414-S17 Crypto

Shankar

April 18, 2017

Outline

Overview

Symmetric Crypto
- Block Cipher
- Encryption Modes for Variable-size Messages
- Message Authentication Codes (MACs)
- MAC and Confidentiality

Asymmetric Crypto (aka Public-Key Crypto)
- Introduction
- A Little Bit of Number Theory
- RSA
- Diffie-Helman

Overview

- Crypto is everywhere
  - communications: https, IPsec, 802.11, WPA2, ...
  - files on disk: Bitlocker, FileVault, ...
  - user authentication: Kerberos, ...
  - ...
- Crypto enables secure data communication and storage
  - Confidentiality: only the intended receiver can read the data
  - Integrity: the intended receiver detects any changes to the data
  - Authentication: data received was sent by the specified sender
  - Non-repudiation: third party can verify that the data was sent by the specified sender

Symmetric and Asymmetric Crypto

- Key generation: generate encryption and decryption keys
- Encryption E: plaintext + encryption key $\longrightarrow$ ciphertext
- Decryption D: plaintext $\longleftarrow$ ciphertext + decryption key

- Symmetric crypto
  - encryption key = decryption key
  - eg, AES, MD5, SHA-1, SHA-256, ...
  - fast
- Asymmetric (aka public-key) crypto
  - encryption key $\neq$ decryption key
  - eg, RSA, DH, DSS, ...
  - very slow

Desired properties of crypto functions

- Correctness
  - For any encryption key key_E and its decryption key key_D: if $E(key_E, ptxt)$ returns $ctxt$ then $D(key_D, ctxt)$ returns $ptxt$

- Security: Assuming keys are chosen uniformly randomly
  - Given cyphertext, hard to get plaintext.
  - Given plaintext and ciphertext, hard to get key.
  - Hard: requires brute-force search of key-space (eg, 2^{128} keys)

- Attacker models (from weakest to strongest)
  - Ciphertext-only attack
  - Known plaintext attack: one matching pair
  - Chosen plaintext attack: encryption oracle
  - Chosen ciphertext attack: encryption oracle + decryption oracle

Achieving secure communication

- A and B separated by insecure channel, share secret key k.
- Confidentiality:
  - A sends $E(k, \mathit{plaintext})$
  - B receives and does $D(k, \mathit{ciphertext})$
- Integrity:
  - *mac*: $E(k, \mathit{hash}(\mathit{plaintext}))$
  - A sends $[\mathit{plaintext}, \mathit{mac}]$
  - B receives and verifies *mac*
- Authentication:
  - A sends a random r_A to B, and expects $E(k, r_A)$ back
  - B sends a random r_B to A, and expects $E(k, r_B)$ back

Outline

Overview

Symmetric Crypto

- Block Cipher
- Encryption Modes for Variable-size Messages
- Message Authentication Codes (MACs)
- MAC and Confidentiality

Asymmetric Crypto (aka Public-Key Crypto)

- Introduction
- A Little Bit of Number Theory
- RSA
- Diffie-Helman

Outline

Overview

Symmetric Crypto
- Block Cipher
- Encryption Modes for Variable-size Messages
- Message Authentication Codes (MACs)
- MAC and Confidentiality

Asymmetric Crypto (aka Public-Key Crypto)
- Introduction
- A Little Bit of Number Theory
- RSA
- Diffie-Helman

Ideal block cipher

- Fixed-size messages of d bits (eg, 64, 128)
- Fixed-size keys of k bits (eg, 128, 256)
  - any random k bits is a valid key
- Encryption E : d-bit msg + k-bit key $\longrightarrow$ d-bit output
- To decrypt, E must be 1-1 mapping of msgs to outputs
- To be secure, E must be "random"
  - $E(key, msg)$ gives no information about *key* or *msg*
  - Msgs and keys that differ (even if only slightly) map to outputs that differ randomly
- Key size k large enough so that searching 2^k is infeasible
- Clearly, E cannot be a "simple" function, eg, $msg \oplus key$

Implementation: d-bit data, k-bit key

- Naive approach
  - Table of a random permutation of d-bit strings // $2^d \times d$ bits
  - $E(i)$ is ith row of table
  - Secure but impractical // table itself is the key!

- Practical approach: localized scrambling and global permutations
  - Generate n "round keys" from the key // n small, eg, 10
  - Repeat n times

    Partition d-bit string into p-bit chunks // 2^p is manageable

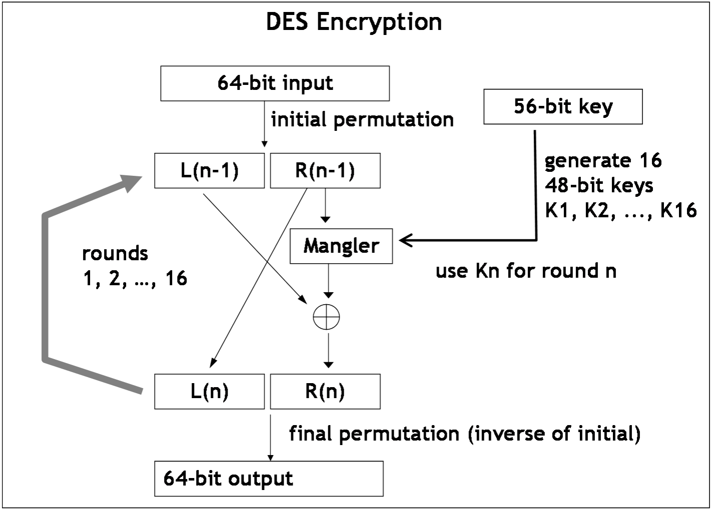
    Scramble each p-bit chunk using $2^p \times p$ tables

    // table's permutation depends on round-n key

    Permute the resulting d-bit string
  - Decryption is similar // often reuse same hardware

~~Exam~~ DES: Data Encryption Standard

- Old standard no longer used: 56-bit keys, 64-bit text



~~Exam~~ DES encryption and decryption

block cipher | symm

DES encryption

a1: $L_0 \mid R_0 \leftarrow perm(pt)$
a2: for $n = 0, \ldots, 15$
a3: $L_{n+1} \leftarrow R_n$
a4: $R_{n+1} \leftarrow mnglr_n(R_n, K_{n+1}) \oplus L_n$
// yields $L_{16} \mid R_{16}$

a5: $L_{17} \mid R_{17} \leftarrow R_{16} \mid L_{16}$
a6: $ct \leftarrow perm^{-1}(R_{16} \mid L_{16})$

// key order: $K_1, \cdots, K_{16}$

DES decryption

b1: $R_{16} \mid L_{16} \leftarrow perm(ct)$ // a6 bw
b2: for $n = 15, \ldots, 0$ // a2 bw
b3: $R_n \leftarrow L_{n+1}$ // a3 bw
b4: $L_n \leftarrow mnglr_n(R_n, K_n) \oplus R_{n+1}$ // a4 bw
// sets L_n to X such that
// $R_{n+1} \leftarrow mnglr_n(R_n, K_n) \oplus X$
// yields $R_0 | L_0$
b5: $L_0 | R_0 \leftarrow R_0 | L_0$ // a5 bw
b6: pt $\leftarrow perm^{-1}(L_0 | R_0)$ // a1 bw

// key order $K_{16}, \cdots, K_1$

~~Exam~~ Multiple DES: EDE or 3DES

- Makes DES more secure
  - Encryption: encrypt key1 $\rightarrow$ decrypt key2 $\rightarrow$ encrypt key1
  - Decryption: decrypt key1 $\rightarrow$ encrypt key2 $\rightarrow$ decrypt key1
- encrypt key1 $\rightarrow$ encrypt key1 is not effective
  - Just equivalent to using another single key.
- encrypt key1 $\rightarrow$ encrypt key2 is not so good

AES: Advanced Encryption Standard

- Current standard encryption algorithm
- Different key sizes: 128, 192, 256
- Data block size: 128 bits
- Algorithm overview ~~Exam~~
  - 10, 12, or 14 rounds // depending on key size
  - Round keys generated from the cipher key
  - Data block treated as 4×4 matrix of bytes
  - Each round involves operations in a finite field
    - permutation of the bytes // lookup table
    - cyclic shifting of rows
    - mixing bytes in each column

Outline

Overview

Symmetric Crypto
- Block Cipher
- Encryption Modes for Variable-size Messages
- Message Authentication Codes (MACs)
- MAC and Confidentiality

Asymmetric Crypto (aka Public-Key Crypto)
- Introduction
- A Little Bit of Number Theory
- RSA
- Diffie-Helman

Handling variable-size messages

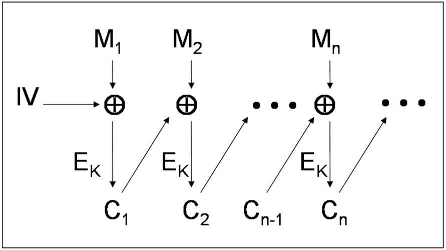
- Encrypting variable-size msg given d-bit block cipher
  - Pad message to multiple of block size:
    $msg \longrightarrow m_1, m_2, \cdots$
  - Use block encryption repeatedly to get ciphertext
    $m_1, m_2, \cdots \longrightarrow c_1, c_2, \cdots$
- Desired
  - $c_j \neq c_k$ even if $m_j = m_k$ // like block encryption
  - Repeated encryptions of *msg* yield different *ciphertxt* // unlike block encryption
- Various modes: ECB, CBC, CFB, OFB, CTR, others

ECB: Electronic Code Book

- Encryption: $m_1, m_2, \cdots \quad \longrightarrow \quad c_1, c_2, \cdots$
- Natural approach: encrypt each block independently
- Encryption: $c_i = E(key, m_i)$
- Decryption: $m_i = D(key, c_i)$
- Not good: repeated blocks get same cipherblock
- Never use ECB
  - Amazingly, the default mode for some crypto libraries

CBC: Cipher Block Chaining

- Encryption: $m_1, m_2, \cdots \quad \longrightarrow \quad c_1, c_2, \cdots$
- Use c_{i-1} as a "random" pad to m_i before encrypting.
  - $c_0 \leftarrow$ random IV
  - $c_i \leftarrow E(key, m_i \oplus c_{i-1})$
  - send $IV, c_1, c_2, \cdots$
  - Can be attacked if IV is predictable



- Decryption: $c_1, c_2, \cdots \quad \longrightarrow \quad m_1, m_2, \cdots$
  - $m_i \leftarrow D(key, c_i) \oplus c_{i-1} \qquad$ for $i = 1, 2, \cdots$
  - Can be done in parallel

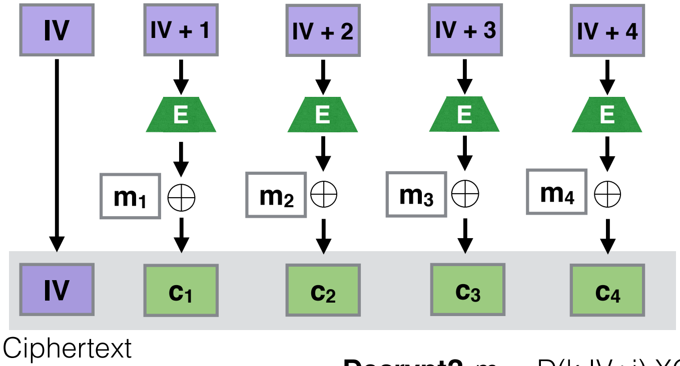
OFB: Output Feedback Mode

- Encryption: $m_1, m_2, \cdots \quad \longrightarrow \quad c_1, c_2, \cdots$
- Generate pad $b_0, b_1, \cdots$:
  - $b_0 \leftarrow$ random IV
  - $b_i \leftarrow E(key, b_{i-1})$
- $c_i \leftarrow b_i \oplus m_i$
- One-time pad that can be generated in advance.
  - attacker with <plaintxt, ciphertxt> can obtain b_i's.

CFB: Cipher Feedback Mode

- Like OFB except that output c_{i-1} is used instead of b_i
  - c_0 is IV
  - $c_i \leftarrow m_i \oplus E(key, c_{i-1})$
- Cannot generate one-time pad in advance.

Counter mode (CTR)



Decrypt? $m_i = D(k, IV+i) \text{ XOR } c_i$

CTR

- Like OFB, can encrypt in parallel
- Initial IV must be random
  - Don't use IV for one msg and $IV + 1$ for another msg

Outline

Overview

Symmetric Crypto

- Block Cipher
- Encryption Modes for Variable-size Messages
- Message Authentication Codes (MACs)
- MAC and Confidentiality

Asymmetric Crypto (aka Public-Key Crypto)

- Introduction
- A Little Bit of Number Theory
- RSA
- Diffie-Helman

MAC: Message Authenication Code

- A MAC detects any change to a msg // integrity
- Signing S: $\quad msg + key \longrightarrow tag$ // send msg, tag
- Verification V: $\quad msg + tag + key \longrightarrow$ YES or NO
  - YES iff *msg* was exactly that sent by *key* holder

- A MAC is secure if an attacker (w/o *key*)
  - can issue msgs $m_1, m_2, \cdots$ and get their tags $t_1, t_2, \cdots$ but still cannot produce the valid tag t for any new msg m
    // Existential forgery

- MACs currently used: ECBC, SHA, SHA-3, others

MACs from Block Ciphers

- Encrypting msg (eg, CBC, CFB, OFB) does not provide integrity
  - Modified ciphertext still decrypts to something

- Encrypted CBC (ECBC): yields a MAC from a block cipher
- Signing S
  - Input: msg, key, key'
  - Apply CBC on msg using key and no IV // $IV = 0$
  - Only the last cipherblock, say c, is needed
  - $tag = E(key', c)$

- Verifying V
  - Input: msg, key, key', tag
  - YES iff $S(msg, key, key')$ equals tag

ECBC vs CBC

- Output only one block // coz not recovering plaintext
- Need two keys, otherwise attacker
  - issues msg $[m_1, \cdots, m_n]$, gets tag $t = c_n$
  - creates single-block msg m', gets tag t' for $t \oplus m'$
  - t' is valid tag for $m||m'$ // "$||$" is concatenation
- Both CBC and ECBC must be computed sequentially
  - There are CTR-like MACs which permit parallel computation
- Would using only one key with a random IV work?
  - msg's tag is last cipherblock of $CBC(key, IV||msg)$

MACs from Hash Functions

Hashes

- Hash function H
  - arbitrary message $\longrightarrow$ k-bit hash
    (pre-image) (digest)
  - msg space $\gg$ hash space $(= 2^k)$ // not 1-1
  - Does not take a key as input
- H is cryptographically secure if // "one-way"
  - Pre-image resistant: hard to find m given $H(m)$
  - Collision-resistant: hard to find $m \neq m'$ s.t. $H(m) = H(m')$
  - In fact, for any $m \neq m'$, the probability that $H(m)$ and $H(m')$ are equal at any given bit index i is $1/2$

How large for collision-resistance

- Assuming H is random, how large should k be?
- Pr(collision in N random msgs $m_1, \cdots, m_N$)

$$\begin{aligned} &= Pr[H(m_1) = H(m_2) \textit{ or } H(m_1) = H(m_3) \textit{ or } \cdots] \\ &\approx N(N-1)/2 \times (1/2^k) \\ &\approx N^2/2^k \end{aligned}$$

- Pr significant if $N^2 \approx 2^k$, ie, if $N \approx \sqrt{2^k}$
- Choose k so that searching through $\sqrt{2^k}$ msgs is hard
  - So $k = 128$ assumes searching through 2^{64} msgs is hard

Some current secure hash functions

- MD5 (Message digest 5): 128-bit digest
  - Known collision attacks, still frequently used
- SHA family
  - SHA-1: 160-bit hash // theoretically broken, but used
  - SHA-256: 256-bit hash
  - SHA-512
  - etc
- SHA-3 (224, 256, 385, 512) // standardized Aug 2015

~~Exam~~ Internals of MD4 (128-bit hash)

- Step 1: Pad *msg* to multiple of 512 bits
  - $pmsg \leftarrow msg|\text{one }1|\ p\ 0\text{'s}|\ (\text{64-bit encodng of } p)$ // p in 1..512

- Step 2: Process *pmsg* in 512-bit chunks to get hash *md*
  - treat 128-bit *md* as 4 words: d_0, d_1, d_2, d_3
    - initialize to 01|23|...|89|ab|cd|ef|fe|dc|...|10
  - For each successive 512-bit chunk of *pmsg*:
    - treat 512-bit chunk as 16 words: $m_0, m_1, \cdots, m_{15}$
    - $e_0..e_3 \leftarrow d_0..d_3$ // save for later
    - pass 1 using mangler $H1$ and permutation J

      // for $i = 0, ..., 15$: $d_{J(i)} \leftarrow H1(i, d_0, d_1, d_2, d_3, m_i)$
    - pass 2: same but with mangler $H2$
    - pass 3: same but with mangler $H3$
    - $d_0..d_3 \leftarrow d_0..d_3 \oplus e_0..e_3$
  - $md \leftarrow d_0..d_3$

MACs from hash functions

- MAC of a *msg* is a hash of some combination of *msg* and *key*
  - $MAC(msg) = H(key, msg)$
- But need to be careful in how *key* and *msg* are combined
- In particular, $key||msg$ is not good // "||" is concatenation
  - This is because usually $H(m_1 || m_2)$ is $H(H(m_1) || m_2)$
  - Given a msg m_1 and $H(key || m_1)$, attacker can get $H(key || m_1 || m_2)$ by doing $H(H(key, m_1) || m_2)$

HMAC: Hash-MAC

- HMAC: standard way to get MACs from Hashes
- HMAC takes any hash function H and any size *key*
- $HMAC(key, msg, H)$

  $= \; H(\,(key' \oplus opad\,) \,||\, H(\,(key' \oplus ipad\,) \,||\, msg\,)\,)$

  - $key' \leftarrow key$ padded with 0's to H's input block size
    if *key* size $>$ H's block size, first hash *key*
  - $opad \; = \;$ 0x5c5c...5c of H's block size // outer padding
  - $ipad \; = \;$ 0x3636...36 of H's block size // inner padding

Aside: Keyed-hash $\equiv$ Block Cipher

- Encryption: $m_1, m_2, \cdots \longrightarrow c_0, c_1, c_2, \cdots$
  - Generate pad: $b_i \leftarrow H(key, b_{i-1})$ where B_0 is IV
  - $c_i \leftarrow b_i \oplus m_i$
  - send $IV, c_1, c_2, \cdots$
- Decryption identical

Outline

Overview

Symmetric Crypto
- Block Cipher
- Encryption Modes for Variable-size Messages
- Message Authentication Codes (MACs)
- MAC and Confidentiality

Asymmetric Crypto (aka Public-Key Crypto)
- Introduction
- A Little Bit of Number Theory
- RSA
- Diffie-Helman

Integrity and confidentiality

- Encrypt || MAC: send $E(msg) \mid\mid MAC(msg)$
  - $MAC(msg)$ may reveal something about *msg*
  - Do not use
- MAC then Encrypt: send $E(msg \mid\mid MAC(msg))$
  - Can be insecure for some E and MAC combinations
  - Do not use
- Encrypt then MAC: send $E(msg) \mid\mid MAC(E(msg))$
  - MAC may reveal something of ciphertext, but that's ok
  - Use this

Outline

Overview

Symmetric Crypto
- Block Cipher
- Encryption Modes for Variable-size Messages
- Message Authentication Codes (MACs)
- MAC and Confidentiality

Asymmetric Crypto (aka Public-Key Crypto)
- Introduction
- A Little Bit of Number Theory
- RSA
- Diffie-Helman

Outline

Overview

Symmetric Crypto
- Block Cipher
- Encryption Modes for Variable-size Messages
- Message Authentication Codes (MACs)
- MAC and Confidentiality

Asymmetric Crypto (aka Public-Key Crypto)
- Introduction
- A Little Bit of Number Theory
- RSA
- Diffie-Helman

Asymmetric Crypto: Encryption

- Key generation
  - Input: source of randomness and max key length L
  - Output: pair of keys, each of size $\leq L$
    - pk: "public" key // publicly disclosed
    - sk: "secret (aka "private") key // shared with no one
- Encryption $E_P(pk, m)$ // executed by public
  - Input: public key pk; msg m (size $\leq L$)
  - Add random pad to m // PKCS, OAEP
  - Output: ciphertext c (size $\leq L$)
- Decryption $D_P(sk, c)$ // executed by sk owner
  - Input: secret key sk; ciphertext c (size $\leq L$)
  - Output: original msg m

Asymmetric Crypto: Encryption

- Key pair $[pk, sk]$
- Correctness
  - $D_P(sk,\ E_P(pk, m)) \ = \ m$
- Security
  - $E_P(pk, m)$ appears random // one-way
  - Can only be decrypted with sk // trapdoor
  - Hard to get sk from pk
- Hybrid encryption for arbitrary-size msg m
  - generate symmetric key k
  - symmetric encrypt m: $c_m = E(k, m)$
  - public-key encrypt k: $c_k = E_P(pk, k)$
  - send $[c_m, c_k]$

Asymmetric Crypto: Signatures

- Key generation: public key pk, secret key sk // as before
- Signing $Sgn(sk, m)$ // executed by sk owner
  - Input: secret key sk; msg m (size $\leq L$)
  - Output: signature s (size $\leq L$)
- Verification function $Vfy(pk, m, s)$ // executed by public
  - Input: public key pk; msg m, signature s
  - Output: YES iff s is a valid signature of m using sk
- Correctness: $Vfy(pk,\ m,\ Sgn(sk, m)) \ = \ \text{YES}$
- Security: Even with pk and many $[msg, sgn]$ examples, cannot produce existential forgery

Asymmetric Crypto Examples

- RSA, ECC: encryption and signatures
- ElGamal, DSS: signatures
- Diffie-Hellman: establishment of a shared secret

Outline

Overview

Symmetric Crypto
- Block Cipher
- Encryption Modes for Variable-size Messages
- Message Authentication Codes (MACs)
- MAC and Confidentiality

Asymmetric Crypto (aka Public-Key Crypto)
- Introduction
- A Little Bit of Number Theory
- RSA
- Diffie-Helman

De-mystifying asymmetric crypto

- Asymmetric crypto is based on modulo-n arithmetic
- It seems magical. but it can be de-mystified with a bit of effort
- What follows is brief look at some number theory
  - Prime numbers
  - Modulo-n addition, multiplication and exponentiation
  - Euler's totient function and a theorem

Prime numbers

- Integer p is prime iff it is exactly divisible only by itself and 1.
- $gcd(p, q)$: greatest common denominator of integers p and q
  - Largest integer that divides both exactly.
- p and q are relatively prime iff $gcd(p, q) = 1$
- Infinitely many primes, but they thin out as numbers get larger
  - 25 primes less than 100
  - Pr[random 10-digit number is a prime] $\approx 1/23$
  - Pr[random 100-digit number is a prime] $\approx 1/230$
  - Pr[random k-digit number is a prime] $\approx 1/(k \cdot \ln 10)$

Modulo-n operator

- $Z_n = \{0, 1, \cdots, n-1\}$

- Modulo-n: integers $\longrightarrow Z_n$ // includes negative integers

- x mod-n for any integer x

  $= \;\; y$ in Z_n st $x = y + k \cdot n$ for some integer k

  $= \;\;$ non-negative remainder of x/n

- Examples
  - 3 mod-$10 = 3$ // $3 = 3 + 0 \cdot 10$
  - 23 mod-$10 = 3$ // $23 = 3 + 2 \cdot 10$
  - -27 mod-$10 = 3$ // $-27 = 3 + (-3) \cdot 10$

    Note: mod-n of negative number is non-negative

Modulo-n addition

- $(a+b)$ mod-n for any integers a and b
- Examples
  - $(3+7)$ mod-10 $=$ 10 mod-10 $=$ 0
  - $(3-7)$ mod-10 $=$ -4 mod-10 $=$ 6
- Additive-inverse-mod-n of x // aka $-x$ mod-n
  - y st $(x+y)$ mod-$n = 0$ // *st*: such that
  - exists for every x
  - easily computed: $(n-x)$ mod-n

Modulo-n multiplication

- $(a \cdot b)$ mod-n for any integers a and b
- Examples
  - $(3 \cdot 7)$ mod-10 $=$ 21 mod-10 $=$ 1 // "$\cdot$" is multiplication
  - $8 \cdot (-7)$ mod-10 $=$ -56 mod-10 $=$ 4
- Multiplicative-inverse-mod-n of x // aka x^{-1} mod-n
  - y st $(x \cdot y)$ mod-$n = 1$
  - exists iff $gcd(x, n) = 1$ // x relatively prime to n
  - Easily computed by Euclid's algorithm // ~~Exam~~
    - $Euclid(x, n)$ returns u, v st $gcd(x, n) = u \cdot x + v \cdot n$
    - if $gcd(x, n) = 1$: $\boxed{u = x^{-1} \text{ mod-}n}$ and $\boxed{v = n^{-1} \text{ mod-}x}$

Modulo-n exponentiation

- (a^b) mod-n for any integer a and integer $b > 0$
- Examples
  - 3^2 mod-10 $=$ 9
  - 3^3 mod-10 $=$ 27 mod-10 $=$ 7
  - $(-3)^3$ mod-10 $=$ -27 mod-10 $=$ 3
- Exponentiative-inverse-mod-n of x
  - y st (x^y) mod-$n = 1$
  - exists iff $gcd(x, n) = 1$
  - easily computed given prime factors of n // only way known

Euler's Totient Function

- $Z_n{}^* \; = \; \{x \text{ in } Z_n,\, gcd(x,n) = 1\}$ // $Z_{10}^* \; = \; \{1,3,7,9\}$
- $\phi(n)$: number of elements in $Z_n{}^*$ // $\phi(10) \; = \; 4$
- Euler's Totient Function // ~~Exam~~

$$\phi(n) = \begin{cases} n-1 & \text{if } n \text{ prime} \\ \phi(p)\cdot\phi(q) & \text{if } n = p\cdot q \text{ and } gcd(p,q) = 1 \\ (p-1)\cdot p^a - 1 & \text{if } n = p^a,\; p \text{ prime, } a > 0 \\ \phi(p_1{}^{a_1})\cdots\phi(p_K{}^{a_K}) & \text{if } n = p_1{}^{a_1}\cdots p_K{}^{a_K} \end{cases}$$

- If p, q distinct primes: $\phi(p\cdot q) \; = \; (p-1)\cdot(q-1)$

Euler's Theorem:

If $n = p \cdot q$ for distinct primes p and q, then

$$a^{(k \cdot \phi(n) + 1)} \text{ mod-}n = a \text{ mod-}n$$

for any a and $k > 0$

Outline

Overview

Symmetric Crypto
- Block Cipher
- Encryption Modes for Variable-size Messages
- Message Authentication Codes (MACs)
- MAC and Confidentiality

Asymmetric Crypto (aka Public-Key Crypto)
- Introduction
- A Little Bit of Number Theory
- RSA
- Diffie-Helman

RSA

- RSA: Rivest, Shamir, Adleman
- Key size variable and much longer than secret keys
  - at least 1024 bits (250 decimal digits)
- Data block size is variable but smaller than key size
- Ciphertext block is same size as key size.
- Orders slower than symmetric crypto algorithms (eg, AES)
  - So use hybrid encryption for large messages

RSA: Generating [public key, private key] pair

RSA asymm

- Choose two large primes, p and q // keep p and q secret
- Let $n = p \cdot q$
- Choose e relatively prime to $\phi(n)$ // $\phi(n) = (p-1) \cdot (q-1)$
- Public key $= [e, n]$ // make this public
- Let $d =$ mult-inverse-mod-$\phi(n)$ of e // $e \cdot d$ mod-$\phi(n) = 1$
- Private key $= [d, n]$ // keep d secret

RSA: Encryption and Decryption

- Encryption of message *msg* using public key
  - $m \leftarrow$ add random pad to *msg* // PKCS, OASP
  - ciphertext $c \leftarrow m^e$ mod-n
- Note:
  - PKCS and OASP are padding standards
  - m must be less than n
- Decryption of ciphertext c using private key
  - plaintext $m \leftarrow c^d$ mod-n // coz $m^{e \cdot d}$ mod-n $=$ m
  - $msg \leftarrow$ remove pad from m

Why is $m^{e \cdot d}$ equal to $m \pmod{n}$

- $m^{e \cdot d}$ mod-n

$= m^{1+k \cdot \phi(n)}$ mod-n for some k // $e \cdot d$ mod-$\phi(n) = 1$

$= m$ mod-n // Euler's theorem

$= m$ // m in Z_n

RSA: Signing and Verifying

- Signing message *msg* using private key
  - $m \leftarrow$ add pad to *msg* // PKCS
  - signature $s \leftarrow m^d$ mod-n
- Verifying signature s using public key
  - $m \leftarrow s^e$ mod-n // coz $m^{e \cdot d}$ mod-$n = m$
  - YES iff m equals *msg* with pad

Why RSA is believed to be secure

- Only known way to obtain m from $x = m^e$ mod-n is by x^d mod-n where $d = e^{-1}$ mod-$\phi(n)$
- Only known way to obtain $\phi(n)$ is with p and q
- Factoring number is believed to be hard, so hard to obtain p and q given n
  - Best current algorithms: $\exp(n.len^{1/3})$
  - Currently $n.len$ of 1024 for OK security
  - Use $n.len$ of 2048 to be sure
  - Decade: $n.len$ of 3072 to be sure

RSA requires efficient modulo exponentation

- RSA operations (encrypt, decrypt, etc) require computing m^e mod-n for large (eg, 200-digit) numbers m, e, n
- Simple approach is not feasible
  - Multiply m with itself, take mod n; repeat e times.
  - e multiplications and divisions of large numbers.
- Much better:
  - Exploit $m^{2x} = m^x \cdot m^x$ and $m^{2x+1} = m^{2x} \cdot m$
  - $\log e$ multiplications and divisions

- $(x_0, x_1, \cdots, x_k) \leftarrow$ e in binary // $x_0 = 1$
- initially $y \leftarrow m$; $j \leftarrow 0$ // $y = m^{x_0}$
- while $j < k$
  - // loop invariant: $y = m^{(x_0, \cdots, x_j)}$ mod-n
  - $y \leftarrow y \cdot y$ mod-n; // $y = m^{(x_0, \cdots, x_j, 0)}$ mod-n
  - if $x_{j+1} = 1$
    $y \leftarrow y \cdot m$ mod-n // $y = m^{(x_0, \cdots, x_j, 1)}$ mod-n
  - $j \leftarrow j + 1$ // $y = m^{(x_0, \cdots, x_j)}$ mod-n
- // $y = m^e$ mod-n

~~Exam~~ Example: 123^{54} mod-678

- 54 in binary is $(1101110)_2$
- $123^{(1)}$ mod-678 $= 123$
- $123^{(10)}$ mod-678 $= 123{\cdot}123$ mod-678 $= 15129$ mod-678 $= 213$
- $123^{(11)}$ mod-678 $= 213{\cdot}123$ mod-678 $= 26199$ mod-678 $= 435$
- $123^{(110)}$ mod-678 $= 435{\cdot}435$ mod-678 $= 189225$ mod-678 $= 63$
- $123^{(1100)}$ mod-678 $= 63{\cdot}63$ mod-678 $= 3969$ mod-678 $= 579$
- $123^{(1101)}$ mod-678 $= 579{\cdot}123$ mod-678 $= 71217$ mod-678 $= 27$
- $123^{(11010)}$ mod-678 $= 27{\cdot}27$ mod-678 $= 729$ mod-678 $= 51$
- $123^{(11011)}$ mod-678 $= 51{\cdot}123$ mod-678 $= 6273$ mod-678 $= 171$
- $123^{(110110)}$ mod-678 $= 171{\cdot}171$ mod-678 $= 29241$ mod-678 $= 87$

RSA key generation is very expensive

- There are two parts to RSA key generation
  - Finding big primes p and q
  - Finding e relatively prime to $\phi(p \cdot q)$ // $= (p-1) \cdot (q-1)$
  - Note: given e, easy to obtain $d = e^{-1}$ mod-$\phi(n)$

Finding a big prime n

- Choose random n and test for prime. If not prime, retry.
- No practical deterministic test.
- Simple probabilistic test
  - Generate random n and random a in $1..n$
  - Pass if a^{n-1} mod-n $= 1$ // converse to Euler's theorem
  - Prob failure is low // $\approx 10^{-13}$ for 100-digit n
  - Can improve by trying different a's.
  - But Carmichael numbers: $561, 1105, 1729, 2465, 2821, 6601, \cdots$
- Miller-Rabin probabilistic test
  - Better and handles Carmichael numbers

Finding e relatively prime to $p \cdot q$

- Approach 1
  - Choose random primes p and q as described above
  - Choose e at random until e relatively prime to $\phi(p.q)$
- Approach 2
  - Fix e st m^e easy to compute (i.e., few 1's in binary)
  - Choose random primes p and q st e relatively prime to $\phi(p.q)$
  - Common choices
    - $e = 2^1 + 1 = 3$ // m^3 requires 2 multiplications
    - $e = 2^{16} + 1 = 65537$ // m^e requires 17 multiplications

PKCS: Public Key Cryptography Standard

- PKCS #1 v1.5
  - Defines padding of msg being encrypted/signed in RSA
  - Padded msg is 1024 bits
- Encryption (fields are octets)
  -

0	2	$\geq$ eight random non-zero octets	0	data

- Signing (fields are octets)
  -

0	1	$\geq$ eight $9F_{16}$ octets	0	digest type and digest

Outline

Overview

Symmetric Crypto
- Block Cipher
- Encryption Modes for Variable-size Messages
- Message Authentication Codes (MACs)
- MAC and Confidentiality

Asymmetric Crypto (aka Public-Key Crypto)
- Introduction
- A Little Bit of Number Theory
- RSA
- Diffie-Helman

DH: Diffie-Helman

- Establishes a key over open channel without a pre-shared secret
- Inputs (public): prime p and generator g for p
  - $1 < g < p$ st g^i mod-p ranges over $1, \cdots, p-1$
- Protocol

Alice	Bob
choose random x	
$A \leftarrow g^x$ mod-p	
send A	choose random y
	$B \leftarrow g^y$ mod-p
	send B
	$K \leftarrow A^y$ mod-p
$K \leftarrow B^x$ mod-p	

- Alice.K $=$ Bob.K $=$ $g^{x \cdot y}$ mod-p // shared key

Why is DH secure

- Hard to get $g^{x \cdot y}$ mod-p from p, g, g^x and g^y
  - Multiplying g^x and g^y yields g^{x+y} // not useful
  - Hard to get x from g^x mod-p // Discrete-log problem
  - Hard to get y from g^y mod-p

DH does not authenticate

- DH allows two principals who share nothing to establish a shared secret over an insecure channel
- DH does not authenticate the principals to each other
  - Alice may be talking to Trent claiming to be Bob
- For authentication, principals must already share something, eg:
  - Alice and Bob share a secret symmetric key
  - Alice and Bob each have the other's public key
  - Alice and Bob each share a key with a trusted third party
    - it generates a new key and sends it securely to Alice and Bob
    - it securely sends the public keys of Alice and Bob to the other

Authenticated DH using pre-shared keys

- DH that incorporates a pre-shared key to provide authentication

- Suppose Alice and Bob share a secret symmetric-crypto key k
- Can do authenticated DH by using k to encrypt the DH msgs
  - Alice sends $E(k, g^x \text{ mod-}p)$
  - Bob sends $E(k, g^y \text{ mod-}p)$
  - If principals are Alice and Bob: get shared key ($g^{x \cdot y}$ mod-p)
    Otherwise the principals would not achieve a shared key, so ok

- Can do similar authenticated DH if Alice and Bob have each other's public key

Why do DH given pre-shared secret

- If Alice and Bob share a secret key k, they can achieve secure communication simply by encrypting msgs with k
- What is gained by using k to do authenticated DH
  - The DH key would be strong whereas k may be weak (eg, obtained from a password)
  - Perfect-forward secrecy: If they forget their DH private keys after their session, then the session data remains secure even if k is later exposed