Computer Security Foundations
Week 8: Symmetric Encryption

Bernardo Portela

L.EIC - 24

Context

People usually think of *encryption* when they hear *cryptography*

... which is not unreasonable. But it is only one of many cryptographic techniques

Context

People usually think of *encryption* when they hear *cryptography*

... which is not unreasonable. But it is only one of many cryptographic techniques

Encryption guarantees *confidentiality*, but real-world applications often require other guarantees to be considered secure systems

- Authenticity, non-repudiation, unpredictability, anonymity, ...

Context

People usually think of *encryption* when they hear *cryptography*

... which is not unreasonable. But it is only one of many cryptographic techniques

Encryption guarantees *confidentiality*, but real-world applications often require other guarantees to be considered secure systems

- Authenticity, non-repudiation, unpredictability, anonymity, ...

Also, there are many kinds of encryption

- Symmetric, asymmetric, authenticated, homomorphic, ...

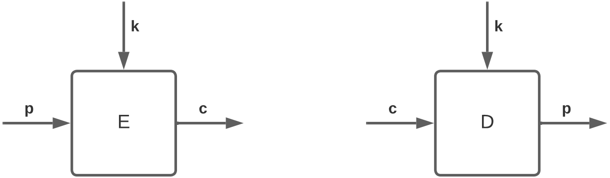
What is encryption?

Q1: What do you think encryption means?

What is encryption?

Q1: What do you think encryption means?

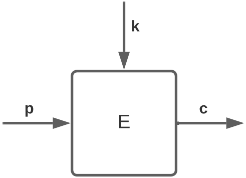
Encryption transforms *plaintexts* into *ciphertexts* using a *key*



What is encryption?

Q1: What do you think encryption means?

Encryption transforms *plaintexts* into *ciphertexts* using a *key*



We will use the following notation to talk about algorithms

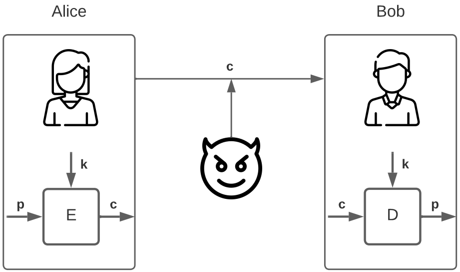
- $c \leftarrow_{\$} \mathrm{E}(k, p)$ - Encryption is (usually) randomized
  - **Q2: Why?**
- $p \leftarrow \mathrm{D}(k, c)$ - Decryption is deterministic

We begin with **symmetric** encryption: same key on both ends

What we talk about when we talk about Security - Part 1

Meet Alice and Bob

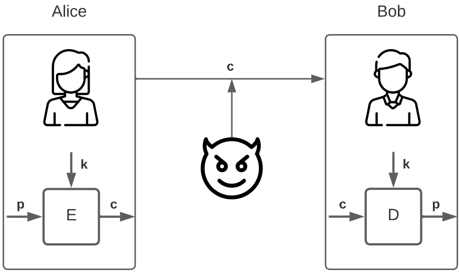
- Alice wants to send a message to Bob
- The message must be secure against an attacker (the devil)



What we talk about when we talk about Security - Part 1

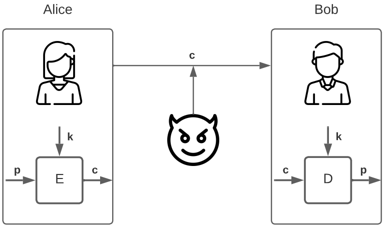
Meet Alice and Bob

- Alice wants to send a message to Bob
- The message must be secure against an attacker (the devil)



Q: What do we mean for the encryption to be "secure"?

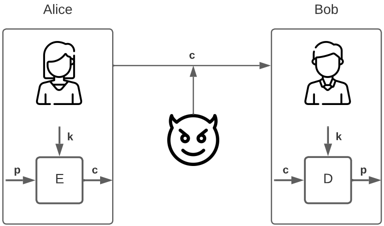
What we talk about when we talk about Security - Part 2



Suppose my message is “banana”

- Attempt #1: I don’t want “*banana*” to be revealed

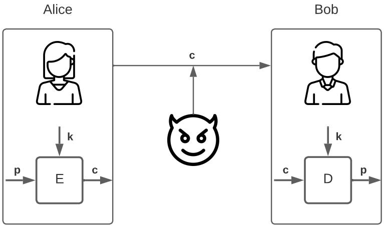
What we talk about when we talk about Security - Part 2



Suppose my message is "banana"

- Attempt #1: I don't want "*banana*" to be revealed
  - But if a scheme reveals "bananb" I am also not happy.
- Attempt #2: I don't want any characters to be retrieved

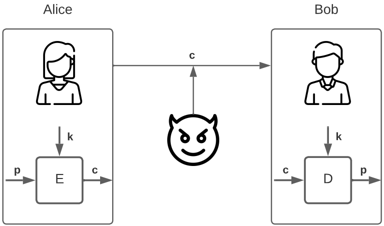
What we talk about when we talk about Security - Part 2



Suppose my message is “banana”

- Attempt #1: I don’t want “*banana*” to be revealed
    - But if a scheme reveals “bananb” I am also not happy.
- Attempt #2: I don’t want any characters to be retrieved
    - But if a scheme reveals “cbobob” I am also not happy.

What we talk about when we talk about Security - Part 2



Suppose my message is "banana"

- Attempt #1: I don't want "*banana*" to be revealed
  - But if a scheme reveals "bananb" I am also not happy.
- Attempt #2: I don't want any characters to be retrieved
  - But if a scheme reveals "cbobob" I am also not happy.
- A more rigorous approach to define security must be taken

Caesar Cipher

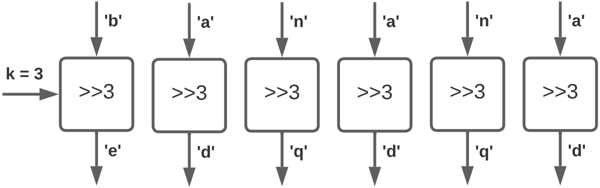
- One of the earliest known ciphers, used in Roman times

Caesar Cipher

- One of the earliest known ciphers, used in Roman times

Algorithm

- Take the plaintext, e.g., "banana"
- Shift the plaintext 3 letters upward
- Key is fixed, but we can choose other shift sizes as keys

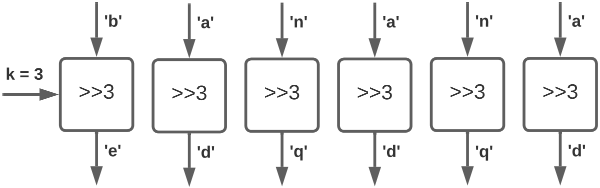


Caesar Cipher

- One of the earliest known ciphers, used in Roman times

Algorithm

- Take the plaintext, e.g., "banana"
- Shift the plaintext 3 letters upward
- Key is fixed, but we can choose other shift sizes as keys



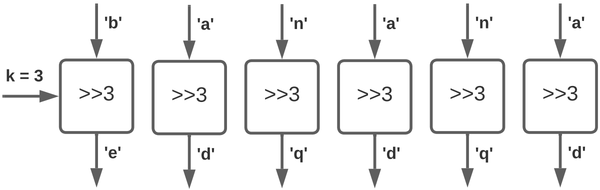
- **Q1: How can we decrypt?**

Caesar Cipher

- One of the earliest known ciphers, used in Roman times

Algorithm

- Take the plaintext, e.g., "banana"
- Shift the plaintext 3 letters upward
- Key is fixed, but we can choose other shift sizes as keys



- **Q1: How can we decrypt?**
- **Q2: Why is this cipher insecure?** *Very small key space!*

Substitution Ciphers

- We can choose different shifts for different letters
  - E.g. $'a' \rightarrow' f'$; $'b' \rightarrow' a'$; $'c' \rightarrow' z'$; ...
- Shift is a particular class of permutations over the alphabet
  - **Q: How many permutations are there over the alphabet?**
  - **A.k.a. how large is the key space?**

Substitution Ciphers

- We can choose different shifts for different letters
  - E.g. $'a' \rightarrow' f'$; $'b' \rightarrow' a'$; $'c' \rightarrow' z'$; ...
- Shift is a particular class of permutations over the alphabet
  - **Q: How many permutations are there over the alphabet?**
  - **A.k.a. how large is the key space?**

- $26! \approx 2^{88}$: It's a pretty big number
- Not possible to brute force without massive investment
- Surely it will be safe... Right?

Frequency letter attacks

Q1: Which of these is most common in Portuguese?

1. 'l'
2. 'a'
3. 's'
4. 'z'

Frequency letter attacks

Q1: Which of these is most common in Portuguese?

1. 'l' - 2.78%
2. 'a' - 14.63%
3. 's' - 6.81%
4. 'z' - 0.47%

Frequency letter attacks

Q1: Which of these is most common in Portuguese?

1. 'l' - 2.78%
2. 'a' - 14.63%
3. 's' - 6.81%
4. 'z' - 0.47%

Q2: How can we use this to attack this encryption scheme?

Frequency letter attacks

Q1: Which of these is most common in Portuguese?

1. 'l' - 2.78%
2. 'a' - 14.63%
3. 's' - 6.81%
4. 'z' - 0.47%

Q2: How can we use this to attack this encryption scheme?

- Gather many ciphertexts and count the frequency of letters
- Match that frequency with the frequency of plaintext alphabet
  - With good odds, the most common letter in the ciphertexts will match the most common letter in the plaintext alphabet
- Can be done using a statistical hypothesis (χ^2) test

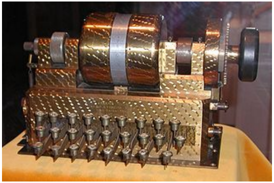
Rotor machines

Hebern machine (left)

- Key is the disk, encoding a substitution table
- On key press, the output is encrypted and the disk rotates

The Enigma (right)

- Key is the initial setting of rotors by multiple rotors (3-5)
- Rotors rotate with different frequencies



The one-time pad - Part 1

- Patent issued in 1917 by Gilbert Vernam

Algorithm

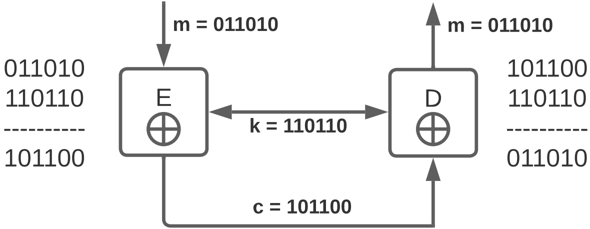
- Lets work with 0s and 1s
- Choose a random bit string $k \leftarrow \{0,1\}^m$
- To encrypt, compute the bit-wise XOR of m and k: $m \oplus k$
- To decrypt, compute the bit-wise XOR of c and k: $c \oplus k$

The one-time pad - Part 1

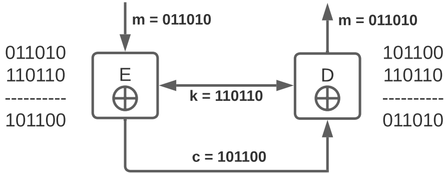
- Patent issued in 1917 by Gilbert Vernam

Algorithm

- Lets work with 0s and 1s
- Choose a random bit string $k \leftarrow \{0, 1\}^m$
- To encrypt, compute the bit-wise XOR of m and k: $m \oplus k$
- To decrypt, compute the bit-wise XOR of c and k: $c \oplus k$

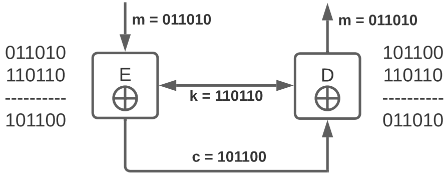


The one-time pad - Part 2



Q1: Is this secure?

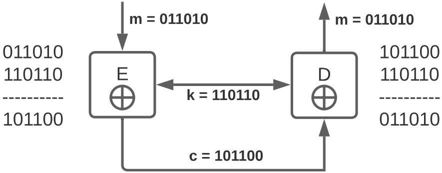
The one-time pad - Part 2



Q1: Is this secure?

- It is *perfectly secure* (as long as keys are used only once)

The one-time pad - Part 2

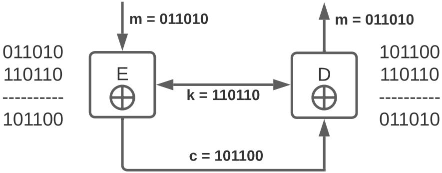


Q1: Is this secure?

- It is *perfectly secure* (as long as keys are used only once)

Q2: Why is this not used to encrypt everything?

The one-time pad - Part 2



Q1: Is this secure?

- It is *perfectly secure* (as long as keys are used only once)

Q2: Why is this not used to encrypt everything?

- Keys must have *the same size* as the messages
  - To send a 2 Gb file, I must use a 2 Gb key!
- How can we pre-share and store such huge keys?
- But it is used everywhere in cryptography as a building block

Kerckhoffs's Principle

- Long ago, it was common for encryption systems to be secret
- The idea is: the less people know, the harder it is to attack
- Also known as *Security through obscurity*
- We now know that **this is a bad idea**

Kerckhoffs's Principle

- Long ago, it was common for encryption systems to be secret
- The idea is: the less people know, the harder it is to attack
- Also known as *Security through obscurity*
- We now know that **this is a bad idea**

Kerckhoffs's Principle

- All details of a cryptosystem's operation are public
- The only secret is the key
- Why? Public knowledge promotes scrutiny
  - Designs of systems we will study are all public knowledge
  - Cryptographic schemes can be analyzed by everyone
  - Real-world security built on top of open standards
  - Methodology that revolutionized the way we approach security

Never Use Your Own Crypto

A point we will hammer home.

- It is very easy to make mistakes
- It is very hard to find mistakes

Never Use Your Own Crypto

A point we will hammer home.

- It is very easy to make mistakes
- It is very hard to find mistakes

A plethora of bad apples

- Cryptography can be poorly designed
  - 2G mobile phone standard using A5/2 encryption
  - Broken after 10 years

Never Use Your Own Crypto

A point we will hammer home.

- It is very easy to make mistakes
- It is very hard to find mistakes

A plethora of bad apples

- Cryptography can be poorly designed
  - 2G mobile phone standard using A5/2 encryption
  - Broken after 10 years
- Cryptography may not match the application
  - TLS servers try to decrypt incoming traffic
  - What if errors are observed on the network?
  - Security definition does not capture this
  - Padding-oracle attacks are an example of this mismatch

Never Use Your Own Crypto

A point we will hammer home.

- It is very easy to make mistakes
- It is very hard to find mistakes

A plethora of bad apples

- Cryptography can be poorly designed
  - 2G mobile phone standard using A5/2 encryption
  - Broken after 10 years
- Cryptography may not match the application
  - TLS servers try to decrypt incoming traffic
  - What if errors are observed on the network?
  - Security definition does not capture this
  - Padding-oracle attacks are an example of this mismatch
- Cryptography can be poorly implemented
  - Timing attacks used to break theoretically secure crypto
  - Implementation errors can leak secret keys (e.g. heartbleed)

Key Takeaways

- Encryption is a combination of two main algorithms:
  - Encryption takes plaintexts and produces ciphertexts
  - Decryption takes ciphertexts and produces plaintexts

Key Takeaways

- Encryption is a combination of two main algorithms:
  - Encryption takes plaintexts and produces ciphertexts
  - Decryption takes ciphertexts and produces plaintexts
- Classical ciphers fail because of multiple reasons:
  - Small key space - key is easy to guess
  - Ciphertexts reveal patterns in the original message
  - One-time pad, on the other hand, is *perfectly secure*
  - But it is very *inefficient* to use in practice

Key Takeaways

- Encryption is a combination of two main algorithms:
  - Encryption takes plaintexts and produces ciphertexts
  - Decryption takes ciphertexts and produces plaintexts
- Classical ciphers fail because of multiple reasons:
  - Small key space - key is easy to guess
  - Ciphertexts reveal patterns in the original message
  - One-time pad, on the other hand, is *perfectly secure*
  - But it is very *inefficient* to use in practice
- Kerkhoff's Principle
  - Algorithms should always be open-source
  - Security comes from the strength of the key
  - Many systems (today) rely on closed-source crypto

Key Takeaways

- Encryption is a combination of two main algorithms:
  - Encryption takes plaintexts and produces ciphertexts
  - Decryption takes ciphertexts and produces plaintexts
- Classical ciphers fail because of multiple reasons:
  - Small key space - key is easy to guess
  - Ciphertexts reveal patterns in the original message
  - One-time pad, on the other hand, is *perfectly secure*
  - But it is very *inefficient* to use in practice
- Kerkhoff's Principle
  - Algorithms should always be open-source
  - Security comes from the strength of the key
  - Many systems (today) rely on closed-source crypto
- Do not write your own crypto!
  - It's easy to f-up
  - Testing correctness and security is very nuanced

Defining Block Ciphers

A block cipher is defined by two deterministic algorithms

Encrypt: $E(k, p)$

- Takes a key $k \in \{0, 1\}^\lambda$
- Takes a plaintext block $p \in \{0, 1\}^B$
- Outputs a ciphertext block $c \in \{0, 1\}^B$

Defining Block Ciphers

A block cipher is defined by two deterministic algorithms

Encrypt: $E(k, p)$

- Takes a key $k \in \{0, 1\}^\lambda$
- Takes a plaintext block $p \in \{0, 1\}^B$
- Outputs a ciphertext block $c \in \{0, 1\}^B$

Decrypt: $D(k, c)$

- Takes a key $k \in \{0, 1\}^\lambda$
- Takes a ciphertext block $c \in \{0, 1\}^B$
- Outputs a plaintext block $p \in \{0, 1\}^B$

A block cipher is **invertible**: k defines a **permutation**

Advanced Encryption Standard (AES)

AES was standardized in 2000

- DES was still standard (56-bit keys)
- 3DES was a common solution for short keys (112-bit security)
- 3DES: use DES 3 times with 3 independent keys
- 3DES chains $E(k_1, D(k_2, E(k_3, p)))$

Advanced Encryption Standard (AES)

AES was standardized in 2000

- DES was still standard (56-bit keys)
- 3DES was a common solution for short keys (112-bit security)
- 3DES: use DES 3 times with 3 independent keys
- 3DES chains $E(k_1, D(k_2, E(k_3, p)))$

AES is now the most used block cipher, by far

- Available in mainstream CPUs as HW implementation

Selected as a result of a competition

- 1997-2000 public competition run by NIST
- This process has since become the norm
- Criteria: performance and resistance to cryptanalysis

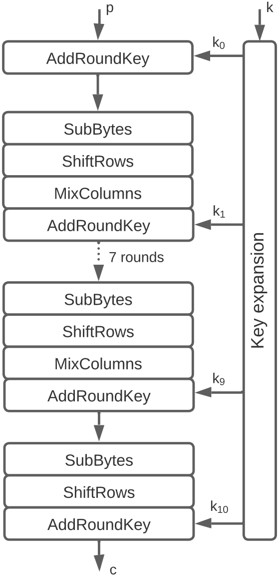
Internals of AES

- Block size 128-bits and varying key size (128, 192, 256)-bits
- Keeps a 128-bit internal state: 4 x 4 array of 16-bits
- State is transformed using a substitution-permutation network

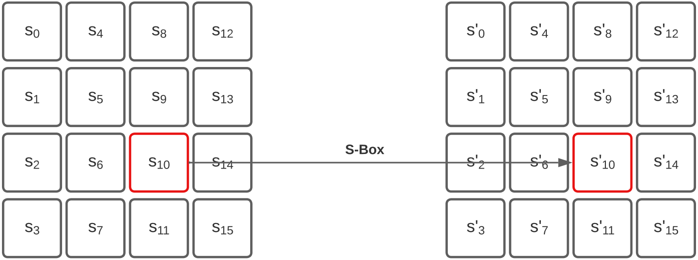
s_0	s_4	s_8	s_{12}
s_1	s_5	s_9	s_{13}
s_2	s_6	s_{10}	s_{14}
s_3	s_7	s_{11}	s_{15}

Substitutions/permutations have an algebraic description

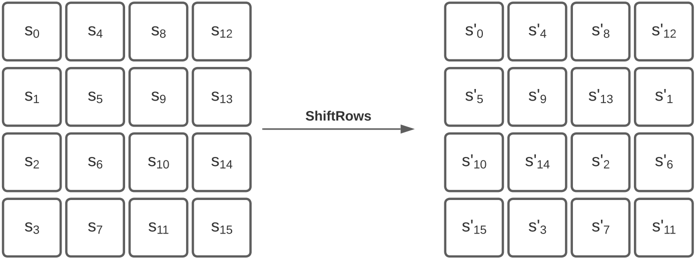
Internals of AES - High Level View



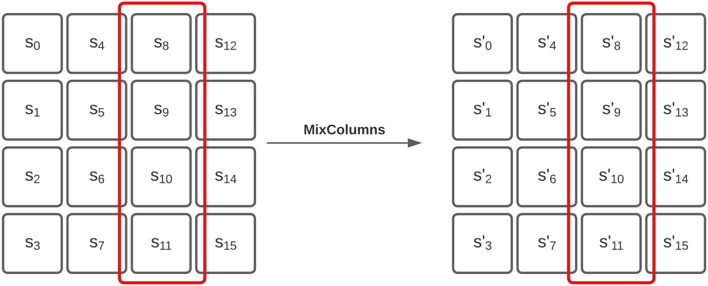
Internals of AES - SubBytes



Internals of AES - ShiftRows



Internals of AES - MixColumns



Using Block Ciphers Directly

Recall our secure block cipher building block:

Encrypt: $E(k, p)$

- Takes a key $k \in \{0, 1\}^\lambda$
- Takes a plaintext block $p \in \{0, 1\}^B$
- Outputs a ciphertext block $c \in \{0, 1\}^B$

Decrypt: $D(k, c)$

- Takes a key $k \in \{0, 1\}^\lambda$
- Takes a ciphertext block $c \in \{0, 1\}^B$
- Outputs a plaintext block $p \in \{0, 1\}^B$

Q: What issues arise when using this to encrypt messages?

Modes of Operation

Modern cryptography clearly defines these concepts

- Block-ciphers are a **primitive**
- On their own, they're not very useful
- There are **insecure** ways to encrypt with a block cipher
- Encryption schemes have their own security definitions
- Encryption schemes built <u>from</u> block ciphers
- We prove encryption secure <u>assuming</u> a block cipher is secure

Insecure Encryption from Secure Block Ciphers

Electronic-Code-Book Mode (ECB)

- Break message into plaintext blocks $p_0, \ldots, p_n$
- Last block may need padding
  - That's a can of worms in and of itself
  - More on that later
- Independently encrypt each block $c_i \leftarrow E(k, p_i)$

Insecure Encryption from Secure Block Ciphers

Electronic-Code-Book Mode (ECB)

- Break message into plaintext blocks $p_0, \ldots, p_n$
- Last block may need padding
  - That's a can of worms in and of itself
  - More on that later
- Independently encrypt each block $c_i \leftarrow E(k, p_i)$
- **Q: Why is this insecure?**

Insecure Encryption from Secure Block Ciphers

Electronic-Code-Book Mode (ECB)

- Break message into plaintext blocks $p_0, \ldots, p_n$
- Last block may need padding
  - That's a can of worms in and of itself
  - More on that later
- Independently encrypt each block $c_i \leftarrow E(k, p_i)$
- **Q: Why is this insecure?**

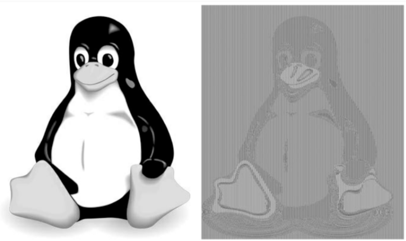
ECB is broken because you can see the penguin!

Insecure Encryption from Secure Block Ciphers

Electronic-Code-Book Mode (ECB)

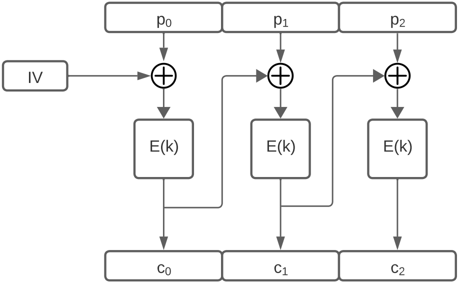
- Break message into plaintext blocks $p_0, \ldots, p_n$
- Last block may need padding
  - That's a can of worms in and of itself
  - More on that later
- Independently encrypt each block $c_i \leftarrow E(k, p_i)$
- **Q: Why is this insecure?**

ECB is broken because you can see the penguin!



Cipher Block Chaining

Engineers designed a secure encryption scheme before security proofs were well understood



- Main difference to ECB is the Initialization Vector (IV)
- Blocks depend on each other

CBC: Padding

There are several padding methods

- Some schemes require message size as multiple of block size
- Padding schemes re-encode message so that is true
- To avoid ambiguity: **padding is always added**

CBC: Padding

There are several padding methods

- Some schemes require message size as multiple of block size
- Padding schemes re-encode message so that is true
- To avoid ambiguity: **padding is always added**

The most common padding scheme is specified in PKCS#7:

- Let $k > |M|$ be the next multiple of B (in bytes)
- Add $k - |M|$ bytes with value $k - |M|$
- The last byte always reveals how much padding was added
  - $0\mathrm{x}01$ means 1 byte of padding with that value
  - $0\mathrm{x}03$ means 3 bytes of padding with that value

CBC: Padding

There are several padding methods

- Some schemes require message size as multiple of block size
- Padding schemes re-encode message so that is true
- To avoid ambiguity: **padding is always added**

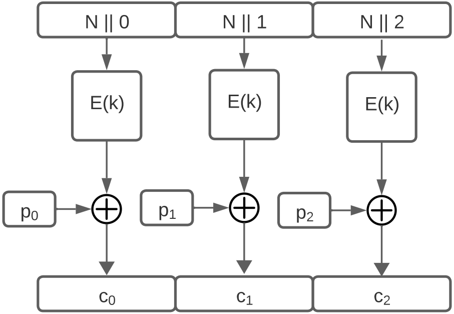
The most common padding scheme is specified in PKCS#7:

- Let $k > |M|$ be the next multiple of B (in bytes)
- Add $k - |M|$ bytes with value $k - |M|$
- The last byte always reveals how much padding was added
  - $0\mathrm{x}01$ means 1 byte of padding with that value
  - $0\mathrm{x}03$ means 3 bytes of padding with that value

Q: What is the minimum and maximum of added padding?

Counter Block Mode

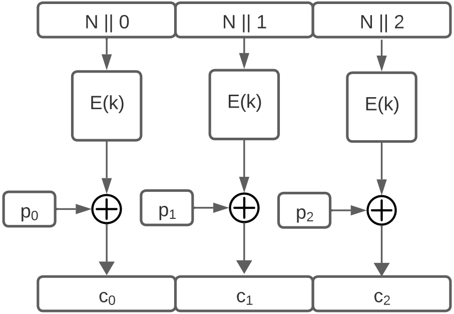
Often Counter Block Mode (CTR) is used in Nonce-based form



- N must be unique, but not necessarily random
- Encryption becomes stateful

Counter Block Mode

Often Counter Block Mode (CTR) is used in Nonce-based form



- N must be unique, but not necessarily random
- Encryption becomes stateful
- **Q: How can this be faster than CBC?**

Advantages of CTR

Counter mode is very efficient

- Key stream can be pre-processed
  - Block cipher not applied to the message!
- Any part of the data can be accessed efficiently
- This includes read/write access
- Decryption/encryption can be parallelized

As such, many modern protocols rely on CTR mode

Key Takeaways

- AES selected via public competition
  - ... as all modern ciphers are

Key Takeaways

- AES selected via public competition
  - ... as all modern ciphers are
- SubBytes; ShiftRows; MixColumns; AddRoundKey

Key Takeaways

- AES selected via public competition
  - ... as all modern ciphers are
- SubBytes; ShiftRows; MixColumns; AddRoundKey
- Currently the *de facto* standard for block ciphers

Key Takeaways

- AES selected via public competition
  - ... as all modern ciphers are
- SubBytes; ShiftRows; MixColumns; AddRoundKey
- Currently the *de facto* standard for block ciphers
- Block ciphers by themselves are **insecure**

Key Takeaways

- AES selected via public competition
  - ... as all modern ciphers are
- SubBytes; ShiftRows; MixColumns; AddRoundKey
- Currently the *de facto* standard for block ciphers
- Block ciphers by themselves are **insecure**
- So we rely on modes of encryption: ~~ECB~~, CBC, CTR

On Keys

- All of this uses and generates keys
- How is this done?

On Keys

- All of this uses and generates keys
- How is this done?

For Symmetric Crypto

- Generated uniformly at random
- Derived using a Key Derivation Function
  - From a password or low entropy secret
  - From a high-entropy master key from key exchange protocol

On Keys

- All of this uses and generates keys
- How is this done?

For Symmetric Crypto

- Generated uniformly at random
- Derived using a Key Derivation Function
  - From a password or low entropy secret
  - From a high-entropy master key from key exchange protocol

For Asymmetric Crypto

- Key generation algorithm $\rightarrow$ key pair
- Private key holder generates both keys; publishes public key
- Asymmetric keys are typically much larger
  - RSA keys take roughly 4096-bits for 128-bit security
  - Elliptic-curve keys take roughly 400-bits for 128-bit security

Storage and Generation

Keys are often *the most sensitive material* a secure system holds

Storage and Generation

Keys are often *the most sensitive material* a secure system holds

Ideally, in an external secure hardware

- Hardware Security Module (HSM)
- Smartcard or similar cryptographic token

Storage and Generation

Keys are often *the most sensitive material* a secure system holds

Ideally, in an external secure hardware

- Hardware Security Module (HSM)
- Smartcard or similar cryptographic token

Key wrapping

- Long-term keys are often *wrapped* before storage
- To encrypt with another key
- Password-based encryption (low security)
- Wrap with HW-protected master key (standard security)
- Master key stored in trusted hardware (high security)

To Be Random

Q1: Which of these numbers are random?

1. 00000000
2. 10101010
3. 00100100
4. 10011101

To Be Random

Q1: Which of these numbers are random?

1. 00000000 - Not random!
2. 10101010 - Not random (pattern)
3. 00100100 - Maybe not random?
4. 10011101 - Seems random...

To Be Random

Q1: Which of these numbers are random?

1. 00000000 - Not random!
2. 10101010 - Not random (pattern)
3. 00100100 - Maybe not random?
4. 10011101 - Seems random...

Randomness is not a property of a bit string, but rather:

- The bit generation process
- The bit string sampling procedure

To Be Random

Q1: Which of these numbers are random?

1. 00000000 - Not random!
2. 10101010 - Not random (pattern)
3. 00100100 - Maybe not random?
4. 10011101 - Seems random...

Randomness is not a property of a bit string, but rather:

- The bit generation process
- The bit string sampling procedure

Q2: Which of these numbers will more likely appear in a fair randomness generator?

Randomness Distributions

Randomized processes described using *randomness distributions*.

We start with the **uniform distribution** over a finite field S.

A process U samples from the uniform distribution if

$$\forall s^* \in S, \Pr[s = s^* : s \leftarrow_{\$} U] = \frac{1}{|S|}$$

Randomness Distributions

Randomized processes described using *randomness distributions*.

We start with the **uniform distribution** over a finite field S.

A process U samples from the uniform distribution if

$$\forall s^* \in S, \Pr[s = s^* : s \leftarrow_\$ U] = \frac{1}{|S|}$$

Q1: If we roll a fair dice, what is the probability of getting 1?

Randomness Distributions

Randomized processes described using *randomness distributions*.

We start with the **uniform distribution** over a finite field S.

A process U samples from the uniform distribution if

$$\forall s^* \in S, \Pr[s = s^* : s \xleftarrow{\$} U] = \frac{1}{|S|}$$

Q1: If we roll a fair dice, what is the probability of getting 1?

$$\frac{1}{6} \approx 0.1667$$

Randomness Distributions

Randomized processes described using *randomness distributions*.

We start with the **uniform distribution** over a finite field S.

A process U samples from the uniform distribution if

$$\forall s^* \in S, \Pr[s = s^* : s \xleftarrow{\$} U] = \frac{1}{|S|}$$

Q1: If we roll a fair dice, what is the probability of getting 1?

$$\frac{1}{6} \approx 0.1667$$

Q2: If we do a fair sampling of a byte, what is the probability of getting 00000000 **or** 10011101?

Randomness Distributions

Randomized processes described using *randomness distributions*.

We start with the **uniform distribution** over a finite field S.

A process U samples from the uniform distribution if

$$\forall s^* \in S, \Pr[s = s^* : s \leftarrow_\$ U] = \frac{1}{|S|}$$

Q1: If we roll a fair dice, what is the probability of getting 1?

$$\frac{1}{6} \approx 0.1667$$

Q2: If we do a fair sampling of a byte, what is the probability of getting 00000000 **or** 10011101?

Both are $\frac{1}{2^8} \approx 0.0078$

Caution: statistical tests are not sufficient

- **Q: What type of tests can we do over "random" inputs?**

Caution: statistical tests are not sufficient

- **Q: What type of tests can we do over "random" inputs?**
  - Count number of 1s and 0s
  - Check distribution of 8-bit words
  - Look for patterns
  - ...

Irrelevant for Security

- Possible to pass statistical tests
- Totally insecure for cryptographic purposes

Linux systems

- PRG is accessible at */dev/urandom*
  - In $*$nix-style, PRG is mapped to a file
  - Careful to make sure system calls are successful!

Linux systems

- PRG is accessible at */dev/urandom*
  - In $*$nix-style, PRG is mapped to a file
  - Careful to make sure system calls are successful!

Link to code from **LibreSSL**

In some variants, there is a blocking */dev/random* based on an entropy simulator

- Check if there is "sufficient entropy"
- Blocks otherwise
- Current consensus indicates that, for most applications, this is not useful (see **this link** for more information)

Concrete Numbers

Some numbers for scale

- Not easy to perceive very very large numbers
- The estimated age of the universe in nanosecs is around 2^{88}
- The number of atoms in the universe is roughly 2^{256}

Concrete Numbers

Some numbers for scale

- Not easy to perceive very very large numbers
- The estimated age of the universe in nanosecs is around 2^{88}
- The number of atoms in the universe is roughly 2^{256}

A common security parameter

- A common size for keys is 128 bits
- Consider the following events
  - Winning a lottery with 9 million participants (all of Portugal)
  - Guessing a 2^{128} size key at the first try

Concrete Numbers

Some numbers for scale

- Not easy to perceive very very large numbers
- The estimated age of the universe in nanosecs is around 2^{88}
- The number of atoms in the universe is roughly 2^{256}

A common security parameter

- A common size for keys is 128 bits
- Consider the following events
  - Winning a lottery with 9 million participants (all of Portugal)
  - Guessing a 2^{128} size key at the first try

Q1: Which event is more likely?

Concrete Numbers

Some numbers for scale

- Not easy to perceive very very large numbers
- The estimated age of the universe in nanosecs is around 2^{88}
- The number of atoms in the universe is roughly 2^{256}

A common security parameter

- A common size for keys is 128 bits
- Consider the following events
  - Winning a lottery with 9 million participants (all of Portugal)
  - Guessing a 2^{128} size key at the first try

Q1: Which event is more likely?

Q2: By how much?

Quantifying Security

Lower bound on the work required for a successful attack

Number of steps of the best attack

- n-bits security
- Best attack to break the scheme requires 2^n steps
- n-bit keys cannot ever give more than n-bit security

Quantifying Security

Lower bound on the work required for a successful attack

Number of steps of the best attack

- n-bits security
- Best attack to break the scheme requires 2^n steps
- n-bit keys cannot ever give more than n-bit security
  - **Q1: Why?**

Quantifying Security

Lower bound on the work required for a successful attack

Number of steps of the best attack

- n-bits security
- Best attack to break the scheme requires 2^n steps
- n-bit keys cannot ever give more than n-bit security
  - **Q1: Why?**
- Brute-force attack allows finding the correct key
- l-bit keys could lead to n-bit security s.t. $n << t$

Quantifying Security

Lower bound on the work required for a successful attack

Number of steps of the best attack

- n-bits security
- Best attack to break the scheme requires 2^n steps
- n-bit keys cannot ever give more than n-bit security
  - **Q1: Why?**
- Brute-force attack allows finding the correct key
- l-bit keys could lead to n-bit security s.t. $n << t$
  - **Q2: When?**

Quantifying Security

Lower bound on the work required for a successful attack

Number of steps of the best attack

- n-bits security
- Best attack to break the scheme requires 2^n steps
- n-bit keys cannot ever give more than n-bit security
  - **Q1: Why?**
- Brute-force attack allows finding the correct key
- l-bit keys could lead to n-bit security s.t. $n << t$
  - **Q2: When?**
  - Best attack is more efficient than brute-force
  - Common in asymmetric cryptography
  - Keys must follow specific structures, not random bit strings
- Quantifying using n-bit security permits comparing schemes

Good Security Values for Real-world Crypto

The 2^{128} rule of thumb

- Designs for which best attacks require roughly 2^{128} attempts

Good Security Values for Real-world Crypto

The 2^{128} rule of thumb

- Designs for which best attacks require roughly 2^{128} attempts

For how long do we need security to hold?

- Moore's law: computational power doubles every 2 years
- $n + 1$ bit security every 2 years
- This no longer seems to be true, but...
- Maybe we will have quantum computers soon

Long-term security: $\approx$ 256-bit keys

Short-term security: $\approx$ 80-bit keys may be OK

Key Takeaways

- Randomness is an important and challenging topic
  - Necessary to ensure strength of cryptographic keys
  - which (recall Kerkhoff's principle) is where security lies
  - It is a property of the generator, not the values

Key Takeaways

- Randomness is an important and challenging topic
  - Necessary to ensure strength of cryptographic keys
  - which (recall Kerkhoff's principle) is where security lies
  - It is a property of the generator, not the values
- For real-world systems
  - Generators use entropy to gather randomness
  - Hard-to-predict events
  - Upon setup, entropy is low, so *be careful!*

Key Takeaways

- Randomness is an important and challenging topic
  - Necessary to ensure strength of cryptographic keys
  - which (recall Kerkhoff's principle) is where security lies
  - It is a property of the generator, not the values
- For real-world systems
  - Generators use entropy to gather randomness
  - Hard-to-predict events
  - Upon setup, entropy is low, so *be careful!*
- Concrete security
  - Security relates to the size of the key
  - Maximum security is the key size (recall one-time-pad!)
  - 2^{128} bit security is often a good number
  - Long-term: 256-bit keys; short-term 80-bit sometimes suffices

Computer Security Foundations
Week 8: Symmetric Encryption

Bernardo Portela

L.EIC - 24