

Playing with CBC

André Mariën

Abstract

The padding oracle attack in combination of Cipher Block Chaining (**CBC**) proves to have great practical implications. But just using CBC does have some risks that are insufficiently described. In this paper we looked at risk-related properties of CBC. We present options to reduce these risks while keeping to the core CBC operation.

Introduction

Cipher Block Chaining (CBC) (Dworkin, ISO) recently got serious attention (Vaudenay, Yau, Paterson) because an information leak via a block padding oracle turned out to be a lethal combination when coupled with CBC. With a padding oracle it is possible to decrypt the last block – the padding block – by trial-and-error, with a very low complexity. Making any block the last block allows a complete communication to be recovered.

The padding oracle attack is considered a form of a side-channel attack (Yau, Black). In the application security world, it would be a plain vulnerability. Error message analysis is a standard part of the common assessment methodology. The term “error oracle” (Mitchell) is therefore a very accurate one.

This unexpected vulnerability by itself is troublesome, but feels like more theory than a practical concern. In his Ph.D. thesis, Yau (Yau) analyses the different forms of padding, and how to attack them. In the second part, he investigates the vulnerability of well-known protocols for this attack. The results of this study increase the concerns.

If a vulnerability is found with high potential, such as this, fix it. Otherwise, some day it will be used in a more complex or unexpected scenario. As could be predicted, this happened. A padding oracle may seem unlikely at first. In some cases, timing based analysis was proposed which increases the noise an attack would make. But then a more interesting observation was made (Rizzo): if a block can be decrypted, a block can be encrypted. And not just the last block, but a whole new communication stream. Now you have the attention of main stream security.

In this paper, CBC will be investigated under various conditions and see if anything interesting is possible. The target is CBC itself: we do not care neither about the cipher method nor the block size. We will not look at specific faults that enable encryption or decryption.

Are there any practical implications? The padding oracle made it to the Black Hat conference 2010 (Rizzo). If one should investigate this, please consider not just standard protocols. Many applications use some form of encryption and may incorporate CBC. The hope is that this paper will be helpful to understand some of the risks so that they can be mitigated.

Notation

In the sequel we often need to look at two consecutive encrypted blocks and the decrypted block of the second one:

$$(E_{i-1}, E_i) \xrightarrow{D} D_i$$

Or the reverse, encryption based on the previous block and the plaintext:

$$(E_{i-1}, D_i) \xrightarrow{E} E_i$$

Starting with the basics

CBC is extremely simple and therefore attractive. Take any block cipher, add exclusive or (xor) and done. Each block is encrypted as: $E_i = \text{enc}(D_i \text{ xor } E_{i-1})$ and decrypted as: $D_i = \text{dec}(E_i) \text{ xor } E_{i-1}$

We already can make important observations:

1. When encrypting the same data, the result will be different if the previous encrypted block is different.
This is as designed.
2. An encrypted block by itself can be the encryption of any data block. One needs to consider two blocks to determine the plaintext data.
The good thing is that it can be any data block, as expected and designed.
But changing the first of two blocks of ciphertext does allow to produce any plaintext, and if controlled by the attacker, the effects and possibilities must be investigated. It feels too easy to make designs that do not consider the possibilities.

Basic attacks

Collision in encrypted blocks

A documented attack is based on producing or finding two instances of the same block in streams encrypted with the same key.

If we have both $E = \text{enc}(D_i \text{ xor } E_{i-1})$ and $E = \text{enc}(D_j \text{ xor } E_{j-1})$, then $D_i \text{ xor } E_{i-1} = D_j \text{ xor } E_{j-1}$

Since we know both E_{i-1} and E_{j-1} , we know: $D_i \text{ xor } D_j = E_{j-1} \text{ xor } E_{i-1}$

This information might be sufficient to find probable plaintexts.

In common practice, collisions are unlikely, but if the block size is too small this attack can become practical.

Same IV

Same IV has been well-documented as a bad practice. One can compare the risks with the well-known protection of passwords with salted hashes. What happens if the salt is always the same? Some proposals are to hide the IV altogether.

We have both $E_{i1} = \text{enc}(D_{i1} \text{ xor } IV)$ and $E_{j1} = \text{enc}(D_{j1} \text{ xor } IV)$

or in shorthand notation: $(IV, D_{i1}) \xrightarrow{E} E_{i1}$ and $(IV, D_{j1}) \xrightarrow{E} E_{j1}$

If the first blocks of stream i and j are identical their encrypted streams are identical.

In a chosen plaintext attack, we enumerate the possibilities for the first data block of the unknown stream. Typically, the possible set is much smaller than the full size. A dictionary attack becomes an option. Of course, we need cooperation of the system to encrypt all our tests and be able to read the ciphertext back. This is not too unrealistic.

Playing

Encrypt(0) = E0

The basic encryption formula is:

$$E_i = \text{enc}(D_i \text{ xor } E_{i-1})$$

What if $D_i = E_{i-1}$? In a typical setting we send a plaintext block and get back the next encrypted block. If we send the just received encrypted block as the new data block, we do exactly this:

$$E_i = \text{enc}(E_{i-1} \text{ xor } E_{i-1})$$

$$E_i = \text{enc}(0)$$

This result $\text{enc}(0)$ will be referred to as $E0$. It is nice to play with.

“Neutral” element: (X, E0) D=> X

$$\text{If } E_i = E0 \text{ We have: } D_i = \text{dec}(E_i) \text{ xor } E_{i-1} \Leftrightarrow D_i = 0 \text{ xor } E_{i-1} \Leftrightarrow D_i = E_{i-1}$$

So, if we have $E0$ anywhere, its decrypted value is the value of the previous encrypted block, or:

$$(X, E0) D=> X$$

Producing 0 in the decrypted stream: (0, E0) D=> 0

From the previous:

$$(X, E0) D=> X$$

So:

$$(0, E0) D=> 0$$

Detecting same key: compute E(0)

Forcing $E0$ can be used to check for identical key. Typically one needs to know the IV and be able to do chosen plaintext encryption. The $E0$ method works with chosen plaintext only.

After receiving an encrypted block E , send it back as the next plaintext. The next encrypted block, the answer, is $E0$. This works under the most restrictive operation.

If $E0$ is the same, the key is the same.

Exchange plaintext and IV: (E_{i-1}, E_i) D=> D_i and (D_i, E_i) D=> E_{i-1}

We have:

$E_i = \text{enc}(D_i \text{ xor } E_{i-1})$ with $D_i = \text{dec}(E_i) \text{ xor } E_{i-1}$

And therefore obviously also:

$E_i = \text{enc}(E_{i-1} \text{ xor } D_i)$ with $E_{i-1} = \text{dec}(E_i) \text{ xor } D_i$

In other words

$(E_{i-1}, E_i) \xrightarrow{D} D_i$ and $(D_i, E_i) \xrightarrow{D} E_{i-1}$

Related pairs

If we produce another pair (X, Y) such that:

$E_{i-1} \text{ xor } D_i = X \text{ xor } Y$

Then we have

$(E_{i-1}, D_i) \xrightarrow{E} E_i$ but also $(X, Y) \xrightarrow{E} E_i$

Inserting (X, E_i) in a stream produces Y as plaintext:

$(X, E_i) \xrightarrow{D} Y$

Changing the decrypted text, chosen bits

If

$(E_{i-1}, E_i) \xrightarrow{D} D_i$

Then

$(E_{i-1} \text{ xor } X, E_i) \xrightarrow{D} D_i \text{ xor } X$

Changing the decrypted text, into chosen one

If

$(E_{i-1}, E_i) \xrightarrow{D} D_i$

Then

$(E_{i-1} \text{ xor } D_i \text{ xor } X, E_i) \xrightarrow{D} X$

Stream cross-over

This idea is also present in (Rizzo).

Given

$E_0 E_1 \dots E_{i-1} E_i E_{i+1} \dots E_{n-1} E_n \xrightarrow{D} D_0 D_1 \dots D_{i-1} D_i D_{i+1} \dots D_{n-1} D_n$

And

$E_0 E_1 \dots E_{j-1} E_j E_{j+1} \dots E_{m-1} E_m \xrightarrow{D} C_0 C_1 \dots C_{i-1} C_i C_{i+1} \dots C_{n-1} C_n$

One can create:

$E_0 E_1 \dots E_{i-1} E_i E_{j-1} E_j E_{j+1} \dots E_{m-1} E_m D \Rightarrow D_0 D_1 \dots D_{i-1} D_i X C_j C_{j+1} \dots D_{n-1} D_n$

Everything but the block X will be “normal” cleartext.

If the length must be the same (it typically is required), we get:

$E_0 E_1 \dots E_{i-1} E_j E_{j+1} \dots E_{m-1} E_n D \Rightarrow D_0 D_1 \dots D_{i-1} X C_{i+1} \dots D_{n-1} D_n$

Message prepend

If the IV is not actually an IV, but some block of another stream, the other stream can be prepended to the new one. The padding might cause some issues, but if the prepended message was well prepared, this could pass unnoticed. (see also Fib’ikov’a).

Segregation of rights to encrypt - decrypt

In this section we will examine if it possible to segregate the rights to encrypt and decrypt.

Given that we have two interfaces, one to decrypt and one to encrypt: can we build on top of the isolation of encryption and encryption? In neither case we have access to the key, that is secretly held by the service. As an example consider the use of tamper resistant devices: the key never leaves the device.

Encryption

We may look at two times two options:

Bootstrap:

- a) Start() -> IV
- b) genIV() -> IV

Steps:

- a) $E = \text{next}(D)$
- b) $E = \text{CBCstep}(PE, D)$

In both cases, a is more restrictive than b. If we can do something with (a,a), we can do it always.

Generating E0

In either system, we can generate E0 blocks.

We get EX back from the last operation, and we feed this as the next plaintext block.

$(EX, EX) E \Rightarrow \text{enc}(EX \text{ xor } EX) = E0$

Note:

“one advantage of CBC mode is that it is an online algorithm. An online algorithm is one in which encryption or decryption can be performed on-the-fly, as and when blocks of data are received” (Watson)

It is exactly this strength that allows adaptive behavior, like producing E0.

Generating a validly encrypted (IV, block) pair for any data block

Now we can use the pair (D, E0) $D \Rightarrow D$.

So, we constructed a validly encrypted pair of blocks (D,E0) for any plaintext D.

Brute force decryption?

Suppose we have:

$(IV,DX) E \Rightarrow EX$

Can we decrypt without having to guess the key?

A known issue is that there are often two attacks: brute force theoretical space, or enumerate possible plaintext, a subset thereof.

If we have encryption authorization, we can brute force by encrypting with the previous block of the target and enumerating all the possible plaintexts we look for, like for example encrypted passwords.

Encryption to decryption?

There are no obvious ways to leverage the encryption capability into decryption. This is good news.

Decrypt

Decrypt to encrypt? Yes!

When someone has decryption capability, it also means he can encrypt. This was first reported in (Rizzo).

Decryption:

$(PE,E) D \Rightarrow D$

Pick random X_{n-1} and E_n

$(X_{n-1}, E_n) D \Rightarrow Y_{n-1}$

$E_{n-1} = X_{n-1} \text{ xor } Y_{n-1} \text{ xor } D_n$

$(E_{n-1}, E_n) D \Rightarrow X_{n-1} \text{ xor } Y_{n-1} \text{ xor } D_n$

$(E_{n-1}, E_n) D \Rightarrow D_n$

Now we can repeat the process to determine the previous block and the previous block etc.

Note that this is specific for CBC. There is no equivalent for ECB, for instance.

So we can encrypt a stream backwards while decrypting a fake stream forward.

Examples

The following examples are not real, specific attacks. They are just intended to clarify that the whole discussion is less academic than it may seem.

The token

Suppose we use a session token that is an encrypted uid+timestamp. The uid is 8 positions, the time stamp is 8 positions, the cipher block size is 32 (say, AES 256).

Each token will be an IV and one encrypted block.

We have a known plaintext (our token has plaintext X) with ciphertext, the pair (IVX, C).

(IVX, X) $E \Rightarrow C$

We use our encrypted block with another IV to create a new token.

If the token we want to create has value Y, then we compute IVY as:

$IVY = IVX \text{ xor } X \text{ xor } Y$

Now we have:

(IVY, EX) $D \Rightarrow Y$

Note: this finding was reported and used to encrypt new messages by obtaining incrementally plaintext.

Keep your token alive

If we have a valid token, and now the time position, we can just modify time to suit our needs.

So I steal your token, but you say: no worry, it will expire. Maybe.

Stream cross-over

An option way to fool the system is merging two streams. While this seems like a dumb and bound to fail idea, it is best to have a quick look at the options.

If I have two streams of encrypted data I can take some head from the first and some tail from the last and paste these together. Clearly, the first block of the tail will come out mangled: it uses the wrong preceding block.

But, with CBC one usually reasons:

- all blocks are chained
- the last block must have correct padding
- so if the last block shows correct padding, the integrity of the communication is ok

But clearly we can merge streams and pass this test.

Another assumption is that if we have a bad block somewhere in the plaintext, the message will not parse or otherwise be recognized as a fake. This may not be the case:

- as the stream was encrypted by a trusted party, input validation is unlikely very strict

Many communications contain free text blocks that may be used for some functions, but not authentication or authorization purposes. So, the switch over may be carefully planned to coincide with a don't care position.

Just looking at the size of for instance a viewstate makes one wonder about the options.

Note: even in certificates such a block was found and used to brute force a signature check collision (md5).

We concur with NN that padding is not integrity checking and that it would be wise to consider a better mechanism.

Padbuster

Holyfield wrote the tool "padbuster" that uses padding oracle to decrypt blocks. He also gives a good explanation of how it works. But his example is interesting for another reason: there is absolutely no need to use a padding oracle in the example he gives.

He presents an application that uses a token that is a 3DES CBC encrypted string "account;uid;gid;". This type of system can trivially be subverted. The first block contains the key information.

IV 3DES(IV xor "account;...") ...

Many systems have fixed width user names. So it would be very easy to switch user names:

IV xor (account| |0..0) xor (newaccount| |0...0) 3DES(IV xor "account;...")

will decrypt to newaccount;...

Solutions?

Complicate playing with preceding block

The core problem is that the xor operation is very simple. How attractive it may be, it is at the core of the attacks. Two obvious variations attempt to fix this:

Encrypt the previous block before xor:

Encryption:

$$E_i = \text{enc}(D_i \text{ xor } \text{enc}(E_{i-1}))$$

Decryption:

$$D_i = \text{dec}(E_i) \text{ xor } \text{dec}(E_{i-1})$$

Hash the previous block before xor:

Encryption:

$$E_i = \text{enc}(D_i \text{ xor } \#(E_{i-1}))$$

Decryption:

$$D_i = \text{dec}(E_i) \text{ xor } \#(E_{i-1})$$

The variant with the extra encryption clearly feels bad. This feeling can be related to its relationships with a one-time pad based on a block cipher, and competing schemes with CBC, namely Cipher feedback (CFB) and Output Feedback (OFB). In both these cases there is only one encryption step, yet they are perceived as realistic alternatives.

The variant with the hash is therefore the only one explored. Its main advantage is that the attacker now has no direct control anymore. Hashes are supposed to protect against finding plaintext given a desired outcome. The padding attack for instance will be rather difficult to execute.

The padding attack can still be partially executed in this case. The attacker can generate (text,#) pairs where the hash has the desired ending. The longer the ending, the more impractical this becomes, but some padding schemes can be abused to leak ending bytes this way.

In a similar approach, the first encrypted block can also partially be attacked to change the last few bytes.

Both padding attack and first block attack are more difficult if one would use the encryption variant. It would be much more difficult to construct (plaintext, cipher text) pairs with the desired pattern.

Prevent first block attack

We look at two proposals that make reuse of existing CBC systems easy.

The first proposal is to use as first data block a "signature" block: The encrypted stream becomes:

$$\text{IV enc}(\text{IV xor SIGN}) \text{ enc}(E1 \text{ xor } D1) \dots$$

This proposal enforces a correct start: it is not possible to play with the IV, as the second block should decrypt to the SIGN block. Detection is early in the stream, not at the end like with some CRC or padding check. Of course, it adds another block of data to the encrypted stream.

A second proposal is even more direct: encrypt the IV. The encrypted stream becomes:

$$\text{enc}(\text{IV}) \text{ enc}(\text{IV xor } D1) \text{ enc}(E1 \text{ xor } D2) \dots$$

Now it becomes hard to attack the first block, even if one has a known plaintext and IV. The basic attack requires a split of (IV xor D1) into (IV' xor F1), and now the attacker must find enc(IV').

This also is a nice example of theory versus practice: in theory the IV can be public, in practice better not.

Conclusion

Cryptography is required for our internet-infested society. We need it to help protect our privacy, our safety and our money. But cryptography is a minefield where only the specialists should go. For anyone else we need pre-canned, thoroughly tested solutions at a high enough level to avoid being victims. A false sense of security is as bad as no security, maybe worse.

This little exploration in the world of cryptography once more confirms the above feeling: it is just too difficult to get cryptography right. OWASP consist of top expert in application security, yet even they failed against the padding oracle attacks. Will you be able to do better? Unlikely.

During the writing of this text I learned a few things I did not expect when I started. Is everything I designed in the past resistant to these attacks? No. Is the full extent of what someone can come up with given these options clear? No. But one thing is sure: there is a real risk, as the topic made it to black hat: it must be practical, and that audience now knows about it, for better or worse.

References

- J. Black and H. Urtubia. Side-Channel Attacks on Symmetric Encryption Schemes: The Case for Authenticated Encryption. Proceedings of the 11th USENIX Security Symposium, pages 327{338, 2002. URL: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.91.4700&rep=rep1&type=pdf>
- D. Bleichenbacher. Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS#1. In H. Krawczyk, editor, Proceedings of CRYPTO '98, volume 1462 of LNCS, pages 1{12, 1998.
- M. Dworkin. Recommendation for Block Cipher Modes of Operation: Methods and Techniques. NIST Special Publication 800-38A, NIST, 2001.
- Fib'ikov'a, L., Provable Secure Scalable Block Ciphers, Ph.D. thesis, Universitat Duisburg-Essen, 2003, URL: http://deposit.ddb.de/cgi-bin/dokserv?idn=971192898&dok_var=d1&dok_ext=pdf&filename=971192898.pdf
- International Organization for Standardization. ISO/IEC 10116 (3rd edition): Information technology | Security techniques | Modes of operation for an n-bit block cipher, 3rd edition, 2006.
- Holyfield, B, URL: <http://www.gdssecurity.com/l/b/2010/09/14/automated-padding-oracle-attacks-with-padbuster/>
- TBD: C. Madson and N. Doraswamy. The ESP DES-CBC Cipher With Explicit IV. RFC 2405, IETF, 1998.
- C. B. McCubbin, A. A. Selcuk, and D. Sidhu. Initialization vector attacks on the IPsec protocol suite. In 9th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2000), pages 171-175. IEEE Computer Society, 2000.
- A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. Handbook of Applied Cryptography. CRC Press, 1996.
- C. J. Mitchell. Error Oracle Attacks on CBC Mode: Is There a Future for CBC Mode Encryption? In J. Zhou, J. Lopez, R. H. Deng, and F. Bao, editors, Proceedings of ISC 2005, volume 3650 of LNCS, pages 244-258. Springer-Verlag, 2005.
- 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 LNCS, pages 305{323. Springer-Verlag, 2004
- Rizzo, J. Duongy, T., Practical Padding Oracle Attacks, Black Hat Europe, 2010, URL: https://media.blackhat.com/bh-eu-10/whitepapers/Duong_Rizzo/BlackHat-EU-2010-Duong-Rizzo-Padding-Oracle-wp.pdf
- B. Schneier. Applied Cryptography. John Wiley & Sons, second edition, 1996
- 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 LNCS, pages 534-545. Springer-Verlag, 2002
- Yau, A. K. L., Side Channel Analyses of CBC Mode Encryption, Ph.D. thesis, University of London, 2009

A. K. L. Yau, K. G. Paterson, and C. J. Mitchell. Padding Oracle Attacks on CBCMode Encryption with Secret and Random IVs. In H. Gilbert and H. Handschuh, editors, Proceedings of FSE 2005, volume 3557 of LNCS, pages 299-319. Springer-Verlag, 2005