6.857 Recitation 4: Block Cipher Review, AES, HMAC

TA: Sean Fraser

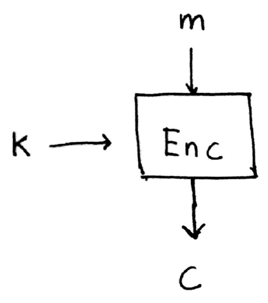
Friday March 1st, 2019

Agenda

- Review of Block Ciphers
- Discuss Design of AES in more detail
- Review Modes of Operation
- Authentication (continued) - HMAC
- Questions

1 Review of Block Ciphers

A block cipher takes a fixed-length key k and a fixed-length input m, and produces a fixed-length output (usually the same size as the input).



We'll deal with variable length input soon (we use a "mode of operation" for this).

1.1 What properties must a block cipher have?

- it must be invertible
- it must be "secure". How is secure defined?
  - ideal cipher model: for every secret and random key k, encryption should behave as a random function from its inputs to its outputs. That is, the ciphertext should look like a random permutation (computationally indistinguishable). This basically means that we shouldn't be able to find any patterns that reveal info about a key or input.
  - cryptanalysis: people should work very hard to break it, and if they fail, thats a good thing. People usually try to break a few rounds, and then when designing a cipher, that number plus a safty margin is used (for example in AES, there was a theoretical break for 6 rounds, but the lowest number of rounds used is 10).
  - statistical tests: output computationally indistinguishable from random. e.g. flipping one bit of the key or plaintext flips half the bits of the ciphertext in a random-looking way.
- it is efficient to compute. In software? (e.g. Speck Cipher). In hardware? (e.g. Simon Cipher)
- easy of implementation?

1.2 Two main design techniques

- **Feistel Structure** (as in DES, Speck, Simon). Looked at Simon Cipher link here: `https://en.wikipedia.org/wiki/Simon_(cipher)`
- **Substitution-Permutation Networks (SPN)** (as in AES). Also known as a Confusion/Diffusion Model.

2 Design of AES

- uses SPN structure
- block size 128 bits
- secret key k either 128, 192 or 256 bits (those use 10, 12 and 14 rounds total respectively). Tradeoff between performance and security for more rounds.
- three main ideas:
  - confusion - 'obscure relationship between input and output as much as possible'
  - diffusion - 'spread out message as much as possible over output'

  - secrecy only in the key - similar to Kerchoff's principle from recitation 2 - assume adversary knows design of AES

- byte-oriented design. Operates on elements of $GF(2^8)$. 128 bit input can be viewed as a 4x4 grid of bytes. Look at the following visualization here: `https://www.youtube.com/watch?v=mlzxpkdXP58`.

- all operations in Galois (finite) field, $GF(2^8)$.

- for an approachable overview of AES, see the following guide: `http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html`.

Each round:

- SubBytes (uses "S-boxes" or substitution boxes) (*subsitution/confusion*)

- ShiftRows (*permutation / diffusion*)

- MixColumns (*permutation / diffusion*)

- AddRoundKey (*add in secrecy*)

All of the above needs to be invertible for decryption.

What does $GF(2^8)$ mean? I aim to provide a high level overview or some intuition, there will be a more formal definition and more uses next week.

- Informally, a finite field is a set on which operations of multiplication, addition, subtraction and division are defined, and satisfy certain basic rules such as associativity, commutativity etc. (See lecture 8 for a formal definition).

Example: All bytes in AES algorithm are interpreted as finite field elements. For example addition may be viewed as following. The addition is performed with the XOR operation, or modulo 2. To give some intuition for how this might work, see the example below:

$$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2 \qquad \textbf{(polynomial notation);}$$

$$\{01010111\} \oplus \{10000011\} = \{11010100\} \qquad \textbf{(binary notation);}$$

$$\{57\} \oplus \{83\} = \{d4\} \qquad \textbf{(hexadecimal notation).}$$

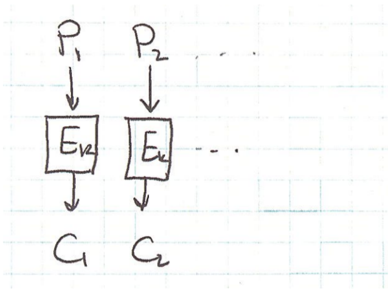
Multiplication is a little more complicatated. For a more detailed overview of how this all works, see the official documentation for AES here: `https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf`.

In practice, AES does pretty well. For practical uses, we can treat AES as an ideal block cipher, i.e. $Enc(K, \cdot)$ is a random permutation.

3 Cipher Modes of Operation

How do you use a block cipher on a message that is longer than its block size, i.e. variable length?

- Idea # 1: ECB (Electronic Codebook)

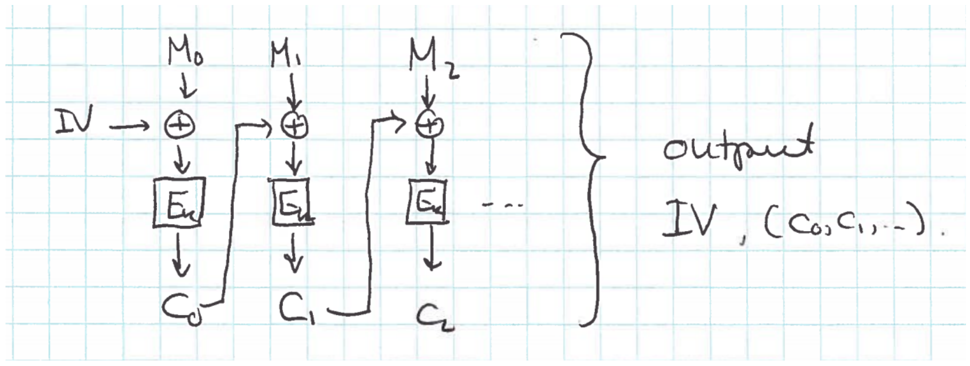


Pick a cipher and use key for each block independently.
Problem: encrypting same plaintext twice gives same output. See a visualization of this here: `https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation` under ECB.

- Idea # 2: Use a different key for every block

Problem: incredibly inefficient.

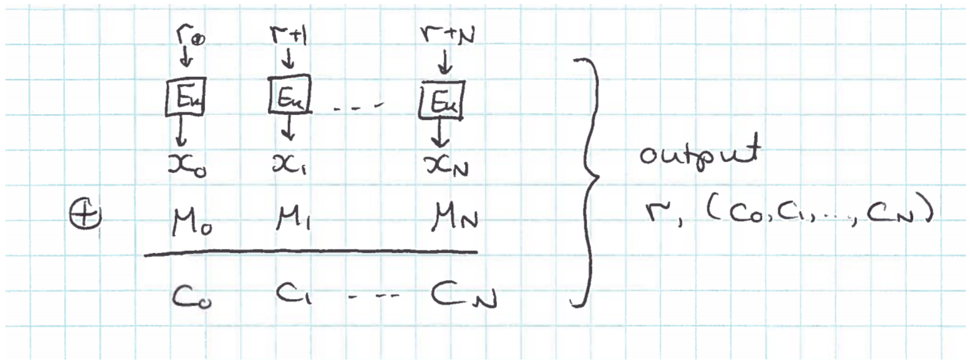
- Idea # 3: CBC (Cipher Block Chaining)



Make sure initialization vector (IV) is random or unique.
Good: No two encryptions of the same block will look the same.
Bad: If part of the message is corrupted, whole message is corrupted.

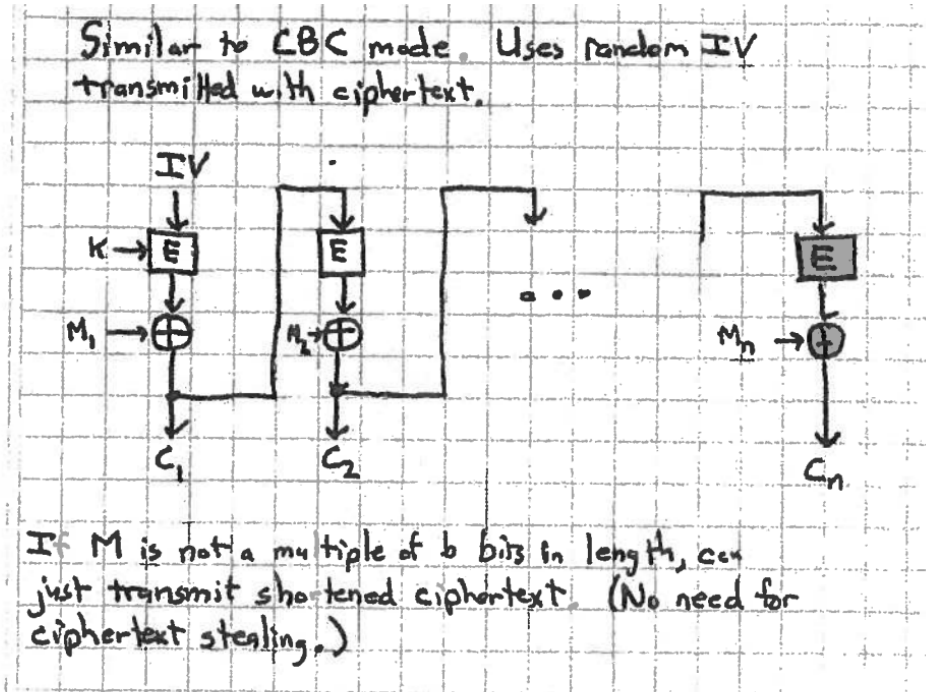
- Idea # 4: CTR (Counter)

Make sure nonce (r) is random and unique. Generate the bits to XOR with the message (like OTP) by repeatedly encrypting arbitrary values. Usual case is count up



from 1 to n, where n is the number of blocks. Assuming ideal cipher, this solves ECB problem because we never repeat an input to the block cipher.
Good: Highly parallelizable.

- Idea # 5: CFB (Cipher Feedback)


Similar to CBC mode. Uses random IV transmitted with ciphertext.

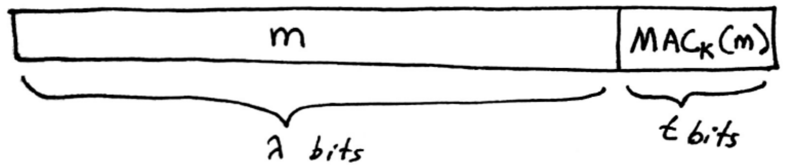
If M is not a multiple of b bits in length, can just transmit shortened ciphertext. (No need for ciphertext stealing.)

Essentially generating long sequence of bits that your XOR with the plaintext to get ciphertext. Like "stretching" the key, except that plaintext also affects the pattern.

4 Authentication - HMAC

Recall: Message Authentication Codes (MAC)

Suppose Alice is trying to communicate a message to Bob, but Eve, an active eavesdropper, is listening. She can intercept the message, tamper with it, and send it to Bob. How does Bob ensure the authenticity of the message (i.e. that is hasn't been tampered with). Similar to signatures, but slightly different. We tag on a MAC to the end of our message, that Bob has to himself generate a $MAC_k(m)$ based on the message and compare with to ensure validity. We assume Alice and Bob share a secret key k.



Our security goal (roughly - refer to Lecture notes for actual definition): adversary shouldn't be able to generate a new $(m, MAC_K(m))$ pair that is valid, even given pairs to m's of his/her choosing. Note that the tag or MAC cannot be too short, otherwise we could guess with probability $\frac{1}{2^t}$.

- <u>0^{th} attempt:</u> $MAC_k(m) = h(m)$.
  - BAD. Adversary can simply compute $h(m)$ with a changed message of his/her choosing, and produce a valid tag. No use of the secret key k.
- <u>1^{st} attempt:</u> $MAC_k(m) = h(k||m)$.
  - We use our secret k as a prefix construction.
  - May be vulnerable to length extension attacks. (Given $h(k||m)$, one can efficiently compute $h(k||m||m')$ for any m').
  - This vulnerability exists for Merkle-Damgard constructions. E.g. SHA-256 and SHA-512 are vulnerable to this type of attack. This is because adversary can use $h(k||m)$ to reproduce the 'state' used in the 'serial mode of operation' Merkle-Damgard construction so can append to the message and still get a valid hash of the old message with the appended new message.
- <u>2^{nd} attempt:</u> $MAC_k(m) = h(m||k)$.
  - Length extension attacks don't work.
  - Possibly insecure if attacker knows a known collision $h(m_1) = h(m_2)$. For Merkle-Damgard (or SHA-256), this means that if we have $h(m_1) = h(m_2) \implies h(m_1||k) = h(m_2||k)$ by same argument as above.
- <u>3^{rd} attempt:</u> HMAC Construction (used in IPSec, SSH, TLS).

- We define $HMAC_k(m) = h(k_1||h(k_2||m))$
- $k_1 = k \oplus$ `opad` (outer padding - fixed constant `0x5c`)
- $k_2 = k \oplus$ `ipad` (inner padding - fixed constant `0x36`)
- This solves the problems above by adding another layer of hashing. Cannot do length extension attacks etc. and even if known collision, cannot reproduce what is inside the outer hash.