Padding Oracle Attacks

We discuss in this addendum padding oracle attacks, which are a limited form of CCA attacks that have proven incredibly damaging in practical settings. At a high level, the problem is as follows. Encryption schemes are almost always defined via a Pad-then-Encrypt methodology. First, a plaintext is padding according to some padding rules captured by a padding function Pad. Then an encryption scheme $\mathcal{E}$ is applied to the result. During decryption, one first applies the decryption algorithm of $\mathcal{E}$ is used, and then the resulting string is checked to see if it is consistent with the padding rules of Pad. If not, a special symbol is returned (here $\perp$) and the ciphertext is rejected.

In practice, implementors often have made it so that padding errors are reported in a manner distinguishable from other types of decryption errors. That means that an attacker can send a (chosen) ciphertext to a party with the secret key, and observe whether that ciphertext had valid padding or not. Here we develop attacks based on this observation. We focus on CBC$_s$ mode since this seems the most vulnerable to such padding oracle attacks (POAs).

### 0.1 Pad-then-Encrypt

Let $D = (\{0, 1\}^n)^+$ be the set of all strings of length a multiple of $n$. Let $\mathcal{E} = (K, E, D)$ be a symmetric encryption algorithm with message space $D$. Examples are CBC$_s$ and CTR$_s$. A padding function Pad: $\{0, 1\}^* \rightarrow D$ determines how to unambiguously map arbitrary bit strings to a string in $D$. We assume an inverse function Unpad: $D \rightarrow (\{0, 1\}^* \cup \{\perp\})$. Both must be efficiently computable. Then the Pad-then-Encrypt scheme $\mathcal{PTE} = (K, \mathcal{PTE}.E, \mathcal{PTE}.D)$ associated to $\mathcal{E}$ and Pad has the same key generation algorithm as $\mathcal{E}$ and the following encryption and decryption algorithms.

$$
\begin{align*}
\text{Alg } \mathcal{PTE}.E_K(M) & \quad \text{Alg } \mathcal{PTE}.D_K(C) \\
X & \leftarrow \text{Pad}(M) \\
\text{Ret } \mathcal{E}_K(X) & \quad X & \leftarrow D_K(C) \\
& \quad \text{If } X = \perp \text{ then Return } \perp \\
& \quad \text{Ret Unpad}(X)
\end{align*}
$$

For schemes like CBC$_s$ for which $D$ never returns $\perp$, we have that $\mathcal{PTE}.D$ returning $\perp$ indicates a padding error. Assume that our target message space only includes messages that are a multiple of 8 bits (1 byte), that $n$ is a multiple of 8, and that $n \leq 255 \cdot 8$. For any number $p \in [0 .. 255]$ let $\langle p \rangle_8$ represent the 8-bit string containing some canonical encoding of the number $p$. Let $Y_8$ represent the number encoded (under the same encoding) in the 8-bit string $Y$. Let $X' \parallel Y \leftarrow \text{LastByte}(X)$ be the function that parses $X$ as $X' \parallel Y$ with $|Y| = 8$. A slightly simplified version of the padding mechanism used by TLS is the following:
Game $\text{POA}_{\mathcal{E}}$

**procedure** Initialize

\( K \leftarrow \mathcal{K} \); \( M^* \leftarrow \{0, 1\}^n \)

Return \( \mathcal{E}_K(M^*) \)

**procedure** CheckPad(C)

\( M \leftarrow \mathcal{D}_K(C) \)

If \( M \neq \perp \) then Return 1

Return 0

**procedure** Finalize(M)

Return \((M^* = M)\)

Figure 1: POA attack game.

\[ \text{Alg Pad}(M) \]

\( p \leftarrow (n + (|M| \mod n))/8 \)

If \( p = 0 \) then \( p = n/8 \)

\( Y \leftarrow (p)_8 \)

Ret \( M \parallel Y \parallel Y \parallel \cdots \parallel Y \)

\[ \text{Alg Unpad}(X) \]

\( X_1 \parallel Y_1 \leftarrow \text{LastByte}(X) \)

\( p \leftarrow Y_1(8) \)

If \( p > n/8 \) then Return \( \perp \)

If \( p = 1 \) then Return \( X_1 \)

For \( i = 2 \) to \( p \) do

\( X_i \parallel Y_i \leftarrow \text{LastByte}(X_{i-1}) \)

If \( Y_i \neq Y_1 \) then Return \( \perp \)

Ret \( X_p \)

where the number of \( Y \)'s repeated in the string returned by Pad is exactly \( p \).

For the remainder we let \( \mathcal{PTE} \) denote the Pad-then-CBC$ construction. This uses the just-given padding functions and CBC$ mode.

### 0.2 A Notion of Padding Oracle Security

We define a game $\text{POA}_{\mathcal{E}}$ in Fig. 1 to formalize POAs. In line with our example of CBC$, the game assumes that \( \{0, 1\}^n \) is a subset of the domain of \( \mathcal{E} \). The game requires an adversary to recover a message \( M^* \) chosen uniformly given only its encryption and access to an oracle that tells the adversary whether decryption is successful or not. A POA adversary expects input a ciphertext, can query CheckPad a number of times (adaptively), and outputs a string in \( \{0, 1\}^n \). We define POA advantage by

\[
\text{Adv}^\text{poa}_{\mathcal{E}}(A) = \Pr \left[ \text{POA}^A_{\mathcal{E}} \Rightarrow \text{true} \right].
\]

### 0.3 POA against Pad-then-CBC$

We prove the following claim.

**Claim 0.3.1** Let \( \mathcal{PTE} \) be the Pad-then-CBC$ encryption scheme as defined above. Then there exists a POA adversary \( A \) such that

\[
\text{Adv}^\text{poa}_{\mathcal{PTE}}(A) = 1
\]

and \( A \) makes \( 512 + 256 \cdot 15 \) queries to its CheckPad.
adversary $A(C^*)$

Parse $C^*$ as $n$-bit strings $C^*[0], C^*[1], C^*[2]$ 
Parse $C^*[0]$ as $8$-bit strings $C^*_1, \ldots, C^*_1$

$X_1 \leftarrow \text{FindFirstByte}(C^*_1, C^*[1])$

For $j = 2$ to 16 do

$X_j \leftarrow \text{FindOtherByte}(j, C^*_1, \ldots, C^*_1, C^*[1], X_{j-1}, \ldots, X_1)$

Return $X_{16} \parallel \cdots \parallel X_1$

subroutine $\text{FindFirstByte}(C^*_1, C^*[1])$

For $i = 0$ to 255 do

$R \leftarrow \{0, 1\}^{n-8}$

$R' \leftarrow R \oplus 1^{n-8}$

$C'[0] \leftarrow R \parallel \langle i \rangle_8$

$C''[0] \leftarrow R' \parallel \langle i \rangle_8$

$d \leftarrow \text{CheckPad}(C'[0] \parallel C^*[1])$

$d' \leftarrow \text{CheckPad}(C''[0] \parallel C^*[1])$

If $(d = 1 \land d' = 1)$ then

$\text{Ret } C^*_1 \oplus \langle i \rangle_8 \oplus \langle 1 \rangle_8$

subroutine $\text{FindOtherByte}(j, C^*_1, \ldots, C^*_1, C^*[1], X_{j-1}, \ldots, X_1)$

For $i = 0$ to 255 do

$R \leftarrow \{0, 1\}^{n-8j}$

$C'[0] \leftarrow R \parallel \langle i \rangle_8 \parallel (X_{j-1} \oplus \langle j \rangle_8 + C^*_j) \parallel \cdots \parallel (X_1 \oplus \langle j \rangle_8 + C^*_1)$

$d \leftarrow \text{CheckPad}(C'[0] \parallel C^*[1])$

If $(d = 1)$ then

$\text{Ret } C^*_j \oplus \langle i \rangle_8 \oplus \langle j \rangle_8$

Figure 2: POA adversary against Pad-then-CBC$^\$.

Here we give a POA adversary against $PTE$ when $SE$ is CBC$^\$ and $n = 16 \cdot 8$ (as in the case of AES). See Fig. 2. Adversary $A$ attempts to recover one byte at a time from the ciphertext by making cleverly constructed ciphertexts that are queried to the $\text{CheckPad}$ oracle. The goal is to use the padding rules of $\text{Unpad}$ in order to infer what the byte is.

We will justify that

$$Adv_{PTE}^{\text{poa}}(A) = \Pr\left[PTE^A_{PTE} \Rightarrow \text{true} \right] = 1 .$$

Let

$$M^* = M^*_{16} \parallel \cdots \parallel M^*_1,$$

$$Z^*[0] = Z^*_1 \parallel \cdots \parallel Z^*_1 = E^{-1}_K(C^*[1]),$$

$$C[0] = C^*_1 \parallel \cdots \parallel C^*_1$$

$$Y_k = Z^*_k \oplus C_k \text{ for } 1 \leq k \leq 16,$$

$$C'[0] = C'_1 \parallel \cdots \parallel C'_1 \text{ and}$$

$$Y_k' = Z^*_k \oplus C'_k \text{ for } 1 \leq k \leq 16.$$

We use subscripts to index the byte-offset within a block. Thus, the first definition labels the 16 1-byte strings of the challenge message $A$ is attempting to find; the second labels the 16 1-byte strings
of $E_K^{-1}(C^*[1])$; the third labels the 16 1-byte strings that make up each of the $256 \cdot 16$ blocks $C[0]$ used in the CheckPad queries; and the fourth labels the values generated during a CheckPad query after running $D_K(C[0]C^*[1])$, but before applying Unpad. The last two definitions there label the values generated during CheckPad on the $C'[0]C^*[1]$ used in FindFirstByte.

We split the analysis into first showing that FindFirstByte always returns the correct value $X_1 = M_1^*$. Then we will show that when $X_1 = M_1^*$ the subroutine FindOtherByte always succeeds.

The routine FindFirstByte in each iteration prepares two ciphertexts $C[0] \| C^*[1]$ and $C'[0] \| C^*[1]$ such that the first $n-8$ bits of $C[0]$ and $C'[0]$ are different, but the last 8 bits are the same (an encoding of the iteration counter $i$). It calls CheckPad twice, one for each ciphertext. We have that $d = d' = 1$ iff $Y_1 = Y_1' = \langle 1 \rangle_8$. Note that $Y_1 = Y_1'$ because the first byte of $C[0]$ and $C'[0]$ is always the same and $C^*[1]$ is used in both queries. Moreover, since we try all values of $i$, it must be that for one iteration we have that $Y_1 = \langle 1 \rangle_8$. To see why other values for $Y_1$ could not lead to $d = d' = 1$, consider if $Y_1 \neq \langle 1 \rangle_8$. Then necessarily $d' = 0$, since our choice of the first $n-8$ bits of $C[0]$ and $C'[0]$ ensures then that $Y_2' \neq Y_2$. In turn, Unpad will return ⊥ if $Y_1 \neq \langle 1 \rangle_8$ and $Y_1 \neq Y_2'$.

Now consider the first run of FindOtherByte, with $X_1 = M_1^*$. Then

$$\text{FindOtherByte}(2, C^*_1, \ldots, C^*_1, C^*[1], X_1)$$

sets $C[0]$ to be a random $n-16$ bit string followed by an 8-bit encoding of $i$ followed by

$$X_1 \oplus \langle 2 \rangle_8 \oplus C_1^* = M_1^* \oplus \langle 2 \rangle_8 \oplus C_1^* = Z_1^* \oplus \langle 2 \rangle_8.$$ 

During decryption, then, in the CheckPad oracle, we have that

$$Y_1 = (Z_1^* \oplus \langle 2 \rangle_8) \oplus Z_1^* = \langle 2 \rangle_8$$

which means that Unpad will read a first byte that encodes 2. This means that Unpad will return true exactly if the second value $Y_2 = \langle 2 \rangle_8$. This occurs only when

$$\langle 2 \rangle_8 = \langle i \rangle_8 \oplus Z_2^* = \langle i \rangle_8 \oplus M_2^* \oplus C_2^*.$$ 

Thus here Unpad only returns one in the case that $M_2^* = C_2^* \oplus \langle i \rangle_8 \oplus \langle 2 \rangle_8$, which is exactly what is returned by FindOtherByte. Moreover, since FindOtherByte tries all 256 values of $i$ it is guaranteed to find the exact byte $M_2^*$. A simple inductive argument justifies that the rest of the values $X_3, \ldots, X_{16}$ are likewise correct.