CSc 466/566

Computer Security

5 : Cryptography — Basics

Version: 2013/02/13 16:54:59

Department of Computer Science
University of Arizona

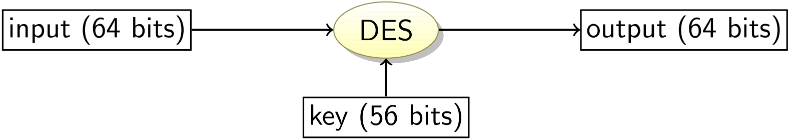
collberg@gmail.com

Christian Collberg

Outline

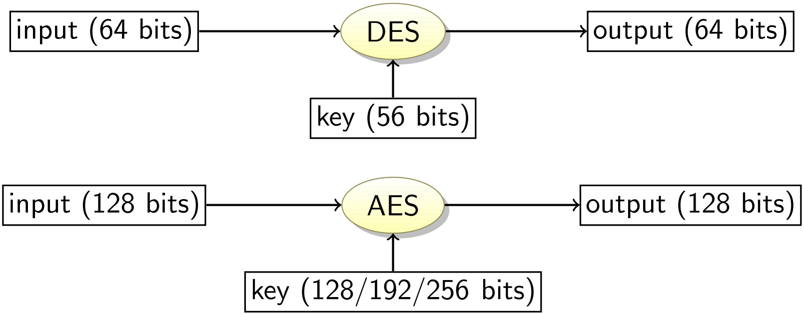
Block Ciphers

- Block ciphers work on one block of data at a time. Different ciphers use different block size and key length:



Block Ciphers

- Block ciphers work on one block of data at a time. Different ciphers use different block size and key length:



Misuses of Cryptosystem

- Cryptographic systems are sensitive to the environment.
- The strength of a cryptosystem depends on how it is used.
- Just because a cryptosystem is mathematically strong doesn't mean it's secure – it can be vulnerable to various attacks when used incorrectly.
- Attacks can be carried out in many ways besides guessing the key.

Misuses of Cryptosystem. . .

Precomputing the Possible Message: If the plaintexts is drawn from a small set, attacker can just encipher all the plaintexts using the public key and search the intercepted ciphertext in database to find the corresponding plaintext (dictionary attack).

Misuses of Cryptosystem...

Precomputing the Possible Message: If the plaintexts is drawn from a small set, attacker can just encipher all the plaintexts using the public key and search the intercepted ciphertext in database to find the corresponding plaintext (dictionary attack).

Misordered Blocks: If different parts of ciphertext are not bound together, the attacker can delete, replay and reorder the ciphertext without being detected.

Misuses of Cryptosystem...

Precomputing the Possible Message: If the plaintexts is drawn from a small set, attacker can just encipher all the plaintexts using the public key and search the intercepted ciphertext in database to find the corresponding plaintext (dictionary attack).

Misordered Blocks: If different parts of ciphertext are not bound together, the attacker can delete, replay and reorder the ciphertext without being detected.

Statistical Regularities: If each part of a message is enciphered separately the ciphertext can give away information about the structure of the message, even if the message itself is unintelligible.

Block Cipher: Performance Criteria

- Key size – decides the upper bound of security using exhaustive search.
- Block size – larger block is harder to crack but more costly to implement.
- Complexity of cryptographic mapping – affect the implementation cost and real-time performance
- Data expansion – it is desirable not to increase the size of the data.

Outline

Block Cipher: Modes

- Modes of operation deal with how to encrypt a message of arbitrary length using a block cipher.
- To be useful, a mode must be at least as secure and as efficient as the underlying cipher.
- The most common modes for block ciphers are:
    1. Electronic Code Book (ECB)
    2. Cipher Block Chaining (CBC)
    3. Cipher Feedback (CFB)
    4. Output Feedback(OFB)
    5. Counter (CTR)

ECB Mode

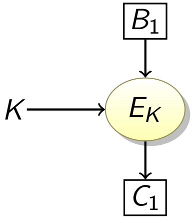
- Electronic Codebook
- In ECB mode, each plaintext block is encrypted independently with the block cipher.
- Encryption:

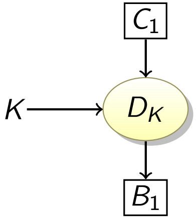
$$C_i \leftarrow E_K(B_i)$$

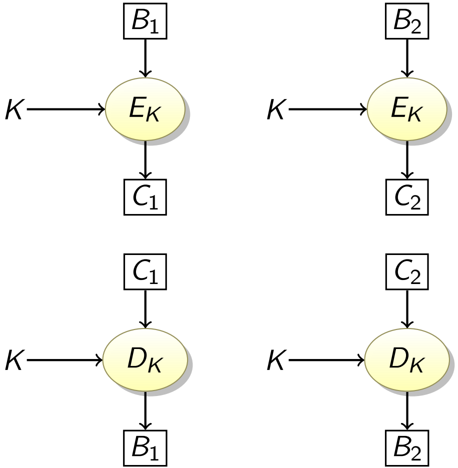
- Decryption:

$$B_i \leftarrow D_K(C_i)$$

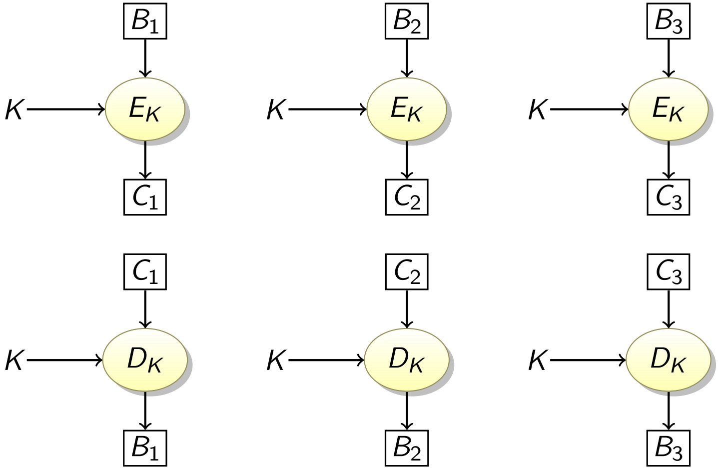
- Notation:
  - B_i is the i:th plaintext block.
  - C_i is the i:th ciphertext block.


B1
K
EK
C1


C1
K
DK
B1



B_1
K
E_K
C_1
B_2
K
E_K
C_2
C_1
K
D_K
B_1
C_2
K
D_K
B_2



B1
B2
B3
K
EK
C1
C2
C3
DK

ECB Mode: Analysis

- Pros:
  - Simple.
  - Tolerates blocks lost in transit.
  - Easy to parallelize.
- Cons:
  - Identical plaintext blocks (eg. blocks of sky in a jpg) result in identical ciphertext $\Rightarrow$ data patterns aren't hidden.
- Not suitable for encrypting message longer than one block.
- Example (en.wikipedia.org/wiki/Block_cipher_modes_of_operation):

  > *the* Phantasy Star Online: Blue Burst *online video game uses Blowfish in ECB mode. Before the key exchange system was cracked leading to even easier methods, cheaters repeated encrypted* **monster killed** *message packets, each an encrypted Blowfish block, to illegitimately gain experience points quickly.[citation needed]*

Message Padding

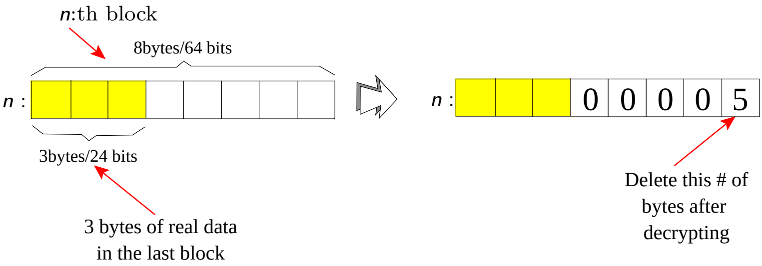
- What happens if the last plaintext block is not completely full?

Message Padding

- What happens if the last plaintext block is not completely full?
- The message must be padded to a multiple of the cipher block size.

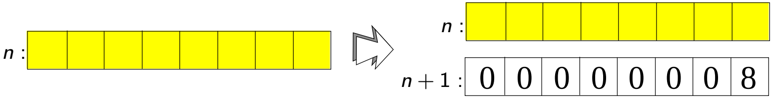
Message Padding

- What happens if the last plaintext block is not completely full?
- The message must be padded to a multiple of the cipher block size.
- One way to do this is to pad with 0:s and make the last byte be the number of bytes to remove from the last block:



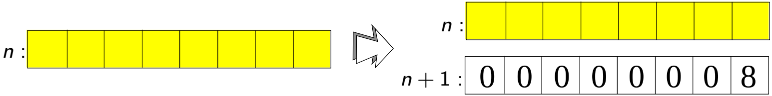
Message Padding. . .

- With this method you *have to* pad every message, even if it ends on a block boundary:



Message Padding. . .

- With this method you *have to* pad every message, even if it ends on a block boundary:



- Another method called ciphertext stealing doesn't add any extra blocks.

CBC Mode

- Cipher-Block Chaining
- In CBC mode, each plaintext block is XORed with the previous ciphertext block and then encrypted. An initialization vector IV is used as a seed for encrypting the first block.
- Initialization:

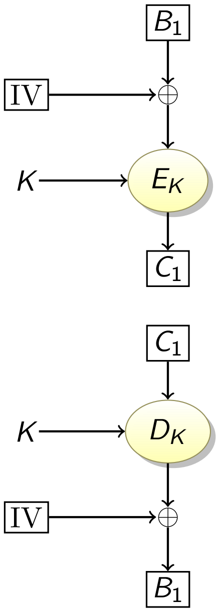
$$C_0 \leftarrow \mathrm{IV}$$

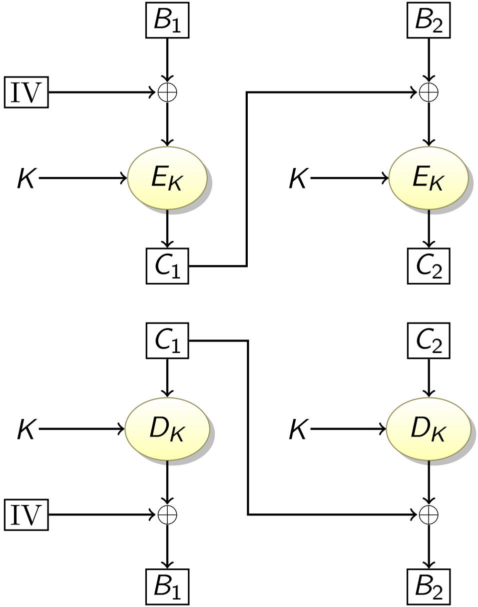
- Encryption:

$$C_i \leftarrow E_K(B_i \oplus C_{i-1})$$

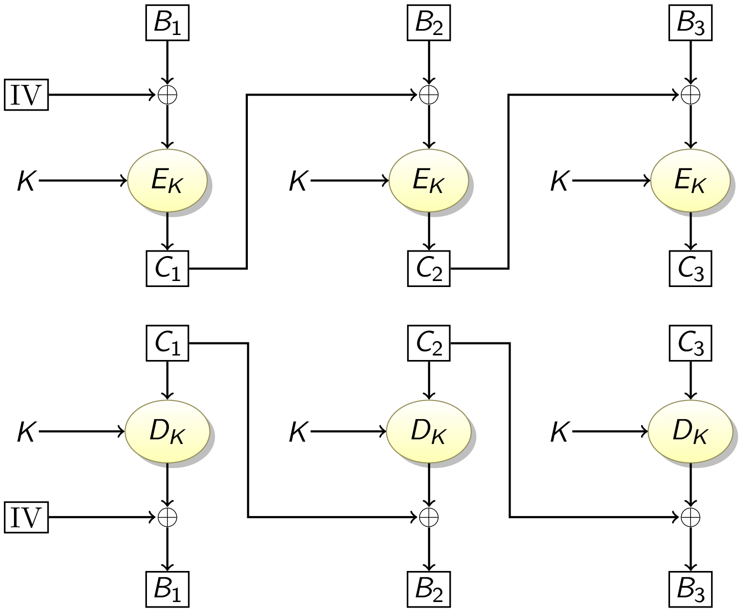
- Decryption:

$$B_i \leftarrow D_K(C_i) \oplus C_{i-1}$$


B_1
IV
K
E_K
C_1
C_1
K
D_K
IV
B_1



B1
B2
IV
K
EK
C1
C2
C1
C2
K
DK
IV
B1
B2



B_1
B_2
B_3
IV
K
E_K
K
E_K
K
E_K
C_1
C_2
C_3
C_1
C_2
C_3
K
D_K
K
D_K
K
D_K
IV
B_1
B_2
B_3

CBC Mode: Analysis

- Pros:
  - Identical plaintext blocks will yield different ciphertext blocks.
  - Decryption can be parallelized if all ciphertext blocks are available.
  - If block C_i is lost, C_{i+1} can't be decrypted, but C_{i+2} can.
- Cons:
  - Encryption can't be parallelized.
- Most commonly used mode of operation.
- A one-bit change in a plaintext or IV affects all following ciphertext blocks.

CFB Mode

- Cipher-FeedBack
- In CFB mode, the previous ciphertext block is encrypted and the output produced is combined with the plaintext block using XOR to produce the current ciphertext block.
- CFB can use feedback that is less than one full data block.
- An initialization vector IV is used as a seed for the first block.
- Initialization:

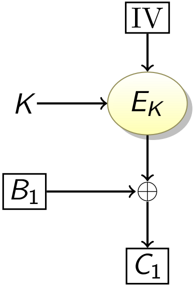
$$C_0 \leftarrow \mathrm{IV}$$

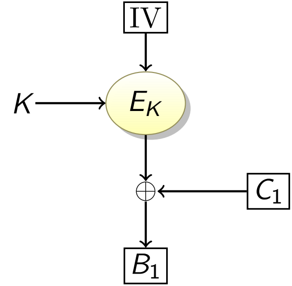
- Encryption:

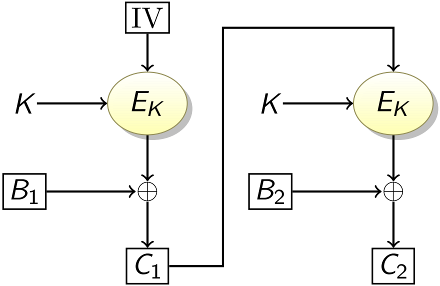
$$C_i \leftarrow E_K(C_{i-1}) \oplus B_i$$

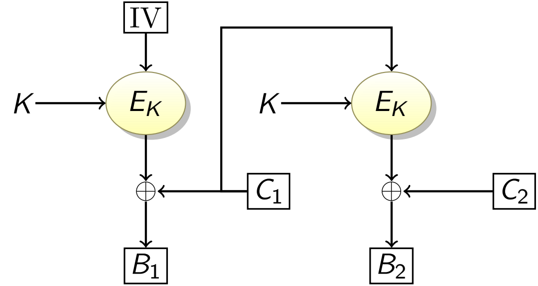
- Decryption:

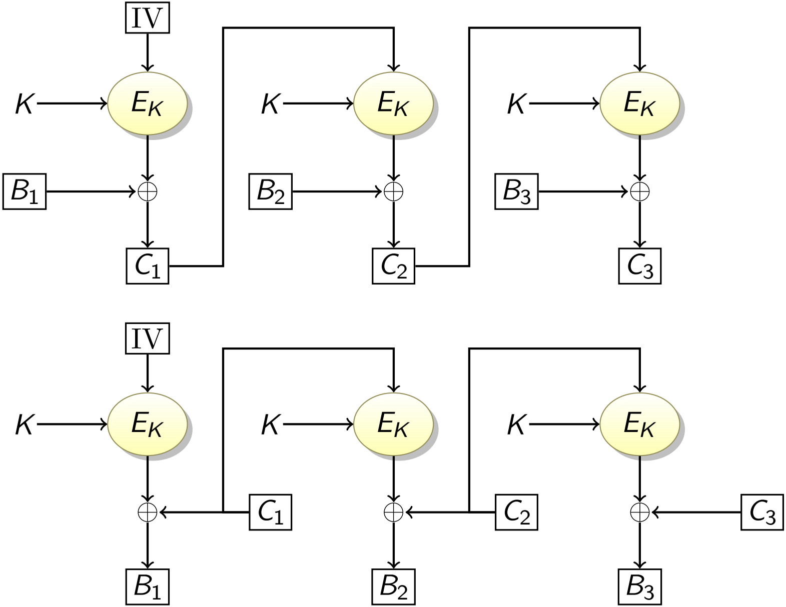
$$B_i \leftarrow E_K(C_{i-1}) \oplus C_i$$


IV
K
E_K
B_1
C_1


IV
K
E_K
C_1
B_1


IV
K
E_K
K
E_K
B_1
B_2
C_1
C_2


IV
K
E_K
K
E_K
C_1
C_2
B_1
B_2


IV
K
E_K
B_1
C_1
B_2
C_2
B_3
C_3

CFB Mode: Analysis

- Pros:
  - CFB mode is self-synchronizing similar to CBC.
  - Decryption can be parallelized.
  - Decryptor is never used.
- Cons:
  - Encryption cannot be parallelized.
  - When decrypting, a one-bit change in the ciphertext corrupts the following 2 plaintext blocks.
  - When decrypting, a one-bit change in the plaintext block, corrupts 1 following plaintext block.

OFB Mode

- Output-FeedBack Mode
- OFB mode is similar to CFB mode except that the quantity XORed with each plaintext block are vectors generated independently of both the plaintext and ciphertext.
- Stream cipher
- Initialization:

$$V_0 \leftarrow \mathrm{IV}$$

- Create vectors:

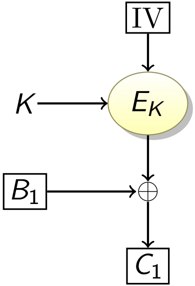
$$V_i \leftarrow E_K(V_{i-1});$$

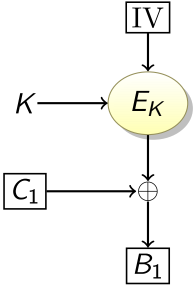
- Encryption:

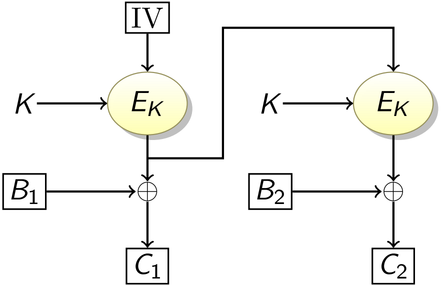
$$C_i \leftarrow V_i \oplus B_i;$$

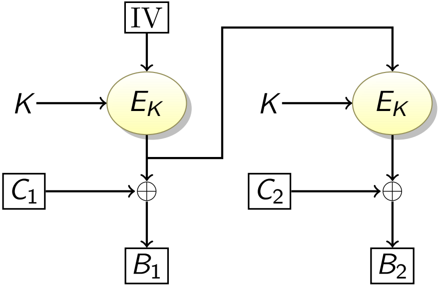
- Decryption:

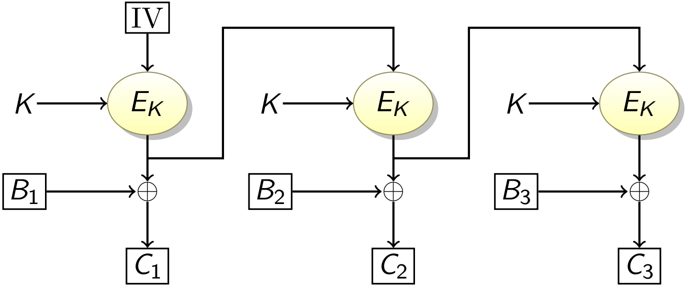
$$B_i \leftarrow V_i \oplus C_i;$$

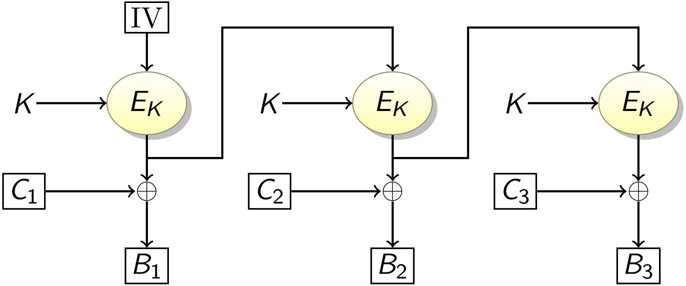

IV
K
E_K
B_1
C_1


IV
K
E_K
C_1
B_1


IV
K
E_K
K
E_K
B_1
B_2
C_1
C_2


IV
K
E_K
K
E_K
C_1
C_2
B_1
B_2


IV
K
E_K
K
E_K
K
E_K
B_1
B_2
B_3
C_1
C_2
C_3


IV
K
E_K
K
E_K
K
E_K
C_1
C_2
C_3
B_1
B_2
B_3

OFB Mode: Analysis

- Pros:
  - Encryption and decryption can be done in parallel if the vectors have been precomputed.
  - If i:th ciphertext bit is flipped, the i:th plaintext bit is also flipped. This property helps with many error correcting codes.
- The keystream is plaintext independent.

CTR Mode

- Counter Mode
- CTR mode is similar to OFB: encryption is performed by XORing with a pad.
- Vectors are generated by encrypting $\mathrm{seed}+0, \mathrm{seed}+1, \mathrm{seed}+1, \ldots$ given a random seed.
- Create vectors:

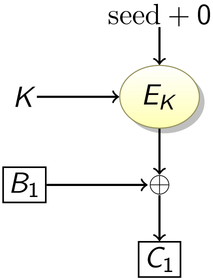
$$V_i \leftarrow E_K(\mathrm{seed}+i-1);$$

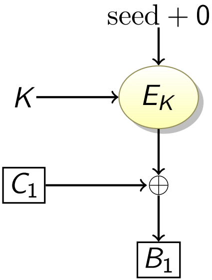
- Encryption:

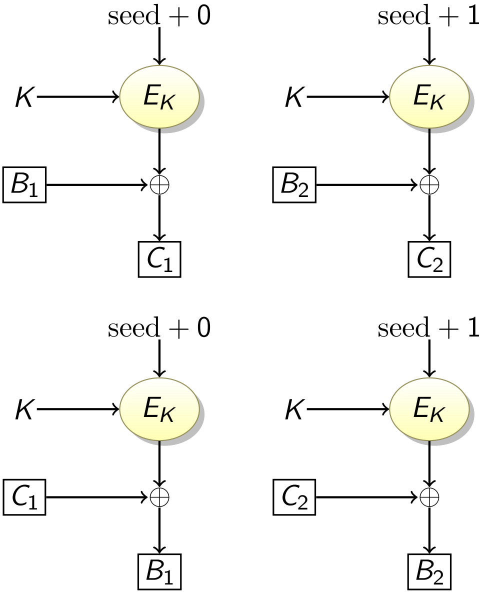
$$C_i \leftarrow V_i \oplus B_i;$$

- Decryption:

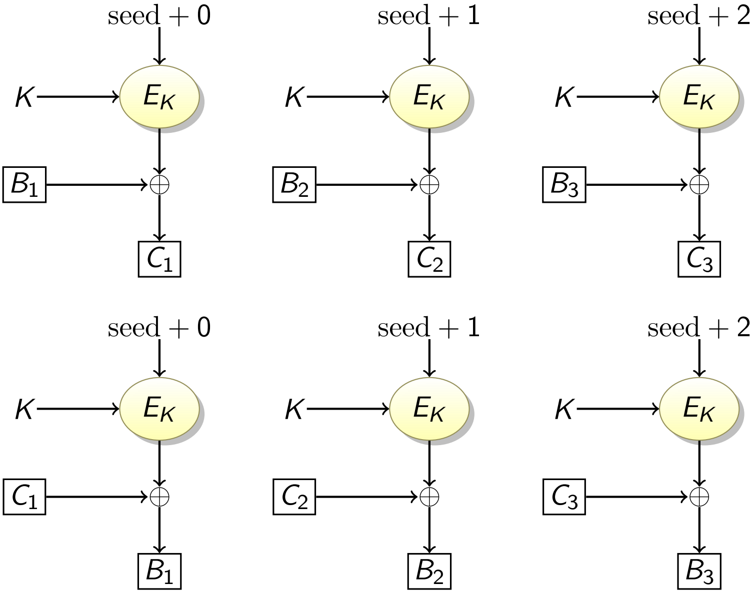
$$B_i \leftarrow V_i \oplus C_i;$$


seed + 0
K
E_K
B_1
C_1


seed + 0
K
E_K
C_1
B_1



seed + 0
seed + 1
K
E_K
K
E_K
B_1
B_2
C_1
C_2
seed + 0
seed + 1
K
E_K
K
E_K
C_1
C_2
B_1
B_2



seed + 0
seed + 1
seed + 2
K
E_K
B_1
B_2
B_3
C_1
C_2
C_3

CTR Mode: Analysis

- Pros:
  - Vector generation, encryption, decryption can be all be done in parallel.
  - We can recover from dropped blocks.
- Cons:
  - There are attacks (Hardware Fault Attack) that are based on the use of simple counter function.

Outline

Attacks on Block Ciphers

Differential cryptanalysis: By careful analysis of the ciphertext of two related plaintexts encrypted under the same key, probabilities can be assigned to each of the possible keys, and eventually the most probable key is identified as the correct one.

Linear cryptanalysis: Use a linear approximation to describe the behavior of the block cipher. Given sufficient pairs of plaintext and corresponding ciphertext, bits of information about the key can be obtained.

Weak keys: Weak keys are secret keys with a certain value for which the block cipher in question will exhibit certain regularities in encryption or, in other cases, a poor level of encryption. For instance, with DES there are four keys for which encryption is exactly the same as decryption.

Outline

Modular Arithmetic

- Block ciphers operate on blocks as large numbers.
- We can't deal with overflow: the output has to fit in the same size block as the input.
- We therefore perform arithmetic modulo n.
- After each arithmetic operation return the remainder after dividing by n.
- We're performing arithmetic in Z_n:

$$Z_n = \{0, 1, 2, \ldots, n - 1\}$$

Modular Arithmetic

- Addition, subtraction, multiplication are done by reducing the result to values in Z_n:

$$
\begin{aligned}
(a+b) \bmod n &= ((a \bmod n) + (b \bmod n)) \bmod n \\
(a-b) \bmod n &= ((a \bmod n) - (b \bmod n)) \bmod n \\
(a*b) \bmod n &= ((a \bmod n) * (b \bmod n)) \bmod n
\end{aligned}
$$

$$
\begin{aligned}
23 &\equiv 11 \bmod 12 \\
23 &\equiv 2 \bmod 7 \\
(10+13) \bmod 7 &= ((10 \bmod 7) + (13 \bmod 7)) \bmod 7 \\
&= (3+6) \bmod 7 = 2
\end{aligned}
$$

Modular Arithmetic: Addition

- Addition table for Z_{10}, $(x + y)$ mod 10.

$+$	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	0
2	2	3	4	5	6	7	8	9	0	1
3	3	4	5	6	7	8	9	0	1	2
4	4	5	6	7	8	9	0	1	2	3
5	5	6	7	8	9	0	1	2	3	4
6	6	7	8	9	0	1	2	3	4	5
7	7	8	9	0	1	2	3	4	5	6
8	8	9	0	1	2	3	4	5	6	7
9	9	0	1	2	3	4	5	6	7	8

Modular Inverses

- y is the modular inverse of x, modulo n, if

$$xy \bmod n = 1$$

- Not every number in Z_n has an inverse.
- If n is prime then every number in Z_n has an inverse.
- Examples:
  1. $4 \cdot 3 \bmod 11 = 12 \bmod 11 = 1 \Rightarrow 4$ is the inverse of 3 in Z_{11}.

Modular Inverses...

- The inverse of 4 is $\frac{1}{4}$. Modular inverses are harder.
- To find the the inverse of 4 modulo 7 we want to compute:

$$4 * x = 1 \bmod 7$$

which is the same as finding integers x and k such that

$$4x = 7k + 1$$

- This is also written: $4^{-1} = x \bmod n$.
- Sometimes inverses exist, sometimes not:

$$\begin{aligned} 5^{-1} &= 3 \bmod 14 \\ 2^{-1} &= ? \bmod 14 \end{aligned}$$

Modular Inverses

- Multiplication table for Z_{10}, xy mod 10.
- Elements that have a modular inverse have been highlighted.

×	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9
2	0	2	4	6	8	0	2	4	6	8
3	0	3	6	9	2	5	8	1	4	7
4	0	4	8	2	6	0	4	8	2	6
5	0	5	0	5	0	5	0	5	0	5
6	0	6	2	8	4	0	6	2	8	4
7	0	7	4	1	8	5	2	9	6	3
8	0	8	6	4	2	0	8	6	4	2
9	0	9	8	7	6	5	4	3	2	1

Modular Inverses

- Multiplication table for Z_{11}, $xy \bmod 11$.

$\times$	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	10
2	0	2	4	6	8	10	1	3	5	7	9
3	0	3	6	9	1	4	7	10	2	5	8
4	0	4	8	1	5	9	2	6	10	3	7
5	0	5	10	4	9	3	8	2	7	1	6
6	0	6	1	7	2	8	3	9	4	10	5
7	0	7	3	10	6	2	9	5	1	8	4
8	0	8	5	2	10	7	4	1	9	6	3
9	0	9	7	5	3	1	10	8	6	4	2
10	0	10	9	8	7	6	5	4	3	2	1

In-Class Exercise

- Create the modular multiplication table for Z_5, xy mod 5.

Modular Exponentiation

- Modular exponentiation is an important operation in cryptography:

$$x^y \bmod n = \overbrace{x * x * \cdots * x}^{y} \bmod n$$

Modular Exponentiation...

- For which x and n do there exist modular powers equal to 1?

$$x^y \bmod n \stackrel{?}{=} 1$$

  - If n is prime then every non-zero element of Z_n has a power $=$ 1.
  - If n is not prime, only x for which $\mathrm{GCD}(x, n) = 1$ (x and n are relatively prime) have a power $= 1$.
- Example: For Z_{13}^*

$$\begin{aligned} 1^1 \bmod 13 &= 1 \\ 2^{12} \bmod 13 &= 1 \\ 3^3 \bmod 13 &= 1 \\ 4^6 \bmod 13 &= 1 \\ 5^4 \bmod 13 &= 1 \\ 6^{12} \bmod 13 &= 1 \end{aligned}$$

Modular Exponentiation...

- Z_n^* is the subset of Z_n of elements relatively prime with n:

$$Z_n^* = \{x \in Z_n \text{ such that } \mathrm{GCD}(x, n) = 1\}$$

- Examples:
  1. $Z_{10}^* = \{1, 3, 7, 9\}$

Modular Exponentiation...

- Z_n^* is the subset of Z_n of elements relatively prime with n:

$$Z_n^* = \{x \in Z_n \text{ such that } \mathrm{GCD}(x, n) = 1\}$$

- Examples:
  1. $Z_{10}^* = \{1, 3, 7, 9\}$
  2. $Z_{13}^* = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$

Modular Exponentiation...

- Z_n^* is the subset of Z_n of elements relatively prime with n:

$$Z_n^* = \{x \in Z_n \text{ such that } \mathrm{GCD}(x, n) = 1\}$$

- Examples:
  1. $Z_{10}^* = \{1, 3, 7, 9\}$
  2. $Z_{13}^* = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$

Modular Exponentiation...

- Z_n^* is the subset of Z_n of elements relatively prime with n:

$$Z_n^* = \{x \in Z_n \text{ such that } \mathrm{GCD}(x, n) = 1\}$$

- Examples:
  1. $Z_{10}^* = \{1, 3, 7, 9\}$
  2. $Z_{13}^* = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$
- In general, $Z_n^* = \{1, 2, \ldots, n-1\}$ if n is prime

Modular Exponentiation...

- Modular exponentiation table for Z_{10}, $x^y \bmod 10$.
- Elements in Z_n that have some power equal to 1 have been highlighted.

	y								
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
$\mathbf{1}^y$	1	1	1	1	1	1	1	1	1
$\mathbf{2}^y$	2	4	8	6	2	4	8	6	2
$\mathbf{3}^y$	3	9	7	1	3	9	7	1	3
$\mathbf{4}^y$	4	6	4	6	4	6	4	6	4
$\mathbf{5}^y$	5	5	5	5	5	5	5	5	5
$\mathbf{6}^y$	6	6	6	6	6	6	6	6	6
$\mathbf{7}^y$	7	9	3	1	7	9	3	1	7
$\mathbf{8}^y$	8	4	2	6	8	4	2	6	8
$\mathbf{9}^y$	9	1	9	1	9	1	9	1	9

Modular Exponentiation: Z_{13}, $x^y \bmod 13$

	y											
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
$\mathbf{1}^y$	1	1	1	1	1	1	1	1	1	1	1	1
$\mathbf{2}^y$	2	4	8	3	6	12	11	9	5	10	7	1
$\mathbf{3}^y$	3	9	1	3	9	1	3	9	1	3	9	1
$\mathbf{4}^y$	4	3	12	9	10	1	4	3	12	9	10	1
$\mathbf{5}^y$	5	12	8	1	5	12	8	1	5	12	8	1
$\mathbf{6}^y$	6	10	8	9	2	12	7	3	5	4	11	1
$\mathbf{7}^y$	7	10	5	9	11	12	6	3	8	4	2	1
$\mathbf{8}^y$	8	12	5	1	8	12	5	1	8	12	5	1
$\mathbf{9}^y$	9	3	1	9	3	1	9	3	1	9	3	1
$\mathbf{10}^y$	10	9	12	3	4	1	10	9	12	3	4	1
$\mathbf{11}^y$	11	4	5	3	7	12	2	9	8	10	6	1
$\mathbf{12}^y$	12	1	12	1	12	1	12	1	12	1	12	1

In-Class Exercise: Modular Exponentiation

- Create the modular exponentiation table for Z_5, x^y mod 5.

Outline

Euler's Totient Function

- $\phi(n)$ is the totient of n, the number of elements of Z_n^*:

$$\phi(n) = |Z_n^*|$$

- Examples:
  1. $Z_{10}^* = \{1, 3, 7, 9\} \Rightarrow \phi(10) = 4$

Euler's Totient Function

- $\phi(n)$ is the totient of n, the number of elements of Z_n^*:

$$\phi(n) = |Z_n^*|$$

- Examples:
  1. $Z_{10}^* = \{1, 3, 7, 9\} \Rightarrow \phi(10) = 4$
  2. $Z_{13}^* = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\} \Rightarrow \phi(13) = 12$

Euler's Totient Function

- $\phi(n)$ is the totient of n, the number of elements of Z_n^*:

$$\phi(n) = |Z_n^*|$$

- Examples:
  1. $Z_{10}^* = \{1, 3, 7, 9\} \Rightarrow \phi(10) = 4$
  2. $Z_{13}^* = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\} \Rightarrow \phi(13) = 12$

Euler's Totient Function

- $\phi(n)$ is the totient of n, the number of elements of Z_n^*:

$$\phi(n) = |Z_n^*|$$

- Examples:
  1. $Z_{10}^* = \{1,3,7,9\} \Rightarrow \phi(10) = 4$
  2. $Z_{13}^* = \{1,2,3,4,5,6,7,8,9,10,11,12\} \Rightarrow \phi(13) = 12$
- In general, if n is prime,
  $Z_n^* = \{1,2,\ldots,n-1\} \Rightarrow \phi(n) = n-1$.

Euler's Totient Function Values

n	$\phi(n)$	List of Divisors
1	1	1
2	1	1, 2
3	2	1, 3
4	2	1, 2, 4
5	4	1, 5
6	2	1, 2, 3, 6
7	6	1, 7
8	4	1, 2, 4, 8
9	6	1, 3, 9
10	4	1, 2, 5, 10
11	10	1, 11
12	4	1, 2, 3, 4, 6, 12
13	12	1, 13
14	6	1, 2, 7, 14
15	8	1, 3, 5, 15
16	8	1, 2, 4, 8, 16
17	16	1, 17
18	6	1, 2, 3, 6, 9, 18

n	$\phi(n)$	List of Divisors
19	18	1, 19
20	8	1, 2, 4, 5, 10, 20
21	12	1, 3, 7, 21
22	10	1, 2, 11, 22
23	22	1, 23
24	8	1, 2, 3, 4, 6, 8, 12, 24
25	20	1, 5, 25
26	12	1, 2, 13, 26
27	18	1, 3, 9, 27
28	12	1, 2, 4, 7, 14, 28
29	28	1, 29
30	8	1, 2, 3, 5, 6, 10, 15, 30
31	30	1, 31
32	16	1, 2, 4, 8, 16, 32
33	20	1, 3, 11, 33
34	16	1, 2, 17, 34
35	24	1, 5, 7, 35
36	12	1, 2, 3, 4, 6, 9, 12, 18, 36

Euler's Totient Function...

- You can calculate $\phi(n)$ as

$$\phi(n) = n(1 - \frac{1}{p_1}) \cdots (1 - \frac{1}{p_m})$$

  where $p_1, \ldots, p_m$ are the the prime factors of n.
- Example:
  1. $\phi(35) = 35(1 - \frac{1}{5})(1 - \frac{1}{7}) = 35 \cdot \frac{4}{5} \cdot \frac{6}{7} = 24$

In-Class Exercise

1. What's $\phi(37)$?

In-Class Exercise

1. What's $\phi(37)$?
2. What's $\phi(38)$?

Euler's Theorem

- $\phi(n)$ is the number of positive integers relatively prime with n.
- If p is prime, $\phi(p) = p - 1$.
- If $n = pq$ is the product of two primes p and q, then $\phi(n) = (p-1)(q-1)$.

Theorem (Euler)

Let x be any positive integer that's relatively prime to the integer $n > 0$, then

$$x^{\phi(n)} \bmod n = 1$$

Euler's Theorem...

- Euler's theorem holds for each element x of Z_n^*:

$$x^{\phi(n)} \bmod n = 1$$

- Examples:
  1. $7^{\phi(10)} \bmod 10 \equiv 7^4 \bmod 10 = 1$ since $\mathrm{GCD}(7, 10) = 1$ and $7 \in Z_{10}^*$:

$$7^4 \bmod 10 \equiv 2401 \bmod 10 = 1$$

Euler's Theorem...

Theorem (Corollary to Euler's theorem)

Let x be any positive integer that's relatively prime to the integer $n > 0$, and let k be any positive integer, then

$$x^k \bmod n = x^{k \bmod \phi(n)} \bmod n$$

- Euler's theorem allows us to reduce the exponent modulo $\phi(n)$:

$$x^y \bmod n = x^{y \bmod \phi(n)} \bmod n$$

- Examples:
    1. $7^{27} \bmod 13 \equiv 7^{27 \bmod \phi(13)} \bmod 13 \equiv 7^{27 \bmod 12} \bmod 13 \equiv 7^3 \bmod 13 = 5$

In-Class Exercise: Goodrich & Tamassia R-8.17

1. What's $\phi(143)$?

In-Class Exercise: Goodrich & Tamassia R-8.17

1. What's $\phi(143)$?
2. What's $7^{120} \bmod 143$?

In-Class Exercise: Goodrich & Tamassia C-8.8

1. What are the prime factors of 10403?

In-Class Exercise: Goodrich & Tamassia C-8.8

1. What are the prime factors of 10403?
2. What's $\phi(10403)$?

In-Class Exercise: Goodrich & Tamassia C-8.8

1. What are the prime factors of 10403?
2. What's $\phi(10403)$?
3. Use Euler's theorem to compute 20^{10203} mod 10403.

Euler's Theorem...

Theorem (Corollary to Euler's theorem)

Given two prime numbers p and q, integers $n = pq$ and $0 < m < n$, and an arbitrary integer k, then

$$m^{k\phi(n)+1} \bmod n = m^{k(p-1)(q-1)+1} \bmod n = m \bmod n$$

- This relationship will be useful in the proof of correctness of the RSA algorithm.

Fermat's Little Theorem

Theorem (Fermat's Little)

Let p be a prime number and g any positive integer $g < p$, then

$$g^{p-1} \bmod p = 1$$

- Euler's theorem is a generalization of Fermat's little theorem.
- Examples:
  1. $10^{13-1} \bmod 13 = 10^{12} \bmod 13 = 1$

Outline

Euler's GCD Algorithm

- $\mathrm{GCD}(a, b)$ is the largest number d that divides a and b evenly.

Euler's GCD Algorithm

- $\mathrm{GCD}(a, b)$ is the largest number d that divides a and b evenly.
- Euler's algorithm $\mathrm{GCD}(a, b)$ returns a triple (d, i, j).

Euler's GCD Algorithm

- $\mathrm{GCD}(a, b)$ is the largest number d that divides a and b evenly.
- Euler's algorithm $\mathrm{GCD}(a, b)$ returns a triple (d, i, j).
- Based on the observation that if x divided a and b, it also divides $a - b$. We need to find the largest such x.

Euler's GCD Algorithm

- $\mathrm{GCD}(a, b)$ is the largest number d that divides a and b evenly.
- Euler's algorithm $\mathrm{GCD}(a, b)$ returns a triple (d, i, j).
- Based on the observation that if x divided a and b, it also divides $a - b$. We need to find the largest such x.
- Key observation: If

$$d = \mathrm{GCD}(a, b) \text{ and } b > 0$$

then

$$d = \mathrm{GCD}(b, a \bmod b)$$

Euler's GCD Algorithm...

```
function gcd(int a, int b) : (int,int,int) =
    if b = 0 then
        return (a,1,0)
    q ← ⌊a/b⌋
    (d,k,l) ← gcd(b, a mod b)
    return (d,l,k − lq)
```

Euler's GCD Algorithm...

- Example:

$$\mathrm{GCD}(546, 198) = \mathrm{GCD}(198, 546 \bmod 198) = \mathrm{GCD}(198, 150)$$

Euler's GCD Algorithm. . .

- Example:

$$\begin{aligned} \mathrm{GCD}(546, 198) &= \mathrm{GCD}(198, 546 \bmod 198) = \mathrm{GCD}(198, 150) \\ &= \mathrm{GCD}(150, 198 \bmod 150) = \mathrm{GCD}(150, 48) \end{aligned}$$

Euler's GCD Algorithm...

- Example:

$$\begin{aligned}\mathrm{GCD}(546, 198) &= \mathrm{GCD}(198, 546 \bmod 198) = \mathrm{GCD}(198, 150)\\ &= \mathrm{GCD}(150, 198 \bmod 150) = \mathrm{GCD}(150, 48)\\ &= \mathrm{GCD}(48, 150 \bmod 48) = \mathrm{GCD}(48, 6)\end{aligned}$$

Euler's GCD Algorithm...

- Example:

$$
\begin{aligned}
\mathrm{GCD}(546, 198) &= \mathrm{GCD}(198, 546 \text{ mod } 198) = \mathrm{GCD}(198, 150) \\
&= \mathrm{GCD}(150, 198 \text{ mod } 150) = \mathrm{GCD}(150, 48) \\
&= \mathrm{GCD}(48, 150 \text{ mod } 48) = \mathrm{GCD}(48, 6) \\
&= \mathrm{GCD}(6, 48 \text{ mod } 6) = \mathrm{GCD}(6, 0)
\end{aligned}
$$

Euler's GCD Algorithm...

- Example:

$$
\begin{aligned}
\mathrm{GCD}(546, 198) &= \mathrm{GCD}(198, 546 \bmod 198) = \mathrm{GCD}(198, 150) \\
&= \mathrm{GCD}(150, 198 \bmod 150) = \mathrm{GCD}(150, 48) \\
&= \mathrm{GCD}(48, 150 \bmod 48) = \mathrm{GCD}(48, 6) \\
&= \mathrm{GCD}(6, 48 \bmod 6) = \mathrm{GCD}(6, 0) \\
&= 6
\end{aligned}
$$

Euler's GCD Algorithm. . .

- Compute GCD by hand:
  1. divide the larger one by the smaller;
  2. write an equation of the form

$$\text{larger} = \text{smaller} \times \text{quotient} + \text{remainder};$$

  3. repeat using the two numbers smaller and remainder;
  4. when you get a 0 remainder, the previous line will be the gcd of the original two numbers.

Euler's GCD Algorithm...

- Find $\mathrm{GCD}(421, 111)$.

$$421 = 111 \times 3 + 88$$

Euler's GCD Algorithm...

- Find $\text{GCD}(421, 111)$.

$$\begin{aligned} 421 &= 111 \times 3 + 88 \\ 111 &= 88 \times 1 + 23 \end{aligned}$$

Euler's GCD Algorithm...

- Find $\mathrm{GCD}(421, 111)$.

$$\begin{aligned} 421 &= 111 \times 3 + 88 \\ 111 &= 88 \times 1 + 23 \\ 88 &= 23 \times 3 + 19 \end{aligned}$$

Euler's GCD Algorithm...

- Find $\mathrm{GCD}(421, 111)$.

$$\begin{aligned} 421 &= 111 \times 3 + 88 \\ 111 &= 88 \times 1 + 23 \\ 88 &= 23 \times 3 + 19 \\ 23 &= 19 \times 1 + 4 \end{aligned}$$

Euler's GCD Algorithm...

- Find $\mathrm{GCD}(421, 111)$.

$$
\begin{aligned}
421 &= 111 \times 3 + 88 \\
111 &= 88 \times 1 + 23 \\
88 &= 23 \times 3 + 19 \\
23 &= 19 \times 1 + 4 \\
19 &= 4 \times 4 + 3
\end{aligned}
$$

Euler's GCD Algorithm...

- Find $\mathrm{GCD}(421, 111)$.

$$\begin{aligned}
421 &= 111 \times 3 + 88 \\
111 &= 88 \times 1 + 23 \\
88 &= 23 \times 3 + 19 \\
23 &= 19 \times 1 + 4 \\
19 &= 4 \times 4 + 3 \\
4 &= 3 \times 1 + \boxed{1}
\end{aligned}$$

Euler's GCD Algorithm...

- Find $\mathrm{GCD}(421, 111)$.

$$
\begin{aligned}
421 &= 111 \times 3 + 88 \\
111 &= 88 \times 1 + 23 \\
88 &= 23 \times 3 + 19 \\
23 &= 19 \times 1 + 4 \\
19 &= 4 \times 4 + 3 \\
4 &= 3 \times 1 + \boxed{1} \\
3 &= 1 \times 3 + 0
\end{aligned}
$$

- The last non-zero remainder is $1 \Rightarrow \mathrm{GCD}(421, 111) = 1$.

In-Class Exercise

- Compute $\mathrm{GCD}(196, 42)$. Show your work.

Bezout's identity

Theorem (Bezout's identity)

Given any integers a and b, not both zero, there exist integers i and j such that $\mathrm{GCD}(a, b) = ia + jb$.

- Example:

$$\mathrm{GCD}(819, 462) = (-9) \times 819 + 16 \times 462 = 21.$$

- We use the Extended GCD Algorithm to compute i and j.

Bezout's identity: Extended GCD Algorithm

- Start by finding $\mathrm{GCD}(819, 462) = 21$:

$$0: \quad 819 \quad = \quad 462 \times 1 + 357$$

Bezout's identity: Extended GCD Algorithm

- Start by finding $\mathrm{GCD}(819, 462) = 21$:

$$\begin{aligned} 0: \quad 819 &= 462 \times 1 + 357 \\ 1: \quad 462 &= 357 \times 1 + 105 \end{aligned}$$

Bezout's identity: Extended GCD Algorithm

- Start by finding $\mathrm{GCD}(819, 462) = 21$:

$$\begin{aligned} 0: \quad 819 &= 462 \times 1 + 357 \\ 1: \quad 462 &= 357 \times 1 + 105 \\ 2: \quad 357 &= 105 \times 3 + 42 \end{aligned}$$

Bezout's identity: Extended GCD Algorithm

- Start by finding $\mathrm{GCD}(819, 462) = 21$:

$$\begin{aligned} 0: \quad 819 &= 462 \times 1 + 357 \\ 1: \quad 462 &= 357 \times 1 + 105 \\ 2: \quad 357 &= 105 \times 3 + 42 \\ 3: \quad 105 &= 42 \times 2 + 21 \end{aligned}$$

Bezout's identity: Extended GCD Algorithm

- Start by finding $\mathrm{GCD}(819, 462) = 21$:

$$
\begin{aligned}
0: \quad 819 &= 462 \times 1 + 357 \\
1: \quad 462 &= 357 \times 1 + 105 \\
2: \quad 357 &= 105 \times 3 + 42 \\
3: \quad 105 &= 42 \times 2 + 21 \\
4: \quad 42 &= 21 \times 2 + 0
\end{aligned}
$$

Bezout's identity: Extended GCD Algorithm

- Start by finding $\mathrm{GCD}(819, 462) = 21$:

$$\begin{aligned} 0: \quad 819 &= 462 \times 1 + 357 \\ 1: \quad 462 &= 357 \times 1 + 105 \\ 2: \quad 357 &= 105 \times 3 + 42 \\ 3: \quad 105 &= 42 \times 2 + 21 \\ 4: \quad 42 &= 21 \times 2 + 0 \end{aligned}$$

- Now work backwards, substituting one equation into the previous one.

Bezout's identity: Extended GCD Algorithm...

Step 3:		
3 :	105	$= 42 \times 2 + 21$

Bezout's identity: Extended GCD Algorithm...

Step 3:

3 :	105	$=$	$42 \times 2 + 21$
$3a$:	$1 \times 105 + (-2) \times 42$	$=$	21

Bezout's identity: Extended GCD Algorithm...

Step 3:

$3:$	105	$=$	$42 \times 2 + 21$
$3a:$	$1 \times 105 + (-2) \times 42$	$=$	21

Step 2:

$2:$	357	$=$	$105 \times 3 + 42$

Bezout's identity: Extended GCD Algorithm...

Step 3:

$3:$	105	$=$	$42 \times 2 + 21$
$3a:$	$1 \times 105 + (-2) \times 42$	$=$	21

Step 2:

$2:$	357	$=$	$105 \times 3 + 42$
$2a:$	$357 + (-3) \times 105$	$=$	42

Bezout's identity: Extended GCD Algorithm...

Step 3:

3 :	105	$=$	$42 \times 2 + 21$
$3a$:	$1 \times 105 + (-2) \times 42$	$=$	21

Step 2:

2 :	357	$=$	$105 \times 3 + 42$
$2a$:	$357 + (-3) \times 105$	$=$	42
$2b[2a \times (-2)]$	$(-2) \times 357 + (-2)(-3) \times 105$	$=$	$(-2) \times 42$

Bezout's identity: Extended GCD Algorithm...

Step 3:

3 :	105	$=$	$42 \times 2 + 21$
$3a$:	$1 \times 105 + (-2) \times 42$	$=$	21

Step 2:

2 :	357	$=$	$105 \times 3 + 42$
$2a$:	$357 + (-3) \times 105$	$=$	42
$2b[2a \times (-2)]$	$(-2) \times 357 + (-2)(-3) \times 105$	$=$	$(-2) \times 42$
$2c[2b \text{ in } 3a]$:	$(-2) \times 357 + (-2)(-3) \times 105$	$=$	$21 - 1 \times 105$

Bezout's identity: Extended GCD Algorithm...

Step 3:

$3:$	105	$=$	$42 \times 2 + 21$
$3a:$	$1 \times 105 + (-2) \times 42$	$=$	21

Step 2:

$2:$	357	$=$	$105 \times 3 + 42$
$2a:$	$357 + (-3) \times 105$	$=$	42
$2b[2a \times (-2)]$	$(-2) \times 357 + (-2)(-3) \times 105$	$=$	$(-2) \times 42$
$2c[2b \text{ in } 3a]:$	$(-2) \times 357 + (-2)(-3) \times 105$	$=$	$21 - 1 \times 105$
$2d[\text{simplify } 2c]:$	$(-2) \times 357 + 7 \times 105$	$=$	21

Bezout's identity: Extended GCD Algorithm...

Step 3:

$3:$	105	$=$	$42 \times 2 + 21$
$3a:$	$1 \times 105 + (-2) \times 42$	$=$	21

Step 2:

$2:$	357	$=$	$105 \times 3 + 42$
$2a:$	$357 + (-3) \times 105$	$=$	42
$2b[2a \times (-2)]$	$(-2) \times 357 + (-2)(-3) \times 105$	$=$	$(-2) \times 42$
$2c[2b \text{ in } 3a]:$	$(-2) \times 357 + (-2)(-3) \times 105$	$=$	$21 - 1 \times 105$
$2d[\text{simplify } 2c]:$	$(-2) \times 357 + 7 \times 105$	$=$	21

Bezout's identity: Extended GCD Algorithm...

Step 1:		
1 :	462	$= 357 \times 1 + 105$

Bezout's identity: Extended GCD Algorithm...

Step 1:

$1:$	462	$=357 \times 1 + 105$
$1a:$	$462 + (-1) \times 357$	$=105$

Bezout's identity: Extended GCD Algorithm...

Step 1:

$1:$	462	$= 357 \times 1 + 105$
$1a:$	$462 + (-1) \times 357$	$= 105$
$1b[1a \times 7]:$	$7 \times 462 + 7(-1) \times 357$	$= 7 \times 105$

Bezout's identity: Extended GCD Algorithm...

Step 1:

$1:$	462	$=357 \times 1 + 105$
$1a:$	$462 + (-1) \times 357$	$=105$
$1b[1a \times 7]:$	$7 \times 462 + 7(-1) \times 357$	$=7 \times 105$
$1c[1b \text{ in } 2d]:$	$(-2) \times 357 + 7 \times 462 + (7)(-1) \times 357$	$=21$

Bezout's identity: Extended GCD Algorithm...

Step 1:

$1:$	462	$=357 \times 1 + 105$
$1a:$	$462 + (-1) \times 357$	$=105$
$1b[1a \times 7]:$	$7 \times 462 + 7(-1) \times 357$	$=7 \times 105$
$1c[1b \text{ in } 2d]:$	$(-2) \times 357 + 7 \times 462 + (7)(-1) \times 357$	$=21$
$1d[\text{simplify } 1c]:$	$(-9) \times 357 + 7 \times 462$	$=21$

Bezout's identity: Extended GCD Algorithm...

Step 1:

$1:$	462	$=357 \times 1 + 105$
$1a:$	$462 + (-1) \times 357$	$=105$
$1b[1a \times 7]:$	$7 \times 462 + 7(-1) \times 357$	$=7 \times 105$
$1c[1b \text{ in } 2d]:$	$(-2) \times 357 + 7 \times 462 + (7)(-1) \times 357$	$=21$
$1d[\text{simplify } 1c]:$	$(-9) \times 357 + 7 \times 462$	$=21$

Step 0:

$0:$	819	$=462 \times 1 + 357$

Bezout's identity: Extended GCD Algorithm...

Step 1:

$1:$	462	$=357 \times 1 + 105$
$1a:$	$462 + (-1) \times 357$	$=105$
$1b[1a \times 7]:$	$7 \times 462 + 7(-1) \times 357$	$=7 \times 105$
$1c[1b \text{ in } 2d]:$	$(-2) \times 357 + 7 \times 462 + (7)(-1) \times 357$	$=21$
$1d[\text{simplify } 1c]:$	$(-9) \times 357 + 7 \times 462$	$=21$

Step 0:

$0:$	819	$=462 \times 1 + 357$
$0a:$	$819 + (-1) \times 462$	$=357$

Bezout's identity: Extended GCD Algorithm...

Step 1:

$1:$	462	$=357 \times 1 + 105$
$1a:$	$462 + (-1) \times 357$	$=105$
$1b[1a \times 7]:$	$7 \times 462 + 7(-1) \times 357$	$=7 \times 105$
$1c[1b \text{ in } 2d]:$	$(-2) \times 357 + 7 \times 462 + (7)(-1) \times 357$	$=21$
$1d[\text{simplify } 1c]:$	$(-9) \times 357 + 7 \times 462$	$=21$

Step 0:

$0:$	819	$=462 \times 1 + 357$
$0a:$	$819 + (-1) \times 462$	$=357$
$0b[0a \times (-9)]:$	$(-9) \times 819 + (-9)(-1) \times 462$	$=(-9) \times 357$

Bezout's identity: Extended GCD Algorithm...

Step 1:

$1:$	462	$=357 \times 1 + 105$
$1a:$	$462 + (-1) \times 357$	$=105$
$1b[1a \times 7]:$	$7 \times 462 + 7(-1) \times 357$	$=7 \times 105$
$1c[1b \text{ in } 2d]:$	$(-2) \times 357 + 7 \times 462 + (7)(-1) \times 357$	$=21$
$1d[\text{simplify } 1c]:$	$(-9) \times 357 + 7 \times 462$	$=21$

Step 0:

$0:$	819	$=462 \times 1 + 357$
$0a:$	$819 + (-1) \times 462$	$=357$
$0b[0a \times (-9)]:$	$(-9) \times 819 + (-9)(-1) \times 462$	$=(-9) \times 357$
$0c[0b \text{ in } 1d]:$	$(-9) \times 819 + (-9)(-1) \times 462 + 7 \times 462$	$=21$

Bezout's identity: Extended GCD Algorithm...

Step 1:

$1:$	462	$=357 \times 1 + 105$
$1a:$	$462 + (-1) \times 357$	$=105$
$1b[1a \times 7]:$	$7 \times 462 + 7(-1) \times 357$	$=7 \times 105$
$1c[1b \text{ in } 2d]:$	$(-2) \times 357 + 7 \times 462 + (7)(-1) \times 357$	$=21$
$1d[\text{simplify } 1c]:$	$(-9) \times 357 + 7 \times 462$	$=21$

Step 0:

$0:$	819	$=462 \times 1 + 357$
$0a:$	$819 + (-1) \times 462$	$=357$
$0b[0a \times (-9)]:$	$(-9) \times 819 + (-9)(-1) \times 462$	$=(-9) \times 357$
$0c[0b \text{ in } 1d]:$	$(-9) \times 819 + (-9)(-1) \times 462 + 7 \times 462$	$=21$
$0d[\text{simplify } 0c]:$	$(-9) \times 819 + 16 \times 462$	$=21$

In-Class Exercise

- Compute i and j such that at

$$\mathrm{GCD}(196, 42) = i \times 196 + j \times 42.$$

Show your work.

Computing Modular Multiplicative Inverses

- We can use the GCD routine to compute modular multiplicative inverses.

Computing Modular Multiplicative Inverses

- We can use the GCD routine to compute modular multiplicative inverses.
- Given $x < n$, we want to compute $y = x^{-1} \bmod n$, i.e.

$$yx \bmod n = 1$$

Computing Modular Multiplicative Inverses

- We can use the GCD routine to compute modular multiplicative inverses.
- Given $x < n$, we want to compute $y = x^{-1} \bmod n$, i.e.

$$yx \bmod n = 1$$

- The inverse of x in Z_n exists when $\mathrm{GCD}(n, x) = 1$.

Computing Modular Multiplicative Inverses

- We can use the GCD routine to compute modular multiplicative inverses.
- Given $x < n$, we want to compute $y = x^{-1} \bmod n$, i.e.

$$yx \bmod n = 1$$

- The inverse of x in Z_n exists when $\mathrm{GCD}(n, x) = 1$.
- Call $\mathrm{GCD}(n, x)$ which returns

$$(1, i, j)$$

  such that

$$1 = ix + jn$$

Computing Modular Multiplicative Inverses

- We can use the GCD routine to compute modular multiplicative inverses.
- Given $x < n$, we want to compute $y = x^{-1} \bmod n$, i.e.

$$yx \bmod n = 1$$

- The inverse of x in Z_n exists when $\mathrm{GCD}(n, x) = 1$.
- Call $\mathrm{GCD}(n, x)$ which returns

$$(1, i, j)$$

  such that

$$1 = ix + jn$$

- Then

$$(ix + jn) \bmod n = ix \bmod n = 1$$

  and i is x's multiplicative inverse in Z_n.

Computing Modular Multiplicative Inverses

- We can use the GCD routine to compute modular multiplicative inverses.
- Given $x < n$, we want to compute $y = x^{-1} \bmod n$, i.e.

$$yx \bmod n = 1$$

- The inverse of x in Z_n exists when $\mathrm{GCD}(n, x) = 1$.
- Call $\mathrm{GCD}(n, x)$ which returns

$$(1, i, j)$$

  such that

$$1 = ix + jn$$

- Then

$$(ix + jn) \bmod n = ix \bmod n = 1$$

  and i is x's multiplicative inverse in Z_n.
- If $\mathrm{GCD}(n, x) \neq 1$ then we know that the inverse doesn't exist.

Modular Exponentiation by Repeated Squaring

- Modular exponentiation is an important operation in cryptography.

$$g^n \bmod p = \overbrace{g * g * \cdots * g}^{n} \bmod p$$

Modular Exponentiation by Repeated Squaring

- Modular exponentiation is an important operation in cryptography.

$$g^n \bmod p = \overbrace{g * g * \cdots * g}^{n} \bmod p$$

- Simply iteratively multiplying the g:s together is too slow.

Modular Exponentiation by Repeated Squaring

- Modular exponentiation is an important operation in cryptography.

$$g^n \bmod p = \overbrace{g * g * \cdots * g}^{n} \bmod p$$

- Simply iteratively multiplying the g:s together is too slow.
- Instead, we compute

$$g$$

Modular Exponentiation by Repeated Squaring

- Modular exponentiation is an important operation in cryptography.

$$g^n \bmod p = \overbrace{g * g * \cdots * g}^{n} \bmod p$$

- Simply iteratively multiplying the g:s together is too slow.
- Instead, we compute

$$g$$

Modular Exponentiation by Repeated Squaring

- Modular exponentiation is an important operation in cryptography.

$$g^n \bmod p = \overbrace{g * g * \cdots * g}^{n} \bmod p$$

- Simply iteratively multiplying the g:s together is too slow.
- Instead, we compute

$$\begin{aligned} & g \\ g^2 &= g \cdot g \end{aligned}$$

Modular Exponentiation by Repeated Squaring

- Modular exponentiation is an important operation in cryptography.

$$g^n \bmod p = \overbrace{g * g * \cdots * g}^{n} \bmod p$$

- Simply iteratively multiplying the g:s together is too slow.
- Instead, we compute

$$\begin{aligned} & g \\ g^2 &= g \cdot g \\ g^4 &= g^2 \cdot g^2 \end{aligned}$$

Modular Exponentiation by Repeated Squaring

- Modular exponentiation is an important operation in cryptography.

$$g^n \bmod p = \overbrace{g * g * \cdots * g}^{n} \bmod p$$

- Simply iteratively multiplying the g:s together is too slow.
- Instead, we compute

$$\begin{aligned} & g \\ g^2 &= g \cdot g \\ g^4 &= g^2 \cdot g^2 \\ g^8 &= g^4 \cdot g^4 \end{aligned}$$

- We can then use these powers to compute g^n:

$$g^{25} = g^{16+8+1} = g^{16} \cdot g^8 \cdot g^1$$

Modular Exponentiation by Repeated Squaring

- Modular exponentiation is an important operation in cryptography.

$$g^n \bmod p = \overbrace{g * g * \cdots * g}^{n} \bmod p$$

- Simply iteratively multiplying the g:s together is too slow.
- Instead, we compute

$$\begin{aligned} & g \\ g^2 &= g \cdot g \\ g^4 &= g^2 \cdot g^2 \\ g^8 &= g^4 \cdot g^4 \end{aligned}$$

- We can then use these powers to compute g^n:

$$g^{25} = g^{16+8+1} = g^{16} \cdot g^8 \cdot g^1$$

Modular Exponentiation by Repeated Squaring

- Modular exponentiation is an important operation in cryptography.

$$g^n \bmod p = \overbrace{g * g * \cdots * g}^{n} \bmod p$$

- Simply iteratively multiplying the g:s together is too slow.
- Instead, we compute

$$\begin{aligned} & g \\ g^2 &= g \cdot g \\ g^4 &= g^2 \cdot g^2 \\ g^8 &= g^4 \cdot g^4 \end{aligned}$$

- We can then use these powers to compute g^n:

$$\begin{aligned} g^{25} &= g^{16+8+1} = g^{16} \cdot g^8 \cdot g^1 \\ g^{46} &= g^{32+8+4+2} = g^{32} \cdot g^8 \cdot g^4 \cdot g^2 \end{aligned}$$

Modular Exponentiation by Repeated Squaring...

- Compute g^n mod p:

```
function modexp(int g, int n, int p)
    int q ← 1
    int m ← n
    int square ← g
    while m ≥ 1 do
        if odd(m) then
            g ← q · square mod p
        square ← square · square mod p
        m ← ⌊m/2⌋
```

Primality Testing

- We are given an integer n and want to test if it's prime or not.

Primality Testing

- We are given an integer n and want to test if it's prime or not.
- There exists efficient methods for primality testing.

Primality Testing

- We are given an integer n and want to test if it's prime or not.
- There exists efficient methods for primality testing.
- The number of primes between 1 and n is at least $n/\ln(n)$, for $n \geq 4$.

Primality Testing

- We are given an integer n and want to test if it's prime or not.
- There exists efficient methods for primality testing.
- The number of primes between 1 and n is at least $n/\ln(n)$, for $n \geq 4$.
- To generate a prime number q between $n/2$ and n:

We need to repeat approximately a logarithmic number of times to find a prime.

Primality Testing

- We are given an integer n and want to test if it's prime or not.
- There exists efficient methods for primality testing.
- The number of primes between 1 and n is at least $n/\ln(n)$, for $n \geq 4$.
- To generate a prime number q between $n/2$ and n:
  1. Let $q \leftarrow$ a random number between $n/2$ and n;

We need to repeat approximately a logarithmic number of times to find a prime.

Primality Testing

- We are given an integer n and want to test if it's prime or not.
- There exists efficient methods for primality testing.
- The number of primes between 1 and n is at least $n/\ln(n)$, for $n \geq 4$.
- To generate a prime number q between $n/2$ and n:
  1. Let $q \leftarrow$ a random number between $n/2$ and n;
  2. q is prime with a probability of at least $1/\ln(n)$;

We need to repeat approximately a logarithmic number of times to find a prime.

Primality Testing

- We are given an integer n and want to test if it's prime or not.
- There exists efficient methods for primality testing.
- The number of primes between 1 and n is at least $n/\ln(n)$, for $n \geq 4$.
- To generate a prime number q between $n/2$ and n:
  1. Let $q \leftarrow$ a random number between $n/2$ and n;
  2. q is prime with a probability of at least $1/\ln(n)$;
  3. If $\mathrm{isPrime}(q)$ then return q;

  We need to repeat approximately a logarithmic number of times to find a prime.

Primality Testing

- We are given an integer n and want to test if it's prime or not.
- There exists efficient methods for primality testing.
- The number of primes between 1 and n is at least $n/\ln(n)$, for $n \geq 4$.
- To generate a prime number q between $n/2$ and n:
  1. Let $q \leftarrow$ a random number between $n/2$ and n;
  2. q is prime with a probability of at least $1/\ln(n)$;
  3. If $\mathrm{isPrime}(q)$ then return q;
  4. Repeat from 1.

  We need to repeat approximately a logarithmic number of times to find a prime.

In-Class Exercise: Goodrich & Tamassia R-8.16

- Roughly how many times would you have to call a primality tester to find a prime number between 1,000,000 and 2,000,000?

Outline

Euler's Theorem

Theorem (Euler)

Let x be any positive integer that's relatively prime to the integer $n > 0$, then

$$x^{\phi(n)} \bmod n = 1$$

- Now consider (where a and n are relatively prime)

$$a^m \bmod n = 1.$$

  We know, by Euler's theorem, that there's at least one number m that satisfies this equation: $\phi(n)$!

- The smallest positive m for which the equation holds is called
  - the order of a mod n
  - the length of the period generated by a.

The Order of a mod n

- Consider the powers of 7, mod 19:

$$\begin{aligned}
7^1 &= 7 \bmod 19 \\
7^2 &= 11 \bmod 19 \\
7^3 &= 1 \bmod 19 \\
7^4 &= 7 \bmod 19 \\
7^5 &= 11 \bmod 19 \\
7^6 &= 1 \bmod 19
\end{aligned}$$

- The sequence repeats.

Powers of Integers, Modulo 19

- All the powers of *a*, modulo 19.
- The length of the sequence is highlighted.

a^1	a^2	a^3	a^4	a^5	a^6	a^7	a^8	a^9	a^{10}	a^{11}	a^{12}	a^{13}	a^{14}	a^{15}	a^{16}	a^{17}	a^{18}
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	4	8	16	13	7	14	9	18	17	15	11	3	6	12	5	10	1
3	9	8	5	15	7	2	6	18	16	10	11	14	4	12	17	13	1
4	16	7	9	17	11	6	5	1	4	16	7	9	17	11	6	5	1
5	6	11	17	9	7	16	4	1	5	6	11	17	9	7	16	4	1
6	17	7	4	5	11	9	16	1	6	17	7	4	5	11	9	16	1
7	11	1	7	11	1	7	11	1	7	11	1	7	11	1	7	11	1
8	7	18	11	12	1	8	7	18	11	12	1	8	7	18	11	12	1
9	5	7	6	16	11	4	17	1	9	5	7	6	16	11	4	17	1
10	5	12	6	3	11	15	17	18	9	14	7	13	16	8	4	2	1
11	7	1	11	7	1	11	7	1	11	7	1	11	7	1	11	7	1
12	11	18	7	8	1	12	11	18	7	8	1	12	11	18	7	8	1
13	17	12	4	14	11	10	16	18	6	2	7	15	5	8	9	3	1
14	6	8	17	10	7	3	4	18	5	13	11	2	9	12	16	15	1
15	16	12	9	2	11	13	5	18	4	3	7	10	17	8	6	14	1
16	9	11	5	4	7	17	6	1	16	9	11	5	4	7	17	6	1
17	4	11	16	6	7	5	9	1	17	4	11	16	6	7	5	9	1
18	1	18	1	18	1	18	1	18	1	18	1	18	1	18	1	18	1

Primitive Roots

- All sequences end with 1.

Primitive Roots

- All sequences end with 1.
- Some sequences have length 18. Then we say

Primitive Roots

- All sequences end with 1.
- Some sequences have length 18. Then we say
  - *a* generates the set of nonzero integers, modulo 19.

Primitive Roots

- All sequences end with 1.
- Some sequences have length 18. Then we say
  - a generates the set of nonzero integers, modulo 19.
  - a is a primitive root of the modulus 19.

Primitive Roots

- All sequences end with 1.
- Some sequences have length 18. Then we say
  - a generates the set of nonzero integers, modulo 19.
  - a is a primitive root of the modulus 19.
- If a is a primitive root of n then all its powers

$$a, a^2, \ldots, a^{\phi(n)}$$

are distinct.

Primitive Roots

- All sequences end with 1.
- Some sequences have length 18. Then we say
  - a generates the set of nonzero integers, modulo 19.
  - a is a primitive root of the modulus 19.
- If a is a primitive root of n then all its powers

  $$a, a^2, \ldots, a^{\phi(n)}$$

  are distinct.
- If a is a primitive root of p, and p is prime, then

  $$a, a^2, \ldots, a^p$$

  are distinct $\mod p$.

Primitive Roots...

- The only integers with primitive roots are of the form (p prime, $\alpha > 0$)

$$2, 4, p^{\alpha}, 2p^{\alpha}$$

Primitive Roots. . .

- The only integers with primitive roots are of the form (p prime, $\alpha > 0$)

$$2, 4, p^{\alpha}, 2p^{\alpha}$$

- For 19 (a prime), the primitive roots are $2, 3, 10, 13, 14, 15$.

Primitive Roots...

- The only integers with primitive roots are of the form (p prime, $\alpha > 0$)

$$2, 4, p^\alpha, 2p^\alpha$$

- For 19 (a prime), the primitive roots are $2, 3, 10, 13, 14, 15$.
- g is a primitive root modulo p if, for each integer i in Z_p, there exists an integer k such that

$$i = g^k \bmod p.$$

Primitive Roots...

- For example, looking at the table above, we see that 2 is a primitive root modulo 19:

2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}
2	4	8	16	13	7	14	9	18	17	15	11	3	6	12	5	10	1

because for each integer $i \in Z_{19} = \{1, 2, 3, \ldots, 18\}$ there's an integer k, such that $i = 2^k \bmod 19$.

Primitive Roots...

- For example, looking at the table above, we see that 2 is a primitive root modulo 19:

2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}
2	4	8	16	13	7	14	9	18	17	15	11	3	6	12	5	10	1

  because for each integer $i \in Z_{19} = \{1, 2, 3, \ldots, 18\}$ there's an integer k, such that $i = 2^k \bmod 19$.
- There are $\phi(p - 1)$ generators for Z_p.

In-Class Exercise

1. Compute the table of powers of a, modulo 5, for all positive integers $a < 5$.
2. What are the primitive roots of 5?

In-Class Exercise

1. Compute the table of powers of a, modulo 7, for all positive integers $a < 7$.
2. What are the primitive roots of 7?

Computing Primitive Roots

- Consider the equation

$$y = g^x \bmod p$$

If we have g, x, and p it's easy to calculate y.

Computing Primitive Roots

- Consider the equation

$$y = g^x \bmod p$$

  If we have g, x, and p it's easy to calculate y.
- What if, instead, we're given y, g, and p?

Computing Primitive Roots

- Consider the equation

$$y = g^x \bmod p$$

  If we have g, x, and p it's easy to calculate y.
- What if, instead, we're given y, g, and p?
  - it's hard to take the discrete logarithm, i.e. to compute x.

Computing Primitive Roots

- Consider the equation

$$y = g^x \bmod p$$

  If we have g, x, and p it's easy to calculate y.
- What if, instead, we're given y, g, and p?
  - it's hard to take the discrete logarithm, i.e. to compute x.
- The fastest known algorithm is

$$\mathcal{O}(e^{((\ln p)^{1/3}(\ln(\ln p))^{2/3})})$$

  which is infeasible for large primes p.

Outline

Readings and References

- Chapter 8.1.7, 8.2.1, 8.5.2 in *Introduction to Computer Security*, by Goodrich and Tamassia.

Acknowledgments

Additional material and exercises have also been collected from these sources:

1. Igor Crk and Scott Baker, *620—Fall 2003—Basic Cryptography*.
2. William Stallings, *Cryptography and Network Security*.
3. Bruce Schneier, *Applied Cryptography*.
4. Neal R. Wagner, *The Laws of Cryptography with Java Code*, http://amadousarr.free.fr/java/javacryptobook.pdf.
5. *Euler's Totient Function Values For n = 1 to 500, with Divisor Lists*, http://primefan.tripod.com/Phi500.html
6. Diffie-Hellman calculator: http://dkerr.home.mindspring.com/diffie_hellman_calc.html.