Chosen-Ciphertext Attacks
CMSC 28400, Autumn 2019, Lecture 15

David Cash
University of Chicago
Last time: CPAs and Partial Plaintext Recovery

Picks user data (e.g. username)

```
userdata=... ; token=uchic4gos3cr34
```

Encrypt()

```
3F C2 12 FF 32 12 99 D2 3E 42 4A 2C 27 DA 04 07
```

Observe ciphertext
Today: Chosen-Ciphertext Attacks (CCA)

- Adversary submits ciphertexts to system
- System decrypts them and reacts

\[ C = \text{Enc}(K, M) \]

- \( C' \) produced with knowledge of other ciphertexts!
A Simple CCA: Stream Ciphers and CTR Mode

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

- Inherent to stream-cipher approach to encryption.
- More subtle problems even with ECB, CBC, etc (PPS2!)
- “Malleability” attack:
  1. Change $C$ to $C'$
  2. Learn info about $M'$
  3. Infer info about $M$
Typical Bug: Clear-Reply On Padding-Error

- Sometimes systems return message in clear if padding is invalid

\[ C = \text{Enc}(K, M) \]

- Combines with attack on CTR mode to learn plaintext (exercise)
- With other modes, worse things happen (PPS2)
Next Attack: On AES-CBC with Padding
PKCS7 Padding Reminder

- Need to pad a byte string up to a multiple of 16 bytes
- First look at how many bytes are missing. Here, need 10 bytes
- Fill missing k bytes with value k (k = 10 = 0x0A in example)

- If data is already a multiple of 16 bytes long, add an entire block of 0x10 bytes

Can’t leave data unchanged;
Bytes might be interpreted as padding.

- Un-padding is easy
Fact: Not every byte string is a “valid padding”. Some strings have to be handled as “malformed”.

Invalid. Why?

Valid. Why?

Valid! All strings that end in 0x01 are valid.
Padding Oracle Attack Template

Tell if padding error occurred!

C’
[protocol response]

System (e.g. webserver) K

Real-world attacks against:
- TLS (2003)
- IPSec (2007, 2010)
- Ruby on Rails (2010)
- ASP.NET (2010)
- SecurID Auth Tokens (2012)
- Steam Client (2016)

Decrypt(K, C’)
1. M’ ← CBC-Decrypt(K, C’)
2. If padding format wrong: Output PADDING_ERROR
3. Output M’

Now: How to find one byte of plaintext.
Attack Setting

- First two blocks of long valid ciphertext are pictured.

Want this:

Have this, plus the padding oracle.
Initial goal: Learn last byte of a block

- First observation: If we truncate the long ciphertext down to these blocks, what will padding oracle say?
  - *Padding_Error*

- What can we infer from the padding error response?
  - Last plaintext byte of that block could not have been 0x01.

Let's learn this:
Initial goal: Learn last byte of a block.

- What will padding oracle say?
  - (still) **Padding_Error**!

After decryption, this byte becomes **0x1E**.

Consider XORing this byte with 01, then submitting ciphertext to oracle.
Initial goal: Learn last byte of a block.

- What will padding oracle say?
  - Padding is valid!
  - Infer value of last byte: $64 \oplus 7A \oplus (\text{last-byte}) = 0x01$, so last-byte=$1F$

Consider instead xoring this byte with 0x7A, then submitting ciphertext to oracle.
Initial goal: Learn last byte of a block.

Upshot: If last byte of plaintext is $X$, and we change last byte of prior ciphertext block to $X \oplus 0 \times 01$, then padding will be valid. Changing that byte to anything else will usually result in invalid padding.

Q: But how do we should we set that byte?  
A: Just try all 256 until one works.
Recovering more plaintext bytes

- Assume we know last byte is 1F.
- Eventually get valid padding with 02 bytes!
- Guess plaintext byte as before.
- Get all 16 bytes, move on to next block

Eventually this becomes 02 too.
Becomes 0x02

Change to 0x1F \( \oplus \) 0x02

Now start changing this byte.
The End