CS 370
Introduction to Security

Quiz 1 Prep

Yeongjin Jang

Oregon State University

Quiz 1 (10/20)

- Released via CANVAS
  - You can see Quiz 1 at 8:30 am
  - Deadline: 10/21 11:59pm (opened for 2 days)
  - Duration: Unlimited, but you can finish it around 30 min
- You will be given up to 3 attempts to take quiz
- Open material; you may refer to
  - Contents at our course website: https://cs370.unexploitable.systems/cal.html
  - Slides
  - Lecture Videos
  - Your code for challenge assignments
  - Use python

Quiz 1 (10/20)

- Released via CANVAS
  - You can see Quiz 1 at 8:30 am
  - Deadline: 10/21 11:59pm (opened for 2 days)
  - Duration: Unlimited, but you can finish it around 30 min
- You will be given up to 3 attempts to take quiz
- Open material; you may refer to
  - Contents at our course website: https://cs370.unexploitable.systems/cal.html
  - Slides
  - Lecture Videos
  - Your code for challenge assi
  - Use python

Communicating with others during Quiz is
not allowed

Quiz 1 (10/20)

- Question type: multiple choices, around 20 questions
  - 1 pts per each question
- All three weeks content will be covered in the Quiz 1
  - Traditional and XOR cipher
  - Block Cipher
  - Block Cipher Modes
  - HMAC (Hash-based message authentication code)
  - RSA and Asymmetric Encryption
  - PKI/Digital Certificates/Diffie--Hellman

Sample Question 1

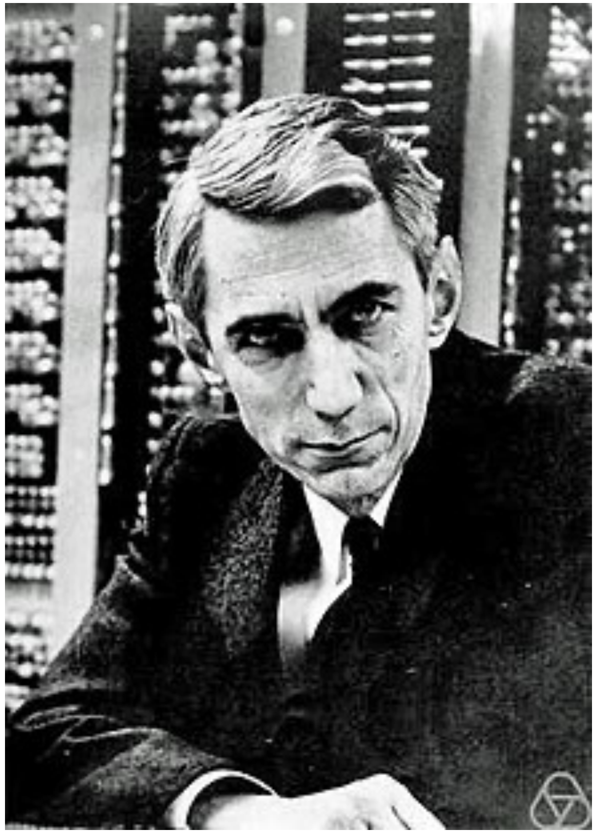
- Decipher the following string, encrypted with the CAESAR cipher

Sample Question 2

- Which of the following description is true for the perfect secrecy?

What is a Secure Cryptography?

- Shannon’s Intuition
  - If attackers cannot distinguish a message M from
  - A random number R
  - Then it is perfectly secure

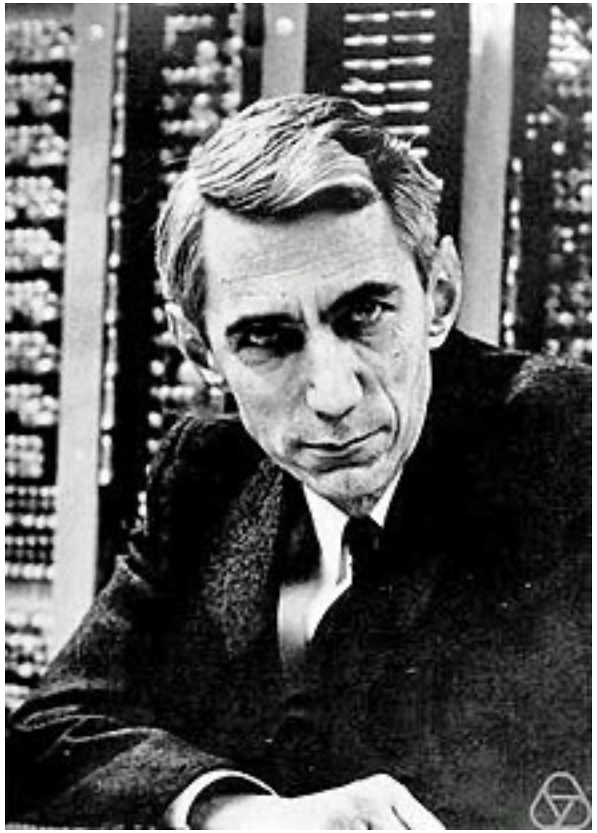


Claude Shannon (1916 ~ 2001)
A Father of Information Theory
and Modern Cryptography

What is a Secure Cryptography?

- More formally
  - A message M has a distribution D
  - D is known to adversary (English, etc..)
  - Adversary observes Ciphertext C which is Enc(M)
  - Knowledge of adversary before observing C
    - Distribution of D
  - Knowledge of adversary after observing C
    - Distribution of D | C

- Shannon Secrecy
  - Distribution of D == Distribution of D|C
  - Then the scheme is perfectly secure
- This intuitively means
  - Observing many Cs does not give any information to adversary



Claude Shannon (1916 ~ 2001)
A Father of Information Theory
and Modern Cryptography

Definition of Perfect Secrecy

- For every pair of m1, m2 in M and for all c,
- Pr [k <- KG : Enc (m1, k) = c] = Pr[k <- KG: Enc(m2, k) = c]

- Implications
  - Probability that the encryption of m1 with k resulting in c
  - is the same as
  - Probability that the encryption of m2 with k resulting in c

  - Adversary cannot distinguish which message the c corresponds to

Sample Question 3

- Which of the following is correct for the XOR cipher?

XOR Cipher

- A simple XOR Cipher is with perfect secrecy

- Scheme
  - For a message M with length L
  - Get a random key K with length L
  - Compute ciphertext C
    - $C = M \oplus K$

How is It Perfectly Secure?

- Key must be selected randomly
  - The distribution of K is random
- Ciphertext distribution is independent to the message distribution
  - $C = M \oplus K$
  - No matter how you choose M, if you choose K randomly, then it's good

CAVEAT

- Re-using the key make the scheme weak

- Suppose the attacker knows
  - HELLO -> 0x9, 0x7, 0xf, 0x8, 0xa

- They can calculate the key by

```
>>> chr(ord('H')^0x9)
'A'
>>> chr(ord('E')^0x7)
'B'
>>> chr(ord('L')^0xf)
'C'
>>> chr(ord('L')^0x8)
'D'
>>> chr(ord('O')^0xa)
'E'
```

Generic Version of XOR Cipher

- One-time Pad
  - https://en.wikipedia.org/wiki/One-time_pad

The resulting ciphertext will be impossible to decrypt or break if the following four conditions are met:[1][2]

1. The key must be at least as long as the plaintext.
2. The key must be random (uniformly distributed in the set of all possible keys and independent of the plaintext), entirely sampled from a non-algorithmic, chaotic source such as a hardware random number generator. It is not sufficient for OTP keys to pass statistical randomness tests as such tests cannot measure entropy, and the number of bits of entropy must be at least equal to the number of bits in the plaintext. For example, using cryptographic hashes or mathematical functions (such as logarithm or square root) to generate keys from fewer bits of entropy would break the uniform distribution requirement, and therefore would not provide perfect secrecy.
3. The key must never be reused in whole or in part.
4. The key must be kept completely secret by the communicating parties.

Sample Question 4

- Stream ciphers are convenient. Why don't we use RC4/RC5 stream ciphers?

Insecure RC4/RC5

- Read the Wikipedia articles for their weaknesses
  - https://en.wikipedia.org/wiki/RC4
  - https://en.wikipedia.org/wiki/RC5
- RC4

RC5

As of 2015, there is speculation that some state cryptologic agencies may possess the capability to break RC4 when used in the TLS protocol.[6] IETF has published RFC 7465 to prohibit the use of RC4 in TLS;[3] Mozilla and Microsoft have issued similar recommendations.[7][8]

Best public cryptanalysis

12-round RC5 (with 64-bit blocks) is susceptible to a differential attack using 2^{44} chosen plaintexts.[1]

- Have no mathematical proof or
- Have an attack against these…

Sample Question 5

- With a block cipher, we can encrypt/decrypt a block of message. For example, in AES, a block size is 16-byte (128-bit). Then, how can we encrypt a message that is shorter than a block size, e.g., a 5-byte message?

Can We Encrypt Blocks That Its Size is Less Than 16 byte?

- YES
  - We can ignore the rest of bits
- How?
  - Via padding (add some meaningless but identifiable data)

- ECB (Electronic Code Book)
  - A padding scheme to indicate the length of the message
  - Pad the length of the padding as byte for the length of the padding…

How ECB Works?

- Suppose you want to encrypt 15 byte of data
  - “0123456789ABCDE” ← a 15-byte string
  - ‘0’ ~ ‘9’ are 0x31 ~ 0x39, ’A’ ~ ‘E’ are 0x41 ~ 0x45

- We need 1 byte padding to make it to be 16-byte block

pos	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
val	0x30	0x31	0x32	0x33	0x34	0x35	0x36	0x37	0x38	0x39	0x41	0x42	0x43	0x44	0x45	0x01

- Make the padding value as the length of the padding
  - 0x1

How ECB Works?

- Suppose you want to encrypt 14 byte of data
  - “0123456789ABCD” ← a 14-byte string
  - ‘0’ ~ ‘9’ are 0x31 ~ 0x39, ’A’ ~ ‘E’ are 0x41 ~ 0x45

- We need 2 bytes padding to make it to be 16-byte block

pos	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
val	0x30	0x31	0x32	0x33	0x34	0x35	0x36	0x37	0x38	0x39	0x41	0x42	0x43	0x44	0x02	0x02

- Make the padding value as the length of the padding
  - 0x2 * 2

How ECB Works?

- Suppose you want to encrypt 1 byte of data
  - “0” ← a 1-byte string
  - ‘0’ ~ ‘9’ are 0x31 ~ 0x39, ’A’ ~ ‘E’ are 0x41 ~ 0x45

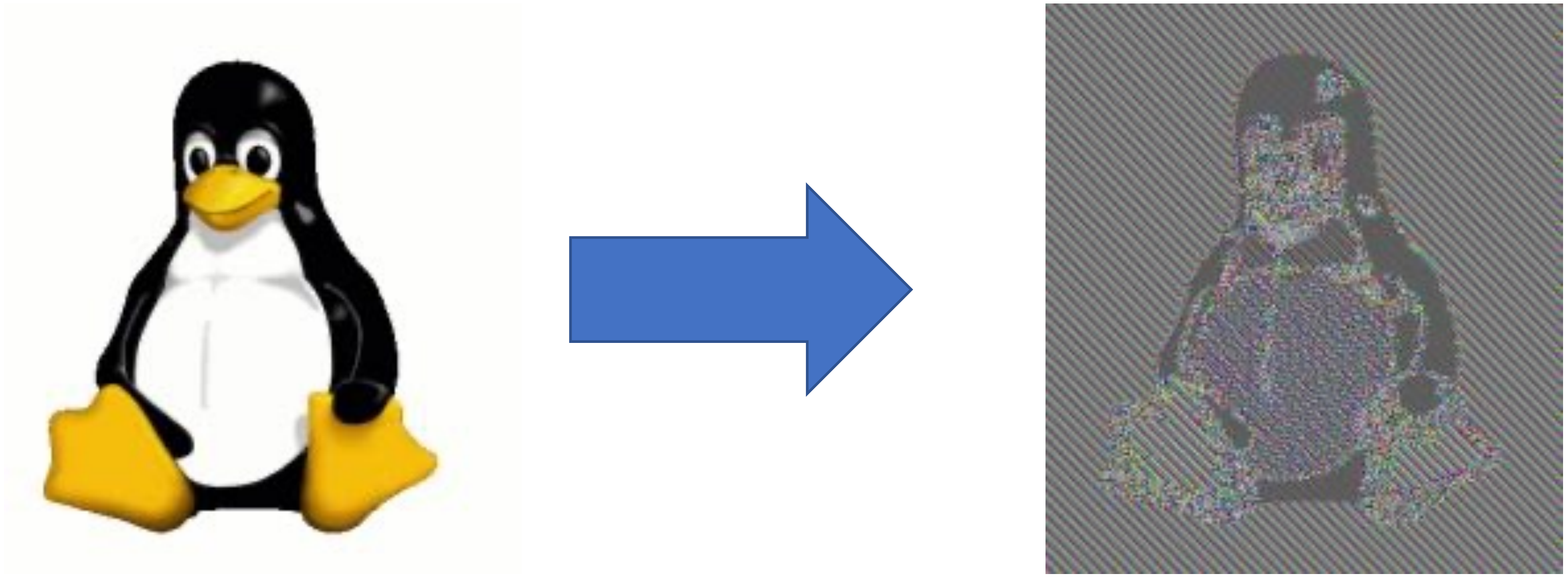
- We need 15 bytes padding to make it to be 16-byte block

pos	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
val	0x30	0x0f	0x0f	0x0f	0x0f	0x0f	0x0f	0x0f	0x0f	0x0f	0x0f	0x0f	0x0f	0x0f	0x0f	0x0f

- Make the padding value as the length of the padding
  - 0xf * 15

Sample Question 6

- What is the reason that we can indirectly see some content of original image if we encrypt a bitmap file in ECB mode?



Are There Any Weaknesses in ECB?

- For the same key, the same plaintext block will result in the same ciphertext

M: 10101010101010101010101010101010

Whenever an attacker observe d303fe9c04a4876930e4a5728f1eda4c
They know that it is the encryption of “\x10” * 16
(maybe the end of the message for 16-byte granularity)

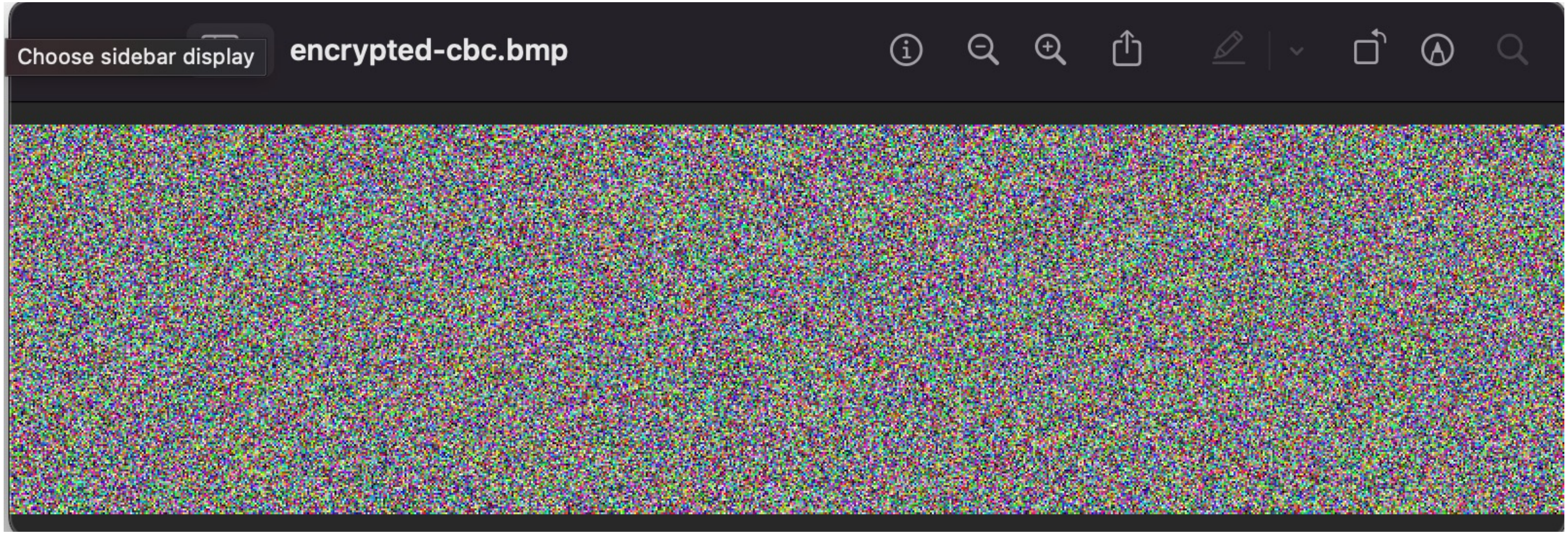
E: d303fe9c04a4876930e4a5728f1eda4c

ECB Weakness

- Suppose a message 0 is encrypted to
  - 0x39827332…
- Suppose a message 1 is encrypted to
  - 0x5a83f874…
- Suppose the attacker knows this via
  - Pattern analysis or something

Sample Question 7

- Why does CBC mode not have the same problem with ECB, in encrypting a bitmap file?

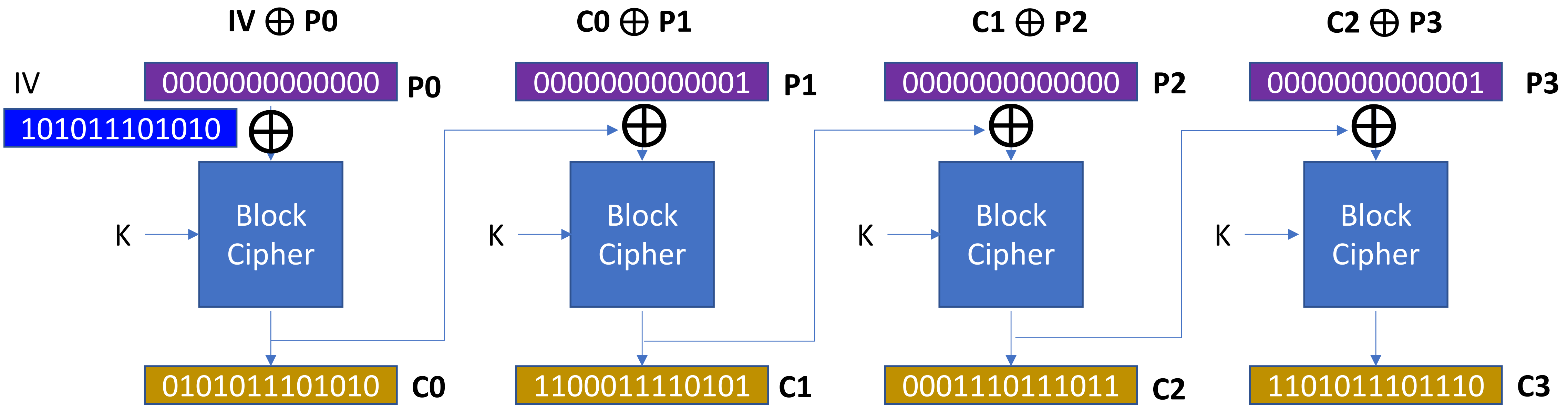


CBC Benefits

- ECB Weakness: 0 always encrypted in a fixed value… we can switch ciphertext to launch an attack
- Let's make each input to the block cipher looks like random
- Solution:
    - 1. Use a random IV, xor that to the plaintext; input will be random
    - 2. If the Block Cipher is PRP, the ciphertext looks like random
    - 3. Chain that random (ciphertext) to the next plaintext block…

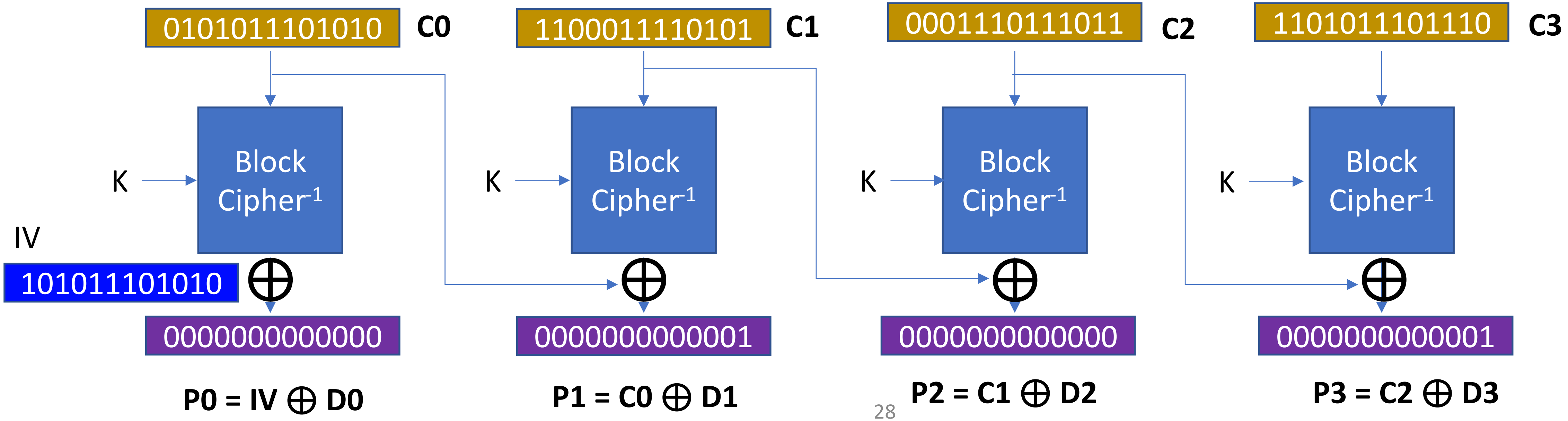
CBC Benefits

- Solution:
  - 1. Use a random IV, xor that to the plaintext; input will be random
  - 2. If the Block Cipher is PRP, the ciphertext looks like random
  - 3. Chain that random (ciphertext) to the next plaintext block…



Sample Question 8

- In CBC, which block do we need to apply the XOR tricks to change the plaintext marked as P2?
- Choices: IV, C0, C1, C2, C3

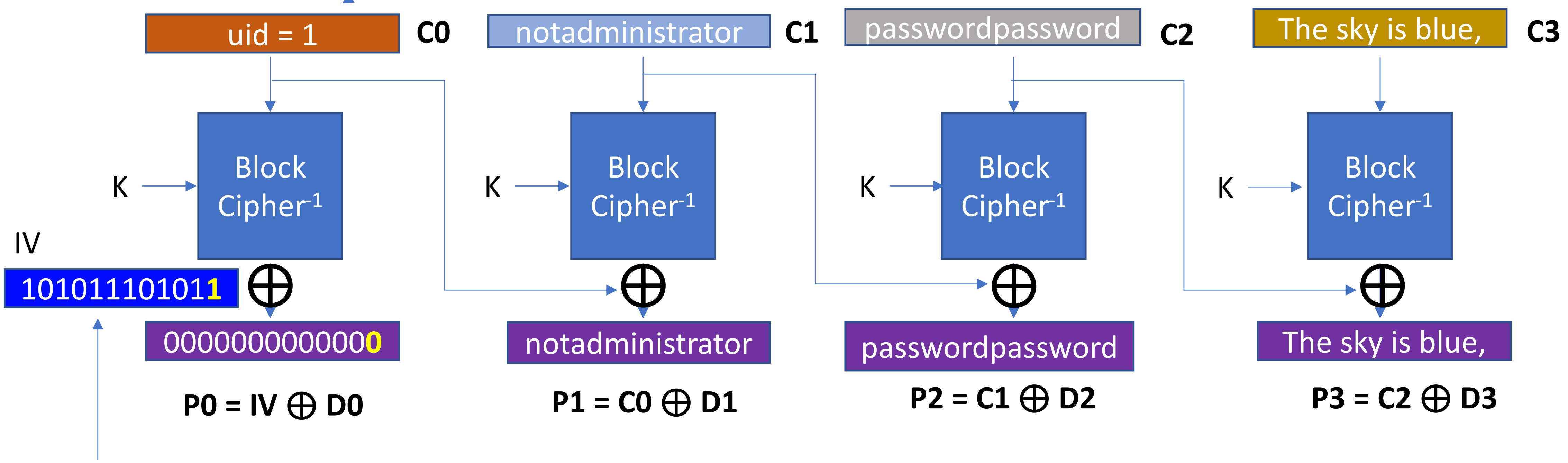


cbc-attack

- uid == 0

This block will be decrypted as:
0000000000000001

We don't manipulate the ciphertext



We manipulate IV; change the last bit from 0 to 1

Sample Question 9

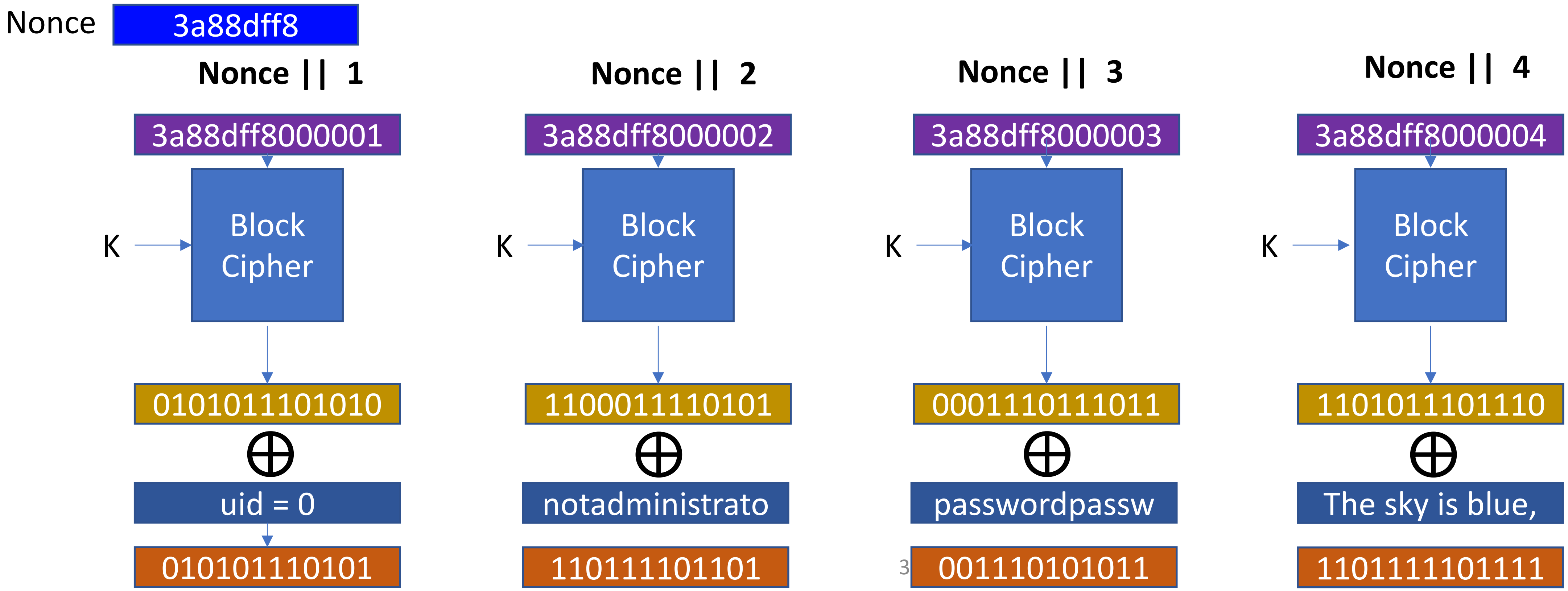
- Which of the following is false for the CTR mode?

Counter Mode

- CTR (Counter mode)
- Start with a random nonce || counter
- It uses the block cipher as random number generator
- V = Enc(nonce || counter)
- Then, XOR this V to the plaintext
  - $C = P \oplus V$

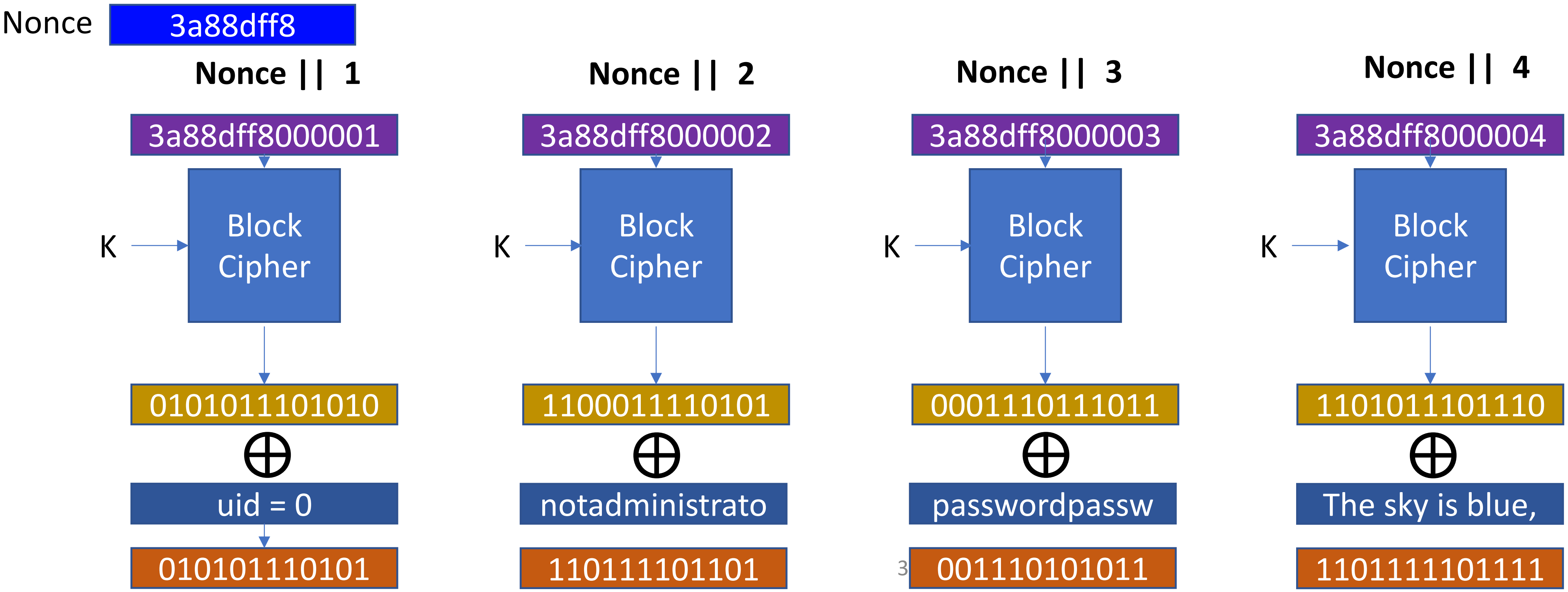
Counter Mode

- CTR (Counter mode)
- Start with a random nonce || counter



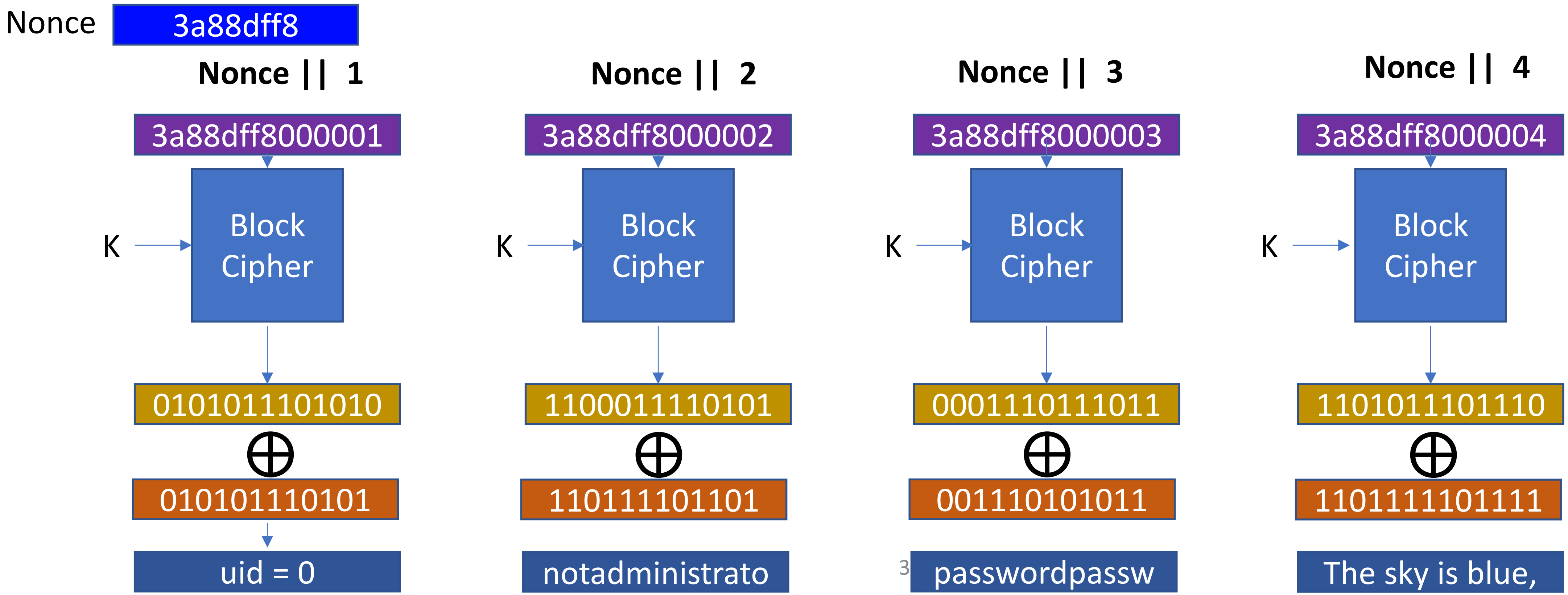
Counter Mode Encryption

- CTR (Counter mode)
- Start with a random nonce || counter



Counter Mode Decryption

- CTR (Counter mode)
- Start with a random nonce || counter



CTR Benefits

- Can encrypt in parallel
- Can decrypt in parallel
  - Basically, encryption and decryption are both encrypting counters
  - We apply XOR to the output with (plaintext/ciphertext)

CTR Weaknesses

- Any bitflip in the ciphertext will be direct bitflip in the plaintext..

Sample Question 10

- Which of the following is true for the cryptographic hash function?

Cryptographic Hash

- A hash function that generates a fingerprint of a data
- SHA256('Hello, world') =
  `03675ac53ff9cd1535ccc7dfcdfa2c458c5218371f418dc136f2d19ac1fbe8a5`

- With characteristics of:
  - 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

SHA256

- Secure Hash Algorithm (SHA)
  - SHA256 is in the SHA2 standard
  - Input can be any length data
  - Output is 256-bit, 32-byte
- SHA256 is a cryptographic hash function that
  - It is one-way function
  - SHA256('Hello, world') = `03675ac53ff9cd1535ccc7dfcdfa2c458c5218371f418dc136f2d19ac1fbe8a5`
  - $SHA256^{-1}$(`03675ac53ff9cd1535ccc7dfcdfa2c458c5218371f418dc136f2d19ac1fbe8a5`) == ???? there could be many..

SHA256 Examples

```
┌─blue9057@blue9057-vm-ctf1 ~/t ‹ruby-head›
└─$ sha256sum *
9a271f2a916b0b6ee6cecb2426f0b3206ef074578be55d9bc94f6f3fe3ab86aa  0
4355a46b19d348dc2f57c046f8ef63d4538ebb936000f3c9ee954a27460dd865  1
53c234e5e8472b6ac51c1ae1cab3fe06fad053beb8ebfd8977b010655bfdd3c3  2
1121cfccd5913f0a63fec40a6ffd44ea64f9dc135c66634ba001d10bcf4302a2  3
7de1555df0c2700329e815b93b32c571c3ea54dc967b89e81ab73b9972b72d1d  4
f0b5c2c2211c8d67ed15e75e656c7862d086e9245420892a7de62cd9ec582a06  5
87428fc522803d31065e7bce3cf03fe475096631e5e07bbd7a0fde60c4cf25c7  a
0263829989b6fd954f72baaf2fc64bc2e2f01d692d4de72986ea808f6e99813f  b
a3a5e715f0cc574a73c3f9bebb6bc24f32ffd5b67b387244c2c909da779a1478  c
8d74beec1be996322ad76813bafb92d40839895d6dd7ee808b17ca201eac98be  d
a2bbdb2de53523b8099b37013f251546f3d65dbe7a0774fa41af0a4176992fd4  e
```

SHA256

- SHA256 is a cryptographic hash function that
  - Hard to find x for given y where H(x) = y
  - Find SHA256(x) for
    - 0000000000000000000000000000000000000000f418dc136f2d19ac1fbe8a5
  - This task requires around the 2^{256} times of search…
- **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

- SHA256 is a cryptographic hash function that
  - 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**

Avalanche Effect

- Even with a slight change in input, we want to have a huge change in output to not making attackers correlate output values based on their inputs…

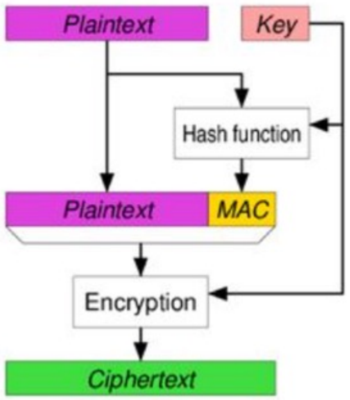
```
┌─blue9057@blue9057-vm-ctf1 ~/t ‹ruby-head›
└─$ sha256sum *
9a271f2a916b0b6ee6cecb2426f0b3206ef074578be55d9bc94f6f3fe3ab86aa  0
4355a46b19d348dc2f57c046f8ef63d4538ebb936000f3c9ee954a27460dd865  1
53c234e5e8472b6ac51c1ae1cab3fe06fad053beb8ebfd8977b010655bfdd3c3  2
1121cfccd5913f0a63fec40a6ffd44ea64f9dc135c66634ba001d10bcf4302a2  3
7de1555df0c2700329e815b93b32c571c3ea54dc967b89e81ab73b9972b72d1d  4
f0b5c2c2211c8d67ed15e75e656c7862d086e9245420892a7de62cd9ec582a06  5
87428fc522803d31065e7bce3cf03fe475096631e5e07bbd7a0fde60c4cf25c7  a
0263829989b6fd954f72baaf2fc64bc2e2f01d692d4de72986ea808f6e99813f  b
a3a5e715f0cc574a73c3f9bebb6bc24f32ffd5b67b387244c2c909da779a1478  c
8d74beec1be996322ad76813bafb92d40839895d6dd7ee808b17ca201eac98be  d
a2bbdb2de53523b8099b37013f251546f3d65dbe7a0774fa41af0a4176992fd4  e
```

Cryptographic Hash: Implications

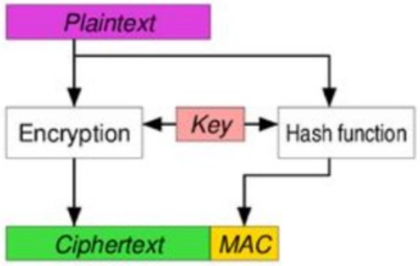
- SHA256 is a cryptographic hash that is included in the SHA2 standard
- SHA256 is a one-way function and
- It is hard to calculate x from y
  - where y = SHA256(x)
- It is hard to calculate x’ from x,y
  - where x’ != x, SHA256(x) = y, SHA256(x’) = y
- It is hard to correlate x and x’ from x, y, y’
  - where SHA256(x) = y, SHA256(x’) = y’

Sample Question 11

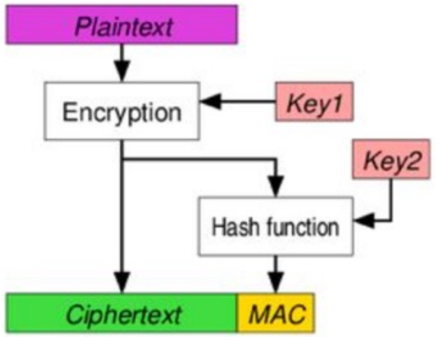
- Which of the following is a secure construction of HMAC that protects the integrity of the ciphertext?



- MAC then Encrypt



- Encrypt and MAC



- Encrypt then MAC

Summary

- Encrypt-then-MAC

- This is the only secure composition of using MAC with Encrypted data

- You must
  - Encrypt the data, and supply the entire encrypted data to HMAC
- No MAC-then-encrypt
  - Cryptanalysis exists (proven to be insecure)

Sample Question 12

- Which of the following is false for the RSA public key cryptography?

Public (Asymmetric) Key Cryptography

- There is a scheme that we use different key to encryption and decryption
  - Why is it important? We will discuss this later about the ‘key exchange’

- RSA (Rivest, Shamir, Adleman)
  - A public-key cryptography algorithm
  - Based on the difficulty of prime factorization
    - i.e., if the prime factorization of a large prime number is difficult, then the cryptography scheme is secure
  - Can be used for digital signature

How RSA Works?

- Choose two large prime number, p and q
- $N = pq$
- $\phi = (p-1)(q-1)$

- Choose public key, say, $e = 65537$ (a prime), that is coprime to ϕ
- Find $de == 1 \pmod{\phi}$
  - d can be efficiently be computed if you know ϕ
  - Blue: public key, Red: private key
  - Attackers don't know ϕ, to know ϕ, you need to factor N

RSA Encryption

- Public key: e, N
- Message: M

$$M^e \bmod N$$

RSA Decryption

- Private key: d
- Public key: e N
- Ciphertext = C = M^e
- ed = 1

$$C^d \bmod N$$

$$(M^e)^d \bmod N$$

$$M^{ed} \bmod N$$

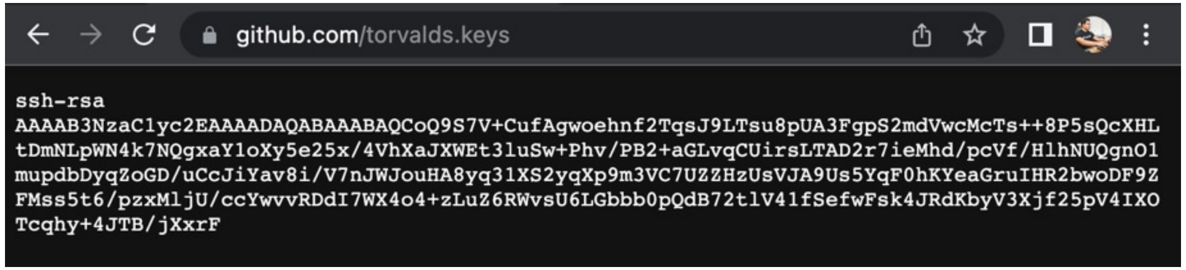
$$M \bmod N$$

Public Key Cryptography

- We can use separate key for encryption and decryption
  - Encryption key: public key (e, N)
  - Decryption key: private key (d)

- Attackers cannot guess the private key from the public key
  - In RSA, attacker must factor the prime number N = pq
  - In creating the key, we choose p and q as a big prime number
  - Factoring a multiplication of two big prime numbers is a difficult task

Characteristics



- We use a public key for encryption
  - We can publicize this key
  - If you publish your key, anyone who can access that can encrypt message
    - (e, N) is public, m^e mod N!
  - Only the holder of the private key can decrypt the message
    - d is private, $m^{ed} == m^1 == m \pmod N$

- Why is this important?
  - Let's talk about the key exchange problem

RSA: Digital Signature

- RSA can be used as a digital signature scheme
- What is that?

- In RSA, encryption is applying the public exponent to the message
  - $M^e \bmod N$
- In RSA, decryption is applying the private exponent to the message
  - $C^d \bmod N$

RSA: What will be the meaning of private encrypt?

- Suppose A encrypts the following message with her private key
  - "I would like to donate $100 to OSU if I get A from CS 370"
- M = 531514063336112570939562934115847599880532289387244271018598830892541197117394868377841674978391417646124501198563959951714555855196 13744
- C = m^d mod N

RSA: What will be the meaning of private encrypt?

- M = 531514063336112570939562934115847599880532289387244271018598830892541197117394868377841674978391417646124501198563959951714555855196137 44
- $C = m^d \bmod N$
- Anyone can have e. That means, anyone can decrypt C
  - $C^e == m^{de} == m^1 == m \pmod N$

RSA: What will be the meaning of private encrypt?

- M = 53151406333611257093956293411584759988053228938724427101859883089254119711739486837784167497839141764612450119856395995171455585519613744
- C = m^d mod N
- Anyone can have e. That means, anyone can decrypt C
  - $C^e == m^{de} == m^1 == m \pmod{N}$
  - m = 53151406333611257093956293411584759988053228938724427101859883089254119711739486837784167497839141764612450119856395995171455585519613744
  - "I would like to donate $100 to OSU if I get A from CS 370"

RSA: What will be the meaning of privat…

We can verify that the encrypted content C contains
The ciphertext that only the holder of private key can generate.
We all have public key, and if that is decrypted to
"I would like to donate $100 to OSU if I get A from CS 370",
then, we know that the holder of private key 'endorsed it'

- M =
  5315140
  1859883
  8563959
- $C = m^d \bmod N$
- Anyone can have e. That means, anyone can decrypt C
  - $C^e == m^{de} == m^1 == m \pmod N$
  - m =
    53151406333611257093956293411584759988053228938724427101859883089254119711739486837784167497839141764612450119856395995171455585519613744
  - "I would like to donate $100 to OSU if I get A from CS 370"

RSA: private_encrypt

- RSA Encryption using the private key is so-called as ‘Signing’
- Why?
  - The ciphertext will be decrypted as a plaintext using the public key
    - Anyone can decrypt!
  - But the ciphertext can only be generated with the private key
    - Only the private key owner can generate it!
- Implication
  - Holder of the private key generated a ciphertext message of message M
  - M is signed, endorsed by the holder’s private key
  - (Because it can only be generated with the private key)

RSA Summary

- Public/Private key Scheme
  - We can publish the public key – encryption key
  - We must hide the private key – decryption key
- Based on the difficulty of prime factorization
  - You cannot correlate the private key from the public key unless
  - You can factor a big number (a multiple of 2 big prime numbers)
- Anyone can encrypt message to the private key owner
  - Enc(public_key, message)
- Only the private key owner can decrypt message
  - Dec(private_key, encrypted_message)

RSA Summary

- Encryption with private key could be a ‘digital-signature’
  - Signed_message = Enc(private_key, message)
  - Message = Dec(public_key, signed_message)

- The correctly decrypted message using public key means that the private key holder have endorsed (‘encrypted’) the data
  - Anyone can verify this using the public key

Sample Question 13

- We have 10 people, and we would like to have secure and private communications among all those people (i.e., if there is users A, B, C, then the encrypted message between A and B must not be decrypted by C, while each of A and B can send encrypted messages to C).

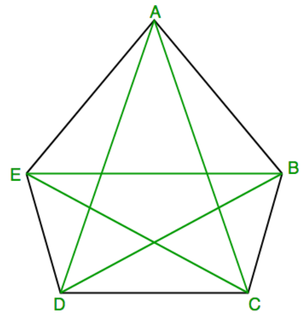
- To construct this kind of environment, how many keys do we need if we use 1) symmetric crypto and 2) asymmetric crypto?

- There will be choices such as a) 100 for symmetric, 10 for asymmetric, etc…

Key Exchange

- Suppose we have 5 people, A, B, C, D, E
  - How many keys do we need to have to make them communicate securely?
  - E.g., if A talks to B, C or others must not see the message
  - But anyone should be able to talk to anyone…
- A block cipher
  - We need 1 key for A and B can talk securely

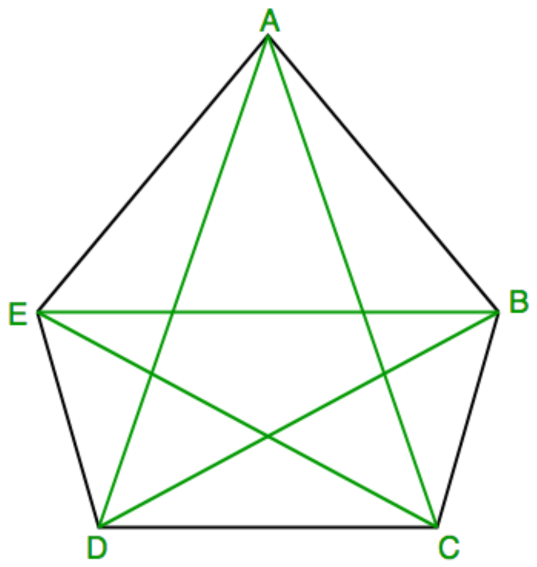
- How many keys do we need to let them communicate securely?
  - A-B, A-C, A-D, A-E
  - B-C, B-D, B-E
  - C-D, C-E
  - D-E
  - 10 keys (5*4/2 = 10)



Symmetric Key Cryptography

- Encryption and the decryption operations are using the same key
  - Block Cipher – encryption key == decryption key
  - You cannot share that other than 2 people

- Key exchange complexity
  - We need 1 key per each pair of people
  - N (N – 1) / 2
  - O(N^2)

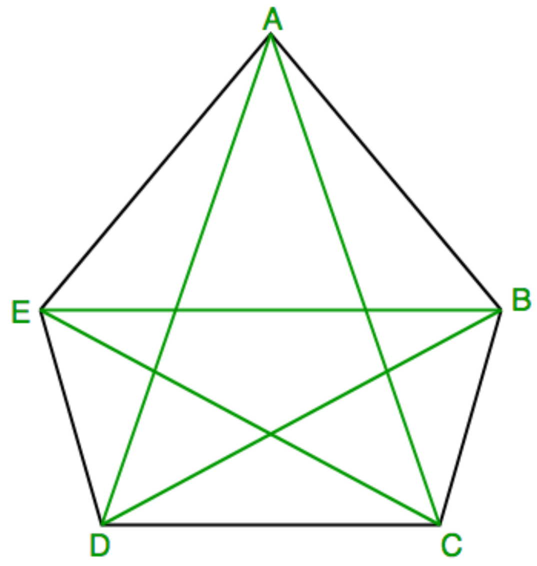


Asymmetric Key Cryptography

- Can we use a different key for the encryption and decryption?
  - K = k1, k2
  - Enc(k1, M) = C, Dec(k2, C) = M?
- Preferably, can we publish the encryption key to public?
  - While keeping the decryption key secret
- Then we need O(N) keys
  - Each member's public key, that's it.

Why O(N)?

- We need $O(N^2)$ keys for symmetric encryption



Why O(N)?

- Suppose each will generate public and private key

- Public_A, Private_A
- Public_B, Private_B
- Public_C, Private_C
- Public_D, Private_D
- Public_E, Private_E

Why O(N)?

- Each will have their own private key, and then,
  - publish all their public keys
- A: Private_A
- B: Private_B
- C: Private_C
- D: Private_D
- E: Private_E
- Public keys: Public_A, Public_B, Public_C, Public_D, Public_E

Can A Send an Encrypted message to B?

- Can A send an encrypted message to B?
  - Yes, encrypt data using Public_B; only B (holder of Private_B) can decrypt it
- Can C send an encrypted message to E?
  - Yes, encrypt data using Public_E; only E (holder of Private_E) can decrypt it

- Can X send an encrypted message to Y?
  - Yes, if X knows the public key of Y

- We only need to know the receiver's public key
  - Sender does not matter, that's why we have O(N)
  - Suppose we have N = 200, we need 19900 keys in symmetric, and we need 400 keys for asymmetric

Sample Question 14

- Alice generated an RSA 4096-bit public/private key pair. Alice published the public key on her Twitter so that anyone can access her public key.

- After that, she created a message M = “I owe Yeongjin $1,000,000”, and then, generate a hash message H = SHA256(M).
- Then, she encrypts H using the RSA private key, i.e.,
  - S = rsa_private_encrypt(key, H)
- Next, she gives S to Yeongjin, and then,

- Yeongjin lend her $1,000,000. How can he do that?

RSA: Digital Signature

- RSA can be used as a digital signature scheme
- What is that?

- In RSA, encryption is applying the public exponent to the message
  - $M^e \bmod N$
- In RSA, decryption is applying the private exponent to the message
  - $C^d \bmod N$

RSA: What will be the meaning of private encrypt?

- Suppose A encrypts the following message with her private key
  - "I would like to donate $100 to OSU if I get A from CS 370"
- M = 53151406333611257093956293411584759988053228938724427101859883089254119711739486837784167497839141764612450119856395995171455585519613744
- C = m^d mod N

RSA: What will be the meaning of private encrypt?

- M = 53151406333611257093956293411584759988053228938724427101859883089254119711739486837784167497839141764612450119856395995171455585519613744
- $C = m^d \bmod N$
- Anyone can have e. That means, anyone can decrypt C
  - $C^e == m^{de} == m^1 == m \pmod N$

RSA: What will be the meaning of private encrypt?

- M = 53151406333611257093956293411584759988053228938724427101859883089254119711739486837784167497839141764612450119856395995171455585519613744
- $C = m^d \bmod N$
- Anyone can have e. That means, anyone can decrypt C
  - $C^e == m^{de} == m^1 == m \pmod N$
  - m = 53151406333611257093956293411584759988053228938724427101859883089254119711739486837784167497839141764612450119856395995171455585519613744
  - "I would like to donate $100 to OSU if I get A from CS 370"

RSA: What will be the meaning of privat

We can verify that the encrypted content C contains
The ciphertext that only the holder of private key can generate.
We all have public key, and if that is decrypted to
"I would like to donate $100 to OSU if I get A from CS 370",
then, we know that the holder of private key 'endorsed it'

- M =
  5315140
  1859883
  8563959
- C = m^d mod N
- Anyone can have e. That means, anyone can decrypt C
  - $C^e == m^{de} == m^1 == m \pmod N$
  - m =
    531514063336112570939562934115847599880532289387244271018598830892541197117394868377841674978391417646124501198563959951714555855196137 44
  - "I would like to donate $100 to OSU if I get A from CS 370"

RSA: private_encrypt

- RSA Encryption using the private key is so-called as ‘Signing’
- Why?
  - The ciphertext will be decrypted as a plaintext using the public key
    - Anyone can decrypt!
  - But the ciphertext can only be generated with the private key
    - Only the private key owner can generate it!

- Implication
  - Holder of the private key generated a ciphertext message of message M
  - M is signed, endorsed by the holder’s private key
  - (Because it can only be generated with the private key)

RSA Summary

- Encryption with private key could be a ‘digital-signature’
  - Signed_message = Enc(private_key, message)
  - Message = Dec(public_key, signed_message)

- The correctly decrypted message using public key means that the private key holder have endorsed (‘encrypted’) the data
  - Anyone can verify this using the public key

Sample Question 15

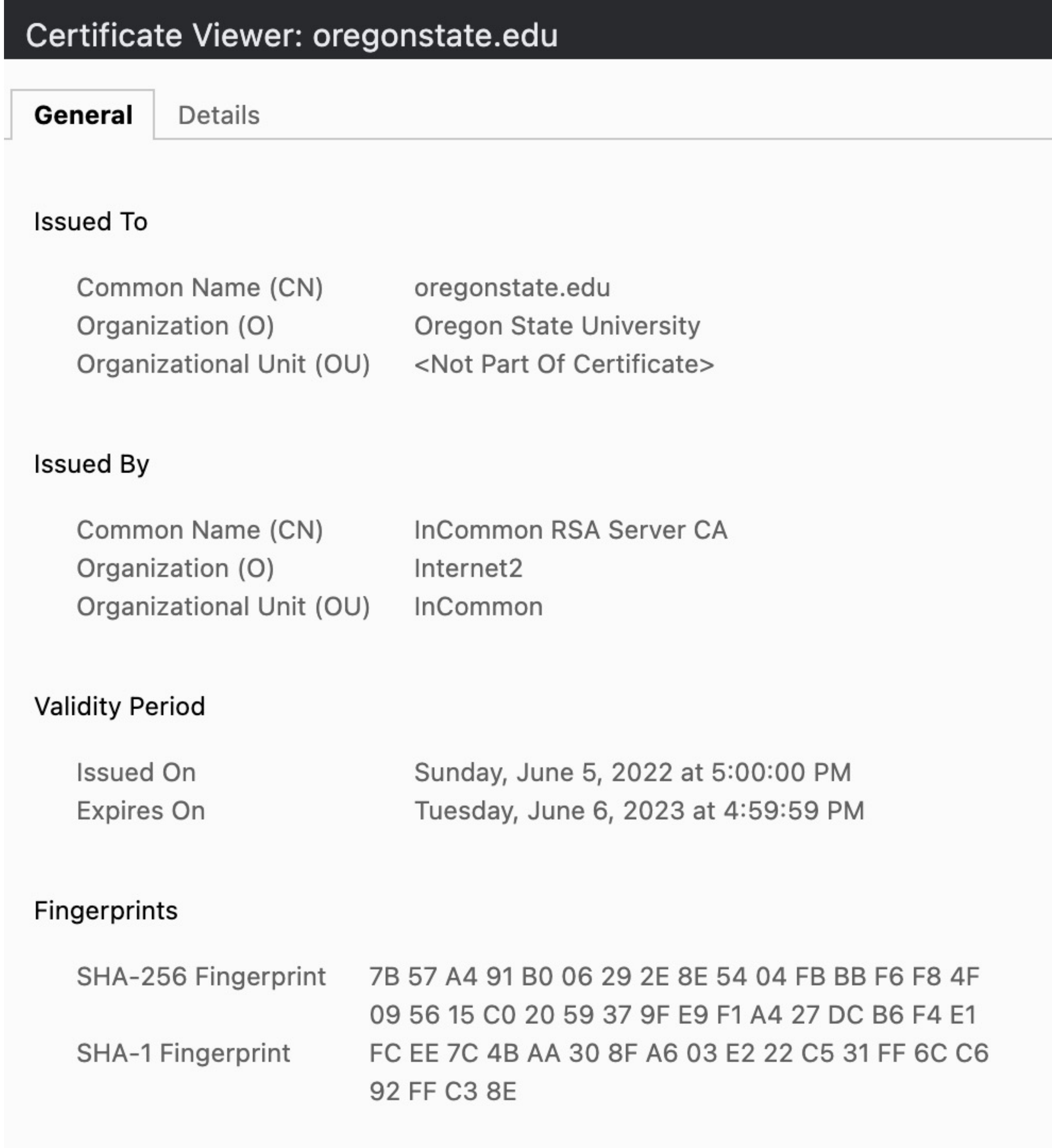
- Which of the following describes how the Public Key Infrastructure (PKI) works?

Public Key Infrastructure (PKI)

- We need an identification method for the key and the real entity
  - We need an online ID card for crypto keys…
- With RSA, we can use public key cryptosystem
  - We can announce the public key
- Let anyone can access and verify it
- How?
  - Where can we publish this and verify it?
  - PKI resolves the problem…

Digital Certificate

- A file that contains
  - Entity info (CN)
  - Issuer info (CN)
  - Public key
  - Signature



Creating a Digital Certificate

- 1. Requester prepares a certificate request
    - Entity information
    - Public key
    - Signature (proving that I have the public key)

Certificate
CN: oregonstate.edu
Will use for:
*.oregonstate.edu

Public Key: 0x112233445566778899aabbccddeeff….
(beaver's public key)

Signature: 0xaabbccddeeff00112233445566778899
(using beaver's private key)

Creating a Digital Certificate

- 1. Requester prepares a certificate request
  - Entity information
  - Public key
  - Signature (proving that I have the public key)

Get SHA256 sum of this part

Sign it with the private key

Certificate

CN: oregonstate.edu

Will use for:

*.oregonstate.edu

Public Key: 0x112233445566778899aabbccddeeff….

(beaver's public key)

Signature: 0xaabbccddeeff00112233445566778899

(using beaver's private key)

Creating a Digital Certificate

- 1. Requester prepares a certificate request
  - Entity information
  - Public key

- 2. Issuer verifies the requester information, and digitally sign the cert
  - 1) Verify the entity information
  - 2) Get a SHA-256 fingerprint of the certificate
  - 3) Sign the fingerprint (with issuer's private key)

```
RSA_encrypt(private_key, SHA-256(certificate))
```

Creating a Digital Certificate

- 2. Issuer verifies the requester information, and digitally sign the cert
  - 1) Verify the entity information
  - 2) Get a SHA-256 fingerprint of the certificate
  - 3) Sign the fingerprint (with issuer's private key)

```
RSA_encrypt(private_key, SHA-256(certificate))
```

Get SHA256 sum of this part

Sign it with the private key

Certificate
CN: oregonstate.edu
Will use for:
*.oregonstate.edu

Public Key: 0x112233445566778899aabbccddeeff....
(beaver's public key)

Signature: 0xffeeddccbbaa00112233445566778899
(with Issuer's private key)

Creating a Digital Certificate

- 1. Requester prepares a certificate request
  - Entity information
  - Public key

- 2. Issuer verifies the requester information, and digitally sign the cert
  - 1) Verify the entity information
  - 2) Get a SHA-256 fingerprint of the certificate
  - 3) Sign the fingerprint (with issuer's private key)

  ```
  RSA_encrypt(private_key, SHA-256(certificate))
  ```

- 3. Anyone with the public key can verify the result
  - Get issuer's public key from their certificate

Digital Certificate Creation Step 1

- The certificate requesting entity fills
  - Entity information
  - Public Key

CN = oregonstate.edu

- Entity can be anyone
  - For google, its *.google.com
  - Can be your website address
- *.unexploitable.systems
  - also has a certificate

Certificate
CN: oregonstate.edu
Will use for:
*.oregonstate.edu

Public Key: 0x112233445566778899aabbccddeeff….
(beaver’s public key)

Signature: 0xaabbccddeeff00112233445566778899
(with beaver’s private key)

Digital Certificate Creation Step 2

- The issuer receives the certificate request
- Verifies the entity for
  - Their identification
  - Owning the target domain name
  - Owning the public key
- Verify the signature
  - Decrypt the signature with public key
  - It must be the same as SHA256 sum
- Verification proves holding of the private key

CN = oregonstate.edu

Certificate
CN: oregonstate.edu
Will use for:
    *.oregonstate.edu

Public Key: 0x112233445566778899aabbccddeeff….
    (beaver’s public key)

Signature: 0xaabbccddeeff00112233445566778899
    (with beaver’s private key)

Digital Certificate Creation Step 2

- The issuer receives the certificate request
- Verifies the entity for
  - Their identification
  - Owning the target domain name
  - etc…
- Then, fill issuer information
  - Issuer information
  - Issuer public key

CN = oregonstate.edu

Certificate
CN: oregonstate.edu
Will use for:
*.oregonstate.edu

Public Key: 0x112233445566778899aabbccddeeff….
(beaver's public key)

Issuer: InCommon RSA
Public Key: 0x2233445566778899001 1aabbccddeeff

Digital Certificate Creation Step 2

- The issuer receives the certificate request
- Verifies the entity for
  - Their identification
  - Owning the target domain name
  - etc…
- Then, fill issuer information
  - Issuer information
  - Issuer public key
- And then, sign the certificate
  - Get SHA-256 fingerprint of the certificate
  - Attach it as a signature!

CN = oregonstate.edu

Certificate
CN: oregonstate.edu
Will use for:
*.oregonstate.edu

Public Key: 0x112233445566778899aabbccddeeff….
(beaver's public key)

Issuer: InCommon RSA
Public Key: 0x2233445566778899000011aabbccddeeff
Signature: 0xffeeddccbbaa00112233445566778899
(InCommon RSA's private key)

Issued Certificate

- Now InCommon RSA verified
  - oregonstate.edu is owned by
  - Oregon State University
  - With a specific Public Key

Subject Public Key Info

Subject Public Key Algorithm

Subject's Public Key

Field Value

Modulus (2048 bits):
C8 7D 2D A8 EB 12 59 6B 90 6D 4F 71 1E 4C FA C2
F7 A1 EC F6 E6 0E 39 52 FF 69 C0 36 CD A9 74 6E
60 72 C8 34 AF CC F7 6F 8E 66 D0 C5 0D E9 9C 66
F0 B2 D1 D8 75 A7 B9 82 E5 E8 C3 3F 13 35 1E 1E
71 F1 92 B4 40 07 EA 27 BE F9 9B AF E8 D2 E3 71

Certificate Viewer: oregonstate.edu

General | Details

Issued To

Common Name (CN)	oregonstate.edu
Organization (O)	Oregon State University
Organizational Unit (OU)	<Not Part Of Certificate>

Issued By

Common Name (CN)	InCommon RSA Server CA
Organization (O)	Internet2
Organizational Unit (OU)	InCommon

Validity Period

Issued On	Sunday, June 5, 2022 at 5:00:00 PM
Expires On	Tuesday, June 6, 2023 at 4:59:59 PM

Fingerprints

SHA-256 Fingerprint	7B 57 A4 91 B0 06 29 2E 8E 54 04 FB BB F6 F8 4F 09 56 15 C0 20 59 37 9F E9 F1 A4 27 DC B6 F4 E1
SHA-1 Fingerprint	FC EE 7C 4B AA 30 8F A6 03 E2 22 C5 31 FF 6C C6 92 FF C3 8E

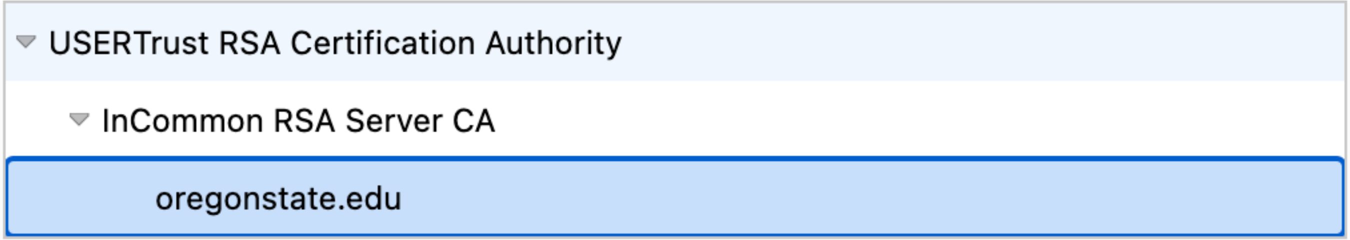
Trust Chain

- oregonstate.edu is owned by Oregon State University
  - Verified by InCommon RSA

- We can verify the certificate using InCommon RSA’s public key
  - Where is it? It is written in InCommon RSA’s certificate

- InCommon RSA, who will verify their identity?
  - oregonstate.edu was verified by InCommon RSA
  - Who will verify InCommon RSA?

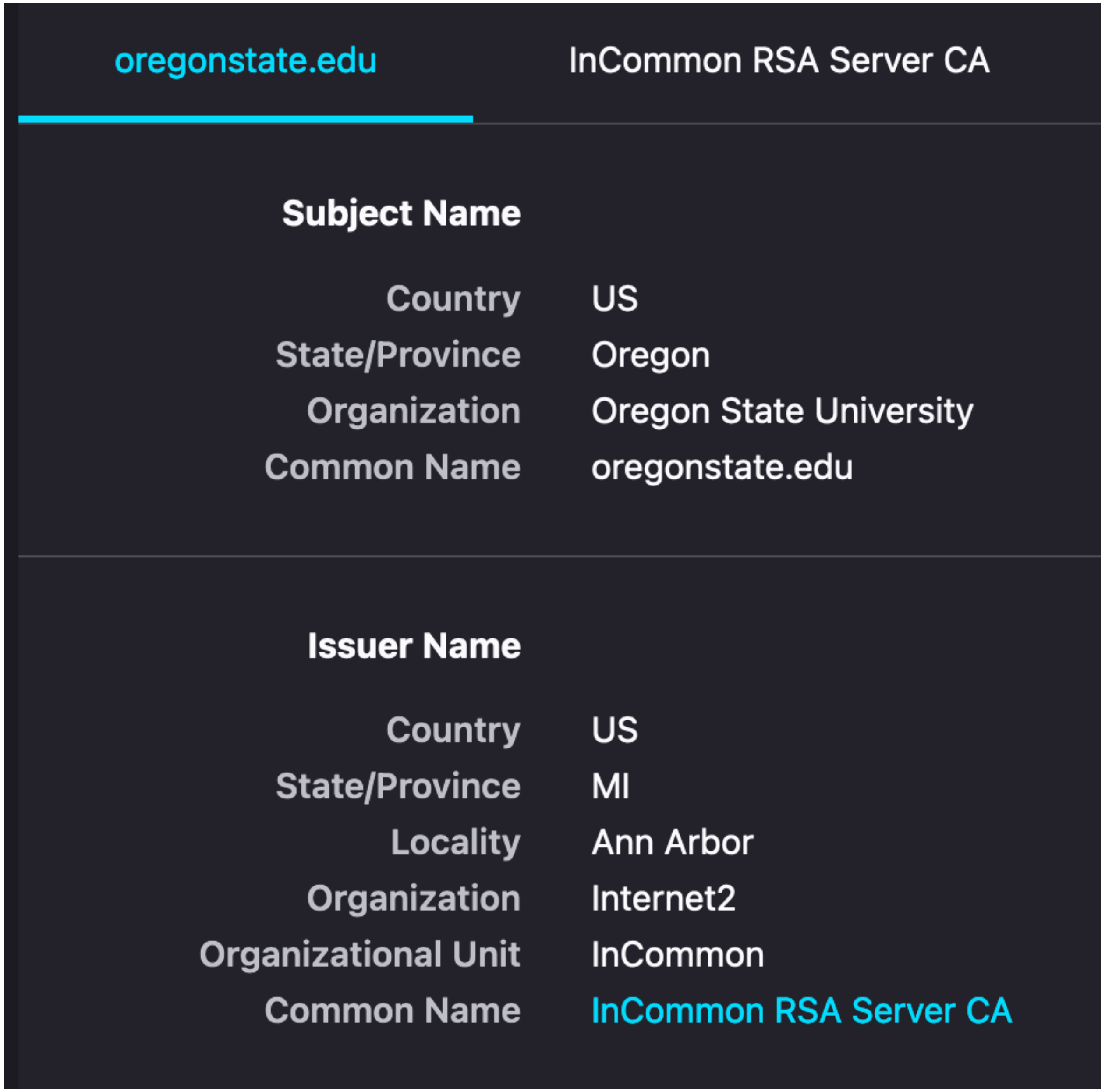
Trust Chain



- oregonstate.edu
  - Verified by InCommon RSA Server CA
- InCommon RSA Server CA
  - Verified by USERTrust RSA Certificate Authority
- USERTrust RSA CA
  - Verified by self

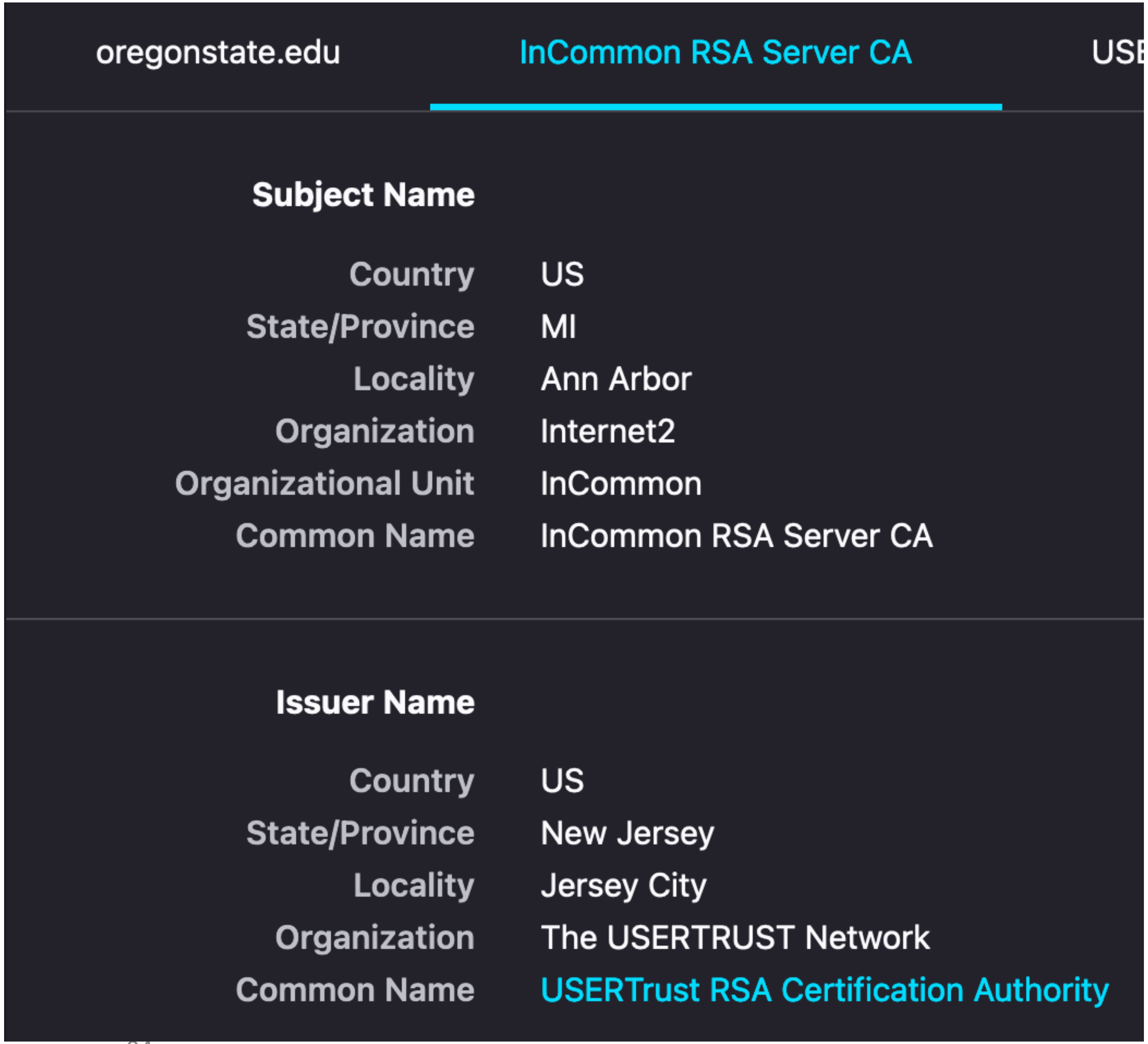
Trust Chain

- oregonstate.edu
  - Verified by InCommon RSA Server CA
- InCommon RSA Server CA
  - Verified by USERTrust RSA Certificate Authority
- USERTrust RSA CA
  - Verified by self



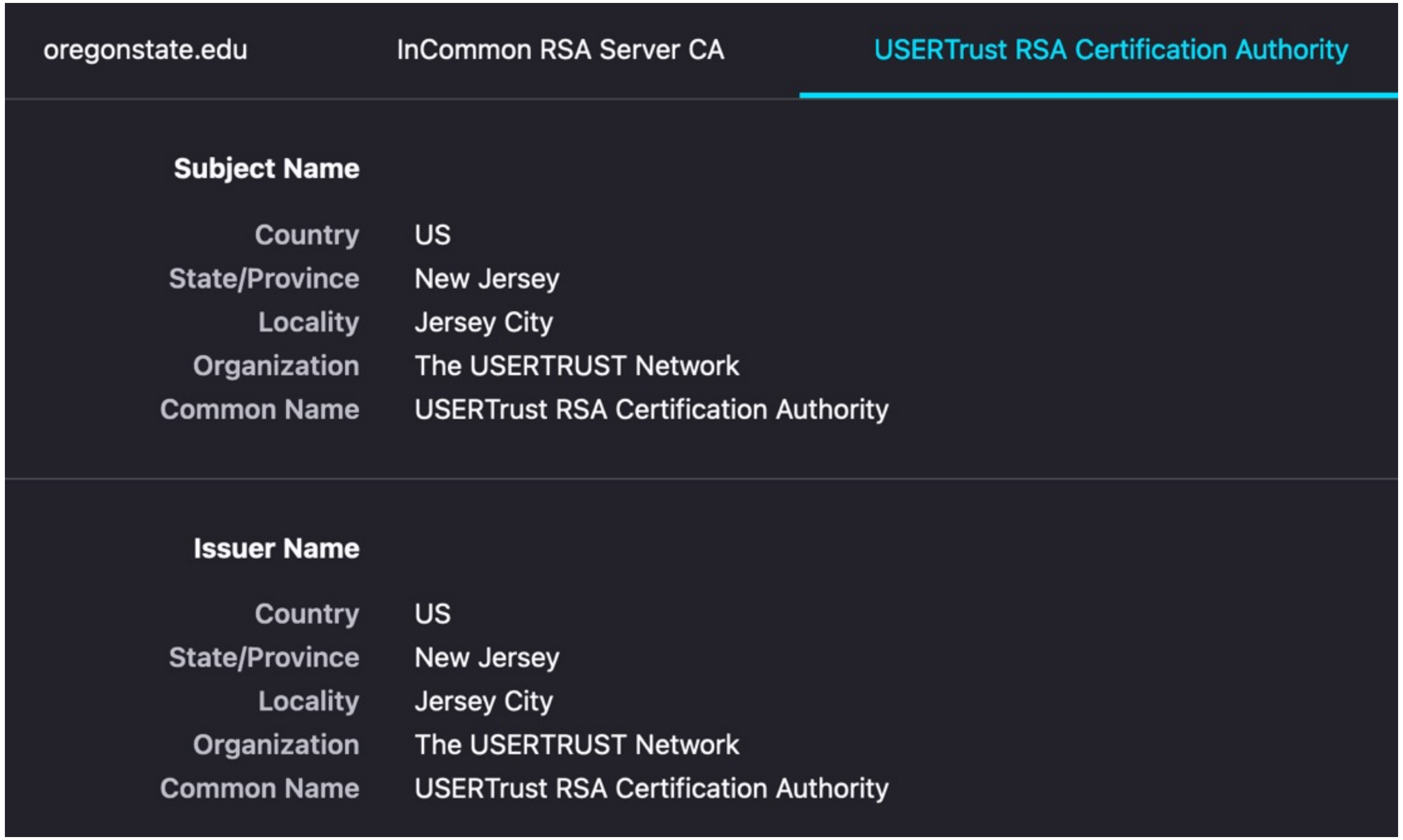
Trust Chain

- oregonstate.edu
  - Verified by InCommon RSA Server CA
- InCommon RSA Server CA
  - Verified by USERTrust RSA Certificate Authority
- USERTrust RSA CA
  - Verified by self



Trust Chain

- oregonstate.edu
  - Verified by InCommon RSA Server CA
- InCommon RSA Server CA
  - Verified by USERTrust RSA Certificate Authority
- USERTrust RSA CA
  - Verified by itself



How Do We Verify Our Identity?

- You as a
  - Student
  - Oregon resident
  - U.S. Citizen

- When issuing the student ID
  - We verify your Oregon ID…

How Do We Verify Our Identity?

- You as a
  - Student
  - Oregon resident
  - U.S. Citizen

- When issuing the Oregon Driver’s License
  - We require either one of your birth certificate, previous Driver’s License, or U.S. passport

How Do We Verify Our Identity?

- You as a
  - Student
  - Oregon resident
  - U.S. Citizen

- When issuing the U.S. passport
  - We require your birth certificate or previously issued passport..

How Do We Verify Our Identity?

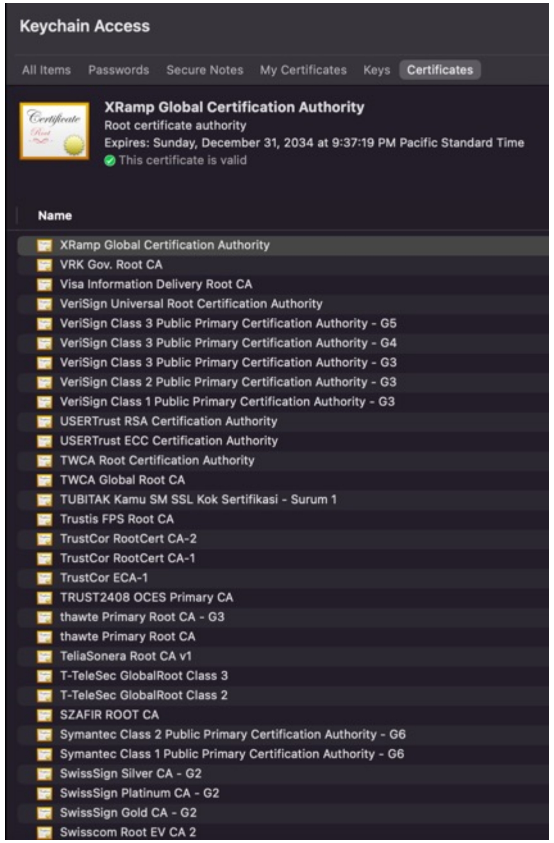
- You as a
  - Student
  - Oregon resident
  - U.S. Citizen

- When issuing the U.S. passport
  - We require your birth certificate or previously issued passport..

We need someone to verify the originality of the proving document…

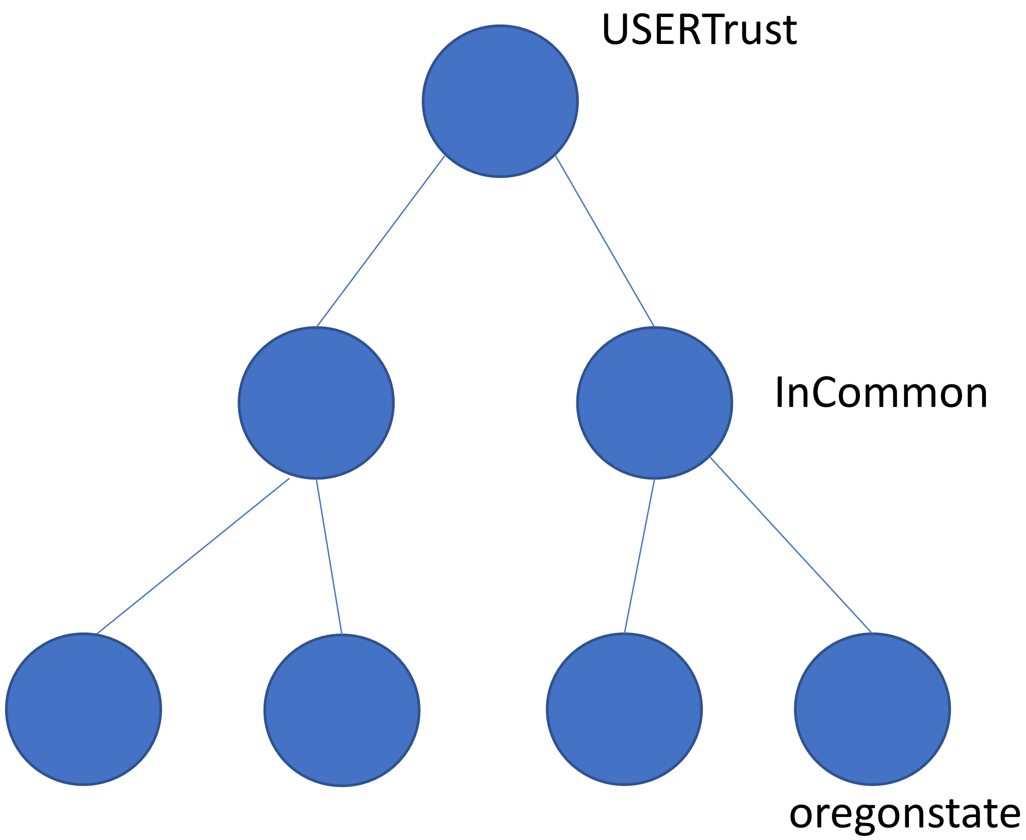
Root Certificate Authority (Root CA)

- We define small set of trustworthy certificate authorities
  - Private companies are authorized by some jurisdiction to run the CA company
    - Google Trust Service (GTS CA)
    - DigiCert
    - Verisign
    - Etc..
- We trust their self-signed certificate
  - Stored in almost every computer machines..

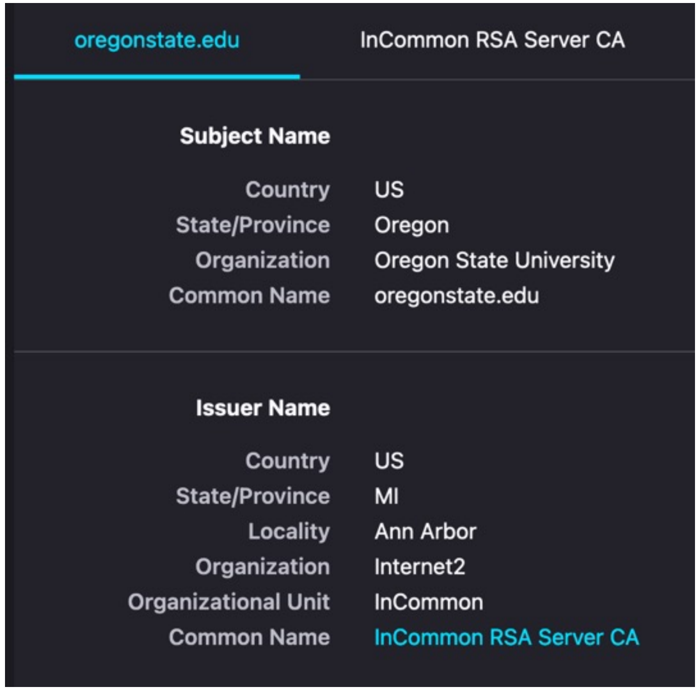


Public Key Infrastructure (PKI)

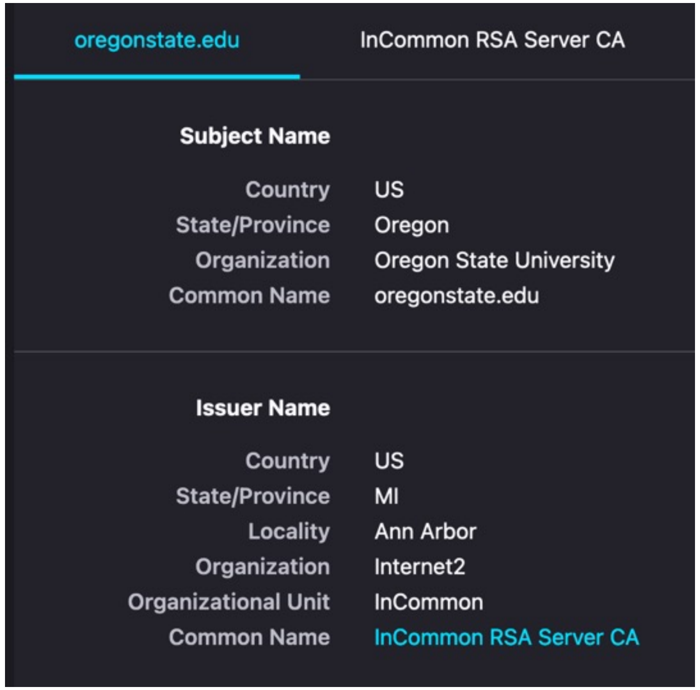
- An Infrastructure that provides public key with certificate chain
- Trust anchor: Root CA
  - We set a small set of entities use self-signed cert
- Verify the certificate chain!
  - We must verify the entire chain

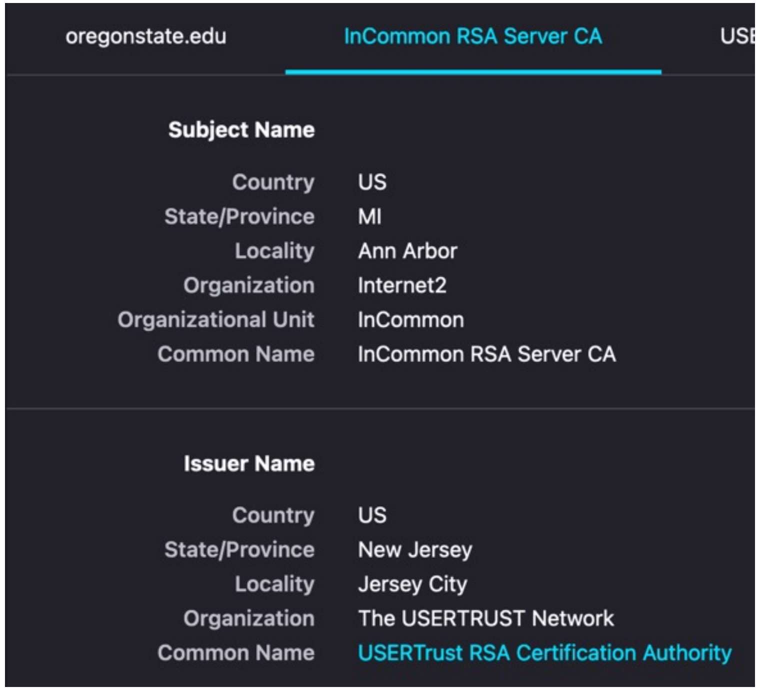


Trust Chain

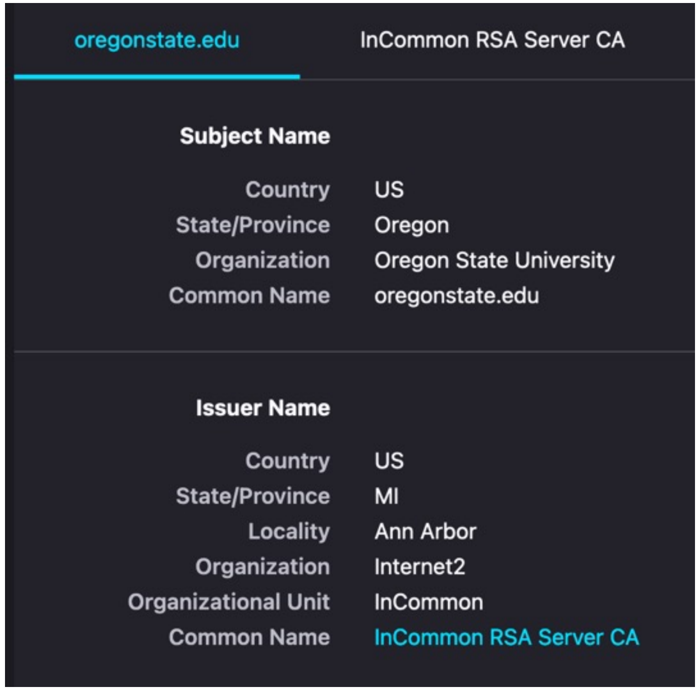

oregonstate.edu
InCommon RSA Server CA
Subject Name
Country US
State/Province Oregon
Organization Oregon State University
Common Name oregonstate.edu
Issuer Name
Country US
State/Province MI
Locality Ann Arbor
Organization Internet2
Organizational Unit InCommon
Common Name InCommon RSA Server CA

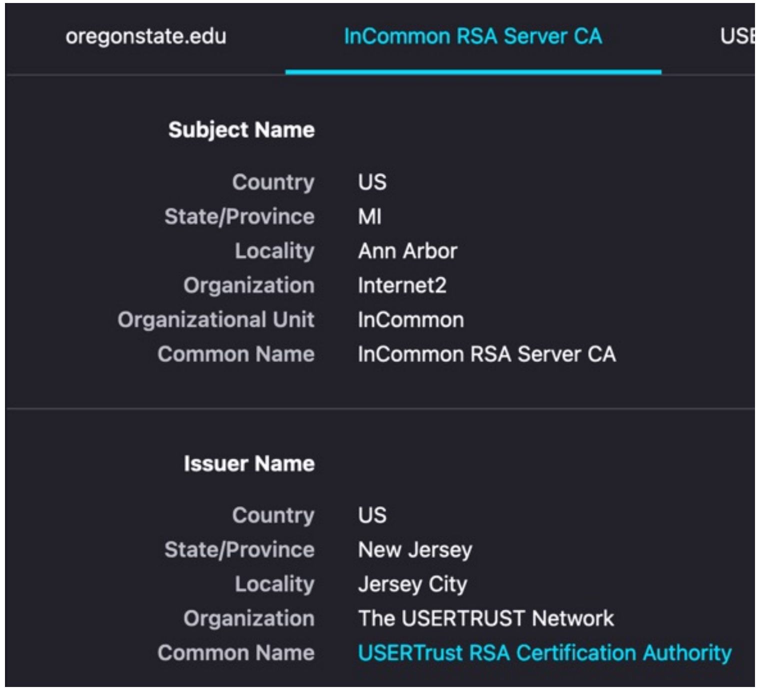
Trust Chain

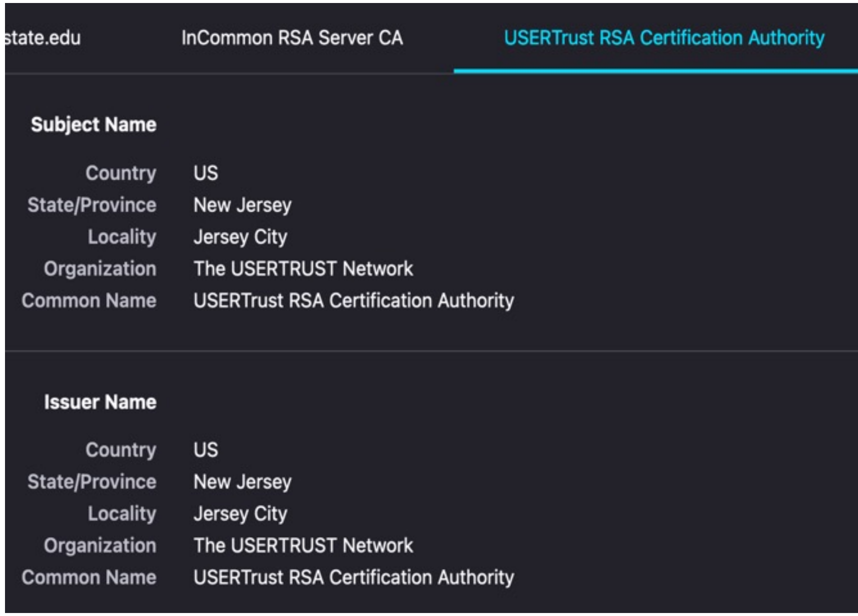

oregonstate.edu
InCommon RSA Server CA
Subject Name
Country US
State/Province Oregon
Organization Oregon State University
Common Name oregonstate.edu
Issuer Name
Country US
State/Province MI
Locality Ann Arbor
Organization Internet2
Organizational Unit InCommon
Common Name InCommon RSA Server CA


oregonstate.edu
InCommon RSA Server CA
Subject Name
Country US
State/Province MI
Locality Ann Arbor
Organization Internet2
Organizational Unit InCommon
Common Name InCommon RSA Server CA
Issuer Name
Country US
State/Province New Jersey
Locality Jersey City
Organization The USERTRUST Network
Common Name USERTrust RSA Certification Authority

Trust Chain


oregonstate.edu
InCommon RSA Server CA
Subject Name
Country US
State/Province Oregon
Organization Oregon State University
Common Name oregonstate.edu
Issuer Name
Country US
State/Province MI
Locality Ann Arbor
Organization Internet2
Organizational Unit InCommon
Common Name InCommon RSA Server CA

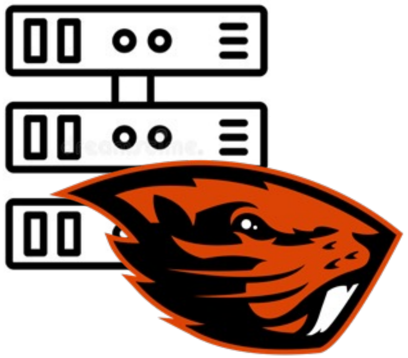

oregonstate.edu
InCommon RSA Server CA
Subject Name
Country US
State/Province MI
Locality Ann Arbor
Organization Internet2
Organizational Unit InCommon
Common Name InCommon RSA Server CA
Issuer Name
Country US
State/Province New Jersey
Locality Jersey City
Organization The USERTRUST Network
Common Name USERTrust RSA Certification Authority


InCommon RSA Server CA
USERTrust RSA Certification Authority
Subject Name
Country US
State/Province New Jersey
Locality Jersey City
Organization The USERTRUST Network
Common Name USERTrust RSA Certification Authority
Issuer Name
Country US
State/Province New Jersey
Locality Jersey City
Organization The USERTRUST Network
Common Name USERTrust RSA Certification Authority

How To Verify oregonstate.edu?

- Using the digital certificate!

Hey, are you oregonstate.edu?
Give me your certificate

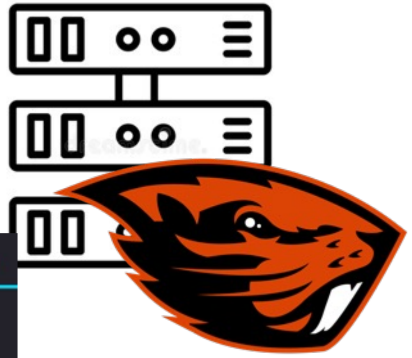


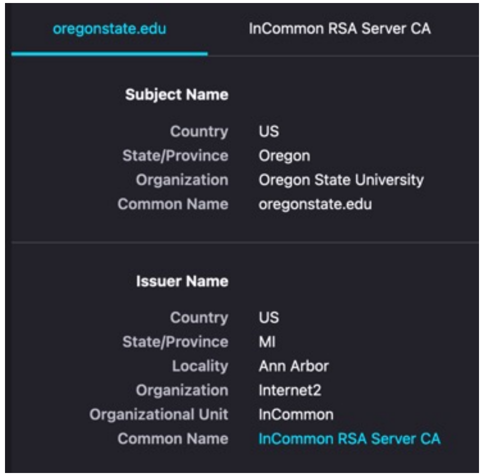
How To Verify oregonstate.edu?

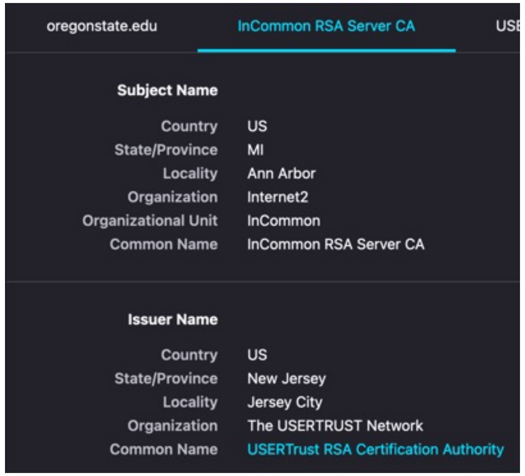
- Using the digital certificate!

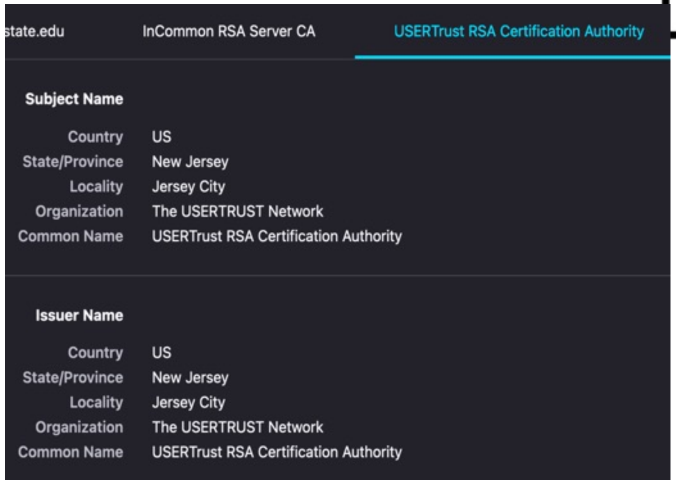
Hey, are you oregonstate.edu?
Give me your certificate

Yes, I am oregonstate.edu!
Here's my cert







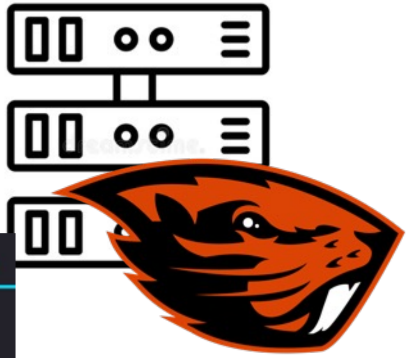


How To Verify oregonstate.edu?

- Using the digital certificate!

Hey, are you oregonstate.edu?
Give me your certificate

Yes, I am oregonstate.edu!
Here's my cert



oregonstate.edu | InCommon RSA Server CA

Subject Name

Country	US
State/Province	Oregon
Organization	Oregon State University
Common Name	oregonstate.edu

Issuer Name

Country	US
State/Province	MI
Locality	Ann Arbor
Organization	Internet2
Organizational Unit	InCommon
Common Name	InCommon RSA Server CA

oregonstate.edu | InCommon RSA Server CA | USE

Subject Name

Country	US
State/Province	MI
Locality	Ann Arbor
Organization	Internet2
Organizational Unit	InCommon
Common Name	InCommon RSA Server CA

Issuer Name

Country	US
State/Province	New Jersey
Locality	Jersey City
Organization	The USERTRUST Network
Common Name	USERTrust RSA Certification Authority

state.edu | InCommon RSA Server CA | USERTrust RSA Certification Authority

Subject Name

Country	US
State/Province	New Jersey
Locality	Jersey City
Organization	The USERTRUST Network
Common Name	USERTrust RSA Certification Authority

Issuer Name

Country	US
State/Province	New Jersey
Locality	Jersey City
Organization	The USERTRUST Network
Common Name	USERTrust RSA Certification Authority

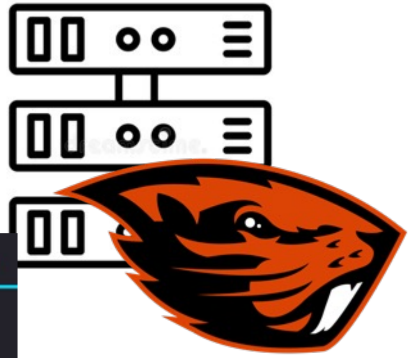
Oregonstate verified by InCommon RSA

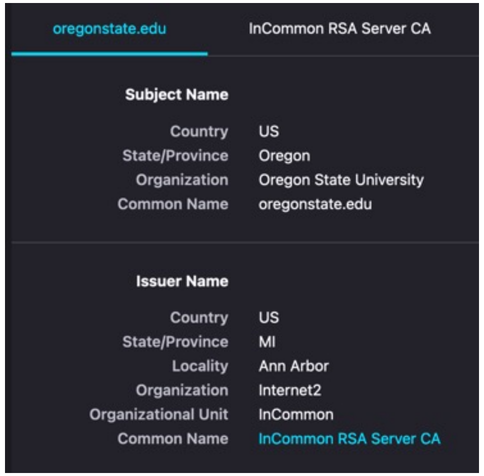
How To Verify oregonstate.edu?

- Using the digital certificate!

Hey, are you oregonstate.edu?
Give me your certificate

Yes, I am oregonstate.edu!
Here's my cert





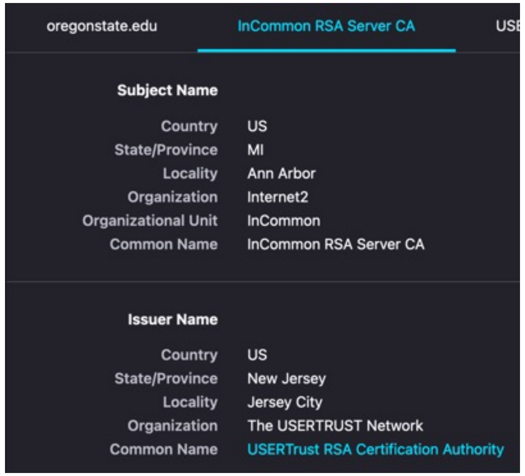
oregonstate.edu | InCommon RSA Server CA

Subject Name

Country	US
State/Province	Oregon
Organization	Oregon State University
Common Name	oregonstate.edu

Issuer Name

Country	US
State/Province	MI
Locality	Ann Arbor
Organization	Internet2
Organizational Unit	InCommon
Common Name	InCommon RSA Server CA



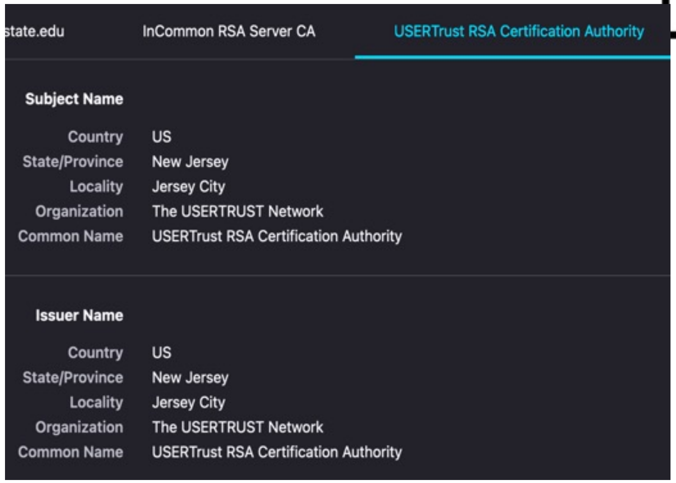
oregonstate.edu | InCommon RSA Server CA | USE

Subject Name

Country	US
State/Province	MI
Locality	Ann Arbor
Organization	Internet2
Organizational Unit	InCommon
Common Name	InCommon RSA Server CA

Issuer Name

Country	US
State/Province	New Jersey
Locality	Jersey City
Organization	The USERTRUST Network
Common Name	USERTrust RSA Certification Authority



state.edu | InCommon RSA Server CA | USERTrust RSA Certification Authority

Subject Name

Country	US
State/Province	New Jersey
Locality	Jersey City
Organization	The USERTRUST Network
Common Name	USERTrust RSA Certification Authority

Issuer Name

Country	US
State/Province	New Jersey
Locality	Jersey City
Organization	The USERTRUST Network
Common Name	USERTrust RSA Certification Authority

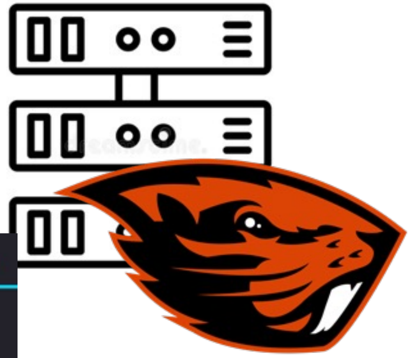
InCommon RSA verified by USERTrust RSA

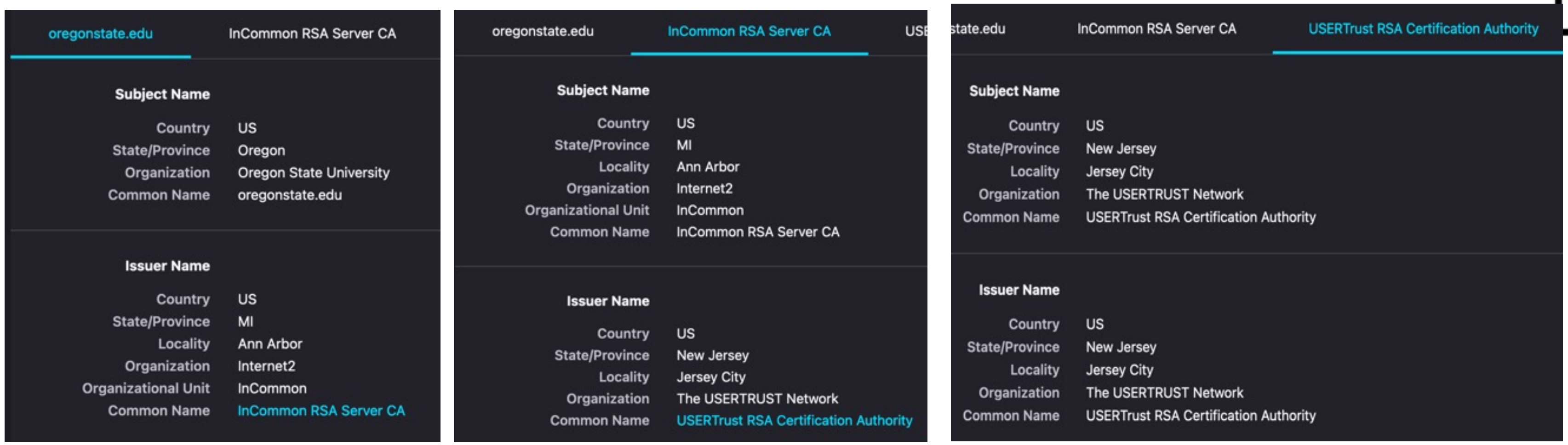
How To Verify oregonstate.edu?

- Using the digital certificate!

Hey, are you oregonstate.edu?
Give me your certificate

Yes, I am oregonstate.edu!
Here's my cert





USERTrust RSA is self-verified (ROOT CA)

Summary

- RSA encryption with private key can be used as digital signature
  - Only the private key holder can generate the message
  - Anyone with public key can verify this!
- We use digital certificates to share public key information
  - Entity name, address, other information with
  - Public key!
- Certificates are signed by other trustful entities
  - Verify the entity info and the public key, and then, sign the certificate!

Summary

- A certificate need to be verified by other entity
  - That other entity is also need to be verified by…
- Root CA is the list of trusted Certificate Authority
  - We accept their self-signed certificate
- We must verify the entire certificate trust chain
  - oregonstate.edu -> InCommon RSA -> USERTrust RSA …

Sample Question 16

- Compute the secret value shared via a Diffie-Hellman key exchange

Example

- $g = 5$, $p = 23$

- A chooses $a = 4$
  - $A = 5^4 \bmod 23 = 625 \bmod 23 = 4$
- B chooses $b = 3$
  - $B = 5^3 \bmod 23 = 125 \bmod 23 = 10$

- $B^4 = 10^4 \bmod 23 = 10000 \bmod 23 = 18$
- $A^3 = 4^3 \bmod 23 = 64 \bmod 23 = 18$
- $5^{(4*3)} = 5^{12} \bmod 23 = 18$