Applied Cryptography
Week 3: Block Ciphers

Bernardo Portela

M:ERSI, M:SI - 23

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**

Defining Security for Block Ciphers

Block cipher should be a **pseudorandom permutation** (PRP)

Q1: How can we define this concretely?

Defining Security for Block Ciphers

Block cipher should be a **pseudorandom permutation** (PRP)

Q1: How can we define this concretely? Using an experiment:

- Experiment samples uniformly at random:
  - $k \in \{0,1\}^\lambda$
  - permutation $\pi : \{0,1\}^B \Rightarrow \{0,1\}^B$
  - bit b

Defining Security for Block Ciphers

Block cipher should be a **pseudorandom permutation** (PRP)

Q1: How can we define this concretely? Using an experiment:

- Experiment samples uniformly at random:
  - $k \in \{0,1\}^\lambda$
  - permutation $\pi : \{0,1\}^B \Rightarrow \{0,1\}^B$
  - bit b
- Attacker can ask for encryptions:
  - Attacker selects $p \in \{0,1\}^B$
  - If $b = 0$, experiment returns $E(k, p)$
  - Otherwise, experiment returns $\pi(p)$

Defining Security for Block Ciphers

Block cipher should be a **pseudorandom permutation** (PRP)

Q1: How can we define this concretely? Using an experiment:

- Experiment samples uniformly at random:
  - $k \in \{0,1\}^\lambda$
  - permutation $\pi : \{0,1\}^B \Rightarrow \{0,1\}^B$
  - bit b
- Attacker can ask for encryptions:
  - Attacker selects $p \in \{0,1\}^B$
  - If $b = 0$, experiment returns $E(k,p)$
  - Otherwise, experiment returns $\pi(p)$
- Attacker outputs b' and wins if $b = b'$

Defining Security for Block Ciphers

Block cipher should be a **pseudorandom permutation** (PRP)

Q1: How can we define this concretely? Using an experiment:

- Experiment samples uniformly at random:
  - $k \in \{0,1\}^\lambda$
  - permutation $\pi : \{0,1\}^B \Rightarrow \{0,1\}^B$
  - bit b
- Attacker can ask for encryptions:
  - Attacker selects $p \in \{0,1\}^B$
  - If $b = 0$, experiment returns $E(k,p)$
  - Otherwise, experiment returns $\pi(p)$
- Attacker outputs b' and wins if $b = b'$

Q2: How do we calculate the adversarial advantage?

Defining Security for Block Ciphers

Block cipher should be a **pseudorandom permutation** (PRP)

Q1: How can we define this concretely? Using an experiment:

- Experiment samples uniformly at random:
  - $k \in \{0,1\}^{\lambda}$
  - permutation $\pi : \{0,1\}^B \Rightarrow \{0,1\}^B$
  - bit b
- Attacker can ask for encryptions:
  - Attacker selects $p \in \{0,1\}^B$
  - If $b = 0$, experiment returns $E(k,p)$
  - Otherwise, experiment returns $\pi(p)$
- Attacker outputs b' and wins if $b = b'$

Q2: How do we calculate the adversarial advantage?

Advantage: $|\Pr[b = b'] - \frac{1}{2}|$

Implications of PRP Security

Our scheme is *indistinguishable* from a random permutation.
What is a random permutation ($\pi : \{0,1\}^B \Rightarrow \{0,1\}^B$), exactly?

Implications of PRP Security

Our scheme is *indistinguishable* from a random permutation.
What is a random permutation ($\pi : \{0,1\}^B \Rightarrow \{0,1\}^B$), exactly?

- Huge table with 2^B entries, indexed by plaintext p
- Each entry contains C
- Each C is sampled uniformly at random, without repeats
  - **Q: Why must Cs never repeat?**

Implications of PRP Security

Our scheme is *indistinguishable* from a random permutation.
What is a random permutation ($\pi : \{0,1\}^B \Rightarrow \{0,1\}^B$), exactly?

- Huge table with 2^B entries, indexed by plaintext p
- Each entry contains C
- Each C is sampled uniformly at random, without repeats
  - **Q: Why must Cs never repeat?**
  - PRPs are invertible!
  - Different from purely random functions

Implications of PRP Security

Our scheme is *indistinguishable* from a random permutation.
What is a random permutation ($\pi : \{0,1\}^B \Rightarrow \{0,1\}^B$), exactly?

- Huge table with 2^B entries, indexed by plaintext p
- Each entry contains C
- Each C is sampled uniformly at random, without repeats
  - **Q: Why must Cs never repeat?**
  - PRPs are invertible!
  - Different from purely random functions

Implications

- Ciphertext blocks look totally random
- Different inputs $\Rightarrow$ independent outputs
- Must be impossible to recover key

Selecting the Block Size

E and D work on bitstrings of size B – the *block size*

Data Encryption Standard (DES, 70s-90s): $B = 64$ (8 bytes)

Advanced Encryption Standard (AES, 2000s-): $B = 128$ (16 bytes)

Selecting the Block Size

E and D work on bitstrings of size B – the *block size*

Data Encryption Standard (DES, 70s-90s): $B = 64$ (8 bytes)

Advanced Encryption Standard (AES, 2000s-): $B = 128$ (16 bytes)

- Block must be small for efficient SW/HW implementation
- Block cannot be too small
    - Constructions based on block ciphers
    - Key space 2^λ
    - Block size must be close to the security parameter $B \approx \lambda$

Some encryption schemes based on block constructions are insecure if the block size is too small (64 can be problematic). More information **here**

Iterated Ciphers: Rounds

Shorter descriptions and code/HW footprints:

- Simple and efficient round algorithm R
- Round algorithm is not as secure as a block cipher
- Block cipher iterates round algorithm n times

Iterated Ciphers: Rounds

Shorter descriptions and code/HW footprints:

- Simple and efficient round algorithm R
- Round algorithm is <u>not</u> as secure as a block cipher
- Block cipher iterates round algorithm n times

- Each round takes a different key
  - Round key *derived* from block cipher key
  - Sequence of round keys called *key schedule*
- Decrypting follows the same method in reverse
- E.g. for a 3 round scheme:

$$c \leftarrow E(k, p) = R_3(k_3, R_2(k_2, R_1(k_1, p)))$$
$$p \leftarrow D(k, c) = R_1^{-1}(k_1, R_2^{-1}(k_2, R_3^{-1}(k_3, c)))$$

Round Functions #1: Substitution-Permutation Networks

- **Substitution:** S-boxes are small lookup tables (4-8 bits) designed to introduce non-linearity in the round function. They create *confusion*
- **Permutation:** Bit-level transformations (e.g. switches) or algebraic functions that introduce dependencies across the whole block (*diffusion*)

Round Functions #1: Substitution-Permutation Networks

- **Substitution:** S-boxes are small lookup tables (4-8 bits) designed to introduce non-linearity in the round function. They create *confusion*
- **Permutation:** Bit-level transformations (e.g. switches) or algebraic functions that introduce dependencies across the whole block (*diffusion*)

Q: Why is diffusion necessary?

Round Functions #1: Substitution-Permutation Networks

- **Substitution:** S-boxes are small lookup tables (4-8 bits) designed to introduce non-linearity in the round function. They create *confusion*
- **Permutation:** Bit-level transformations (e.g. switches) or algebraic functions that introduce dependencies across the whole block (*diffusion*)

Q: Why is diffusion necessary?

Consider the encryption of "Attack at dawn" and "Attack at dusk"

Round Functions #1: Substitution-Permutation Networks

- **Substitution:** S-boxes are small lookup tables (4-8 bits) designed to introduce non-linearity in the round function. They create *confusion*
- **Permutation:** Bit-level transformations (e.g. switches) or algebraic functions that introduce dependencies across the whole block (*diffusion*)

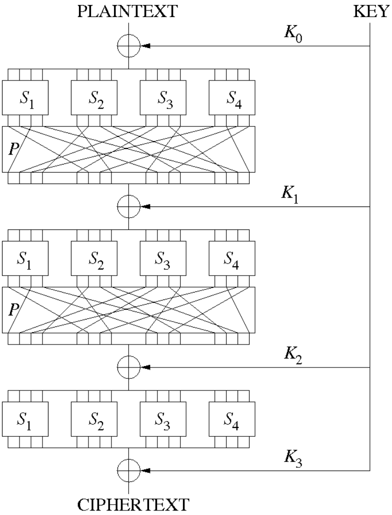
Q: Why is diffusion necessary?
Consider the encryption of "Attack at dawn" and "Attack at dusk"

S-boxes heuristically designed to

- Create complex relations between input and output
- Minimize statistical bias in outputs

Example block cipher: AES

Substitution-Permutation Networks - High-level View



(from Wikipedia)

Round Functions #2: Feistel Networks

Round function processes half of the block

- Input block seen as pair (l, r)
- Output block is $(r \oplus R(k_i, l), l)$
- R is the round function

Round Functions #2: Feistel Networks

Round function processes half of the block

- Input block seen as pair (l, r)
- Output block is $(r \oplus R(k_i, l), l)$
- R is the round function

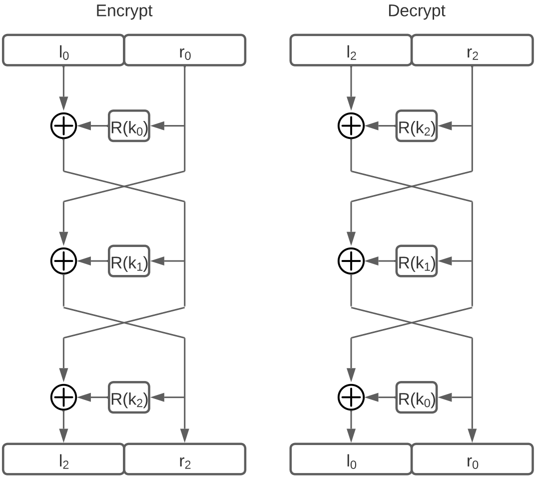
Unprocessed half-block is masked to the next round

Decryption is identical to encryption

- Only key scheduling is inverted
- Very important for HW optimization in the 70s

Example block cipher: DES, GOST

Feistel Networks - High-level View



Round Functions for Feistel Networks

Contrary to SPNs, Feistel Network's R don't have to be invertible

Round Functions for Feistel Networks

Contrary to SPNs, Feistel Network's R don't have to be invertible

- These can be Pseudorandom Functions (PRFs)
- A PRF is similar to a PRP, but not necessarily invertible
- Input size can be different from output size
- Security experiment is similar to that of the PRP:

Round Functions for Feistel Networks

Contrary to SPNs, Feistel Network's R don't have to be invertible

- These can be Pseudorandom Functions (PRFs)
- A PRF is similar to a PRP, but not necessarily invertible
- Input size can be different from output size
- Security experiment is similar to that of the PRP:
  - Experiment chooses a random f
  - Rather than a random permutation π
  - **Q: Is the domain space of random functions larger or smaller than that of all permutations?**

Round Functions for Feistel Networks

Contrary to SPNs, Feistel Network's R don't have to be invertible

- These can be Pseudorandom Functions (PRFs)
- A PRF is similar to a PRP, but not necessarily invertible
- Input size can be different from output size
- Security experiment is similar to that of the PRP:
  - Experiment chooses a random f
  - Rather than a random permutation π
  - **Q: Is the domain space of random functions larger or smaller than that of all permutations?**
- If the round function is secure, 4 rounds ensure a PRP!
- Practical block ciphers use extra rounds
  - Round functions heuristically designed

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 $E(k_1, D(k_2, E(k_3, p)))$
- **Q: Why EDE and not EEE?**

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 $E(k_1, D(k_2, E(k_3, p)))$
- **Q: Why EDE and not EEE?**

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
- Open to proposals, scrutinized by the community
- 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 - Explained

The substitution-permutation network uses:

- **AddRoundKey** - $\oplus$ with the state
- **SubBytes** - Replace each byte using lookup table (S-Box)
- **ShiftRows** - Matrix rows shifted 0..3 positions
- **MixColumns** - Columns transformed

Internals of AES - Explained

The substitution-permutation network uses:

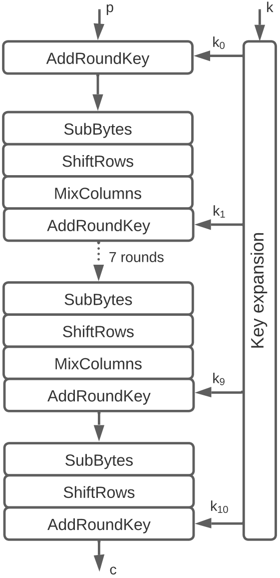
- **AddRoundKey** - $\oplus$ with the state
- **SubBytes** - Replace each byte using lookup table (S-Box)
- **ShiftRows** - Matrix rows shifted 0..3 positions
- **MixColumns** - Columns transformed

SubBytes performs the substitution part

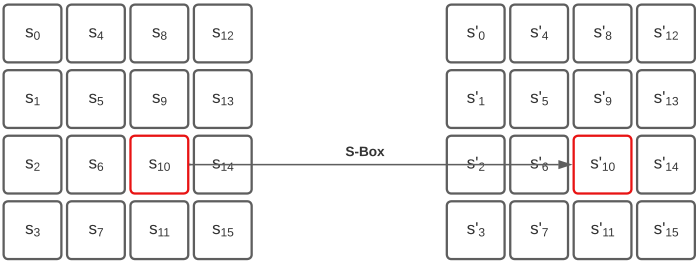
ShiftRows and **MixColumns** are the permutation

Last round has no **MixColumns**. Not necessary. Read more **here**

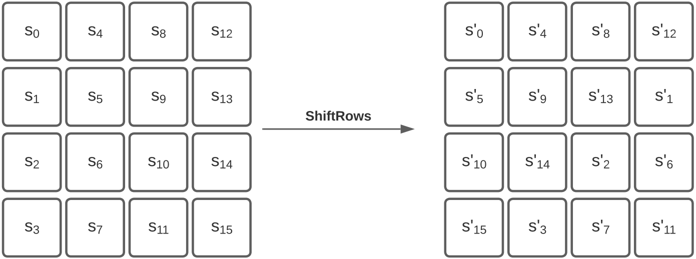
Internals of AES - High Level View



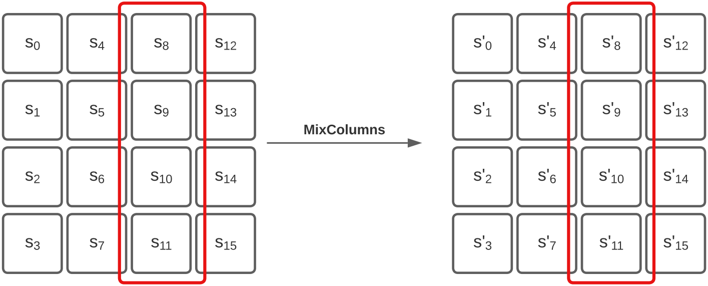
Internals of AES - SubBytes



Internals of AES - ShiftBytes



Internals of AES - MixColumns



Implementing AES

The not so good

- AES is hard to implement in software
- Naive implementations using tables leak via side-channels
- Removing side-channels in software is hard

Implementing AES

The not so good

- AES is hard to implement in software
- Naive implementations using tables leak via side-channels
- Removing side-channels in software is hard

The good

- AES is super fast in mainstream processors
- AES-NI - AES Native Instructions
- From SW one can resort to HW AES

Security of AES

There is no mathematical proof that AES is a PRP

All practical applications based on AES assume this

Security of AES

There is no mathematical proof that AES is a PRP

All practical applications based on AES assume this

AES has been around for 25 years:

- No significant cryptanalysis progress
- AES scrutiny is an important area of research
- Direct attack on AES unlikely to be the weakest link

Security of AES

There is no mathematical proof that AES is a PRP

All practical applications based on AES assume this

AES has been around for 25 years:

- No significant cryptanalysis progress
- AES scrutiny is an important area of research
- Direct attack on AES unlikely to be the weakest link

Assuming AES is a PRP gives us provably secure and very efficient symmetric encryption schemes

Using Block Ciphers Directly

Recall our secure PRP 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 $p \in \{0,1\}^B$

Using Block Ciphers Directly

Recall our secure PRP 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 problem arises in 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 from block ciphers
- We prove encryption secure assuming a block cipher PRP

Defining Symmetric Encryption

Syntax

- Key Generation: Often uniform sampling in $\{0,1\}^\lambda$
- Encryption: Probabilistic algorithm $c \leftarrow_{\$} E(k,m)$
- Decryption: Deterministic algorithm $m/\bot \leftarrow D(k,c)$

Defining Symmetric Encryption

Syntax

- Key Generation: Often uniform sampling in $\{0,1\}^\lambda$
- Encryption: Probabilistic algorithm $c \leftarrow_\$ E(k, m)$
- Decryption: Deterministic algorithm $m/\perp \leftarrow D(k, c)$

Security (IND-CPA)

- Experiment samples k and bit b uniformly at random
- Attacker can query encryptions of chosen messages
- Attacker outputs (m_0, m_1) s.t. $|m_0| = |m_1|$
- Attacker gets $c \leftarrow_\$ E(k, m_b)$
- Attacker outputs b' and wins if $b = b'$

Defining Symmetric Encryption

Syntax

- Key Generation: Often uniform sampling in $\{0,1\}^\lambda$
- Encryption: Probabilistic algorithm $c \leftarrow_\$ E(k, m)$
- Decryption: Deterministic algorithm $m/\bot \leftarrow D(k, c)$

Security (IND-CPA)

- Experiment samples k and bit b uniformly at random
- Attacker can query encryptions of chosen messages
- Attacker outputs (m_0, m_1) s.t. $|m_0| = |m_1|$
- Attacker gets $c \leftarrow_\$ E(k, m_b)$
- Attacker outputs b' and wins if $b = b'$

Advantage: $|\Pr[b = b'] - \frac{1}{2}|$

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?**

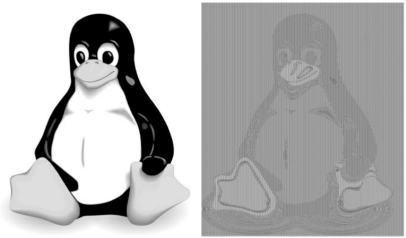
ECB is broken because you can see the penguin!

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!



Breaking ECB

What is the issue?

- Equal input blocks $\Rightarrow$ Equal output blocks
- Preserves patterns that vary slower than block size

Breaking ECB

What is the issue?

- Equal input blocks $\Rightarrow$ Equal output blocks
- Preserves patterns that vary slower than block size

Q1: Can we prove it is insecure (win the game)?

Breaking ECB

What is the issue?

- Equal input blocks ⇒ Equal output blocks
- Preserves patterns that vary slower than block size

Q1: Can we prove it is insecure (win the game)?

- Output $m_0 \neq m_1$, $|m_0| = |m_1|$, get c
- Request an encryption of m_0 to get c^*
- If $b' = 0$ iff $c = c^*$

This attack works against **all** deterministic encryption schemes

Breaking ECB

What is the issue?

- Equal input blocks $\Rightarrow$ Equal output blocks
- Preserves patterns that vary slower than block size

Q1: Can we prove it is insecure (win the game)?

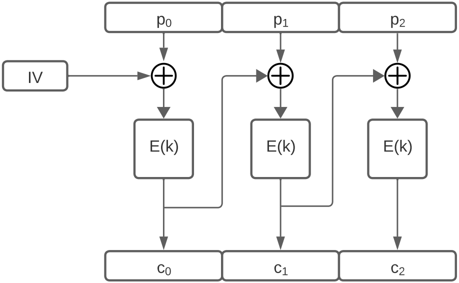
- Output $m_0 \neq m_1$, $|m_0| = |m_1|$, get c
- Request an encryption of m_0 to get c^*
- If $b' = 0$ iff $c = c^*$

This attack works against **all** deterministic encryption schemes

Q2: Can we prove it is insecure not querying exactly m_0/m_1?

Cipher Block Chaining

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



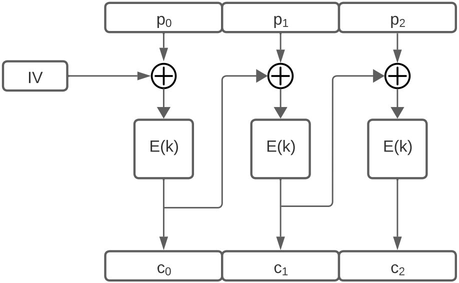
- Main difference to ECB is the Initialization Vector (IV)
- Blocks depend on each other

Cipher Block Chaining: Performance and Security

Intuition of CBC security

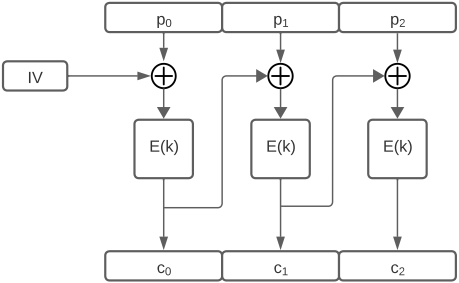
- Random IV makes first block-cipher input random
- Block cipher security implies c_1 looks random and independent
- CBC uses c_1 as the IV for the second block
- Same argument for c_2
- Two encryptions of the same plaintext look independent

Working with CBC



- **Q1: How can we do decryption?**

Working with CBC



- **Q1: How can we do decryption?**
- **Q2: Can we speed encrypt/decrypt with parallelism?**

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
  - $0x01$ means 1 byte of padding with that value
  - $0x03$ 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**

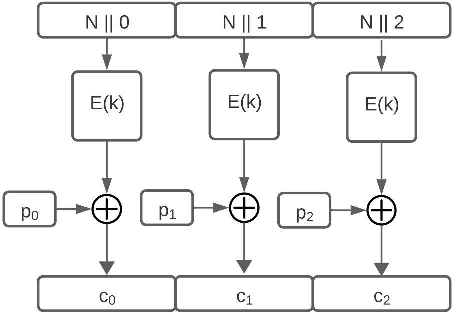
The most common padding scheme is specified in PKCS#7:

- 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
    - $0\text{x}01$ means 1 byte of padding with that value
    - $0\text{x}03$ means 3 bytes of padding with that value

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

Counter Block Mode

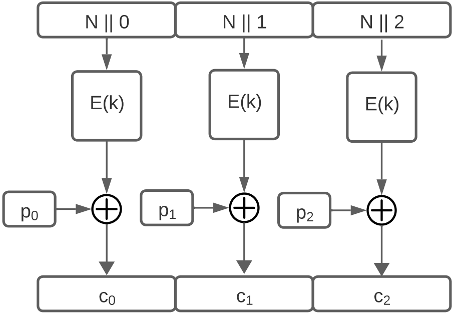
Often Counter Block Mode (CTR) is used in Nonce-based form



- 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

Errors in Designing Modes of Operation

Recall the guarantees of IND-CPA

- Attacker has access to encryptions
- Can't extract any information about messages
- What if it has access to side information on decryption?
- No guarantee that modified ciphertext is rejected: what leaks?

Errors in Designing Modes of Operation

Recall the guarantees of IND-CPA

- Attacker has access to encryptions
- Can't extract any information about messages
- What if it has access to side information on decryption?
- No guarantee that modified ciphertext is rejected: what leaks?

A (very real) practical example:

- Padding oracle attacks against AES-CBC (TLS 1.*)
- Attacker gets to observe padding check error
- This is enough to recover plaintext (e.g. cookies)

Errors in Designing Modes of Operation

Recall the guarantees of IND-CPA

- Attacker has access to encryptions
- Can't extract any information about messages
- What if it has access to side information on decryption?
- No guarantee that modified ciphertext is rejected: what leaks?

A (very real) practical example:

- Padding oracle attacks against AES-CBC (TLS 1.*)
- Attacker gets to observe padding check error
- This is enough to recover plaintext (e.g. cookies)

At the root of the problem: allowing non-authenticated ciphertexts

Applied Cryptography
Week 3: Block Ciphers

Bernardo Portela

M:ERSI, M:SI - 23