Confidentiality

EECS 388, Fall 2017

Administrivia

Homework 1 due Today, September 14 at 6pm!

Project 1 out Today, September 14

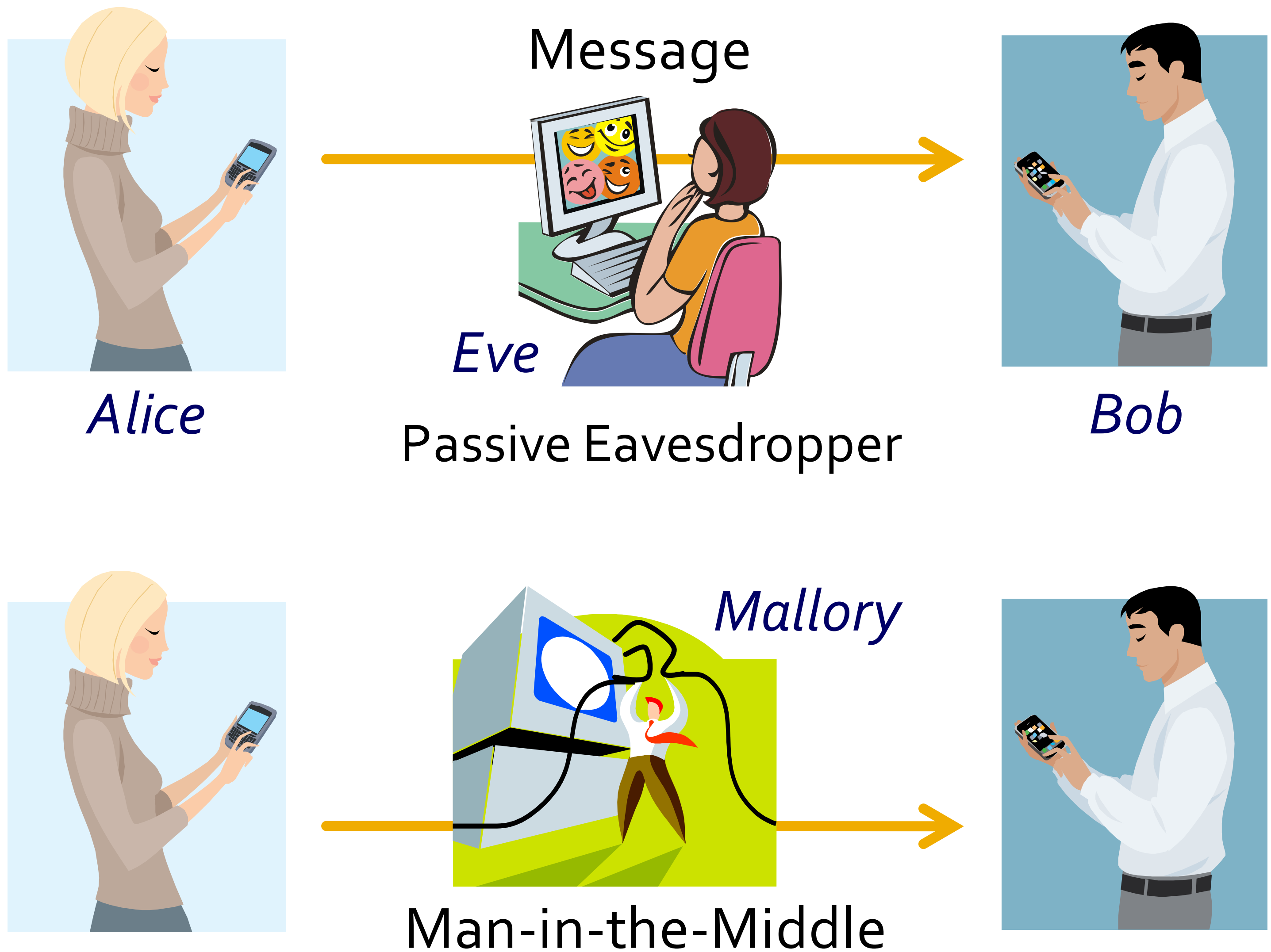
Submit through Canvas

Properties of a Secure Channel

Confidentiality

Integrity

Authentication


Message
Eve
Alice
Passive Eavesdropper
Bob
Mallory
Man-in-the-Middle

Integrity Review

1. Let f be a secure PRF.

2. In advance choose a random k known only to Alice and Bob.

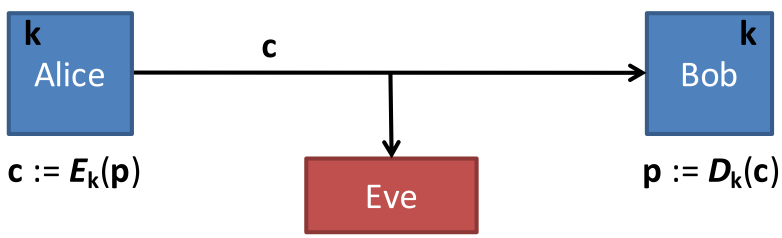
3. Alice computes **v** := $\boldsymbol{f}_{\mathbf{k}}(\mathbf{m})$.

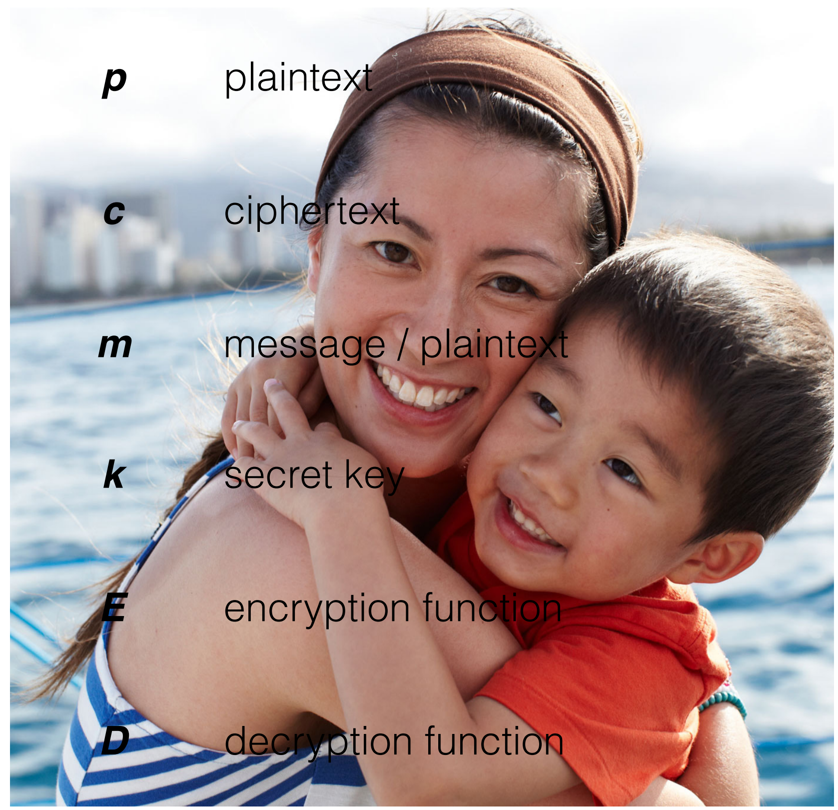
4. 

5. Bob verifies **v′** = $\boldsymbol{f}_{\mathbf{k}}(\mathbf{m'})$, accepts if and only if this is true.

Confidentiality Review

Goal: Keep contents of message p secret from an eavesdropper





One-time Pad Review

Alice and Bob jointly generate a secret, very long string of random bits.

The *one-time pad* $\boldsymbol{k}$

$\boldsymbol{E(p_i) = c_i := p_i}$ xor $\boldsymbol{k_i}$

$\boldsymbol{D(c_i) = p_i := c_i}$ xor $\boldsymbol{k_i}$

Don't reuse pads *[why?]*

Provably secure *[why?]*

Usually impractical *[why?]*

a	***b***	***a* xor *b***
0	0	0
0	1	1
1	0	1
1	1	0

Q: What can we use instead of a truly random pad?

A: Output of a Pseudorandom Generator (PRG)

Secure PRG: *A secure PRG takes as input seed **k**, outputs a stream that is practically indistinguishable from true randomness unless you know **k**.*

Stream Cipher

1. Start with shared secret ***k***
2. Alice and Bob each use ***k*** to seed PRG
3. To encrypt, Alice XORs next bit of her generator's output with next bit of plaintext
4. To decrypt Bob XOR's next bit of his generator's output with next bit of ciphertext

$\boldsymbol{E(p_i) = c_i := p_i}$ xor $\boldsymbol{PRG(k)_i}$

$\boldsymbol{D(c_i) = p_i := c_i}$ xor $\boldsymbol{PRG(k)_i}$

Stream Cipher Caveats

$\boldsymbol{E(p_i) = c_i = p_i}$ xor $\boldsymbol{PRG(k)_i}$

$\boldsymbol{D(c_i) = p_i = c_i}$ xor $\boldsymbol{PRG(k)_i}$

Don’t <u>ever</u> reuse keys

Don’t <u>ever</u> reuse output of PRG

Real-World Stream Ciphers

- ~~RC2~~, ~~RC4~~
- Salsa20

https://github.com/golang/crypto/tree/master/salsa20

Block Ciphers

Functions that encrypt fixed-size blocks with a reusable key

Inverse function decrypts when used with the same key

A block cipher is not a pseudorandom function *[why?]*

Q: What can we use instead of a PRF?

A: Pseudorandom Permutation (PRP)

Secure PRP: *A secure PRP takes as input seed **k**, outputs a permutation that is practically indistinguishable from truly random permutation unless you know **k**.*

Pseudorandom Permutation

Input: ***n***-bit string

Output: ***n***-bit string

Basic Challenge: Design a “hairy” function that is invertible iff you know k

- Highly nonlinear (“confusion”)
- Mixes input bits together (“diffusion”)
- Depends on the key

Secure PRP: *A secure PRP takes as input seed* ***k****, outputs a permutation that is practically indistinguishable from truly random permutation unless you know* ***k****.*

AES (Advanced Encryption Standard)

Designed and standardized by NIST competition, long public comment/discussion period.

Widely believed to be secure, but we don’t know how to prove its security.

128-bit block size

Variable key size,128 or 256 bits

AES Construction

"Round-based" with <u>ten rounds</u>

- Split ***k*** into ten <u>subkeys</u>
- Perform ten rounds of substitution/permutation operations, each time with a different subkey

Foot-Shooting Prevention Agreement

I, ______________ , promise that once
Your Name

I see how simple AES really is, I will not implement it in production code even though it would be really fun.

This agreement shall be in effect until the undersigned creates a meaningful interpretive dance that compares and contrasts cache-based, timing, and other side channel attacks and their countermeasures.

X______________________________ __________
Signature Date

http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html

AES Round

Input: 128-bits “plaintext”, 128-bit subkey
Output: 128 bits “ciphertext”

Picture as operations on a 4x4 grid of 8-bit values

1. **Non-linear substitution**
   - **Run each byte thru a non-linear (but invertible) function (or S-box)**
2. Shift rows
   - Circular-shift each row
   - i^{th} row shifted by i
3. Linear-mix columns
   - Treat each column as a 4-vector
   - Multiplication is in GF(2^8), addition is XOR
4. Key-addition
   - XOR each byte with corresponding byte of round subkey

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

AES Round

<u>Input</u>: 128-bits “plaintext”, 128-bit subkey
<u>Output</u>: 128 bits “ciphertext”

Picture as operations on a 4x4 grid of 8-bit values

1. **Non-linear substitution**
   - **Run each byte thru a non-linear (but invertible) function (or S-box)**

2. Shift rows
   - Circular-shift each row
   - i^{th} row shifted by i

3. Linear-mix columns
   - Treat each column as a 4-vector
   - Multiplication is in GF(2^8), addition is XOR

4. Key-addition
   - XOR each byte with corresponding byte of round subkey

$b_{0,0}$	$b_{0,1}$	$b_{0,2}$	$b_{0,3}$
$b_{1,0}$	$b_{1,1}$	$b_{1,2}$	$b_{1,3}$
$b_{2,0}$	$b_{2,1}$	$b_{2,2}$	$b_{2,3}$
$b_{3,0}$	$b_{3,1}$	$b_{3,2}$	$b_{3,3}$

AES Round

<u>Input</u>: 128-bits “plaintext”, 128-bit subkey
<u>Output</u>: 128 bits “ciphertext”

Picture as operations on a 4x4 grid of 8-bit values

1. Non-linear substitution
   - Run each byte thru a non-linear (but invertible) function (or S-box)

2. **Shift rows**
   - **Circular-shift each row**
   - ***i*th row shifted by *i***

3. Linear-mix columns
   - Treat each column as a 4-vector
   - Multiplication is in GF(2^8), addition is XOR

4. Key-addition
   - XOR each byte with corresponding byte of round subkey

$b_{0,0}$	$b_{0,1}$	$b_{0,2}$	$b_{0,3}$
$b_{1,3}$	$b_{1,0}$	$b_{1,1}$	$b_{1,2}$
$b_{2,2}$	$b_{2,3}$	$b_{2,0}$	$b_{2,1}$
$b_{3,1}$	$b_{3,2}$	$b_{3,3}$	$b_{3,0}$

AES Round

Input: 128-bits “plaintext”, 128-bit subkey
Output: 128 bits “ciphertext”

Picture as operations on a 4x4 grid of 8-bit values

1. Non-linear substitution
   - Run each byte thru a non-linear function (lookup table / “S-boxes”)

2. Shift rows
   - Circular-shift each row
   - ***i***th row shifted by ***i***

3. **Linear-mix columns**
   - **Treat each column as a 4-vector**
   - **Multiplication is in GF(2^8), addition is XOR**

4. Key-addition
   - XOR each byte with corresponding byte of round subkey

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \times \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

AES Round

Input: 128-bits “plaintext”, 128-bit subkey
Output: 128 bits “ciphertext”

Picture as operations on a 4x4 grid of 8-bit values

1. Non-linear substitution
   - Run each byte thru a non-linear (but invertible) function (or S-box)

2. Shift rows
   - Circular-shift each row
   - i^{th} row shifted by i

3. Linear-mix columns
   - Treat each column as a 4-vector
   - Multiplication is in GF(2^8), addition is XOR

4. Key-addition
   - **XOR each byte with corresponding byte of round subkey**

$c_{0,0} \oplus k_{0,0}$	$c_{0,1} \oplus k_{0,1}$	$c_{0,2} \oplus k_{0,2}$	$c_{0,3} \oplus k_{0,3}$
$c_{1,0} \oplus k_{1,0}$	$c_{1,1} \oplus k_{1,1}$	$c_{1,2} \oplus k_{1,2}$	$c_{1,3} \oplus k_{1,3}$
$c_{2,0} \oplus k_{2,0}$	$c_{2,1} \oplus k_{2,1}$	$c_{2,2} \oplus k_{2,2}$	$c_{2,3} \oplus k_{2,3}$
$c_{3,0} \oplus k_{3,0}$	$c_{3,1} \oplus k_{3,1}$	$c_{3,2} \oplus k_{3,2}$	$c_{3,3} \oplus k_{3,3}$

AES Can Go Forwards or Backwards

Input: 128-bits "plaintext", 128-bit subkey
Output: 128 bits "ciphertext"

Picture as operations on a 4x4 grid of 8-bit values

1. Non-linear substitution
   - Run each byte thru a non-linear (but invertible) function (or S-box)
2. Shift rows
   - Circular-shift each row
   - i^{th} row shifted by i
3. Linear-mix columns
   - Treat each column as a 4-vector
   - Multiplication is in GF(2^8), addition is XOR
4. Key-addition
   - XOR each byte with corresponding byte of round subkey

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

Hey, What's an S-Box?

1. Non-linear substitution
   - Run each byte thru a non-linear (but invertible) function (or S-box)

"S-boxes" show up in the design of many block ciphers
- In DES, 6 bits in, 4 bits out, not invertible
- In AES, 8 bits in, 8 bits out, invertible

"S-boxes" serve several purposes
- Confusion: drastically changes data from input to output
- Non-linearity / hard to model: make it harder for a cryptanalyst to write down an equation for the S-box and solve it

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**0a**	**0b**	**0c**	**0d**	**0e**	**0f**
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Other fun properties

AES: designed to run fast in software (8-bit embedded through 64-bit)

DES: specifically designed to run slow in software

- There’s a 64-bit reordering (swap low/high bits)
- Cryptographically meaningless, but slows down any software implementation

AES: Designed by two Belgian cryptographers, open NIST competition, no secrets in its design (late 1990’s)

DES: Designed by IBM (with “help” from NSA), meant for commercial uses (1970’s)

- NSA genuinely helped (made DES resistant to differential cryptanalysis)
- Academics were worried about hidden weaknesses in DES (mysterious S-Boxes were mysterious)
- When differential cryptanalysis was discovered by Biham and Shamir, DES was already resistant to it!

Q: How to encrypt arbitrary length messages with a block cipher?

A: Padding / Block-Cipher Modes

Padding

Can only encrypt in ints of cipher block size, but message might not be multiples of block size.

<u>Solution</u>: Add padding to end of message

Must be able to recognize and remove padding after decryption

<u>Common Approach</u>: Add **n** bytes that have value **n**

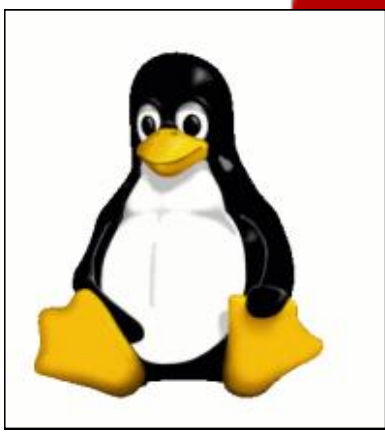
Cipher Modes

How do we handle long, multi-block messages?

Encrypted Codebook (ECB) Mode

Just encrypt each block independently

$C_i := E_k(P_i)$



Plaintext

ECB

Cipher-block Chaining (CBC)

For each block $\boldsymbol{P_i}$, do:

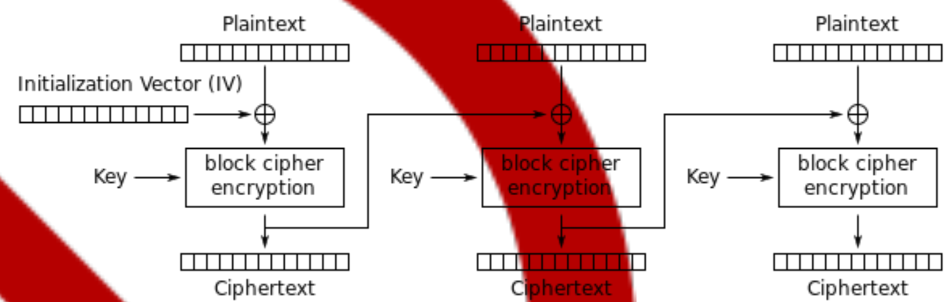
$\boldsymbol{C_0}$ **:=** *initialization vector*

$\boldsymbol{C_i := E_k(P_i \oplus C_{i-1})}$

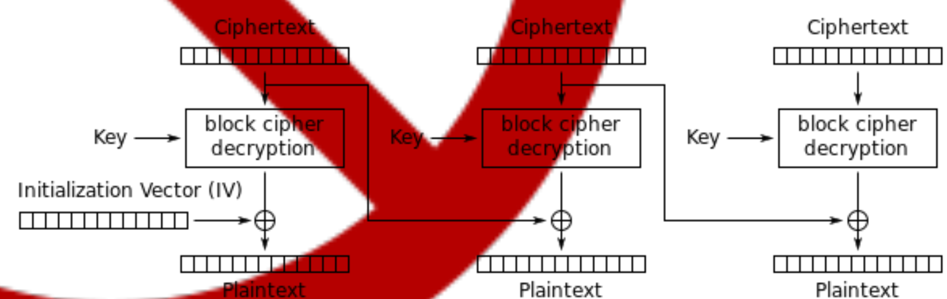
To decrypt $\boldsymbol{C_i}$, do:

$\boldsymbol{C_0}$ **:=** *initialization vector*

$\boldsymbol{P_i := D_k(C_i) \oplus C_{i-1}}$



Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

Counter Mode (CTR)

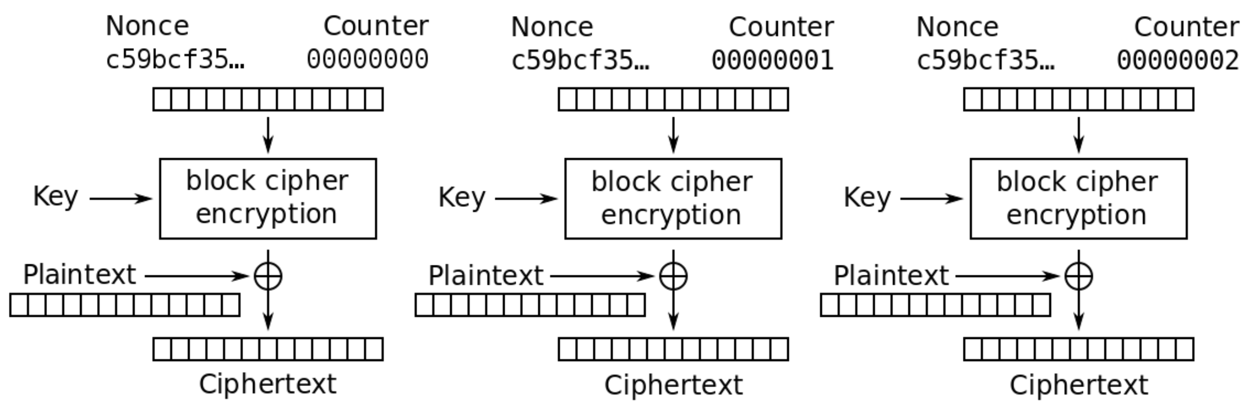
Uses block cipher as a pseudorandom generator

XOR $\boldsymbol{i^{th}}$ block of message with $\boldsymbol{E_K(}$*message_id* || *ctr*$\boldsymbol{)}$

Effectively a stream cipher

Nice side effect: blocks can be decrypted independently

- Useful, e.g., if you're reading an encrypted file on disk



Counter (CTR) mode encryption

Building a Secure Channel

What if you want confidentiality and integrity at the **same time**?

- Encrypt, *then* MAC. Better yet, "authenticated encryption with associated data" (AEAD) *[future lecture]*
    - *Fun reading:* https://blog.cryptographyengineering.com/2012/05/19/how-to-choose-authenticated-encryption/
- Need two (or more) shared keys, but only have one? That's what PRG's are for!
- If there's a reverse channel (Bob to Alice), use *separate keys* for that!

Encryption / Integrity Ordering

Encrypt, then MAC

Encrypt, then MAC

Encrypt, then MAC

Cryptographic Doom Principle: If you have to perform *any* cryptographic operation before verifying the MAC on a message you've received, it will inevitably lead to doom.

Padding Oracles

Distinguish between invalid MAC and invalid padding

Enough to learn plaintext

Vaudenay padding oracle attack

https://en.wikipedia.org/wiki/Padding_oracle_attack

Cipher-block Chaining (CBC)

For each block $\boldsymbol{P_i}$, do:

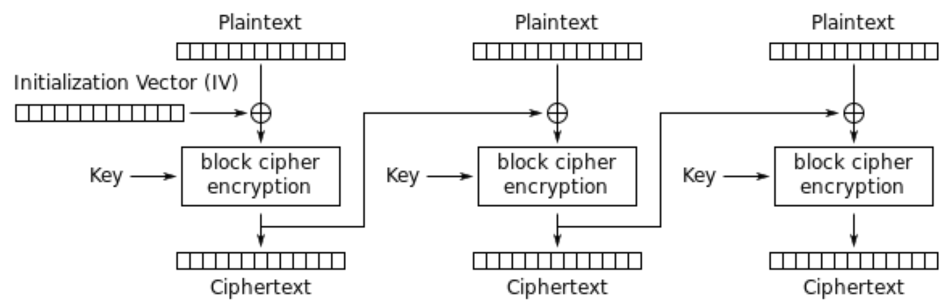
$\boldsymbol{C_0}$ **:=** *initialization vector*

$\boldsymbol{C_i := E_k(P_i \oplus C_{i-1})}$

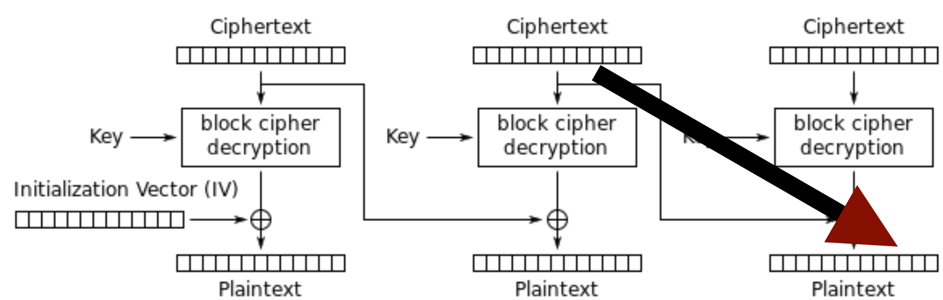
To decrypt $\boldsymbol{C_i}$, do:

$\boldsymbol{C_0}$ **:=** *initialization vector*

$\boldsymbol{P_i := D_k(C_i) \oplus C_{i-1}}$



Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

Manipulate ciphertext, see if plaintext is accepted or rejected by server
("padding oracle attack")

PKCS#7 and padding attacks

If you have one byte left, it's 0x01

Two bytes left? 0x02 0x02, etc.

If the attacker can *set* the last byte (after decryption) to 0x01, then the server "accepts" the message.

256 queries per byte, read out all the non-padding plaintext as well

Padding Oracle Defenses

Don’t use separate errors for MAC vs padding

Always check authentication

Constant-time code, all code paths must be equal

Limit branching

Don’t use CBC mode!

AEAD

Authenticated Encryption and Associated Data

```
ciphertext, auth_tag := Seal(key, plaintext, associated_data)

plaintext := Unseal(key, ciphertext, associated_data, tag)
```

Combine integrity and encryption into a single primitive.

Commonly used is AES-GCM (“Galois Counter Mode”), has hardware support on modern Intel processors.

ChaCha-Poly1305, Salsa20-Poly1305, common on mobile devices.

Key Size

How big should keys be?

<u>Moore's Law</u>: Computer's get twice as good for the same price every 18 months.

Current reasonable safe size: *128 bits*

Worried about quantum computers? *256 bits*

MACs/PRFs need to be 2x cipher key size [*why?*]

So Far

Assuming Alice and Bob share a key

Next Week…

How to derive shared keys securely and in <u>public</u>!

How to authenticate secure channels