Padding Oracle Attacks
“How a 1-bit-leak completely breaks CBC-mode encryption”

Florian Weingarten\textsuperscript{1} \quad Alexander Neumann\textsuperscript{2}

\textsuperscript{1}weingarten@itsec.rwth-aachen.de
Research group IT-Security
RWTH Aachen University

\textsuperscript{2}alexander.neumann@redteam-pentesting.de
RedTeam Pentesting GmbH

5 August 2011
Motivation
Why you want to hear this talk.

Padding Oracle Attacks are ...

... an **efficient** way to use a specific 1-bit-leak to break **any** block cipher, as long as it’s operated in CBC-mode and used without proper authentication.

“Efficient” means linear in the number of ciphertext blocks and linear in the block length! In practice: Within minutes!

- Very nice idea, relatively easy to understand.
- Trivial to implement (even if you do not fully understand why it works).
- People don’t get that encryption and authentication should go hand in hand.
- Allows you to exploit many real-world applications.
- **Examples:** ASP.NET, JSF, Ruby on Rails, numerous custom-made stuff, ...
References


...
Outline

1. Introduction
2. Theory
3. Practice
4. Conclusions
Outline

1 Introduction

2 Theory

3 Practice

4 Conclusions
Cipher Block Chaining
Using block ciphers to encrypt long messages.

Block cipher

- Class of bijective mappings $E_K : \{0, 1\}^\ell \rightarrow \{0, 1\}^\ell$.
- $\ell$ is called the block length, usually a multiple of 8.
- **Examples**: DES ($\ell = 64$), AES ($\ell = 128$), Blowfish ($\ell = 64$), ...
Cipher Block Chaining
Using block ciphers to encrypt long messages.

Block cipher

- Class of bijective mappings \( E_K : \{0, 1\}^\ell \rightarrow \{0, 1\}^\ell \).
- \( \ell \) is called the block length, usually a multiple of 8.
- **Examples:** DES (\( \ell = 64 \)), AES (\( \ell = 128 \)), Blowfish (\( \ell = 64 \)), ...

How to encrypt \( M = (M_1, M_2, \ldots, M_k) \in \{0, 1\}^{k \cdot \ell} \)?

Cipher Block Chaining

- Choose some initialization vector \( C_0 \) (fixed, random, secret, ...)
- **Encryption:** \( C_i = E_K(M_i \oplus C_{i-1}) \) for \( 1 \leq i \leq k \).
- **Decryption:** \( M_i = E_K^{-1}(C_i) \oplus C_{i-1} \) for \( 1 \leq i \leq k \).

\( \oplus \) denotes addition in \( \mathbb{F}_2^\ell \), i.e., bitwise exclusive or on strings of length \( \ell \).
\( (x \oplus y \oplus y) = x \), i.e., addition in \( \mathbb{F}_2^\ell \) coincides with subtraction.
Message padding
Using block ciphers to encrypt variable-length messages.

Block ciphers encrypt \( M_1 \in \{0, 1\}^\ell \).
CBC encrypts \((M_1, \ldots, M_k) \in \{0, 1\}^{k \cdot \ell}\).
How to encrypt \( M \in \{0, 1\}^* \)?

Padding

- If the message length is not a multiple of the block length \( \ell \), use padding.
- Padding scheme must be injective (uniquely invertible).
Message padding
Using block ciphers to encrypt variable-length messages.

Block ciphers encrypt $M_1 \in \{0, 1\}^\ell$.
CBC encrypts $(M_1, \ldots, M_k) \in \{0, 1\}^{k \cdot \ell}$.
How to encrypt $M \in \{0, 1\}^*$?

Padding

- If the message length is not a multiple of the block length $\ell$, use padding.
- Padding scheme must be injective (uniquely invertible).

Notation: $b := \frac{\ell}{8}$ block length in bytes.

Example

- **Bit padding**: Append one 1 and as many 0 as necessary.
- **PKCS#5**: Append $n \geq 1$ bytes of value $n$ such that $|Mn\ldots n|$ is a multiple of $b$.
  - Variation: Append 0x01 0x02 0x03 $\ldots$ n.
  - Variation: Append 0x00 0x00 0x00 $\ldots$ n.
Let $\ell := 64$, i.e., $b = 8$.

**Valid paddings:**

<table>
<thead>
<tr>
<th>$0x12$</th>
<th>$0x34$</th>
<th>$0x56$</th>
<th>$0x05$</th>
<th>$0x05$</th>
<th>$0x05$</th>
<th>$0x05$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0xDE$</td>
<td>$0xAD$</td>
<td>$0xBE$</td>
<td>$0xAF$</td>
<td>$0x04$</td>
<td>$0x04$</td>
<td>$0x04$</td>
</tr>
<tr>
<td>$0x8B$</td>
<td>$0xAD$</td>
<td>$0xF0$</td>
<td>$0x0D$</td>
<td>$0xCA$</td>
<td>$0xFE$</td>
<td>$0xBA$</td>
</tr>
<tr>
<td>$0x08$</td>
<td>$0x08$</td>
<td>$0x08$</td>
<td>$0x08$</td>
<td>$0x08$</td>
<td>$0x08$</td>
<td>$0x08$</td>
</tr>
</tbody>
</table>

**Invalid paddings:**

<table>
<thead>
<tr>
<th>$0x12$</th>
<th>$0x34$</th>
<th>$0x56$</th>
<th>$0x78$</th>
<th>$0x05$</th>
<th>$0x05$</th>
<th>$0x05$</th>
<th>$0x05$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0xDE$</td>
<td>$0xAD$</td>
<td>$0xBE$</td>
<td>$0xAF$</td>
<td>$0x04$</td>
<td>$0x04$</td>
<td>$0x04$</td>
<td>$0x05$</td>
</tr>
<tr>
<td>$0x8B$</td>
<td>$0xAD$</td>
<td>$0xF0$</td>
<td>$0x0D$</td>
<td>$0xCA$</td>
<td>$0xFE$</td>
<td>$0xBA$</td>
<td>$0xBE$</td>
</tr>
</tbody>
</table>
Padding oracles
The basic idea.

**Theory**

A magic black box $\mathcal{P}$ which tells us, whether the plaintext of a given sequence of CBC-encrypted ciphertexts $(C_1, \ldots, C_k)$ has valid padding.
Introduction Theory Practice Conclusions Appendix

Padding oracles

The basic idea.

Theory

A magic black box $\mathcal{P}$ which tells us, whether the plaintext of a given sequence of CBC-encrypted ciphertexts $(C_1, \ldots, C_k)$ has valid padding.

Practice

- Client and server use CBC-mode encryption (connection, cookie, whatever, ...).
- **Question:** How should the server react if an encrypted message he receives is not correctly padded?
- If the server’s reaction is detectable (error message, abort, timing, ...), then this leaks 1 bit of information about the plaintext (i.e., is it padded correctly or not?).
### Theory

A magic black box $\mathcal{P}$ which tells us, whether the plaintext of a given sequence of CBC-encrypted ciphertexts $(C_1, \ldots, C_k)$ has valid padding.

### Practice

- **Client and server** use CBC-mode encryption (connection, cookie, whatever, ...).
- **Question:** How should the server react if an encrypted message he receives is not correctly padded?
- If the server’s reaction is detectable (error message, abort, timing, ...), then this leak *1 bit of information about the plaintext* (i.e., is it padded correctly or not?).

### Consequence

If no message authentication is used, this leak enables us to efficiently decrypt/encrypt any ciphertext/message of our choice.
Outline

1. Introduction
2. Theory
3. Practice
4. Conclusions
For all $1 \leq i \leq k$: $C_i = E_K(M_i \oplus C_{i-1})$ and $M_i = E_K^{-1}(C_i) \oplus C_{i-1}$.

**Observation**

Consider the case $k = 2$:

- $C_1 = E_K(M_1 \oplus C_0)$
- $C_2 = E_K(M_2 \oplus C_1)$
- $M_2 = E_K^{-1}(C_2) \oplus C_1$
- **We can change** $M_2$, **by changing** $C_1$.

**Recall**: $M_1M_2$ has valid padding iff $M_2$ ends with $0x01$, $0x0202$, ..., $0x0808080808080808$.
For all $1 \leq i \leq k$: $C_i = E_K(M_i \oplus C_{i-1})$ and $M_i = E_K^{-1}(C_i) \oplus C_{i-1}$.

**Observation**

Consider the case $k = 2$:

- $C_1 = E_K(M_1 \oplus C_0)$
- $C_2 = E_K(M_2 \oplus C_1)$
- $M_2 = E_K^{-1}(C_2) \oplus C_1$
- We can change $M_2$, by changing $C_1$.

**Recall:** $M_1M_2$ has valid padding iff $M_2$ ends with $0x01, 0x0202, \ldots, 0x0808080808080808$.

**Last Byte Oracle: Idea**

- For target ciphertext block $C_2 := y$, pick a random block $C_1 := r = (r_1, \ldots, r_b)$.
- Change the last byte of $C_1$ until $M_1M_2$ has valid padding, i.e., $P(ry) = 1$.
- If so, we know that $E_K^{-1}(y)$ ends with $r_b \oplus 0x01$ or $r_{b-1}r_b \oplus 0x0202$ or ...
\( \ell := 32 \), i.e., \( b = 4 \).

**Given:** Ciphertext block \( y = (y_1, y_2, y_3, y_4) \). **We want:** The last byte \( E_K^{-1}(y)_4 \) of the plaintext.

<table>
<thead>
<tr>
<th>( C_1 := r )</th>
<th>( C_2 := y )</th>
<th>( P(ry) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_1 \ r_2 \ r_3 \ r_4 \oplus 0x00 )</td>
<td>( y_1 \ y_2 \ y_3 \ y_4 )</td>
<td>( 0 )</td>
</tr>
<tr>
<td>( r_1 \ r_2 \ r_3 \ r_4 \oplus 0x01 )</td>
<td>( y_1 \ y_2 \ y_3 \ y_4 )</td>
<td>( 0 )</td>
</tr>
<tr>
<td>( \ldots )</td>
<td>( \ldots )</td>
<td>( \ldots )</td>
</tr>
<tr>
<td>( r_1 \ r_2 \ r_3 \ r_4 \oplus 0x17 )</td>
<td>( y_1 \ y_2 \ y_3 \ y_4 )</td>
<td>( 1 )</td>
</tr>
</tbody>
</table>

This will take at most \( 2^8 \) trys (\( 2^7 \) on average). Set \( r_4 := r_4 \oplus i \).
\[ \ell := 32, \text{i.e., } b = 4. \]

**Given:** Ciphertext block \( y = (y_1, y_2, y_3, y_4) \). **We want:** The last byte \( E_K^{-1}(y)_4 \) of the plaintext.

<table>
<thead>
<tr>
<th>( C_1 := r )</th>
<th>( C_2 := y )</th>
<th>( P(ry) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_1 ) ( r_2 ) ( r_3 ) ( r_4 \oplus 0x00 )</td>
<td>( y_1 ) ( y_2 ) ( y_3 ) ( y_4 )</td>
<td>0</td>
</tr>
<tr>
<td>( r_1 ) ( r_2 ) ( r_3 ) ( r_4 \oplus 0x01 )</td>
<td>( y_1 ) ( y_2 ) ( y_3 ) ( y_4 )</td>
<td>0</td>
</tr>
<tr>
<td>( \ldots )</td>
<td>( \ldots )</td>
<td>( \ldots )</td>
</tr>
<tr>
<td>( r_1 ) ( r_2 ) ( r_3 ) ( r_4 \oplus 0x17 )</td>
<td>( y_1 ) ( y_2 ) ( y_3 ) ( y_4 )</td>
<td>1</td>
</tr>
</tbody>
</table>

This will take at most \( 2^8 \) trys (\( 2^7 \) on average). Set \( r_4 := r_4 \oplus 1 \).

The plaintext \( M_1M_2 \) of \( ry \) has valid padding, so we have one of the following situations:

| \( E_K^{-1}(y) = M_2 \oplus r \) |
|----------------|----------------|----------------|
| \( * \) | \( * \) | \( * \) | \( r_4 \oplus 0x01 \) |
| \( * \) | \( * \) | \( r_3 \oplus 0x02 \) | \( r_4 \oplus 0x02 \) |
| \( * \) | \( r_2 \oplus 0x03 \) | \( r_3 \oplus 0x03 \) | \( r_4 \oplus 0x03 \) |
| \( r_1 \oplus 0x04 \) | \( r_2 \oplus 0x04 \) | \( r_3 \oplus 0x04 \) | \( r_4 \oplus 0x04 \) |

See appendix for probabilities of the various padding lengths.
Last Byte Oracle
What we got so far.

- We know that $E^{-1}_K(y)$ ends with one of
  - $r_b \oplus 0x01$
  - $r_{b-1} r_b \oplus 0x0202$
  - $r_{b-2} r_{b-1} r_b \oplus 0x030303$
  - ...

- We know all $r_i$ (we picked them ourselves, remember?).
- If we know which case it was, we successfully cracked (at least) one byte.
We know that $E_K^{-1}(y)$ ends with one of
- $r_b \oplus 0x01$
- $r_{b-1}r_b \oplus 0x0202$
- $r_{b-2}r_{b-1}r_b \oplus 0x030303$
- ...

We know all $r_i$ (we picked them ourselves, remember?).

If we know which case it was, we successfully cracked (at least) one byte.

But this is easy!

Go from left to right, change one byte and ask the oracle again.

If we still have $P(ry) = 1$, the byte we changed was not a padding byte.

If we have $P(ry) = 0$, the number of tries gives us the padding length.
Last Byte Oracle

Algorithm

**Input:** Target ciphertext block $y \in \{0, 1\}^{8b}$.

1. Pick random $r := (r_1, ..., r_b) \in \{0, 1\}^{8b}$, set $i := 0$.
2. Set $r_b := r_b \oplus i$.
3. If $P(ry) = 0$, set $i := i + 1$ and go to 2. Otherwise, set $r_b := r_b \oplus i$ and continue.
4. For $i := b$ down to 2 do
   (a) Set $r := (r_1, ..., r_b-i, r_b-i+1 \oplus 0x42, r_b-i+2, ..., r_b)$.
   (b) If $P(ry) = 0$, output $(r_b-i+1 \oplus i) ... (r_b \oplus i)$ and stop.
5. Output $r_b \oplus 1$. 
Algorithm

**Input:** Target ciphertext block \( y \in \{0, 1\}^{8b} \).

1. Pick random \( r := (r_1, ..., r_b) \in \{0, 1\}^{8b} \), set \( i := 0 \).
2. Set \( r_b := r_b \oplus i \).
3. If \( P(ry) = 0 \), set \( i := i + 1 \) and go to 2. Otherwise, set \( r_b := r_b \oplus i \) and continue.
4. For \( i := b \) down to 2 do
   (a) Set \( r := (r_1, ..., r_{b-i}, r_{b-i+1} \oplus 0x42, r_{b-i+2}, ..., r_b) \).
   (b) If \( P(ry) = 0 \), output \((r_{b-i+1} \oplus i)...(r_b \oplus i)\) and stop.
5. Output \( r_b \oplus 1 \).

Discussion

- 1 to 3 find a ciphertext \( r \) such that the plaintext of \( ry \) has valid padding.
- 4 and 5 determine the padding length (and thus at least one byte of plaintext!).
- **Worst case:** \( 256 + b - 1 \) oracle calls (less than \( 128 + b \) on average).
- (Observation: At no point do we care what block cipher is used!)
We can use $\mathcal{P}$ to obtain the last byte of $E_K^{-1}(y)$. How to get the rest?

**Idea:** Iterate the Last Byte Algorithm.

- Let $a = a_1...a_b = E_K^{-1}(y)$. Assume we already cracked $a_j...a_b$ for some $1 < j \leq b$.
- We know $b - j + 1$ bytes.
We can use $P$ to obtain the last byte of $E_K^{-1}(y)$. How to get the rest?

**Idea:** Iterate the Last Byte Algorithm.

- Let $a = a_1...a_b = E_K^{-1}(y)$. Assume we already cracked $a_j...a_b$ for some $1 < j \leq b$.
- We know $b - j + 1$ bytes.
- Pick random $C_1 := r$ such that $M_2$ will decrypt to $b - j + 1$ bytes of value $b - j + 2$. 
We can use $\mathcal{P}$ to obtain the last byte of $E^{-1}_K(y)$. How to get the rest?

Idea: Iterate the Last Byte Algorithm.

- Let $a = a_1...a_b = E^{-1}_K(y)$. Assume we already cracked $a_j...a_b$ for some $1 < j \leq b$.
- We know $b - j + 1$ bytes.
- Pick random $C_1 := r$ such that $M_2$ will decrypt to $b - j + 1$ bytes of value $b - j + 2$.
- We can easily construct such an $r$ by taking $r_i := a_i \oplus (b - j + 2)$ for $i \geq b - j + 1$. 
We can use $\mathcal{P}$ to obtain the last byte of $E_K^{-1}(y)$. How to get the rest?

**Idea:** Iterate the Last Byte Algorithm.

- Let $a = a_1...a_b = E_K^{-1}(y)$. Assume we already cracked $a_j...a_b$ for some $1 < j \leq b$.
- We know $b - j + 1$ bytes.
- Pick random $C_1 := r$ such that $M_2$ will decrypt to $b - j + 1$ bytes of value $b - j + 2$.
- We can easily construct such an $r$ by taking $r_i := a_i \oplus (b - j + 2)$ for $i \geq b - j + 1$.
- **Example:** If $b = 4$ and we know the last 3 bytes, pick

$$r := (r_1, a_2 \oplus 0x04, a_3 \oplus 0x04, a_4 \oplus 0x04).$$
We can use \( P \) to obtain the last byte of \( E_K^{-1}(y) \). How to get the rest?

**Idea:** Iterate the Last Byte Algorithm.

- Let \( a = a_1 \ldots a_b = E_K^{-1}(y) \). Assume we already cracked \( a_j \ldots a_b \) for some \( 1 < j \leq b \).
- We know \( b - j + 1 \) bytes.
- Pick random \( C_1 := r \) such that \( M_2 \) will decrypt to \( b - j + 1 \) bytes of value \( b - j + 2 \).
- We can easily construct such an \( r \) by taking \( r_i := a_i \oplus (b - j + 2) \) for \( i \geq b - j + 1 \).
- **Example:** If \( b = 4 \) and we know the last 3 bytes, pick

\[
  r := (r_1, a_2 \oplus 0x04, a_3 \oplus 0x04, a_4 \oplus 0x04).
\]

\[ E_K^{-1}(y) = (a_1, a_2, a_3, a_4). \]
We can use $P$ to obtain the last byte of $E_K^{-1}(y)$. How to get the rest?

**Idea:** Iterate the Last Byte Algorithm.

- Let $a = a_1...a_b = E_K^{-1}(y)$. Assume we already cracked $a_j...a_b$ for some $1 < j \leq b$.
- We know $b - j + 1$ bytes.
- Pick random $C_1 := r$ such that $M_2$ will decrypt to $b - j + 1$ bytes of value $b - j + 2$.
- We can easily construct such an $r$ by taking $r_i := a_i \oplus (b - j + 2)$ for $i \geq b - j + 1$.
- **Example:** If $b = 4$ and we know the last 3 bytes, pick

\[
r := (r_1, a_2 \oplus 0x04, a_3 \oplus 0x04, a_4 \oplus 0x04).
\]

- $E_K^{-1}(y) = (a_1, a_2, a_3, a_4)$.
- $M_2 = E_K^{-1}(y) \oplus r = (a_1 \oplus r_1, a_2 \oplus a_2 \oplus 0x04, a_3 \oplus a_3 \oplus 0x04, a_4 \oplus a_4 \oplus 0x04)$, so the plaintext suffix $a_j, ..., a_b$ and $r_j, ..., r_b$ will cancel to $b - j + 2$. 
We can use $\mathcal{P}$ to obtain the last byte of $E_K^{-1}(y)$. How to get the rest?

**Idea:** Iterate the Last Byte Algorithm.

- Let $a = a_1...a_b = E_K^{-1}(y)$. Assume we already cracked $a_j...a_b$ for some $1 < j \leq b$.
- We know $b - j + 1$ bytes.
- Pick random $C_1 := r$ such that $M_2$ will decrypt to $b - j + 1$ bytes of value $b - j + 2$.
- We can easily construct such an $r$ by taking $r_i := a_i \oplus (b - j + 2)$ for $i \geq b - j + 1$.
- **Example:** If $b = 4$ and we know the last 3 bytes, pick

$$r := (r_1, a_2 \oplus 0x04, a_3 \oplus 0x04, a_4 \oplus 0x04).$$

- $E_K^{-1}(y) = (a_1, a_2, a_3, a_4)$.
- $M_2 = E_K^{-1}(y) \oplus r = (a_1 \oplus r_1, a_2 \oplus a_2 \oplus 0x04, a_3 \oplus a_3 \oplus 0x04, a_4 \oplus a_4 \oplus 0x04)$, so the plaintext suffix $a_j, ..., a_b$ and $r_j, ..., r_b$ will cancel to $b - j + 2$.
- Last Byte Algorithm (starting with $r_{j-1}$) will find another byte (since we already prepared an “almost valid” padding of length $b - j + 2$).
- We need another 256 queries for that (128 on average).
Let again $\ell := 32$, so $b = 4$ bytes for each block.
Target ciphertext block $y = (y_1, y_2, y_3, y_4) = E_K(a_1, a_2, a_3, a_4) \in \{0, 1\}^\ell$.

**Step 1:** Finding $C_1 C_2 := ry$ such that $M_1 M_2$ has valid padding.
Let again $\ell := 32$, so $b = 4$ bytes for each block. Target ciphertext block $y = (y_1, y_2, y_3, y_4) = E_K(a_1, a_2, a_3, a_4) \in \{0, 1\}^\ell$.

<table>
<thead>
<tr>
<th>$r$</th>
<th>$P(ry)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1 \oplus 0x00$</td>
<td></td>
</tr>
<tr>
<td>$r_2 \oplus 0x00$</td>
<td></td>
</tr>
<tr>
<td>$r_3 \oplus 0x00$</td>
<td></td>
</tr>
<tr>
<td>$r_4 \oplus 0x00$</td>
<td></td>
</tr>
</tbody>
</table>

**Step 1:** Finding $C_1C_2 := ry$ such that $M_1M_2$ has valid padding.

1. Pick a random ciphertext $r = (r_1, r_2, r_3, r_4)$. 
Let again $\ell := 32$, so $b = 4$ bytes for each block.
Target ciphertext block $y = (y_1, y_2, y_3, y_4) = E_K(a_1, a_2, a_3, a_4) \in \{0, 1\}^\ell$.

<table>
<thead>
<tr>
<th>$r$</th>
<th>$\mathcal{P}(ry)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1 \oplus 0x00$</td>
<td>$r_2 \oplus 0x00$</td>
</tr>
</tbody>
</table>

**Step 1:** Finding $C_1 C_2 := ry$ such that $M_1 M_2$ has valid padding.

1. Pick a random ciphertext $r = (r_1, r_2, r_3, r_4)$.
2. Check the oracle response $\mathcal{P}(ry)$. 
Let again $\ell := 32$, so $b = 4$ bytes for each block. Target ciphertext block $y = (y_1, y_2, y_3, y_4) = E_K(a_1, a_2, a_3, a_4) \in \{0, 1\}^\ell$.

<table>
<thead>
<tr>
<th>$r$</th>
<th>$P(ry)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1 \oplus 0x00$</td>
<td></td>
</tr>
<tr>
<td>$r_2 \oplus 0x00$</td>
<td></td>
</tr>
<tr>
<td>$r_3 \oplus 0x00$</td>
<td></td>
</tr>
<tr>
<td>$r_4 \oplus 0x01$</td>
<td>$0$</td>
</tr>
</tbody>
</table>

**Step 1:** Finding $C_1C_2 := ry$ such that $M_1M_2$ has valid padding.

1. Pick a random ciphertext $r = (r_1, r_2, r_3, r_4)$.
2. Check the oracle response $P(ry)$.
3. Change $r_4$ until we hit $P(ry) = 1$. 
Let again $\ell := 32$, so $b = 4$ bytes for each block.
Target ciphertext block $y = (y_1, y_2, y_3, y_4) = E_K(a_1, a_2, a_3, a_4) \in \{0, 1\}^\ell$.

<table>
<thead>
<tr>
<th>$r$</th>
<th>$\mathcal{P}(ry)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1 \oplus 0x00$</td>
<td>$r_2 \oplus 0x00$</td>
</tr>
</tbody>
</table>

**Step 1:** Finding $C_1 C_2 := ry$ such that $M_1 M_2$ has valid padding.

1. Pick a random ciphertext $r = (r_1, r_2, r_3, r_4)$.
2. Check the oracle response $\mathcal{P}(ry)$.
3. Change $r_4$ until we hit $\mathcal{P}(ry) = 1$. 
Let again $\ell := 32$, so $b = 4$ bytes for each block.
Target ciphertext block $y = (y_1, y_2, y_3, y_4) = E_K(a_1, a_2, a_3, a_4) \in \{0, 1\}^\ell$.

<table>
<thead>
<tr>
<th>$r$</th>
<th>$\mathcal{P}(ry)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1 \oplus 0x00$</td>
<td>$r_2 \oplus 0x00$</td>
</tr>
</tbody>
</table>

**Step 1:** Finding $C_1C_2 := ry$ such that $M_1M_2$ has valid padding.
1. Pick a random ciphertext $r = (r_1, r_2, r_3, r_4)$.
2. Check the oracle response $\mathcal{P}(ry)$.
3. Change $r_4$ until we hit $\mathcal{P}(ry) = 1$. 
Block Decryption Oracle

Example

Let again $\ell := 32$, so $b = 4$ bytes for each block.
Target ciphertext block $y = (y_1, y_2, y_3, y_4) = E_K(a_1, a_2, a_3, a_4) \in \{0, 1\}^\ell$.

<table>
<thead>
<tr>
<th>$r$</th>
<th>$\mathcal{P}(ry)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1 \oplus 0x00$</td>
<td>$r_2 \oplus 0x00$</td>
</tr>
<tr>
<td>$r_3 \oplus 0x00$</td>
<td>$r_4 \oplus \ldots$</td>
</tr>
<tr>
<td>$0$</td>
<td></td>
</tr>
</tbody>
</table>

**Step 1:** Finding $C_1 C_2 := ry$ such that $M_1 M_2$ has valid padding.
1. Pick a random ciphertext $r = (r_1, r_2, r_3, r_4)$.
2. Check the oracle response $\mathcal{P}(ry)$.
3. Change $r_4$ until we hit $\mathcal{P}(ry) = 1$. 
Let again $\ell := 32$, so $b = 4$ bytes for each block.

Target ciphertext block $y = (y_1, y_2, y_3, y_4) = E_K(a_1, a_2, a_3, a_4) \in \{0, 1\}^\ell$.

<table>
<thead>
<tr>
<th>$r$</th>
<th>$\mathcal{P}(ry)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1 \oplus 0x00$</td>
<td>$r_2 \oplus 0x00$</td>
</tr>
</tbody>
</table>

**Step 1:** Finding $C_1C_2 := ry$ such that $M_1M_2$ has valid padding.

1. Pick a random ciphertext $r = (r_1, r_2, r_3, r_4)$.
2. Check the oracle response $\mathcal{P}(ry)$.
3. Change $r_4$ until we hit $\mathcal{P}(ry) = 1$. 
Block Decryption Oracle

Example

Let again $\ell := 32$, so $b = 4$ bytes for each block. Target ciphertext block $y = (y_1, y_2, y_3, y_4) = E_K(a_1, a_2, a_3, a_4) \in \{0, 1\}^\ell$.

<table>
<thead>
<tr>
<th>$r$</th>
<th>$\mathcal{P}(ry)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1 \oplus 0x00$</td>
<td>$\mathcal{P}(ry)$</td>
</tr>
<tr>
<td>$r_2 \oplus 0x00$</td>
<td>$\mathcal{P}(ry)$</td>
</tr>
<tr>
<td>$r_3 \oplus 0x00$</td>
<td>$\mathcal{P}(ry)$</td>
</tr>
<tr>
<td>$r_4 \oplus 0x9F$</td>
<td>$1$</td>
</tr>
</tbody>
</table>

Step 1: Finding $C_1C_2 := ry$ such that $M_1M_2$ has valid padding.

1. Pick a random ciphertext $r = (r_1, r_2, r_3, r_4)$.
2. Check the oracle response $\mathcal{P}(ry)$.
3. Change $r_4$ until we hit $\mathcal{P}(ry) = 1$.

- Padding is valid!
Let again $\ell := 32$, so $b = 4$ bytes for each block. Target ciphertext block $y = (y_1, y_2, y_3, y_4) = E_K(a_1, a_2, a_3, a_4) \in \{0, 1\}^\ell$.

<table>
<thead>
<tr>
<th>$r$</th>
<th>$\mathcal{P}(ry)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1 \oplus 0x00$</td>
<td>$r_2 \oplus 0x00$</td>
</tr>
<tr>
<td>$r_3 \oplus 0x00$</td>
<td>$r_4 \oplus 0x9F$</td>
</tr>
</tbody>
</table>

**Step 2:** Determine the padding length.
Let again \( \ell := 32 \), so \( b = 4 \) bytes for each block.
Target ciphertext block \( y = (y_1, y_2, y_3, y_4) = E_K(a_1, a_2, a_3, a_4) \in \{0, 1\}^\ell \).

<table>
<thead>
<tr>
<th>( r )</th>
<th>( P(ry) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_1 \oplus \text{0x42} )</td>
<td>( r_2 \oplus \text{0x00} )</td>
</tr>
</tbody>
</table>

**Step 2:** Determine the padding length.

1. Change bytes from the left.
Let again $\ell := 32$, so $b = 4$ bytes for each block. Target ciphertext block $y = (y_1, y_2, y_3, y_4) = E_K(a_1, a_2, a_3, a_4) \in \{0, 1\}^\ell$.

<table>
<thead>
<tr>
<th>$r$</th>
<th>$P(ry)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1 \oplus 0x42$</td>
<td></td>
</tr>
<tr>
<td>$r_2 \oplus 0x00$</td>
<td></td>
</tr>
<tr>
<td>$r_3 \oplus 0x00$</td>
<td></td>
</tr>
<tr>
<td>$r_4 \oplus 0x9F$</td>
<td></td>
</tr>
</tbody>
</table>

### Step 2: Determine the padding length.

1. Change bytes from the left.
2. Check if padding is still valid.
Let again $\ell := 32$, so $b = 4$ bytes for each block.

Target ciphertext block $y = (y_1, y_2, y_3, y_4) = E_K(a_1, a_2, a_3, a_4) \in \{0, 1\}^\ell$.

<table>
<thead>
<tr>
<th>$r$</th>
<th>$P(ry)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1 \oplus 0x00$</td>
<td></td>
</tr>
<tr>
<td>$r_2 \oplus 0x42$</td>
<td></td>
</tr>
<tr>
<td>$r_3 \oplus 0x00$</td>
<td></td>
</tr>
<tr>
<td>$r_4 \oplus 0x9F$</td>
<td>1</td>
</tr>
</tbody>
</table>

**Step 2:** Determine the padding length.

1. **Change bytes from the left.**
2. **Check if padding is still valid.**
3. **Repeat until padding breaks.**
Let again $\ell := 32$, so $b = 4$ bytes for each block. 
Target ciphertext block $y = (y_1, y_2, y_3, y_4) = E_K(a_1, a_2, a_3, a_4) \in \{0, 1\}^\ell$.

<table>
<thead>
<tr>
<th>$r$</th>
<th>$\mathcal{P}(ry)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1 \oplus 0x00$</td>
<td></td>
</tr>
<tr>
<td>$r_2 \oplus 0x00$</td>
<td></td>
</tr>
<tr>
<td>$r_3 \oplus 0x42$</td>
<td></td>
</tr>
<tr>
<td>$r_4 \oplus 0x9F$</td>
<td>0</td>
</tr>
</tbody>
</table>

**Step 2:** Determine the padding length.

1. Change bytes from the left.
2. Check if padding is still valid.
3. Repeat until padding breaks.

- We broke the padding by changing byte $r_{2+1}$. Therefore, it has length $4 - 2 = 2$. 
Let again $\ell := 32$, so $b = 4$ bytes for each block. Target ciphertext block $y = (y_1, y_2, y_3, y_4) = E_K(a_1, a_2, a_3, a_4) \in \{0, 1\}^\ell$.

<table>
<thead>
<tr>
<th>$r$</th>
<th>$\mathcal{P}(ry)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1 \oplus 0x00$</td>
<td>$r_2 \oplus 0x00$</td>
</tr>
</tbody>
</table>

**Step 2:** Determine the padding length.

1. Change bytes from the left.
2. Check if padding is still valid.
3. Repeat until padding breaks.

- We broke the padding by changing byte $r_{2+1}$. Therefore, it has length $4 - 2 = 2$.
- We just learned two bytes of plaintext: $r_3 \oplus 0x02$ and $(r_4 \oplus 0x9F) \oplus 0x02$. 
  
  $=a_3$  
  $=a_4$
Let again $\ell := 32$, so $b = 4$ bytes for each block. Target ciphertext block $y = (y_1, y_2, y_3, y_4) = E_K(a_1, a_2, a_3, a_4) \in \{0, 1\}^\ell$.

<table>
<thead>
<tr>
<th>$r$</th>
<th>$\mathcal{P}(ry)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1 \oplus 0x00$</td>
<td>$r_2 \oplus 0x00$</td>
</tr>
<tr>
<td>$r_3 \oplus 0x00$</td>
<td>$r_4 \oplus 0x9F$</td>
</tr>
</tbody>
</table>

Step 3: Crack the next byte.
Let again $\ell := 32$, so $b = 4$ bytes for each block. Target ciphertext block $y = (y_1, y_2, y_3, y_4) = E_K(a_1, a_2, a_3, a_4) \in \{0, 1\}^\ell$.

<table>
<thead>
<tr>
<th>$r$</th>
<th>$\mathcal{P}(ry)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1 \oplus 0x00$</td>
<td>$r_2 \oplus 0x00$</td>
</tr>
<tr>
<td>$a_3 \oplus 0x03$</td>
<td>$a_4 \oplus 0x03$</td>
</tr>
</tbody>
</table>

**Step 3**: Crack the next byte.

1. Pick random block with a suffix that will decrypt to an “almost valid” padding.

$M_2 = E_K^{-1}(y) \oplus r$, so the last two bytes of $M_2$ are now $a_3 \oplus a_3 \oplus 0x03$ and $a_4 \oplus a_4 \oplus 0x03$. $=0x00$ $=0x00$
Let again $\ell := 32$, so $b = 4$ bytes for each block.

Target ciphertext block $y = (y_1, y_2, y_3, y_4) = E_K(a_1, a_2, a_3, a_4) \in \{0, 1\}^\ell$.

<table>
<thead>
<tr>
<th>$r$</th>
<th>$\mathcal{P}(ry)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1 \oplus 0x00$</td>
<td>0</td>
</tr>
<tr>
<td>$r_2 \oplus 0x00$</td>
<td>0</td>
</tr>
<tr>
<td>$a_3 \oplus 0x03$</td>
<td>0</td>
</tr>
<tr>
<td>$a_4 \oplus 0x03$</td>
<td>0</td>
</tr>
</tbody>
</table>

**Step 3:** Crack the next byte.

1. Pick random block with a suffix that will decrypt to an “almost valid” padding.
2. Start the whole thing over again, starting with byte $r_2$. 
Block Decryption Oracle

Example

Let again $\ell := 32$, so $b = 4$ bytes for each block. Target ciphertext block $y = (y_1, y_2, y_3, y_4) = E_K(a_1, a_2, a_3, a_4) \in \{0, 1\}^\ell$.

<table>
<thead>
<tr>
<th>r</th>
<th>$P(ry)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1 \oplus 0x00$</td>
<td>$r_2 \oplus 0x01$</td>
</tr>
</tbody>
</table>

Step 3: Crack the next byte.

1. Pick random block with a suffix that will decrypt to an “almost valid” padding.
2. Start the whole thing over again, starting with byte $r_2$.
3. Iterate until padding is valid.
Let again \( \ell := 32 \), so \( b = 4 \) bytes for each block. 
Target ciphertext block \( y = (y_1, y_2, y_3, y_4) = E_K(a_1, a_2, a_3, a_4) \in \{0, 1\}^\ell \).

<table>
<thead>
<tr>
<th>r</th>
<th>( \mathcal{P}(ry) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_1 \oplus 0x00 )</td>
<td>( r_2 \oplus 0x02 )</td>
</tr>
</tbody>
</table>

**Step 3:** Crack the next byte.

1. Pick random block with a suffix that will decrypt to an “almost valid” padding.
2. Start the whole thing over again, starting with byte \( r_2 \).
3. **Iterate until padding is valid.**
Let again $\ell := 32$, so $b = 4$ bytes for each block.

Target ciphertext block $y = (y_1, y_2, y_3, y_4) = E_K(a_1, a_2, a_3, a_4) \in \{0, 1\}^\ell$.

$$
\begin{array}{|c|c|}
\hline
r & \mathcal{P}(ry) \\
\hline
r_1 \oplus 0x00 & r_2 \oplus \ldots & a_3 \oplus 0x03 & a_4 \oplus 0x03 & 0 \\
\hline
\end{array}
$$

**Step 3:** Crack the next byte.

1. Pick random block with a suffix that will decrypt to an “almost valid” padding.
2. Start the whole thing over again, starting with byte $r_2$.
3. Iterate until padding is valid.
Let again $\ell := 32$, so $b = 4$ bytes for each block. Target ciphertext block $y = (y_1, y_2, y_3, y_4) = E_K(a_1, a_2, a_3, a_4) \in \{0, 1\}^\ell$.

<table>
<thead>
<tr>
<th>$r$</th>
<th>$\mathcal{P}(ry)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1 \oplus 0x00$</td>
<td>$r_2 \oplus 0xA1$</td>
</tr>
</tbody>
</table>

**Step 3:** Crack the next byte.

1. Pick random block with a suffix that will decrypt to an “almost valid” padding.
2. Start the whole thing over again, starting with byte $r_2$.
3. Iterate until padding is valid.
Let again $\ell := 32$, so $b = 4$ bytes for each block.

Target ciphertext block $y = (y_1, y_2, y_3, y_4) = E_K(a_1, a_2, a_3, a_4) \in \{0, 1\}^\ell$.

<table>
<thead>
<tr>
<th>$r$</th>
<th>$P(ry)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1 \oplus 0x00$</td>
<td></td>
</tr>
<tr>
<td>$r_2 \oplus 0xA2$</td>
<td></td>
</tr>
<tr>
<td>$a_3 \oplus 0x03$</td>
<td></td>
</tr>
<tr>
<td>$a_4 \oplus 0x03$</td>
<td></td>
</tr>
<tr>
<td>$1$</td>
<td></td>
</tr>
</tbody>
</table>

**Step 3:** Crack the next byte.

1. Pick random block with a suffix that will decrypt to an “almost valid” padding.
2. Start the whole thing over again, starting with byte $r_2$.
3. Iterate until padding is valid.

- **Valid padding!**
Let again $\ell := 32$, so $b = 4$ bytes for each block.
Target ciphertext block $y = (y_1, y_2, y_3, y_4) = E_K(a_1, a_2, a_3, a_4) \in \{0, 1\}^\ell$.

<table>
<thead>
<tr>
<th>$r$</th>
<th>$\mathcal{P}(ry)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1 \oplus 0x00$</td>
<td>$r_2 \oplus 0xA2$</td>
</tr>
</tbody>
</table>

**Step 3:** Crack the next byte.

1. Pick random block with a suffix that will decrypt to an “almost valid” padding.
2. Start the whole thing over again, starting with byte $r_2$.
3. Iterate until padding is valid.

- Valid padding!
- This time, we can be sure that it is $0x03$.
  So we just learned $a_2 = r_2 \oplus 0xA2 \oplus 0x03$. 
Decryption Oracle
From single blocks to sequences of blocks.

- We can use $\mathcal{P}$ to obtain the last byte of $E_K^{-1}(y)$.
- Also, we can iterate this to obtain all bytes of $E_K^{-1}(y)$. 
We can use $\mathcal{P}$ to obtain the last byte of $E_K^{-1}(y)$.

Also, we can iterate this to obtain all bytes of $E_K^{-1}(y)$.

What about ciphertexts consisting of multiple blocks?
- We can use $\mathcal{P}$ to obtain the last byte of $E_K^{-1}(y)$.
- Also, we can iterate this to obtain all bytes of $E_K^{-1}(y)$.
- What about ciphertexts consisting of multiple blocks?
- CBC decryption: $M_i = E_K^{-1}(C_i) \oplus C_{i-1}$. 

Decryption Oracle

From single blocks to sequences of blocks.
Decryption Oracle
From single blocks to sequences of blocks.

- We can use \( P \) to obtain the last byte of \( E^{-1}_K(y) \).
- Also, we can iterate this to obtain all bytes of \( E^{-1}_K(y) \).
- What about ciphertexts consisting of multiple blocks?
- CBC decryption: \( M_i = E^{-1}_K(C_i) \oplus C_{i-1} \).
- Get \( E^{-1}_K(C_i) \) using the Block Decryption Algorithm, the rest is cake.
- **Worst case**: \((256 + b - 1) + 256(b - 1)(N - 1) \leq 256Nb\) oracle queries (if we are extremely unlucky).
- **For example**: Less than \(2048 \cdot N\) queries for AES-CBC on average.
We can use $\mathcal{P}$ to obtain the last byte of $E_K^{-1}(y)$.

Also, we can iterate this to obtain all bytes of $E_K^{-1}(y)$.

What about ciphertexts consisting of multiple blocks?

CBC decryption: $M_i = E_K^{-1}(C_i) \oplus C_{i-1}$.

Get $E_K^{-1}(C_i)$ using the Block Decryption Algorithm, the rest is cake.

**Worst case:** $(256 + b - 1) + 256(b - 1)(N - 1) \leq 256Nb$ oracle queries (if we are extremely unlucky).

**For example:** Less than $2048 \cdot N$ queries for AES-CBC on average.

To decrypt $C_1$, we need the initialization vector $C_0$.

$C_0$ is often attached to the ciphertext (it is usually considered safe to publish it).

(If security depends on the secrecy of $C_0$, the system is flawed anyways.)
CBC-R

Turning *decryption* oracles into *encryption* oracles.

**CBC-R encryption**

- Introduced by Rizzo and Duong in 2010 (*Practical Padding Oracle Attacks*).
- Idea works with every decryption oracle (not just our padding oracles).

**Turning CBC decryption around**

- \( M_i = E_K^{-1}(C_i) \oplus C_{i-1} \) with \( C_0 = IV \). (\( \Rightarrow C_{i-1} = M_i \oplus E_K^{-1}(C_i) \))
- Plaintext depends on current ciphertext and the previous one.
- **Important property:** Changing one ciphertext affects all following ones.
- If we know \( E_K^{-1}(C_i) \) and control \( C_{i-1} \), we can change \( M_i \).
- **This means:** If we want to generate a ciphertext \( C_i \) for some \( M_i \) of our choice, all we have to do is to change the previous one \( C_{i-1} \) accordingly.
- This will turn the plaintext of the previous block into gibberish.
- That’s ok, we iteratively fix every block.
- If we can’t control \( C_0 = IV \), we don’t own the first block :-(
Outline

1 Introduction

2 Theory

3 Practice

4 Conclusions

F. Weingarten, A. Neumann — Padding Oracle Attacks
Alexander Neumann

Working for RedTeam Pentesting GmbH (Aachen).
★ Founded in 2004.
★ Specialized in penetration tests.
★ We only do penetration tests and (a bit) security research.
Padding Oracles in practice

- Prequel: How to identify encrypted data in the wild?
- How do padding oracles look like?
- How to test for an (assumed) padding oracle?
- How to exploit padding oracles?
Encryption in the wild

How to identify encrypted data in the wild?

★ **Answer:** Search for long and random looking strings.
★ Probably base64- or URL-encoded.

VGhpcyBsb29rcyBzdXNwaWNpb3VzCg==
%A9E%B0%D2%8B.%3EK%A2%A2q%14%10%3Dt

★ Tinker with it!
★ See how the application reacts to:
  ★ Changing the last byte.
  ★ Duplicating the last two blocks (⇒ blocksize).
  ★ ...
Encryption in the wild

How to identify encrypted data in the wild?

★ **Answer:** Search for long and random looking strings.

★ Probably base64- or URL-encoded.

VGhpcyBsb29rcyBzdXNwaWNpb3VzCg==
%A9E%B0%D2%8B.%3EK%A2%A2q%14%10%3Dt

★ Tinker with it!

★ See how the application reacts to:
  ★ Changing the last byte.
  ★ Duplicating the last two blocks (⇒ blocksize).
  ★ ...
How do Padding Oracles look like?

- **Classic**: Stacktrace with padding-exception (Java, Ruby).
  - `javax.crypto.BadPaddingException`:
    - Given final block not properly padded
  - `OpenSSL::Cipher::CipherError` in ` ApplicationController`:
    - bad decrypt

- **Timing**: Processing valid encrypted data takes longer.
- **New cookie on invalid data, but error on invalid padding.**

**Summary**: Is it possible to exactly determine, if the padding is valid? Independent from valid/invalid data?
Popular example: Session state

**Situation:**
- Big website.
- Load-balancer appliance.
- 30 backend webservers.

**Question:** How to synchronize session state?

**Answer:** Hold session state in encrypted cookie, load-balancer selects backend webserver at random.
Popular example: Session state

**Situation:**

- Big website.
- Load-balancer appliance.
- 30 backend webservers.

**Question:** How to synchronize session state?

**Answer:** Hold session state in encrypted cookie, load-balancer selects backend webserver at random.
Popular example: Session state

Situation:
- Big website.
- Load-balancer appliance.
- 30 backend webservers.

Question: How to synchronize session state?

Answer: Hold session state in encrypted cookie, load-balancer selects backend webserver at random.
How to exploit Padding Oracles?

- **GUI:** PadBuster, Padding Oracle Exploit Toolkit (POET).
- Generic Python classes.
- Now: Generic Ruby classes.
- Implement yourself (it’s not that hard).
Vulnerabilities in popular services/products

- ASP.net Viewstates
- Apache MyFaces
- Flickr API
- Ruby on Rails
- Many CAPTCHA implementations
- OWASP ESAPI
Now: Demo
Outline

1 Introduction

2 Theory

3 Practice

4 Conclusions
**Conclusion**
What you’ve learned today.

**In general**
- Leaking **anything** about plaintexts potentially breaks **everything**!
- Using strong cryptography does not help if you are doing it wrong.
- Encryption provides confidentiality, **not** data integrity!

**Padding Oracle Attacks**
- Easy to understand.
- Easy to implement (even without understanding them).
- Easy to use (even without implementing them).
- (Suitable for CTF/Lab/Seminar?)
Thanks for listening!

Questions?
Appendix

General formula for padding length probabilities.

- \( P \): Discrete uniform distribution on \( \Omega := \{0, \ldots, 255\}^b \),
- Event \( A_i := \{(x_1, \ldots, x_b) : x_b = \ldots = x_{b-i+1} = i\} \subseteq \Omega \) (Padding has length \( i \)),
- Event \( B := A_1 \cup \ldots \cup A_b \) disjoint (Padding is valid).

We now have:

\[
P(A_i) = \frac{|A_i|}{|\Omega|} = \frac{256^{b-i}}{256^b} = 256^{-i},
\]

\[
P(B) = P(A_1 \cup \ldots \cup A_b) = \sum_{i=1}^{b} P(A_i) = \sum_{i=1}^{b} \left(\frac{1}{256}\right)^i
\]

\[
= 1 - \left(\frac{1}{256}\right)^{b+1} \frac{1 - \frac{1}{256}}{1 - \frac{1}{256}} - 1 = \ldots = \frac{256^b - 1}{256^b \cdot 255}.
\]

Probability that padding has length \( i \), given that padding is valid:

\[
P(A_i \mid B) = \frac{P(A_i \cap B)}{P(B)} = \frac{P(A_i)}{P(B)} = 256^{-i} \cdot \frac{256^b \cdot 255}{256^b - 1} = \frac{256^{b-i} \cdot 255}{256^b - 1}.
\]

No guarantee for correctness ;-)
### Appendix

Some real-world numbers.

| Block length $b$ | Padding length $i$ | $\approx P(A_i|B)$ |
|------------------|--------------------|-------------------|
| 16               | 1                  | 99.6094%          |
| 16               | 2                  | 0.3890%           |
| 16               | 3                  | 0.0015%           |
| 16               | ...                | ...               |