Blockciphers Modes, Authentication
CMSC 23200/33250, Autumn 2018, Lecture 4

David Cash
University of Chicago
Plan

1. Blockciphers recall
2. Blockcipher modes (encrypting large messages)
3. Authentication: MACs
4. Authenticated Encryption
5. Padding Oracle Attacks
Due to Rijmen and Daemen
- Block length n = 128
- Key length k = 128, 192, 256

- Different structure from DES.
- 10 rounds of “substitution-permutation”
Blockcipher Security

- AES is thought to be a good “Pseudorandom Permutation”

- Outputs all look random and independent, even when inputs are maliciously controlled.
- Formal definition in CS284.
Example - AES Input/Outputs

- Keys and inputs are 16 bytes = 128 bits

- $K_1$: 9500924ad9d1b7a28391887d95fcfbd5
- $K_2$: 9500924ad9d1b7a28391887d95fcfbd6

$AES_{K_1}(00..00) = \text{8b805db39f3eee72b43bf95c9ce410f}$
$AES_{K_1}(00..01) = \text{9918e60f2a20b1b81674646dceebdb51}$
$AES_{K_2}(00..00) = \text{1303270be48ce8b8dd8316fdbaa3eb04}$
$AES_{K_2}(00..01) = \text{96ba598a55873ec1286af646073e36f6}$
So we have a blockcipher…

- Now what?

It only processes 16 bytes at a time, and I have a whole lot more data than that.

This next step is where everything flies off the rails in implementations…
Encrypting large files: ECB

- ECB = “Electronic Code Book”

\[
\text{AES-ECB}_K(M)
\]
- Parse \( M \) into blocks \( M_1, M_2, \ldots, M_t \)
  // all blocks except \( M_t \) are 16 bytes
- Pad \( M_t \) up to 16 bytes
- For \( i=1\ldots t \):
  - \( C_i \leftarrow \text{AES}_K(M_i) \)
- Return \( C_1, \ldots, C_t \)
The ECB Penguin

- 16 byte chunks are consecutive pixels

Plaintext

ECB Ciphertext

- It gets even worse…
ECB Security: It gets worse…

- Seeing penguins is bad, but it doesn’t mean you can recover credit card numbers or passwords inside a ciphertext
- “Chosen Plaintext Attack” against ECB can decrypt any ciphertext.
Chosen-Plaintext Attacks (CPA) against Encryption

- Adversary provides inputs to system
- Obtains encryption of message that depends on its inputs
- Sometimes $M = M'$
CPA Example: Encrypted Cookies

```
Webserver

K

```

```
M' ← ...341jkas;SECRET
C ← Enc_K(M')
```

- More later in web security module

**Assignment 1 preview:** ECB is totally insecure in this setting. You will attack it and recover SECRET.
Encrypting large files, Attempt #2: CTR

- CTR = “Counter Mode”
- Idea: Build a nonce-based stream cipher from AES

\[
\text{AES-CTR}_k(\text{IV}, M) = \begin{cases} 
\text{Parse } M \text{ into blocks } M_1, M_2, \ldots, M_t \\
// \text{all blocks except } M_t \text{ are 16 bytes} \\
\text{For } i=1 \ldots t:\n- C_i \leftarrow M_i \oplus \text{AES}_k(\text{IV}+i) \\
\text{Return } \text{IV}, C_1, \ldots, C_t
\end{cases}
\]

Notes:
- No need to pad last block
- Must avoid reusing part of stream
Encrypting large files, Attempt #2: CTR

- CTR = “Counter Mode”
- Idea: Build a nonce-based stream cipher from AES

**Notes:**
- No need to pad last block
- Must avoid reusing part of stream

**AES–CTR**

- Parse \( M \) into blocks \( M_1, M_2, \ldots, M_t \)
  // all blocks except \( M_t \) are 16 bytes
- For \( i=1\ldots t \):
  - \( C_i \leftarrow M_i \oplus \text{AES}_k(IV+i) \)
- Return \( IV, C_1, \ldots, C_t \)

When combined with authentication, CTR is a good cipher.
Penguin Sanity Check

Plaintext  ECB Ciphertext  CTR Ciphertext

Looks random
Encrypting large files, Attempt #3: CBC

- CBC = “Cipher Block Chaining”
- Nonce-based, but not a stream cipher
- Historical option (sometimes

\[
\text{AES-CBC}_k( IV, M )
\]
- Parse \( M \) into blocks \( M_1, M_2, \ldots, M_t \)
  \[ // \text{all blocks except } M_t \text{ are 16 bytes} \]
- Pad \( M_t \) up to 16 bytes
- \( C_0 \leftarrow IV \)
- For \( i=1 \ldots t \):
  - \( C_i \leftarrow \text{AES}_k(M_i \oplus C_{i-1}) \)
- Return \( C_0, C_1, \ldots, C_t \)

![Diagram](image-url)
Encrypting large files, Attempt #3: CBC

- CBC = “Cipher Block Chaining”
- Nonce-based, but not a stream cipher
- Historical option (sometimes

AES\text{–CBC}_k(IV, M)

- Parse M into blocks $M_1, M_2, \ldots, M_t$
  // all blocks except $M_t$ are 16 bytes
- Pad $M_t$ up to 16 bytes
- $C_0 \leftarrow IV$
- For $i = 1 \ldots t$:
  - $C_i \leftarrow AES_k(M_i \oplus C_{i-1})$
- Return $C_0, C_1, \ldots, C_t$

When combined with authentication, CBC is a good cipher.

Warning: Padding creates havoc with authentication. Very difficult to implement.
Blockcipher Summary

- AES is unbroken
- AES-CTR is most robust construction for confidentiality
- AES-CTR/AES-CBC do not provide authenticity/integrity and should almost never be used alone.
Next Up: Integrity and Authentication

- Authenticity: Guarantee that adversary cannot change or insert ciphertexts
- Achieved with MAC = “Message Authentication Code”
Integrity: Preventing message modification
Encryption Integrity: An abstract setting

Encryption satisfies **integrity** if it is infeasible for an adversary to send a new $C'$ such that $\text{Dec}_K(C') \neq \text{ERROR}$. 
AES-CTR does not satisfy integrity

\[ M = \text{please pay ben 20 bucks} \]
\[ C = \text{b0595fafd05df4a7d8a04ced2d1ec800d2daed851ff509b3e446a782871c2d} \]
\[ C' = \text{b0595fafd05df4a7d8a04ced2d1ec800d2daed851ff509b3e546a782871c2d} \]
\[ M' = \text{please pay ben 21 bucks} \]

Inherent to stream-cipher approach to encryption.
AES-CBC does not satisfy integrity

**AES-CBC Decryption:**

- Input: IV, C₁, C₂
- Output: M₁, M₂

**Diagram:**

- IV → C₁ → AES⁻¹ₓ() → M₁
- C₂ → AES⁻¹ₓ() → M₂

- IV, C₁ ⊕ X, C₂ → R, M₂ ⊕ X

Where R is some unpredictable block.
A **message authentication code (MAC)** is an algorithm that takes as input a key and a message, and outputs an “unpredictable” tag.
MAC Security Goal: Unforgeability

MAC satisfies **unforgeability** if it is unfeasible for Adversary to fool Bob into accepting $M'$ not previously sent by Alice.
MAC Security Goal: Unforgeability

Note: No encryption on this slide.

M = please pay ben 20 bucks
T = 827851dc9cf0f92ddc552572ffdd8bc

M′ = please pay ben 21 bucks
T′ = baeaf48a891de588ce588f8535ef58b6

Should be hard to predict T′ for any new M′.
MACs In Practice: Pretty much always use HMAC

- Don't worry about how it works.

- Other options: Poly1305-AES or CBC-MAC (the latter is tricky)
Authenticated Encryption

Encryption that provides **confidentiality** and **integrity** is called **Authenticated Encryption**.

- Built using a good cipher and a MAC.
  - Ex: AES-CTR with HMAC-SHA2
- Best solution: Use ready-made Authenticated Encryption
  - Ex: AES-GCM is the standard
Building Authenticated Encryption

$\text{Encrypt}_{K_1,K_2}(M)$

- $M \xrightarrow{\text{Encrypt}_{K_1,K_2}(\cdot)} (C,T)$
- $C \xrightarrow{\text{MAC}_{K_2}} T$

$\text{Decrypt}_{K_1,K_2}(C,T)$

- $C \xrightarrow{\text{MAC}_{K_2}} T'$
- $T' = T \rightarrow M'$
- $T' \neq T \rightarrow \bot$

- Summary: MAC the ciphertext, not the message
Chosen-Ciphertext Attacks (CCA) against Encryption

- Integrity + Confidentiality = security against CCAs

- Adversary provides ciphertext inputs to system
- Obtains info about decryptions of its ciphertexts
CBC-Based Auth. Enc. Error: Padding and MACs

**Encrypt_{K1,K2}(M)**

1. **IV**
2. **M1**
3. **M2**
4. **AES_{K1}()**
5. **AES_{K1}()**
6. **MAC_{K2}()**
7. **T**
8. **Final output: IV, C1, C2, T**

**Decrypt_{K1,K2}(IV,C1,C2,T)**

1. If tag T wrong:
   - Output REJECT
2. M' ← CBC-Decrypt_{K1}(IV,C1,C2)
3. If padding format wrong:
   - Output PADDING_ERROR
4. Output M'

**Decrypt_{K1,K2}(IV,C1,C2,T)**

1. M' ← CBC-Decrypt_{K1}(IV,C1,C2)
2. If padding format wrong:
   - Output PADDING_ERROR
3. If tag T wrong:
   - Output REJECT.
4. Output M'
Padding Oracle Attacks

Allows decryption of arbitrary ciphertexts by adversary! … also by you, in Assignment 1.

\[
\text{Decrypt}_{K_1,K_2}(IV,C_1,C_2,T) \\
1. M' \leftarrow \text{CBC-Decrypt}_{K_1}(IV,C_1,C_2) \\
2. \text{If padding format wrong:} \\
   \quad \text{Output PADDING\_ERROR} \\
3. \text{If tag T wrong:} \\
   \quad \text{Output REJECT.} \\
4. \text{Output } M' \\
\]

System (e.g. webserver)

K

Broken
Padding Oracle Attacks: It gets worse

Output REJECT lines will take different times to reach:

Attack still possible.

Solutions:
1. Constant-time code (extremely difficult).
2. Use un-padded encryption like CTR.

3. If tag T wrong: Output REJECT.
4. Output M'
The End