



NTNU

Norwegian University of
Science and Technology

PADDING ORACLES: CBC AND SHA

TTM4205 – Lecture 7

Tjerand Silde

04.09.2026

Contents

Introduction

Padding Oracles

The CBC Mode

More CBC Problems

Length Extension Attack

Order of Enc and Auth

Summary

Contents

Introduction

Padding Oracles

The CBC Mode

More CBC Problems

Length Extension Attack

Order of Enc and Auth

Summary

Reference Material

These slides are based on:

- ▶ The referred papers in the slides
- ▶ JPA: parts of chapter 4, 6 and 7
- ▶ DW: parts of chapter 2 to 4

Contents

Introduction

Padding Oracles

The CBC Mode

More CBC Problems

Length Extension Attack

Order of Enc and Auth

Summary

Padding Oracles

By this we mean, on a high level, an API that allows an adversary to learn if some input is correctly formed.

We limit ourselves to inputs with a particular padding.

Padding Oracles

We will look at symmetric and asymmetric padding schemes:

- ▶ in depth on the CBC block cipher mode (today)
- ▶ extension attack against hashing (today)
- ▶ padding attacks against the RSA scheme (next)

Several of which are relevant to the weekly problems.

We will also look at some mitigations to these issues.

Contents

Introduction

Padding Oracles

The CBC Mode

More CBC Problems

Length Extension Attack

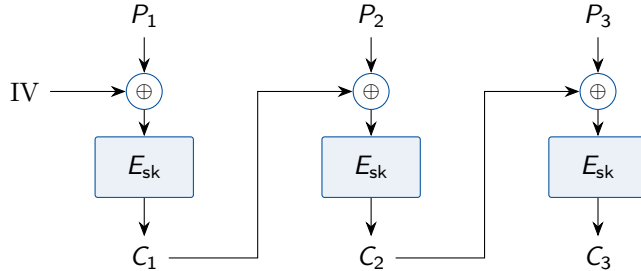
Order of Enc and Auth

Summary

The CBC Mode

- ▶ Cipher mode for symmetric ciphers (e.g. DES, AES)
- ▶ Proposed in 1976, proven IND-CPA-secure in 1997, broken 2002
- ▶ Secure against Chosen Plaintext Attacks (IND-CPA, theory)
- ▶ Not secure against Chosen Ciphertext Attacks (IND-CCA, practice)
- ▶ A variety of padding oracle attacks and patches in practice
- ▶ Removed from TLS in version 1.3 in 2018

CBC Encryption



CBC in Formulas

Let E_{sk} and D_{sk} be the block cipher and its inverse, and let the message be split into blocks $P_1, \dots, P_t$ after padding. With $C_0 = IV$ sampled at random:

$$\text{encryption:} \quad C_i = E_{sk}(P_i \oplus C_{i-1}),$$

$$\text{decryption:} \quad P_i = D_{sk}(C_i) \oplus C_{i-1},$$

for $i = 1, \dots, t$, and the ciphertext is $C_0 \parallel C_1 \parallel \dots \parallel C_t$.

Two things follow. Encryption is sequential while decryption is parallel, and each plaintext block depends on only two ciphertext blocks, so an adversary who controls C_{i-1} controls P_i bit by bit. That is the whole attack.

CBC Attack

- ▶ Each block must be of exactly 128 bits
- ▶ Shorter message leads to padding at the end
- ▶ Add one byte ends with 01, two with 02, etc. ...
- ▶ An API outputs errors when wrong padding

Question: What might an attacker do in this setting?

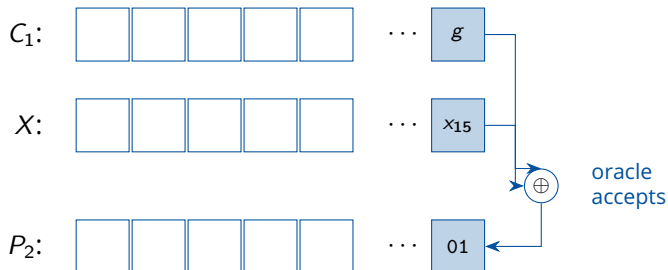
Why the Padding Leaks

Decryption of a two-block ciphertext $C_1 \parallel C_2$ gives $P_2 = D_{\text{sk}}(C_2) \oplus C_1$, and the receiver then strips the padding.

We cannot compute $D_{\text{sk}}(C_2)$, but we do not have to: every bit we flip in C_1 flips the same bit in P_2 . The oracle tells us whether the result ends in valid padding.

One accepted query therefore reveals one byte of $D_{\text{sk}}(C_2)$, and the real plaintext follows by XOR-ing with the ciphertext block that preceded C_2 .

One Query of the Attack



vary g until the oracle accepts, then $x_{15} = g \oplus 01$

$X = D_{sk}(C_2)$ is fixed by the key, and $P_2 = X \oplus C_1$ is what the receiver un pads. One bit of feedback pins one byte of X per 256 queries.

CBC Attack

- ▶ Choose our own C_1 and ask for $C_1 \parallel C_2$ to be decrypted
- ▶ The oracle accepts exactly when $C_1 \oplus X$ has valid padding, for $X = D_{sk}(C_2)$
- ▶ Vary the last byte of C_1 until it does: that pins the last byte of X
- ▶ Then set $C_1[15] = X[15] \oplus 02$ and vary $C_1[14]$, and so on
- ▶ At most $16 \cdot 256$ queries give X , and then $P_2 = X \oplus C_1^{\text{real}}$, block by block

Security Flaws Induced by CBC Padding Applications to 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 from communication protocols in a chosen ciphertext attack since the receiver usually sends an acknowledgment or an error message. This is a side channel.

In this paper we show various ways to perform an efficient side channel attack. We discuss potential applications, extensions to other padding schemes and various ways to fix the problem.

Figure: <https://www.iacr.org/cryptodb/archive/2002/EUROCRYPT/2850/2850.pdf>

Matthew Green in attacks ⌚ October 23, 2011 ☰ 2,314 Words

Attack of the week: XML Encryption

Figure: <https://blog.cryptographyengineering.com/2011/10/23/attack-of-week-xml-encryption>

Contents

Introduction

Padding Oracles

The CBC Mode

More CBC Problems

Length Extension Attack

Order of Enc and Auth

Summary

Authenticated CBC Mode

If we check that the CBC encryption was correctly computed, then we do not need to worry about the padding oracle.

Question: What are possible mitigations for CBC?

Possible Mitigations

- ▶ Compute a MAC on ptx (✗) or ctx (✓)
- ▶ Randomized padding scheme
- ▶ Fixed size padding scheme
- ▶ Additional randomized delay
- ▶ No specific error message

Question: What might go wrong in these cases?

Lucky 13

Removing the error message is not enough. TLS authenticates then encrypts, so the receiver strips the padding *before* checking the MAC, and the amount of data it then hashes depends on how much padding it removed.

More padding means fewer compression function calls, and the difference is visible in the response time: microseconds, recovered by averaging.

Making that path constant time is delicate enough that the real answer is to authenticate before touching the padding.

Lucky 13: Two Message Distributions

The next plot compares the decryption time for two chosen messages:

- ▶ M_0 : 32 arbitrary bytes followed by 256 copies of 0xFF
- ▶ M_1 : 287 arbitrary bytes followed by 0x00

Both are the same length, so they differ only in what the padding check removes: M_0 ends in what looks like 256 bytes of padding, M_1 in a single byte. The receiver then MACs 256 bytes less in the first case, which is four fewer compression function calls.

That difference is a few microseconds, and the two timing distributions separate clearly once enough samples are averaged.

Lucky 13

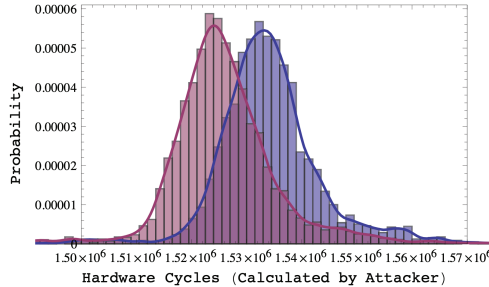


Figure 2. Distribution of timing values (outliers removed) for distinguishing attack on OpenSSL TLS, showing faster processing time in the case of M_0 (in red) compared to M_1 (in blue).

Figure: <https://www.ieee-security.org/TC/SP2013/papers/4977a526.pdf>

General Solutions

- ▶ Always use authenticated encryption
- ▶ Avoid CBC mode if possible (use GCM)
- ▶ Constant time padding check
- ▶ No specific error messages

2013 IEEE Symposium on Security and Privacy

Lucky Thirteen: Breaking the TLS and DTLS Record Protocols

Nadhem J. AlFardan and Kenneth G. Paterson

Information Security Group,

Royal Holloway, University of London

Egham, Surrey TW20 0EX, UK

Email: {nadhem.alfardan.2009, kenny.paterson}@rhul.ac.uk

Figure: <https://www.ieee-security.org/TC/SP2013/papers/4977a526.pdf>

Contents

Introduction

Padding Oracles

The CBC Mode

More CBC Problems

Length Extension Attack

Order of Enc and Auth

Summary

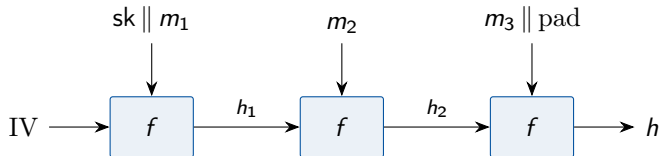
Hashing as MAC

Assume that we are in the following setting:

- ▶ Let sk be a fixed size secret
- ▶ Let m be a known message
- ▶ Let H be the SHA-2 hash function
- ▶ Let the MAC be $h = H(sk \parallel m)$

Question 1: Do you remember how SHA2 works?

SHA-2: Merkle-Damgård



The tag h we are given is the state after the last block. Nothing stops us from carrying on from there.

Hashing as MAC

Assume that we are in the following setting:

- ▶ Let sk be a fixed size secret
- ▶ Let m be a known message
- ▶ Let H be the SHA-2 hash function
- ▶ Let the MAC be $h = H(sk \parallel m)$

Question 2: How can we forge $h' = H(sk \parallel m')$?

Hashing as MAC

The issue is that SHA-2 applies a compression function to the message block by block, and appends *length padding* at the end: a 0x80 byte, zeros, and the total length of $sk \parallel m$ in bits.

The digest *is* the internal state after the last block. Knowing $|sk|$ we know that padding, so we load h as the state and keep hashing our own blocks, never learning the secret.

Can we remove the padding? No. The compression function has already consumed those bytes, so the tag we produce is valid for $sk \parallel m \parallel \text{pad} \parallel m'$ and for nothing shorter. The glue padding has to travel with the forged message.

Hashing as MAC

So the attacker does not get an arbitrary message: the original message, its padding, and only then the chosen suffix.

That is usually enough. In a query string `...&user=bob&role=guest`, the padding is a run of bytes the parser ignores, and `&role=admin` appended at the end simply overrides the earlier value.

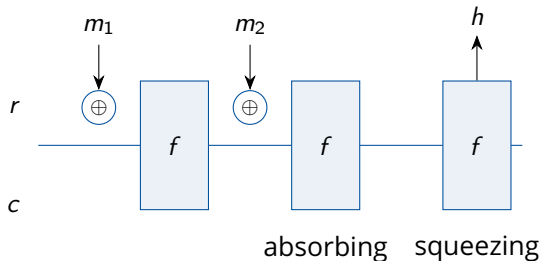
The format was never designed to survive trailing garbage, and the MAC was supposed to be what protected it. This is why $H(sk \parallel m)$ is not a MAC, and why every parser should reject what it does not understand.

Hashing as MAC

The attack applies to SHA-2 and to any Merkle-Damgård hash, but not to SHA-3, which is a sponge and keeps a state larger than its output.

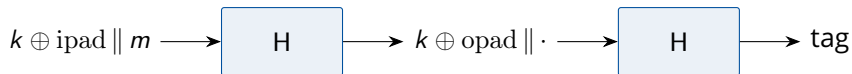
It does not apply to HMAC either: $H(k \oplus \text{opad} \parallel H(k \oplus \text{ipad} \parallel m))$ hashes twice, so the value we see is not the state of the inner hash.

SHA-3: The Sponge



The state is wider than the output: only the rate r is touched by the message and returned as digest, while the capacity c is never exposed. Knowing h therefore does not give the state, and the extension attack does not apply.

HMAC



The value we publish is the output of the *outer* hash, and the state of the inner one is never revealed. Extending the inner chain is therefore not possible, and the two nested keys also fix the length of what the outer hash consumes.

Length Extension in Practice

Length extension attack



Deep RnD · Follow

4 min read · Aug 16, 2019



12



1



What is length extension?

When a Merkle-Damgård based hash is misused as a message authentication code with construction $H(\text{secret} \parallel \text{message})$, and message and the length of secret is known, a length extension attack allows anyone to include extra information at the end of the message and produce a valid hash without knowing the secret. Quick sidebar, before you freak out:

Since HMAC does not use this construction, HMAC hashes are not prone to length extension attacks.

Figure: <https://deeprnd.medium.com/length-extension-attack-bff5b1ad2f70>



Contents

Introduction

Padding Oracles

The CBC Mode

More CBC Problems

Length Extension Attack

Order of Enc and Auth

Summary

Order of Enc and Auth

CPA secure symmetric crypto with input message m :

- ▶ SSL/TLS: $a = \text{Auth}_{sk_A}(m)$, $c = \text{Enc}_{sk_E}(a, m)$, send c
- ▶ IPsec: $c = \text{Enc}_{sk_E}(m)$, $a = \text{Auth}_{sk_A}(c)$, send (a, c)
- ▶ SSH: $c = \text{Enc}_{sk_E}(m)$, $a = \text{Auth}_{sk_A}(m)$, send (a, c)

Order of Enc and Auth

We refer to these methods as:

- ▶ SSL/TLS: authenticate-then-encrypt (AtE)
- ▶ IPSec: encrypt-then-authenticate (EtA)
- ▶ SSH: encrypt-and-authenticate (E&A)

Order of Enc and Auth

We refer to these methods as:

- ▶ SSL/TLS: authenticate-then-encrypt (AtE) (*can* be secure...)
- ▶ IPSec: encrypt-then-authenticate (EtA) (proven secure, ✓)
- ▶ SSH: encrypt-and-authenticate (E&A) (shown broken, ✗)

Interestingly, AtE is proven secure when using CBC mode.

The Order of Encryption and Authentication for Protecting Communications (Or: How Secure is SSL?)*

Hugo Krawczyk**

Figure: <https://iacr.org/archive/crypto2001/21390309.pdf>

Contents

Introduction

Padding Oracles

The CBC Mode

More CBC Problems

Length Extension Attack

Order of Enc and Auth

Summary

What to Remember

- ▶ A padding oracle is any distinguishable answer to a malformed ciphertext: an error, a log entry, or a delay
- ▶ CBC without authentication turns one bit into full decryption, byte by byte
- ▶ Removing the error message is not enough: the timing remains (Lucky 13)
- ▶ Encrypt-then-authenticate, and check the MAC before touching padding
- ▶ A Merkle-Damgård digest is the state, so $H(sk \parallel m)$ is not a MAC: use HMAC

Questions?