# ECE 443/518 – Computer Cyber Security
# Lecture 07 Authenticated Encryption

Professor Jia Wang
Department of Electrical and Computer Engineering
Illinois Institute of Technology

September 14, 2022

# Outline

# Reading Assignment

- This lecture: UC 12, 5.1.6
- Next lecture: UC 6.3

# Outline

Message Authentication Codes

Authenticated Encryption

Complexity Theory

# Motivation

- Cryptographic hash functions help to achieve integrity on an insecure channel with an additional authentic channel.
  - Without using a secret key.
- In the context of symmetric cryptography, since there is already a secret key, can integrity be achieved without the additional authentic channel?
- Message authentication: prove that the message is authentic.
  - I.e. created by a party knowing the secrey key.
- Don't confuse it with user authentication.
  - User authentication: prove you are youself.
  - Preferably unclonable information but usually via a secret.
  - But if Alice proves to Oscar that she is Alice by showing Oscar the secret, how to prevent Oscar to convince Bob that he/she is Alice by showing the same secret?

# Message Authentication Codes (MACs)

- $MAC_k(x)$: a function that returns a fixed-size code that depends on both the message $x$ and the secret key $k$.
- Alice computes $m = MAC_k(x)$ and sends $(x, m)$ to Bob.
  - Since for now we only discuss integrity, everything except $k$ are known by the adversary Oscar.
- Bob receives $(x', m')$ and verifies that $m' == MAC_k(x')$.
  - The active adversary Oscar may change both $x$ and $m$.
- How about use a cryptographic hash function $h$?
  - Secret prefix: $MAC_k(x) = h(k||x)$
  - Secret suffix: $MAC_k(x) = h(x||k)$

# Oscar's Attacks

- Most hash functions consume a message byte by byte.
- Oscar knows $x$ and $m = MAC_k(x) = h(k||x)$.
- Secret prefix: Oscar can compute $h(k||x||y)$ by initializing $h$ with $h(k||x)$ and then proceed with the message $y$.
  - There is no need to know $k$ to compute $MAC_k(x||y) = h(k||x||y)$.
- Secret suffix: if Oscar knows $h(x') == h(x)$ from birthday attack on $h$, then $h(x'||k) == h(x||k)$.
  - There is no need to know $k$ to compute $MAC_k(x') = h(x'||k)$.
- Better solutions?

# HMAC

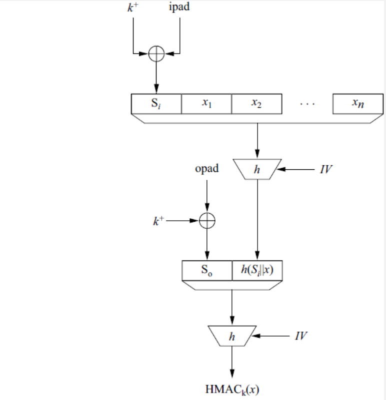



**Fig. 12.2** HMAC construction

(Paar and Pelzl)

- RFC 2104 (1997), FIPS PUB 198-1 (2008)
- Use a cryptographic hash function $h$
  - $k^+$: zero extended to match hash block size.
  - Padding: 0$x$5$c$ for opad and 0$x$36 for ipad.
  - Usually without using the IV.

# CBC-MAC

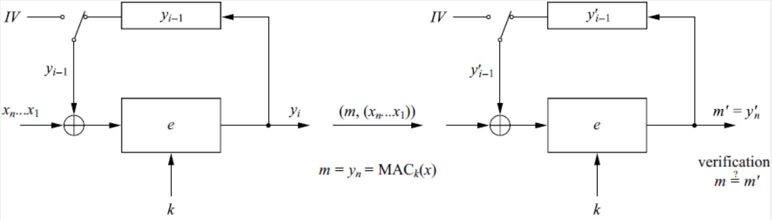



**Fig. 12.3** MAC built from a block cipher in CBC mode

(Paar and Pelzl)

- Use a block cipher. Only need encryption $e()$.
- A lot of pitfalls exist
  - Use a random IV (shown above as suggested by the textbook!)
  - Not include message length.
  - Share the secret key for encryption and MAC.
  - etc.
- Don't implement your own. Use an established library.

# GMAC

- A variant of the Galois Counter Mode (GCM).
- Usually a MAC is used together with a symmetric cipher to provide both confidentiality and integrity so let's delay the discussion of GMAC to GCM.

# Outline

Message Authentication Codes

Authenticated Encryption

Complexity Theory

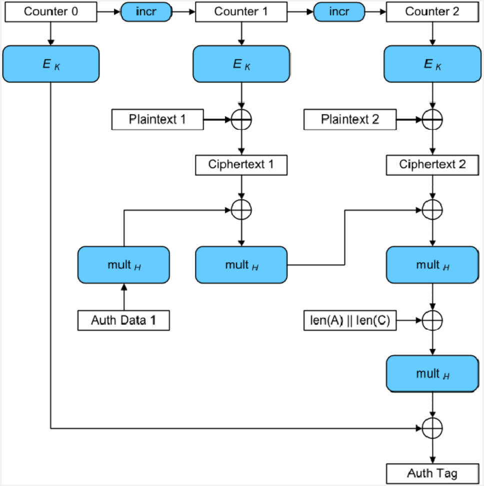
# Motivation

- It is quite intuitive that one may combine a symmetric ciphers and a MAC to achieve confidentiality and integrity (including message authentication) with a secret key.
- Three possible combinations
  - Encrypt-then-MAC: append MAC of ciphertext to ciphertext
  - Encrypt-and-MAC: append MAC of plaintext to ciphertext
  - MAC-then-Encrypt: append MAC of plaintext to plaintext
- Which one?

# Chosen Ciphertext Attacks

- Oscar may create ciphertexts.
  - Usually by modifying ciphertexts sending by Alice.
- Then Oscar may send them to Bob and observe how Bob decrypts/validates them.
  - Bob may response whether the message decrypts/validates correctly.
  - Oscar may further meature time taken by Bob to generate the response (side channel).
- For both Encrypt-and-MAC and MAC-then-Encrypt, the validation is with plaintext so that Oscar may obtain plaintext bit-by-bit if he/she may modify ciphertext to cause a few bits to change in plaintext.
- Not a concern for Encrypt-then-MAC as Bob will reject incorrect ciphertexts without decrypt them and Oscar learns nothing.

# Galois Counter Mode (GCM)




(Wikipedia)

- NIST Special Publication 800-38D (2007), various RFCs
- Work with block ciphers using 128-bit blocks.

# More on GCM

- Encryption/decryption are in the Counter Mode.
  - Counter 0 is derived from the IV.
- MAC
  - Allow to include additional authenticated data (AAD), i.e. Auth Data 1 in the figure, that require only integrity but no confidentiality.
  - Compute authentication subkey $H = e_k(0)$.
  - Treat all 128-bit blocks (padding as needed) as numbers in the Galois field $GF(2^{128})$ and perform multiplications and additions to generate Auth Tag.
- It is critical that the combined choice of $k$ and IV should be unique. Otherwise the GCM mode is not secure.
- In addition to GCM, other modes for authenticated encryption exist.

# GCM Implementation

- Block cipher in counter mode.
  - No need to implement block decryption.
  - Can be parallelized.
  - Usually use AES to leverage existing hardware accelerations.
- MAC essentialy evaluates a polynomial.
  - Can be parallelized.
  - Addition in $GF(2^{128})$ is bitwise XOR.
  - Multiplication can be accelerated by special hardware, accessible on many modern processors through special instructions.

# Outline

# Greatest Common Divisor (GCD)

- $gcd(a, b)$: greatest common divisor of integers $a$ and $b$.
  - Assume at least one of $a$ and $b$ is not 0.
- Examples
  - $gcd(27, 21) = 3$
  - $gcd(10, 12) = 2$
  - $gcd(3, 16) = 1$
  - $gcd(4, 16) = 4$
- Algorithm to compute $gcd()$ on computers?

# Simple GCD Algorithm

```
Input:  two integers a ≥ b > 0
For k = b downto 1:
  If (b mod k == 0) and (a mod k == 0):
    Report gcd(a, b) = k
```

- How efficient is the algorithm?
  - As you may have observed and guessed, the most time consuming parts are the mod operations in the loop.
  - In the worse case when $gcd(a, b) = 1$, there are $2b$ mod operations.
- Still, we need <u>complexity theory</u> to understand how good or how bad that is.

# The Big-$O$ Notation

- Performance of an algorithm
  - Time complexity: how long does it take?
  - Space complexity: how many memory does it consume?
  - Complexities depend on problem sizes.
- The measure should be independent of computer architectures and clock frequencies.
  - A rough measure of trends for large problem sizes.
- The big-$O$ notation: complexity measure of trends
  - $N$: problem size
  - $O(1)$: the complexity is independent of problem size
  - $O(N)$: the complexity grows no faster than $N$
  - $O(N^2)$: the complexity grows no faster than $N^2$
  - $O(2^N)$: the complexity grows no faster than $2^N$
  - And so on ...

# Time Complexity of Simple GCD Algorithm

- Problem size $N$: assume $a$ and $b$ are $N$-bit numbers.
- Complexity of arithmetic operations
  - Addition and subtraction: $O(N)$
  - Multiplication, division, and mod: $O(N^2)$ (could be better)
  - What about power and exponential?
- Time complexity of simple GCD algorithm: $O(2^N N^2)$.

# Cryptography Meets Complexity

- Exponential time vs polymonial time
  - Exponential time: $O(2^N)$, $O(3^N)$, etc.
    - E.g. brute-force attack on $N$-bit keys take $O(2^N)$ time.
  - Polymonial time: $O(N)$, $O(N^2)$, $O(N^{1000})$,etc.
- Exponential time (or worse) algorithms are too slow for computationally bounded parties (for large $N$).
- Computationally bounded parties can execute polynomial time algorithms efficiently (for large $N$).
- Assume all of Alice, Bob, and Oscar have bounded computational power.
  - If there is a problem Alice and Bob could solve in polynomial time,
  - while Oscar need to spend exponential or more time to solve,
  - then Alice and Bob could always choose a large enough $N$ so that they can solve it but Oscar cannot solve it practically.

# Summary

- MAC authenticates the message using the secret key.
- While it appears to be intuitive to create your own MAC for message authentication, or to combining block ciphers with MAC for authenticated encryption, there are a lot of pitfalls for both design and implementaion – you should follow documented standards exactly or use an established library instead.