CS 370: Introduction to Security

04.20: Digital signatures, cryptographic hash, etc.

Tu/Th 4:00 – 5:50 PM

Sanghyun Hong

sanghyun.hong@oregonstate.edu

Oregon State University

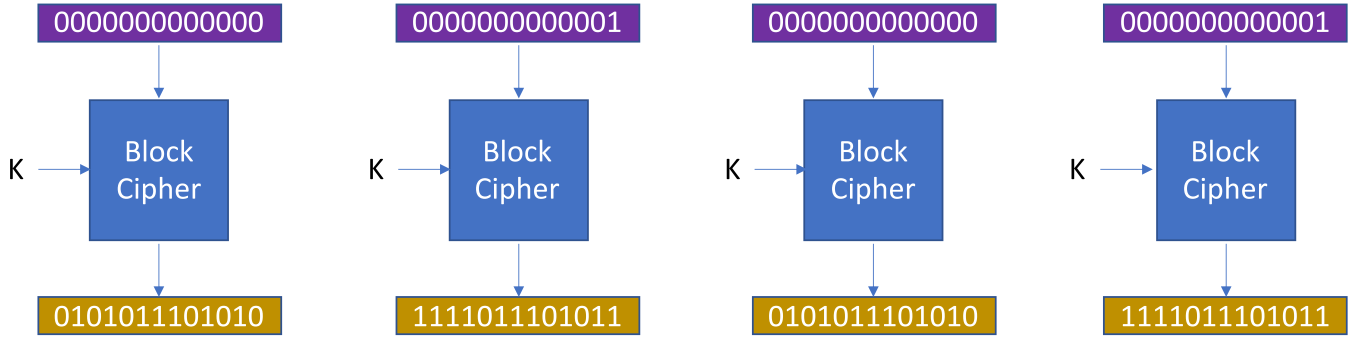
SAIL
Secure AI Systems Lab

Topics for today

- Recap
  - Block cipher modes
  - ECB and CBC
  - ECB and CBC’s weaknesses and exploitations
- Block cipher modes
  - Counter modes (CTR)
  - CTR’s weakness
- Cryptographic hash
  - Message authentication code (MAC)
  - SHA256
  - HMAC

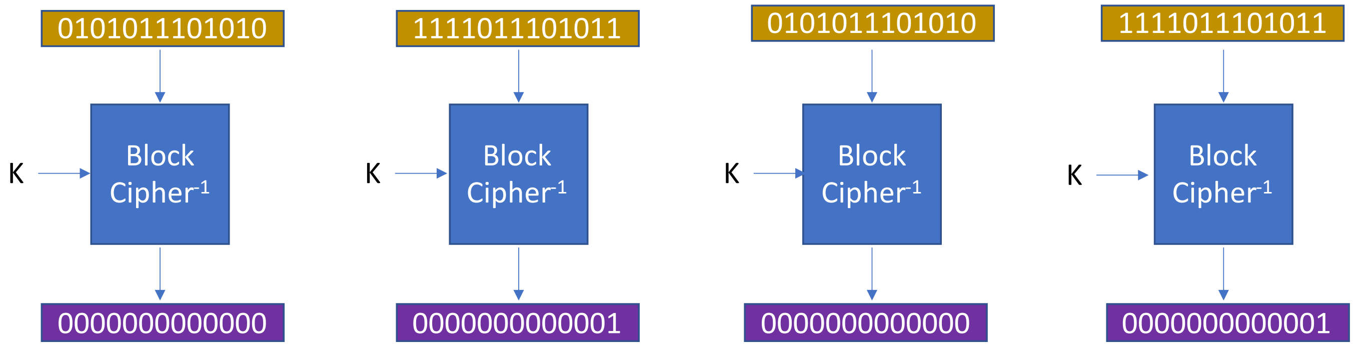
Electronic code block – cont'd

- ECB Operations (and benefits)
  - You can encrypt each block in parallel



Electronic code block – cont'd

- ECB Operations (and benefits)
  - You can encrypt (and decrypt) each block in parallel

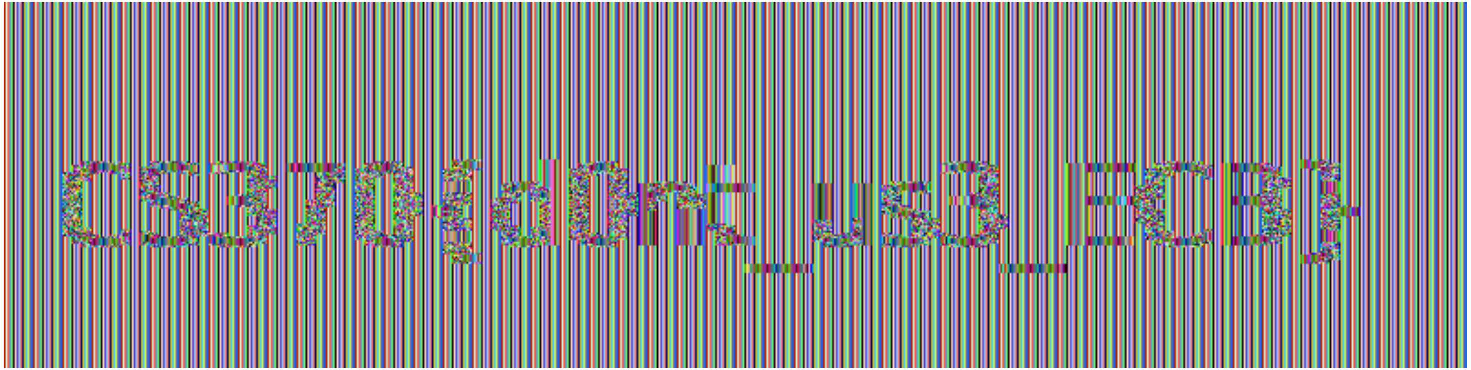


Electronic code block – cont'd

- ECB weakness(es)
  - Using the same key leads to the same ciphertext
  - An adversary can collect the ciphertext and plaintext mappings
    - M: 0 -> C: 0x39827332…
    - M: 1 -> C: 0x5a83f874…
    - …
  - An adversary can alter the plaintext by exploiting the mappings

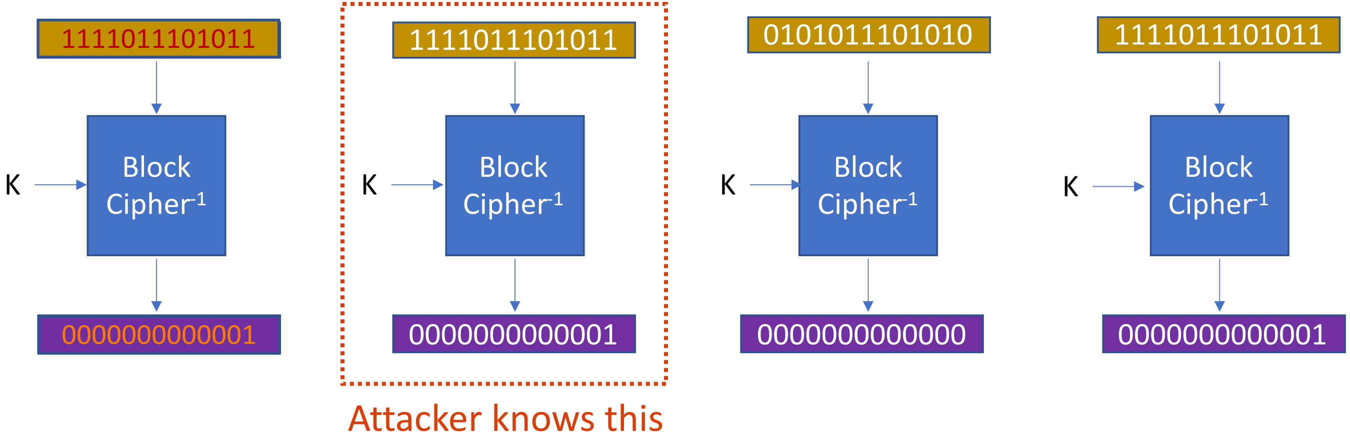
Recap: micro-lab

- ECB weakness
  - We need to guess what is inside this super-secretly encrypted photo



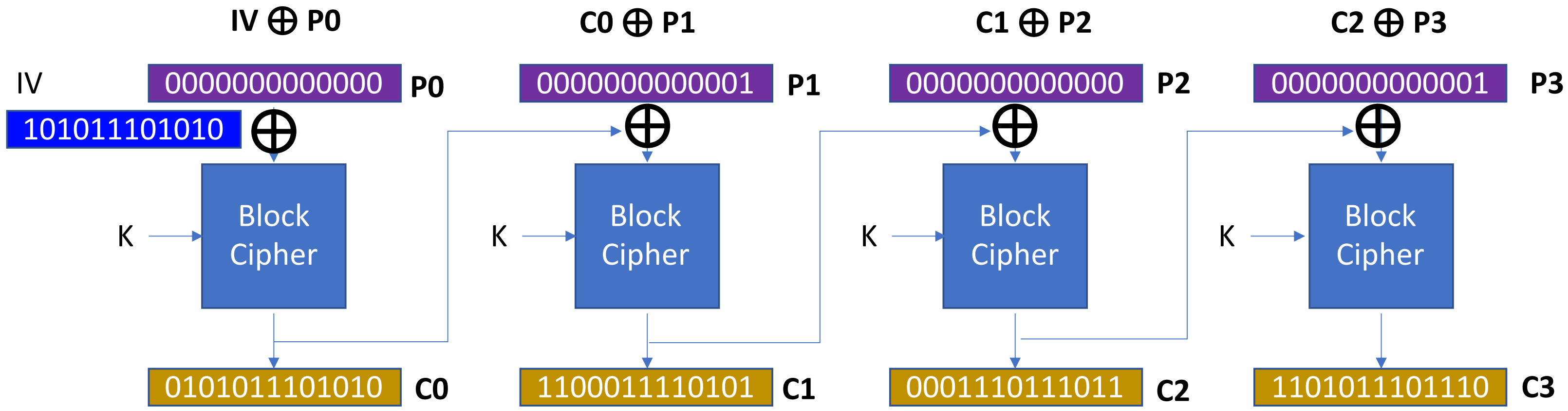
Electronic code block – cont'd

- ECB weakness(es)
  - Using the same key leads to the same ciphertext
  - An adversary can guess the message by looking at the ciphertext
  - An adversary can modify the ciphertext to compromise the plaintext



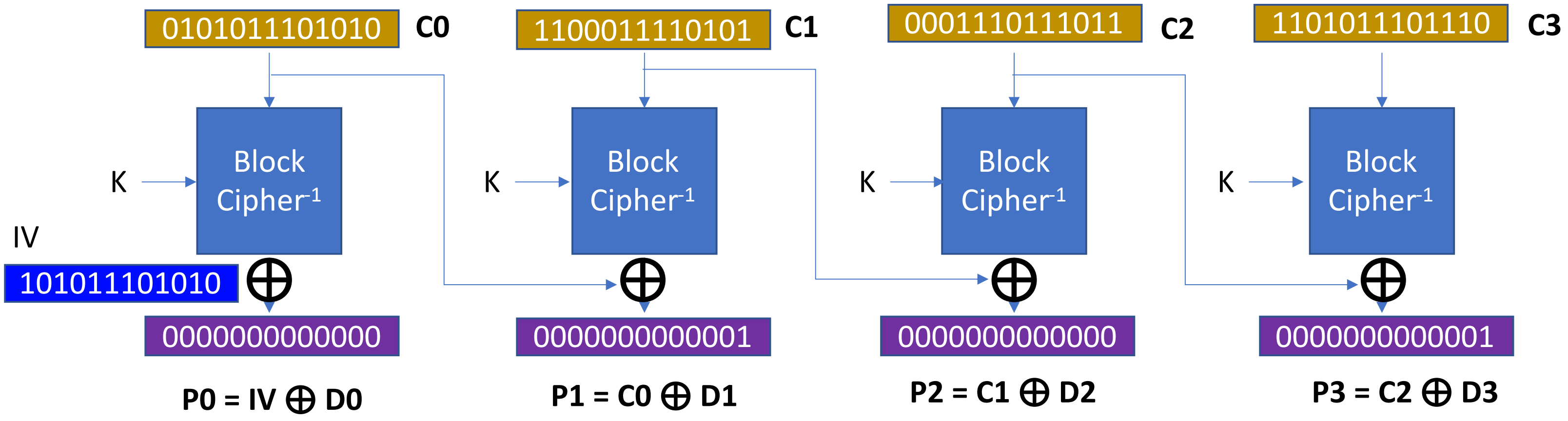
Cipher block chain

- CBC
  - Operations
    - M: XOR between IV (initialization vector) and the P0 (plaintext)
    - Encryption: use the ciphertext from the prev. block as IV and run block encryption



Cipher block chain – cont'd

- CBC
  - Operations
    - M: XOR between IV (initialization vector) and the P0 (plaintext)
    - Encryption: use the ciphertext from the prev. block as IV and run block encryption
    - Decryption: user the ciphertext from the prev. block as IV and run block decryption

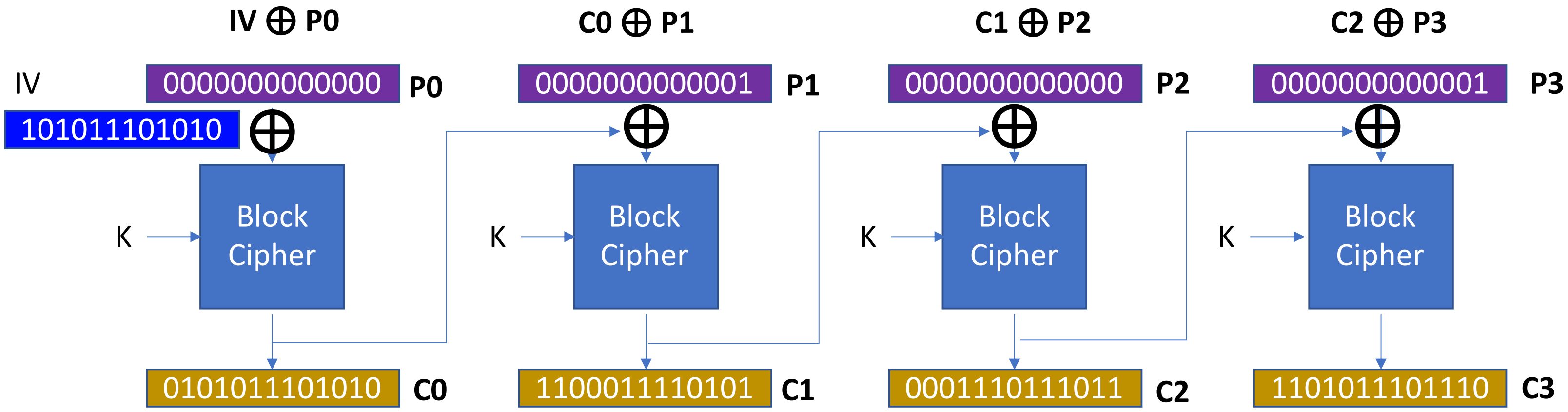


Cipher block chain – cont'd

- CBC
  - Operations
    - M: XOR between IV (initialization vector) and the P0 (plaintext)
    - Encryption: use the ciphertext from the prev. block as IV and run block encryption
    - Decryption: user the plaintext from the prev. block as IV and run block decryption
  - Benefits
    - Address the ECB's weakness
      - Both encryption and decryption are not deterministic
      - We can do this by choosing a random IV
    - Check it out by yourself: link to cbc-encrypted image

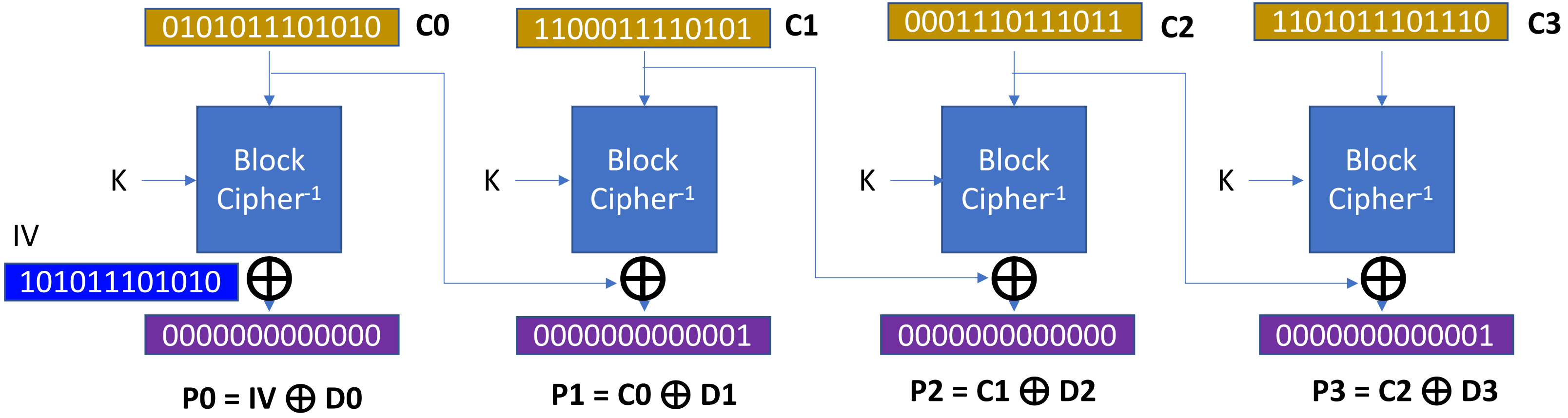
Cipher block chain – cont'd

- CBC weakness
  - Can't run encryption in parallel



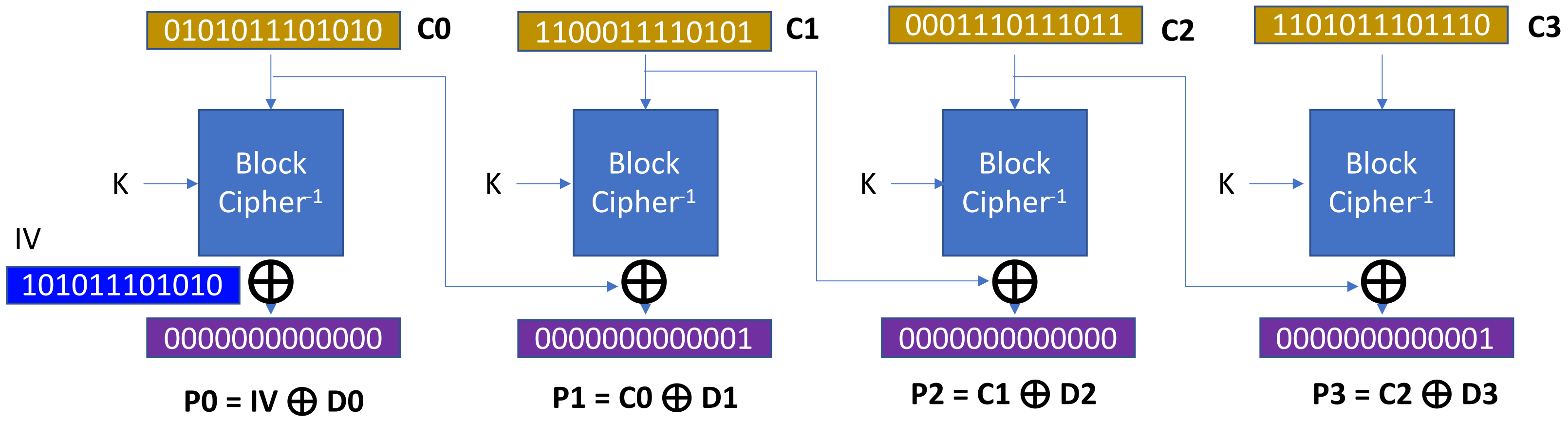
Cipher block chain – cont'd

- CBC weakness
  - Can't run encryption in parallel
  - But can run decryption in parallel (why this is a weakness?)



Cipher block chain – cont'd

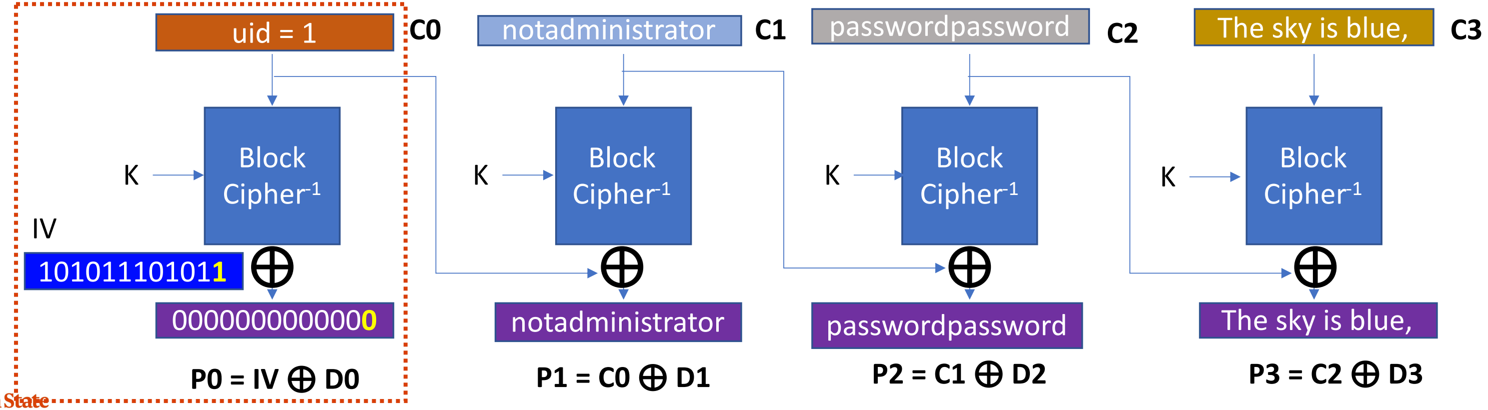
- CBC weakness
  - Can't run encryption in parallel
  - But can run decryption in parallel
  - An attacker can alter the previous block's ciphertext to manipulate the current block's plaintext



Recap: Micro-Lab: Exploiting the Weakness of CBC

- Job 1
  - Create a copy of this data with ‘uid == 0’
  - Use template.py (marked as XXX)
  - (Warning) we cannot use the last block
- Hint
  - Find a way to flip the decrypted value of the 1st block

What if we flip IV’s last bit from 0 to 1

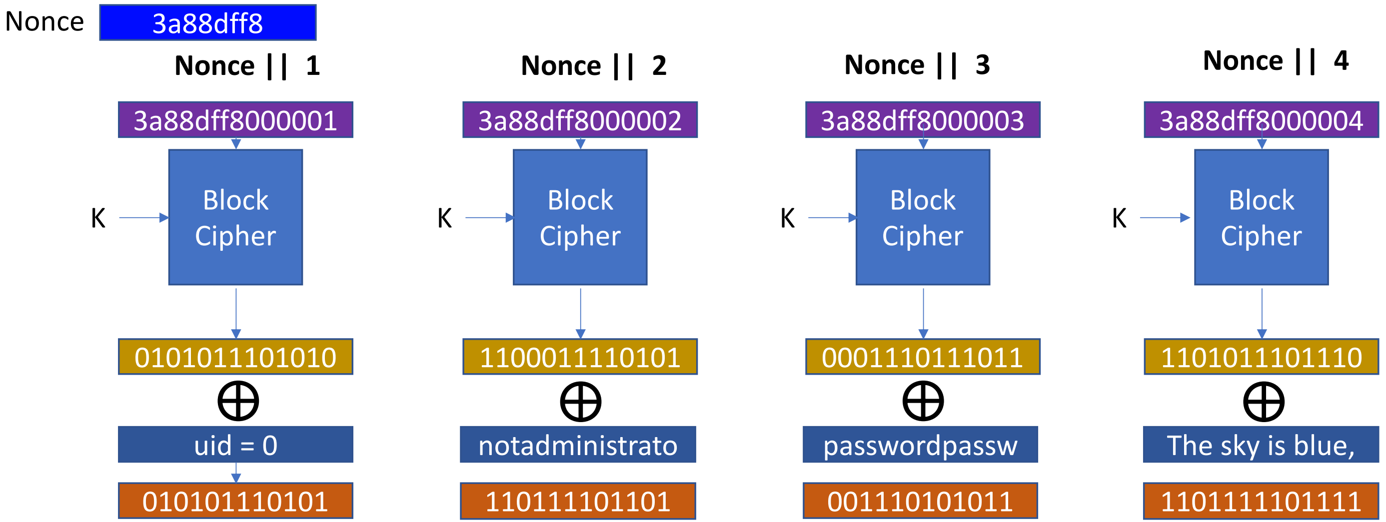


Topics for today

- Recap
  - Block cipher modes
  - ECB and CBC
  - ECB and CBC's weaknesses and exploitations
- Block cipher modes
  - Counter modes (CTR)
  - CTR's weakness
- Cryptographic hash
  - Message authentication code (MAC)
  - SHA256
  - HMAC

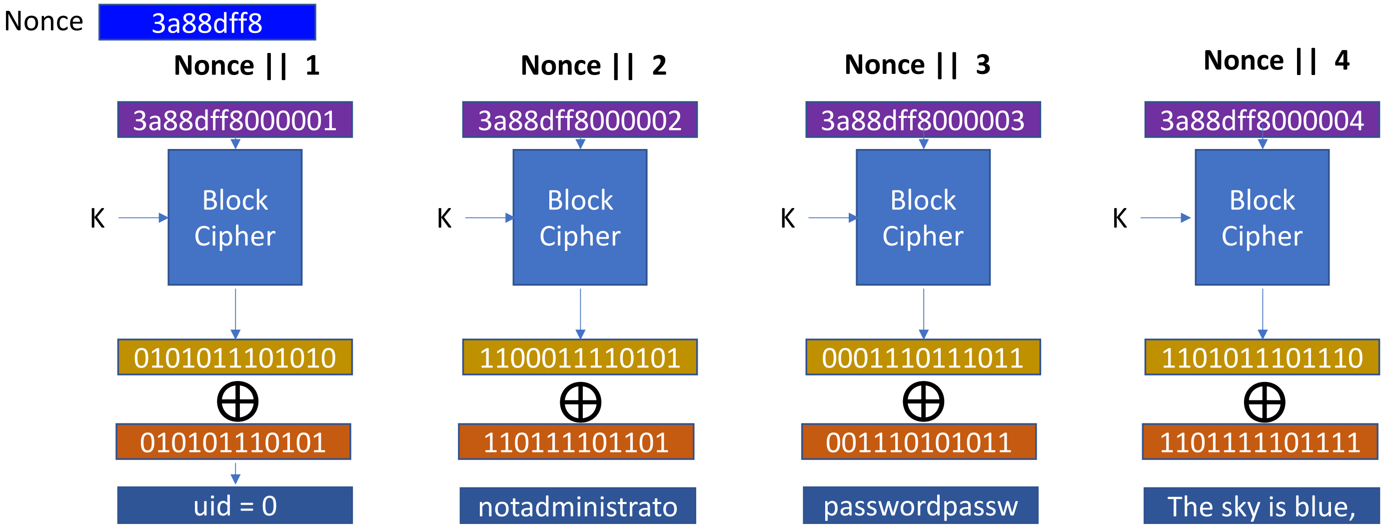
Counter mode: encryption

- CTR
  - A popular block cipher mode
  - Operations
    - Start with a random nonce || counter
    - Encryption: encrypt the random nonce || counter and XOR the result with a plaintext



Counter mode: decryption

- CTR
  - A popular block cipher mode
  - Operations
    - Start with a random nonce || counter
    - Decryption: decrypt the random nonce || counter and XOR the result with a ciphertext

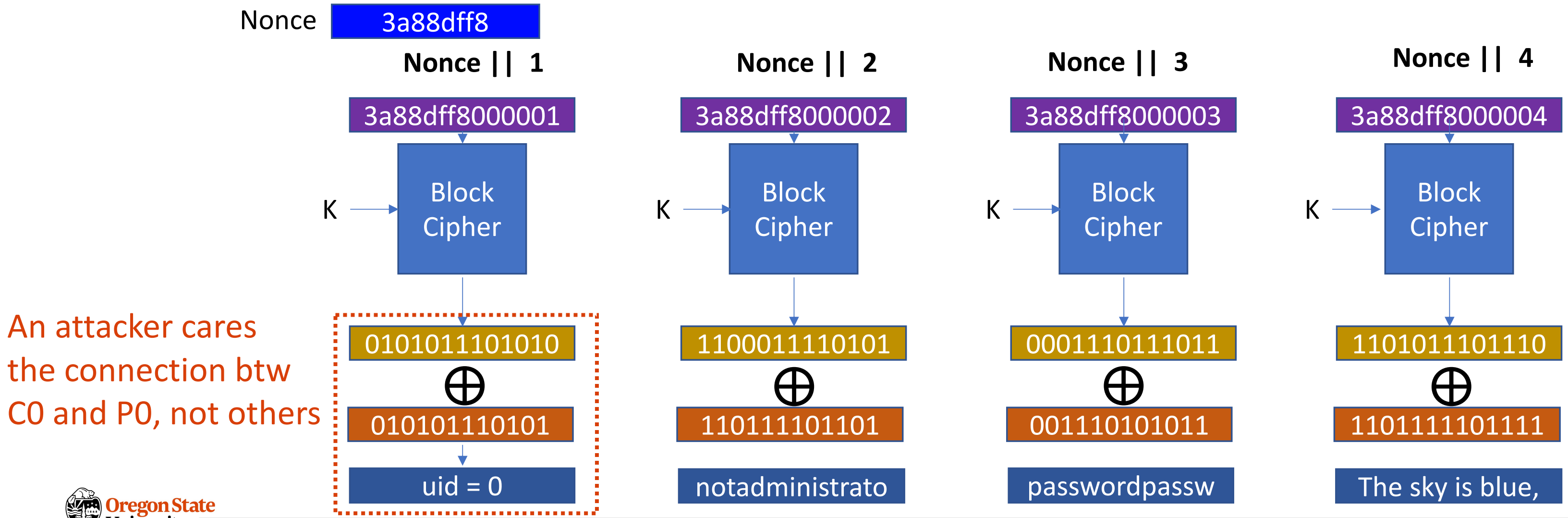


Counter mode

- CTR
  - A mode of block cipher operations
  - Operations
    - Start with a random nonce || counter
    - Encryption: encrypt the random nonce || counter and XOR the result with a plaintext
    - Decryption: decrypt the random nonce || counter and XOR the result with a ciphertext
  - Benefits
    - We can run encryption and decryption in parallel

Counter mode: weakness

- CTR weakness
  - Any alteration in the ciphertext will be reflected on the plaintext
  - Enjoy 3 Micro-labs on ctr-attack ☺



Summary

- ECB, CBC, CTR…
  - Block cipher modes
  - A common weakness
    - An adversary can manipulate encrypted data
    - such a way that they can alter the plaintext data as they want
    - ECB: an adversary can know the mappings btw ciphertext and plaintext and exploit them
    - CBC: an attacker can manipulate the ciphertext of the previous block to do alterations
    - CTR: an attacker can manipulate the ciphertext directly to do alterations

How Can We Address Such Weaknesses?

Topics for today

- Recap
  - Block cipher modes
  - ECB and CBC
  - ECB and CBC's weaknesses and exploitations
- Block cipher modes
  - Counter modes (CTR)
  - CTR's weakness
- Cryptographic hash
  - Message authentication code (MAC)
  - SHA256
  - HMAC

Cryptographic hash

- Cryptographic hash
  - Hash functions with specific properties
    - A function: f(x) = y
    - Generate a fixed-length output (e.g., 256-bit: 32-byte)
    - Desirable security properties
      - Make it difficult to find the inverse: $f^{-1}(y) = x$
      - Knowing the mappings of (x, y) does not help with inferring f(x') = ?
      - (Ideally) X and Y are independent to each other

Cryptographic hash

- Cryptographic hash
  - Hash functions with specific properties
    - A function: f(x) = y
    - Generate a fixed-length output (e.g., 256-bit: 32-byte)
    - Desirable security properties
      - Make it difficult to find the inverse: $f^{-1}(y) = x$
      - Knowing the mappings of (x, y) does not help with inferring f(x') = ?
      - (Ideally) X and Y are independent to each other
  - Benefits (enables MAC)
    - We can check the integrity of the ciphertext before we decrypt
    - The sender sends a ciphertext C with the hash f(salt + C) to receiver
    - The receiver runs f(salt + C) by themselves and see if it matches with the sender's

Cryptographic hash

- Message authentication code (MAC)
  - How to compute?
    - f(salt + C) = MAC
    - f(salt + | IV | Block 0 | Block 1 |) = MAC

Cryptographic hash

- Message authentication code (MAC)
  - How to compute?
    - f(salt + C) = MAC
    - f(salt + [IV | Block 0 | Block 1]) = MAC
  - How to send?
    - Append the MAC block in the end and send to a receiver

IV	Block 0	Block 1	MAC

Cryptographic hash

- Message authentication code (MAC)
  - How to compute?
    - f(salt + C) = MAC
    - f(salt + [IV | Block 0 | Block 1]) = MAC
  - How to send?
    - Append the MAC block in the end and send to a receiver

      [IV | Block 0 | Block 1 | MAC]
  - How to check?
    - Receiver computes
    - f(salt + [IV | Block 0 | Block 1]) = MAC’
    - Checks if MAC’ = MAC

Cryptographic hash

- We can achieve message integrity
  - Suppose an adversary manipulate the ciphertext

- Receiver will compute f(salt + 

) = MAC’
- Receiver will notice MAC’ != MAC
- It’s easy for the receiver to identify MAC’ != MAC as f(x) is designed to make a completely different MAC’ even under a small changes in x

Cryptographic hash

- We can achieve message integrity
  - Suppose an adversary manipulate the ciphertext

    - Receiver will compute f(salt + 

) = MAC’
    - Receiver will notice MAC’ != MAC
    - It’s easy for the receiver to identify MAC’ != MAC as f(x) is designed to make a completely different MAC’ even under a small changes in x

  - Suppose an adversary knows the salt (key) and manipulate the ciphertext

    - Receiver will compute f(salt + 

) = MAC’’
    - Receiver will notice MAC’’ == MAC_X

Cryptographic hash function

- SHA256
  - A hash function that generates a fingerprint of a data
  - It returns 32-byte (256-bit) hashed value for any length data
    - SHA256('Hello, world') =
      `03675ac53ff9cd1535ccc7dfcdfa2c458c5218371f418dc136f2d19ac1fbe8a5`
  - The function has some security properties:
    - One-way function
    - Hard to find x for given y where H(x) = y
    - Hard to find x' for given x,y where x != x', H(x) = y and H(x') = y

Cryptographic hash function

- SHA256
  - SHA256 is in the SHA2 standard
  - Input *x* can be any-length data and output *y* is 256-bit
    (Hash collision: two or more inputs can be mapped to the same hash value)

- Desirable properties of SHA256
  - It is one-way function
  - SHA256(‘Hello, world’) =
    `03675ac53ff9cd1535ccc7dfcdfa2c458c5218371f418dc136f2d19ac1fbe8a5`
  - $SHA256^{-1}$(`03675ac53ff9cd1535ccc7dfcdfa2c458c5218371f418 dc136f2d19ac1fbe8a5`) ==
    ???? there could be many..

SHA256 Examples

```
> sha256sum *
9a271f2a916b0b6ee6cecb2426f0b3206ef074578be55d9bc94f6f3fe3ab86aa  0
4355a46b19d348dc2f57c046f8ef63d4538ebb936000f3c9ee954a27460dd865  1
53c234e5e8472b6ac51c1ae1cab3fe06fad053beb8ebfd8977b010655bfdd3c3  2
1121cfccd5913f0a63fec40a6ffd44ea64f9dc135c66634ba001d10bcf4302a2  3
7de1555df0c2700329e815b93b32c571c3ea54dc967b89e81ab73b9972b72d1d  4
f0b5c2c2211c8d67ed15e75e656c7862d086e9245420892a7de62cd9ec582a06  5
06e9d52c1720fca412803e3b07c4b228ff113e303f4c7ab94665319d832bbfb7  6
10159baf262b43a92d95db59dae1f72c645127301661e0a3ce4e38b295a97c58  7
aa67a169b0bba217aa0aa88a65346920c84c42447c36ba5f7ea65f422c1fe5d8  8
2e6d31a5983a91251bfae5aefa1c0a19d8ba3cf601d0e8a706b4cfa9661a6b8a  9
```

SHA256

- One-way function
  - Hard to find $f^{-1}(y) = x$
  - A brute-force attacker requires 2^{256} times of search for finding the inverse

- Security implication
  - If we know x, it is easy to get SHA256(x) = y
  - But if we don’t know x, even if we know y, it is hard to calculate x

SHA256

- Hash collisions
  - Input space is much larger than the output space
  - Many x exists that satisfy H(x) = y
  - SHA256(‘Hello, world’) = SHA256(‘Something else’)

- Security implication
  - Hard to hit the exact x used by the sender that satisfies SHA256(x) = y

SHA256

- Avalanche effect
  - Hard to find x' for given x,y where x' != x, H(x) = y, H(x') = H(x)
  - SHA256('Hello, world') =
    `03675ac53ff9cd1535ccc7dfcdfa2c458c5218371f418dc136f2d19ac1fbe8a5`
  - Can you find another x' that produces SHA256(x') =
    `03675ac53ff9cd1535ccc7dfcdfa2c458c5218371f418dc136f2d19ac1fbe8a5`
  - Other than 'Hello, world'?

- Implication
  - Even if we know X, Y where SHA256(X) = Y
  - It is hard to find SHA256(X') = Y

SHA256

- Avalanche effect
  - A small change in the input leads to a huge difference in the output

```
> sha256sum *
9a271f2a916b0b6ee6cecb2426f0b3206ef074578be55d9bc94f6f3fe3ab86aa  0
4355a46b19d348dc2f57c046f8ef63d4538ebb936000f3c9ee954a27460dd865  1
53c234e5e8472b6ac51c1ae1cab3fe06fad053beb8ebfd8977b010655bfdd3c3  2
1121cfccd5913f0a63fec40a6ffd44ea64f9dc135c66634ba001d10bcf4302a2  3
7de1555df0c2700329e815b93b32c571c3ea54dc967b89e81ab73b9972b72d1d  4
f0b5c2c2211c8d67ed15e75e656c7862d086e9245420892a7de62cd9ec582a06  5
06e9d52c1720fca412803e3b07c4b228ff113e303f4c7ab94665319d832bbfb7  6
10159baf262b43a92d95db59dae1f72c645127301661e0a3ce4e38b295a97c58  7
aa67a169b0bba217aa0aa88a65346920c84c42447c36ba5f7ea65f422c1fe5d8  8
2e6d31a5983a91251bfae5aefa1c0a19d8ba3cf601d0e8a706b4cfa9661a6b8a  9
```

SHA256

- Avalanche effect
  - A small change in the input leads to a huge difference in the output
  - Input space X is independent to the output space Y (Perfect security?)

- Security implication
  - An adversary cannot find the relationship between x and y
    - x^1, $H(x) = y^1$
    - x^2, $H(x) = y^2$
    - …
  - Even if $x^1 \sim x^2$, y^1 and y^2 are not similar at all

Cryptographic hash with a key (secret or salt)

- Hard to find the inverse
  - H("secret" + message) = hash
  - Hard to find the "secret" from hash

- Hard to generate a valid hash without knowing the secret
  - From given M, h where H ("secret" + M) = h
  - H ("secret" + M') = h' without knowing the "secret"

Topics for today

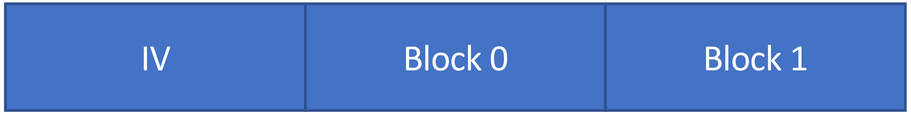
- Recap
  - Block cipher modes
  - ECB and CBC
  - ECB and CBC's weaknesses and exploitations
- Block cipher modes
  - Counter modes (CTR)
  - CTR's weakness
- Cryptographic hash
  - Message authentication code (MAC)
  - SHA256
  - HMAC

HMAC

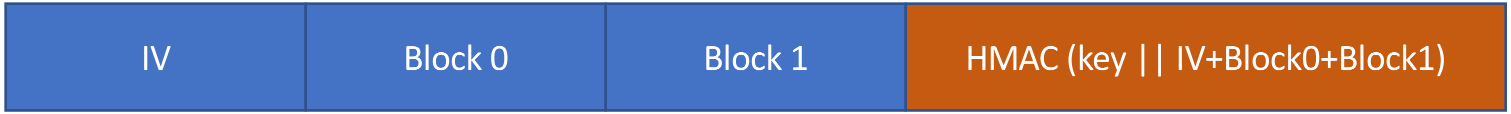
- Hash-based message authentication code (HMAC)
  - H = a hash function (e.g., SHA256)
  - HMAC = H(H(K) || M)
  - K: secret key (salt)
  - H(K): hash of the key
  - M: message or data

HMAC WITH ENCRYPTED DATA

- CBC Data (32-byte blocks)

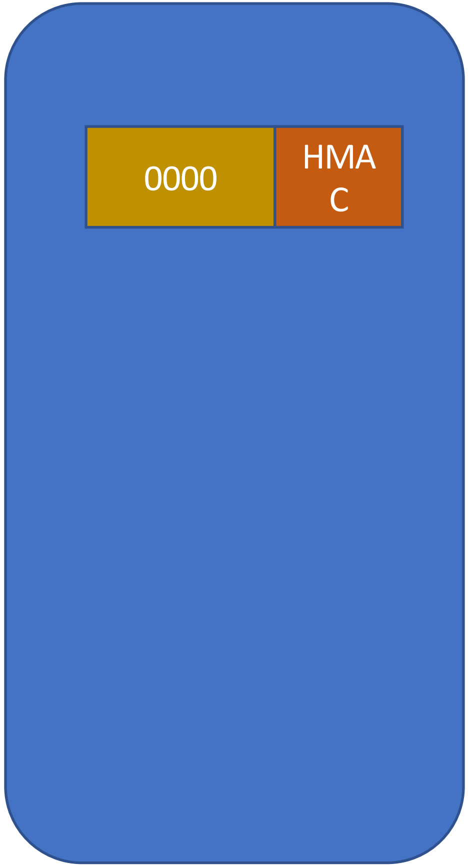


- Suppose you have a hash key = ‘asdf’
  - HMAC = SHA256(SHA256(‘asdf’) || encrypted_data)
  - = 7624e1f89ce009f8ec7e6e39781a42c0a27fa38f94db4f05f78b0f301007e06a



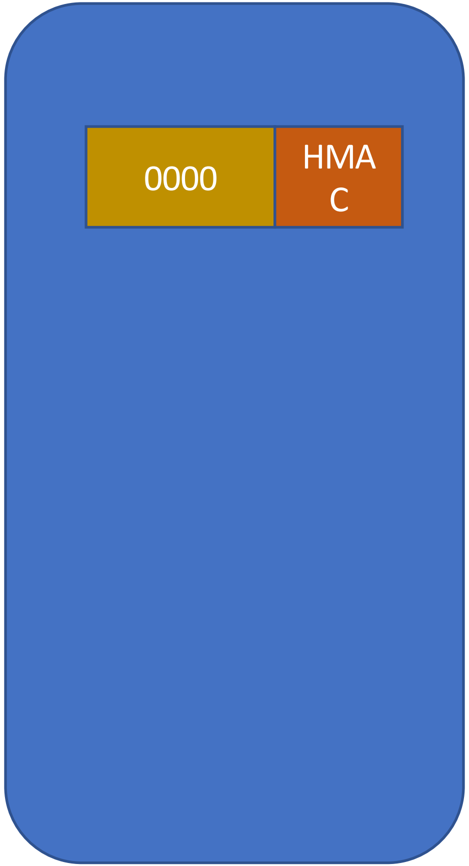
Checking the integrity with HMAC

I encrypt data
& added HMAC!
HMAC(key||0000)



Checking the integrity with HMAC

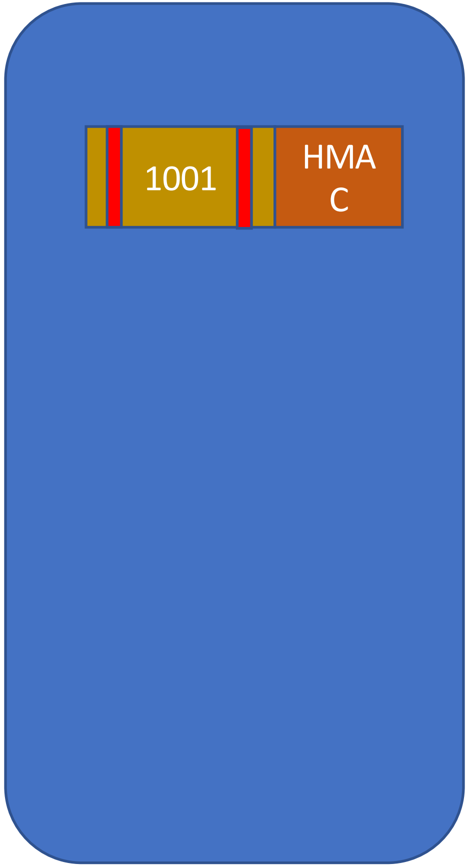
I encrypt data
& added HMAC!
HMAC(key||0000)



Edit data…

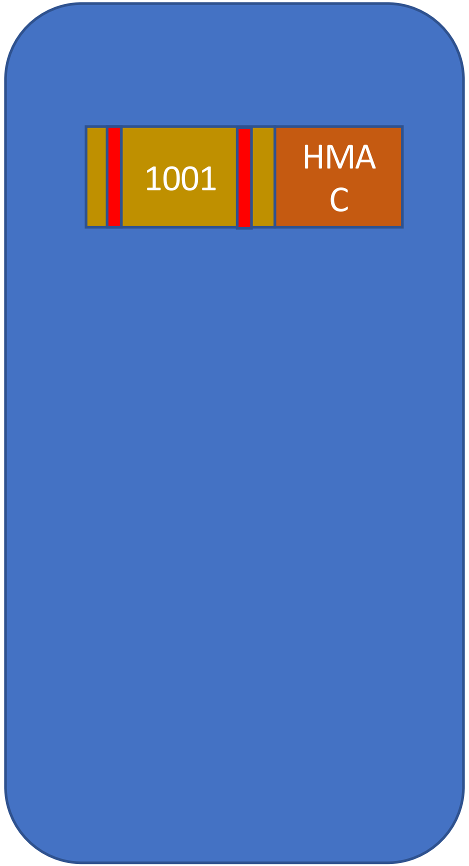
CHECKING THE INTEGRITY WITH HMAC

I encrypt data
& added HMAC!
HMAC(K||0000)



Edit data…

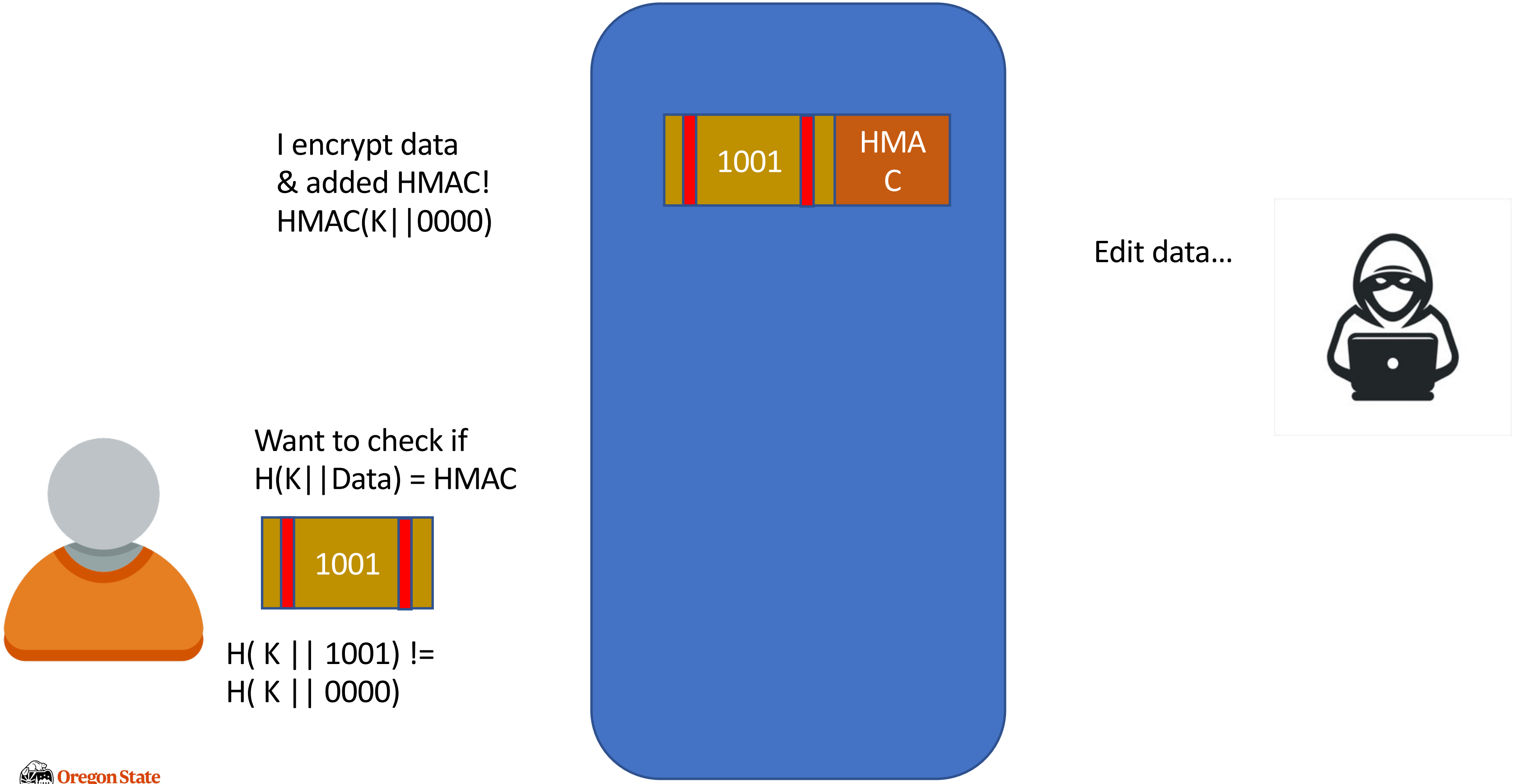
Checking the integrity with HMAC



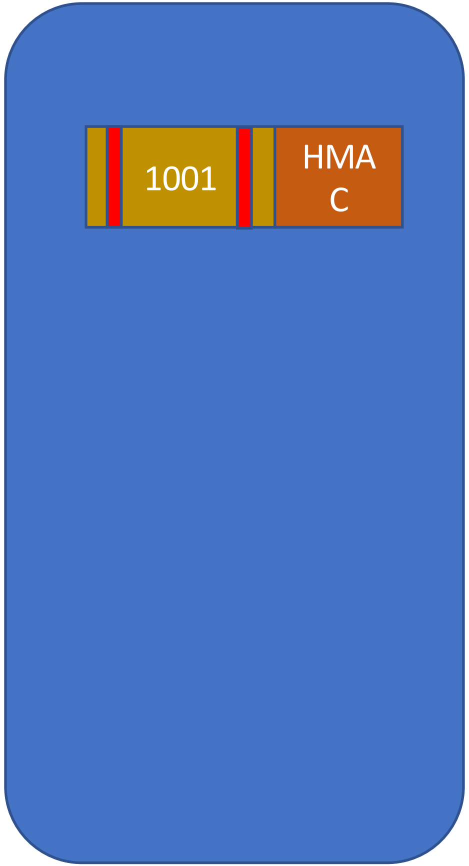
Edit data…

Want to check if
H(K||Data) = HMAC

Checking the integrity with HMAC

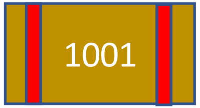


Checking the integrity with HMAC



Edit data…

Want to check if
H(K||Data) = HMAC

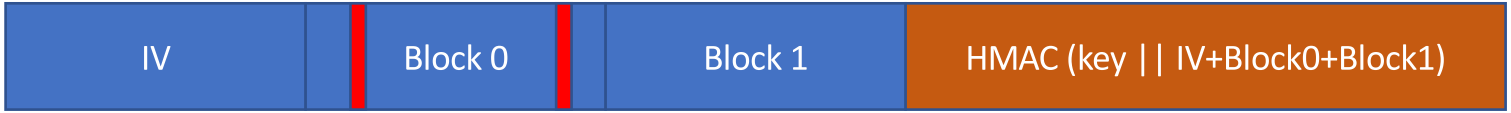


H(K || 1001) !=
H(K || 0000)

Reject!

Checking the integrity with HMAC

- Suppose you have a hash key = 'asdf'
  - HMAC = SHA256(SHA256('asdf') || encrypted_data)
  - = `7624e1f89ce009f8ec7e6e39781a42c0a27fa38f94db4f05f78b0f301007e06a`
- Suppose the attacker changed the encrypted_data



  - HMAC = SHA256(SHA256('asdf') || **encrypted_data**)
  - = `389205904d6c7bb83fc676513911226f2be25bf1465616bb9b29587100ab1414`

- Mismatch with HMAC!

Preserving the integrity with HMAC

- Can an attacker edit HMAC to match that to the edited ciphertext?
  - HMAC = SHA256(SHA256('key') || edited_data)
  - Attackers don't know the key
    - That's why we need to use key to SHA256.
    - Otherwise, anyone can generate valid MAC!
  - Even they know SHA256(SHA256('key')|| encrypted_data)
    - They cannot generate a valid HMAC
    - They cannot correlate that value from this one…

Summary

- Block cipher (mode)s:
  - Encryption/decryption operation is performed as a block-basis
  - But attackers can alter ciphertexts to modify plaintexts (Micro-labs)
  - They only offers data confidentiality
- Cryptographic hash functions
  - Used to offer data integrity
  - Hard to find $f^{-1}(y) = x$ and X and Y (input and output spaces) are independent
  - Work as a certificate that allows receivers to check the integrity of received data
  - MAC and HMAC (advanced version, working with a key)

Summary

- Recommendations
  - Use MAC with encrypted data (not with plaintext data)
  - Do ‘encrypt-then-MAC’
  - Do not do `MAC-then-encrypt`
    - We cannot know the integrity of ciphertext
    - We do not know MAC until we decrypt the data
    - Cryptanalysis attacks…

Topics for today

- Recap
  - Block cipher modes
  - ECB and CBC
  - ECB and CBC’s weaknesses and exploitations
- Block cipher modes
  - Counter modes (CTR)
  - CTR’s weakness
- Cryptographic hash
  - Message authentication code (MAC)
  - SHA256
  - HMAC

Thank You!

Tu/Th 4:00 – 5:50 PM

Sanghyun Hong

sanghyun.hong@oregonstate.edu

Oregon State University

SAIL
Secure AI Systems Lab