This means that the last word of $C^{-1}(y)$ is $r_b \oplus 1$. If $\mathcal{O}(r|y) = 0$, we can try again (by making sure that we picked another r_b: picking the same one twice is not worthwile).

If we are lucky (with probability W^{-1}), we find the last word with the first try. Otherwise we have to try many r_bs. On average, we have to try $W/2$ values.

Odd cases are when the valid padding found is not 1. This is easy to detect by sending $r'|y$ with $r' = r'_1 \ldots r'_b$, $r'_b = r_b$ and $r'_{b-1} \neq r_{b-1}$. If $\mathcal{O}(r'|y) = 1$, the valid padding was 1 indeed. Otherwise, it was a longer one. We can similarly check if it was 22 by sending $r''|y$ with $r'' = r''_1 \ldots r''_b$, $r''_b = r_b$, $r''_{b-1} = r_{b-1}$, and $r''_{b-2} \neq r_{b-2}$...

2.2 Block Decryption Oracle

Now we want to implement an oracle which computes $C^{-1}(y)$ for any y: a "block decryption oracle".

Let $a = a_1 \ldots a_b$ be the word sequence of $C^{-1}(y)$. We can get a_b by using the last word oracle. Assuming that we already managed to get $a_j \ldots a_b$ for some $j \leq b$, we show how to get a_{j-1}, so that we can iterate until we recover the whole sequence.

We let $r_1, \ldots, r_{j-1}$ be random words and $r_k = a_k \oplus (b - j + 2)$ for $k = j, \ldots, b$. Let $r = r_1 \ldots r_b$. We forge a fake ciphertext $r|y$ by

concatenating the two blocks r and y. When calling $\mathcal{O}$ on $r|y$, the second decrypted block is $r \oplus a$ due to Eq. (1), so we have made sure that the last $b-j+1$ words are all equal to $b-j+2$. If $\mathcal{O}(r|y) = 1$, we are thus ensured that $r_{j-1} \oplus a_{j-1} = b-j+2$ from which we can deduce a_{j-1}. When r_{j-1} is random, there is a probability of W^{-1} that this occurs. We thus need $W/2$ trials on average to make this event occur. We can thus recover an additional word within $W/2$ trials. Since there are b words per block, we need $bW/2$ trials on average in order to implement the C^{-1} oracle.

2.3 Decryption Oracle

Now we want to decrypt any message $y_1, \ldots, y_N$ with the help of $\mathcal{O}$. It can be done with $NbW/2$ 2-block oracle calls on average. We just have to call the block decryption oracle on each block y_i and perform the CBC decryption.

One problem remains in the case where IV is secret. Here we cannot decrypt the first block. We can however get the first plaintext block up to an unknown constant. In particular, if two messages are encrypted with the same IV, we can compute the XOR of the two first plaintext blocks.

The attack has a complexity of $O(NbW)$. As an example for $b = 8$ and $W = 256$ we obtain that we can decrypt any N-block ciphertext by making $1024N$ oracle calls on average. The attack is thus extremely efficient.

2.4 Postfix Equality Check Oracle

We can implement a more exotic oracle which has the nice property of using a single $\mathcal{O}$ oracle call. Given a ciphertext $y_1, \ldots, y_N$ and a block sequence $w_1 \ldots w_m$, we want to check if $w_1 \ldots w_m$ is a postfix of the decryption of $y_1, \ldots, y_N$.

Let us first consider that $m \leq b$. We just have to pick a few random words $r_1 \ldots r_{b-m}$, to take $r_{b-m+k} = w_k \oplus m$, and to send $(r \oplus y_{N-1})|y_N$ to the oracle where $r = r_1 \ldots r_b$. If accepted by $\mathcal{O}$, then $w_1 \ldots w_m$ is a postfix of the plaintext (but for an odd case which occurs with probability W^{-1} and which can be ruled out with an extra oracle call). Otherwise it is not a postfix for sure.

For $m > b$, one possibility is to perform the above process several times. But this will use $\mathcal{O}$ more than once. As it will be noticed, some CBC-PAD variant allow to have paddings longer than b (namely at most $W-1$), so we can generalize the previous oracle and check postfixs within a single $\mathcal{O}$ oracle call. This will be used against SSL/TLS in Section 4.1.

3 Other Padding Schemes

In Schneier [10, pp. 190–191], a slightly different padding scheme is proposed: only the last word is equal to the padding length, and all other padded words are equal to zero. The padded sequence is thus $00 \ldots 0n$ instead of $nn \ldots n$. Obviously, a similar attack holds.

IP Encapsulating Security Payload (ESP) [7] uses another slightly different padding: the padded sequence is `1234`$\ldots n$ instead of $nn \ldots n$. Obviously, a similar attack holds.

One can propose to have the last word equal to the padding length and all other padded words chosen at random. The attack still enables the decryption of the last word of any block. We also have another security flaw: if the same message is encrypted twice, it is unlikely that the last encrypted blocks are equal, but in the case where the padding is of length one. We can thus guess what is the padding length when the ciphertexts are equal.

4 The Attack in Real Life

4.1 SSL/TLS

SSL/TLS [5] uses the CBC-PAD scheme with $W = 256$ when using block ciphers (default cipher being the RC4 stream cipher though). The only difference is that the padding length is not necessarily less than b but can be longer (but less than $W - 1$) in order to hide the real length of the plaintext. We can thus expect to use a TLS server like the $\mathcal{O}$ oracle. The attack is however not so practical since the padding format error (the `decryption_failed` error) is a fatal alert and the session must abort. The attacker thus needs to stop as soon as the oracle outputs 0.

However the oracle will output 1 with a probability W^{-1}. Therefore, the attacker can still decrypt one word with a probability of success of W^{-1}, two words with a probability of success of W^{-2}, ...

We furthermore notice that the postfix equality check oracle of Section 2.4 can be implemented since there is essentially a single oracle call.

Interestingly, TLS wants to make secret the real message length itself. We can easily frustrate this feature by implementing a "length equality check oracle" in a very same way: if we want to check whether or not the padding length is equal to n, we take the last ciphertext block y, and we send $r|y$ to the server where the rightmost word of r is set to $n \oplus 1$ and the others are random. Acceptance by $\mathcal{O}$ means that the right length is n with probability at least $1-W^{-1}$. Rejection means that n is not the right length for sure.

Since the padding length is between 1 and W, the above oracle may not look so useful. We can still implement an oracle which answers whether or not the pad length is greater than b, i.e. if the length hiding feature of TLS was used: let y_1 and y_2 be the last two ciphertext blocks. We just send $r|y_1|y_2$ with a random block r to $\mathcal{O}$. Acceptance means that the padding length is at most b with probability at least $1-W^{-1}$. Rejection means that the padding length is at least $b+1$ for sure.

4.2 IPSEC

IPSEC [6] can use CBC-PAD. Default padding scheme is similar, as specified in ESP [7]. Standards clearly mention that the padding should be checked, but the standard behavior in case of invalid padding is quite strange: the server just discards the invalid message and adds a notification in log files for audit and nothing else. This simply means that errors are processed according to non standard rules. It is reasonnable to assume that the lack of activity of the receiver in this case, or the activity of the auditor, can be converted into one bit of information. So our attack may be applicable.

4.3 WTLS

WTLS [1] (which is the SSL variant for WAP) perfectly implements the oracle $\mathcal{O}$ by sending `decryption_failed` warnings in clear. Ac-

tually since mobile telephones have a limited power and CPU resources, key establishment protocols with public key cryptography are limited. So we try to limit the number of session initialization and to avoid breaking them. So seldom errors are fatal alerts. Some implementations of WTLS can however limit the tolerance number of errors within the same session, which can limit the efficiency of the attack. This is however non standard.

In the case of mobile telephones (which is the main application of WTLS), WTLS is usually encapsulated in other protocols which may provide their own encryption protocol, for instance GSM. In this case, the extra encryption layer needs to be bypassed by the attacker.

5 Fixes which Do not Work

5.1 Padding Before the Message

One can propose to put the padding in the first block. This only works for CBC modes in which IV is not sent in clear with the ciphertext (otherwise the same attack holds). This also requires to know the total length (modulo b) of the message that we want to encrypt before starting the encryption. When the plaintext is a word stream, this assumption is not usually satisfied. Therefore we believe that this fix is not satisfactory.

5.2 CBCCBC Mode

Another possibility consists of replacing the CBC mode by a double CBC encryption (i.e. by re-encrypting the $y_1, \ldots, y_N$ sequence in CBC mode). We call it the CBCCBC mode.

Unfortunately, a similar attack holds: given y and z we can recover the value of $u = C^{-1}(y) \oplus C^{-1}(y \oplus C^{-1}(z))$ by sending $r|y|z$ trials to the oracle. This is enough in order to decrypt messages: if y is the $(i-1)$th ciphertext block, z is the ith ciphertext block, and if t is the $(i-2)$th ciphertext block, then the ith plaintext block is nothing but $t \oplus u$!

The same attack holds with a triple CBC mode...

5.3 On-Line Ciphers and HCBC Mode

We can look for another mode of operation which provably leaks no information. One should however try to keep the advantages of the CBC mode: being able to encrypt a stream without knowing the total length, without having to keep an expanding memory, ... In [3], Bellare et al. presented the notion of on-line cipher. This notion is well adapted for these advantages of the CBC mode.

They also proposed the HCBC mode as a secure on-line cipher against chosen plaintext attacks. The idea consists in replacing Eq. (1) by

$$y_i = C(H(y_{i-1}) \oplus x_i)$$

where H is a XOR-universal hash function which includes a part of the secret key. For instance one can propose $H(x) = K_1 x$ in $\mathrm{GF}(W^b)$ where K_1 is a nonzero part of the secret key. (When $a \neq b$, we have $\Pr[H(x) \oplus H(b) = c] \leq 1/(W^b - 1)$ for any c if K_1 is uniformly distributed, thus H is XOR-universal.)

One problem is that this does not protect against the kind of attack we proposed. For instance we can notice that if we get several accepted $r_i|y$ messages with a fixed y, then we deduce that $H(r_i) \oplus x$ ends with a valid padding for an unknown but fixed x. Hence $H(r_i) \oplus H(r_j)$ is likely to end with the word zero. Since this is the last word of $K_1(r_i \oplus r_j)$, we can deduce K_1 from several (i, j) pairs. With the knowledge of K_1 we can then adapt the attack against the raw CBC. It is even more dramatic here since we indeed recover a part of the secret key.

Therefore the notion of on-line cipher resistant against chosen plaintext attacks does not capture security against the kind of side channel cryptanalysis that we have proposed.

5.4 Other Modes of Operation

The standardization process on modes of operation launched by NIST also contains problematic proposals.[1] Several of the proposals can be generalized as follows. The CBC mode is modified in order to have a XOR before and after the block cipher encryption, depending

[1] See `http://csrc.nist.gov/encryption/modes/`

on all previous ciphertext blocks and all previous plaintext blocks. We replace Eq. (1) by

$$y_i = C(x_i \oplus f_i(x, y)) \oplus g_i(x, y)$$

with a public $f_i(x, y)$ and $g_i(x, y)$ functions which depend on i and $x_1, \ldots, x_{i-1}, y_1, \ldots, y_{i-1}$ only. (Note that HCBC is not an example since f_i is not public.)

Assuming that an attacker knows several (x^j, y^j) plaintext-ciphertext pairs written $x^j = x_1^j | \ldots | x_{\ell_j}^j$ and $y^j = y_1^j | \ldots | y_{\ell_j}^j$, and she wants to compute $C^{-1}(y)$ for some given y, she can submit some $y_1^j | \ldots | y_k^j | (y \oplus \delta)$ ciphertexts where $k \leq \ell_j$, $\delta = g_{k+1}(x^j, y^j)$. Acceptance would mean that the block $C^{-1}(y) \oplus f_{k+1}(x^j, y^j)$ ends with a valid padding. Therefore we can decrypt the rightmost word with W samples, two words with W^2 samples, ...

Many other manipulations can also be performed.

6 A Fix which Does Work

One can propose to add a cryptographic checkable redundancy code (crypto-CRC) of the whole padded message (like a hashed value) in the plaintext and encrypt

$$\text{message}|\text{padding}|h(\text{message}|\text{padding}).$$

This way, any forged ciphertext will have a negligible probability to be accepted as a valid ciphertext. Basically, attackers are no longer able to forge valid ciphertexts, so the scheme is virtually resistant against chosen ciphertext attacks.

Obviously it is important to pad before hashing: padding after hashing would lead to the a similar attack. The right enciphering sequence is thus

$$\text{pad, hash, encrypt}$$

Conversly, the right deciphering sequence consists of decrypting, checking the hashed value, then checking the padding value. Invalid hashed value must abort the decipherment.

The drawback is that fault resistance is now quite poor: if a ciphertext block is not well received, the whole ciphertext is rejected by the server. (But recovery is still exceptionally feasible in case of message loss from the client.)

7 Conclusion

We have shown that several popular padding schemes which are used in order to transform block ciphers into variable-input-length encryption scheme introduce an important security flaw. Correctness of the plaintext format is indeed a hard core bit which easily leaks out from the communication protocol.

It confirms that security analysis must not be limited to the block cipher but must rather be considered with the whole environment. This was already well known in the public key cryptography world. We have demonstrated that the situation of symmetric cryptography is virtually the same.

Acknowledgments

I would like to thank Pascal Junod for helpful discussions. I also thank my students from EPFL for having found this attack during their examination. After this attack was releazed at the Rump session of CRYPTO'01, several people provided useful feedback. I would in particular like to thank Bodo Möller, Alain Hiltgen and Wenbo Mao.

References

1. Wireless Transport Layer Security. Wireless Application Protocol WAP-261-WTLS-20010406-a. Wireless Application Protocol Forum, 2001. `http://www.wapforum.org/`
2. R. Baldwin, R. Rivest. The RC5, RC5-CBC, RC5-CBC-Pad, and RC5-CTS Algorithms RFC 2040, 1996.
3. M. Bellare, A. Boldyreva, L. Knudsen, C Namprempre. Online Ciphers and the Hash-CBC Construction. In *Advances in Cryptology CRYPTO'01*, Santa Barbara, California, U.S.A., Lectures Notes in Computer Science 2139, pp. 292–309, Springer-Verlag, 2001.
4. D. Bleichenbacher. Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS#1. In *Advances in Cryptology CRYPTO'98*, Santa Barbara, California, U.S.A., Lectures Notes in Computer Science 1462, pp. 1–12, Springer-Verlag, 1998.
5. T. Dierks, C. Allen. The TLS Protocol Version 1.0. RFC 2246, standard tracks, the Internet Society, 1999.
6. S. Kent, R. Atkinson. Security Architecture for the Internet Protocol. RFC 2401, standard tracks, the Internet Society, 1998.
7. S. Kent, R. Atkinson. IP Encapsulating Security Payload (ESP). RFC 2406, standard tracks, the Internet Society, 1998.

8. J. Manger. A Chosen Ciphertext Attack on RSA Optimal Asymmetric Encryption Padding (OAEP) as Standardized in PKCS#1 v2.0. In *Advances in Cryptology CRYPTO'01*, Santa Barbara, California, U.S.A., Lectures Notes in Computer Science 2139, pp. 230–238, Springer-Verlag, 2001.
9. E. Petrank, C. Rackoff. CBC MAC for Real-Time Data Sources. *Journal of Cryptology*, vol. 13, pp. 315–338, 2000.
10. B. Schneier. *Applied Cryptography*, 2nd Edition, John Wiley & Sons, 1996.
11. S. Vaudenay. Decorrelation over Infinite Domains: the Encrypted CBC-MAC Case. In *Selected Areas in Cryptography*, Waterloo, Ontario, Canada, Lectures Notes in Computer Science 2012, pp. 189–201, Springer-Verlag, 2001. Journal version: *Communications in Information and Systems*, vol. 1, pp. 75–85, 2001.