PURDUE
UNIVERSITY

CS526: Information Security
Prof. Sam Wagstaff

September 16, 2003

Cryptography Basics

Cryptography

- Basic assumptions
  - Message to be encrypted
  - Algorithms (publicly known) to encrypt/decrypt message
  - Key (known only to sender/recipient)
  - *Given only algorithms and encrypted message, nobody knows a method to decrypt that is significantly faster than trying all keys*
- Types of attacks
  - ciphertext only
  - known plaintext
  - chosen plaintext
- Real attacks generally don't break cryptography!
  - *Don't pick the lock, tunnel into the vault*

Secret-Key (Symmetric) Cryptography: Uses

- Prevent eavesdropping
  - Must be secure channel for key exchange
- Secure storage
  - I have to remember my key
- Authentication
  - Challenge/response
  - *Be careful*
- Integrity Check
  - Checksum on the message
  - Encrypt the checksum

Public Key (Assymetric) Cryptography

- First published in 1976 (Diffie-Hellman)
  - More common today: RSA
- Matched pair of keys
  - Public key (*e*) to encrypt
  - Private key (*d*) to decrypt
- For integrity, encrypt checksum with sender's private key
  - Only sender's public key will decrypt properly

Public-Key Cryptography: Uses

- Prevent eavesdropping
- Authentication
- Integrity
- Problem: public key algorithms slow
  - Solution: Use to share secret key

Public Key Cryptography: Non-repudiation

- Message Integrity Checksum (MAC) can convince Recipient that Sender created message
  - Message correct, from right source
- But can't convince anyone else!
  - Sender, recipient share key
  - Either could generate message
- Public key solves this problem
  - Private key required to encrypt
  - Only known to sender

Hash Algorithms

- Transform arbitrarily long message *m* into (short) fixed-length message *h(m)*
  - Must be easy to compute *h(m)*
  - Given *h(m)*, hard to find (an) *m*
  - Hard to find m_1 and m_2 such that $h(m_1)=h(m_2)$
- Uses
  - Password storage (easy to verify that it is probably correct)
  - Integrity: Send *m*, *h(m|s)*
  - Storage integrity

Cryptographic Algorithms What have you covered?

- DES
  - 3DES
- IDEA
- AES
- One-time Pad
  - RC4

Cryptography: Algorithms

- Block encryption: Turn fixed-length block into fixed-length *e(*block*)*
  - Block needs to be large enough to prevent "discovery" of block/*e(*block*)* pairs
  - 64 bits seems adequate in practice
- Goal: appear random
  - Changing one input bit should change each output bit with probability ½
- Approaches:
  - Substitution: Table mapping input to output
  - Permutation: Move bits around
- Do (small) substitutions and permutations in rounds

Encrypting More…

- Electronic Code Book
  - Obvious: Just encrypt each block
  - Leaks information
  - Open to tampering
- Cipher Block Chaining
- *k*-Bit Cipher Feedback Mode
- *k*-Bit Output Feedback Mode
- Counter Mode

Cipher Block Chaining

- Xor first block with 64-bit random before encryption
  - Send random “in the clear”
- Xor each block with previous encrypted block before encryption
- Ensures
  - Identical blocks different in transmitted message
  - A repeated message will look different each time
- Problem: tampering
  - Tampering with one block makes predictable change in the next
  - But destroys first block

Output Feedback Mode

- Use DES to generate one-time pad
  - Start with random value
  - Encrypt with DES to get pad
  - *m* xor pad to encrypt
  - Encrypt pad to get next pad
- Fast, resilient, can stream results bit at a time
- If adversary knows plaintext, ciphertext, can tamper to produce desired result!

Cipher Feedback Mode

- One-time pad like OFB
  - But use ciphertext, not previous pad, to get new pad
- Tampering garbles following block
  - Better than OFB
  - But not as good as CBC
- Counter Mode
  - Increment random before encryption to get next pad

Encryption to generate Message Authentication Codes

- Use CBC
  - Xor each block with previous cipher
  - Then encrypt
- Final block is integrity code
  - Will change if any block changes, or key changes
- Requires sending the plaintext message

Integrity & confidentiality

- Idea: Encrypt, then checksum on encrypted message
  - Requires twice as much encryption!
  - Can we do better?
- Solution: Weak checksum then encrypt
  - Adversary can't see weak checksum to attack it

Hash Algorithms (Message Digest)

- Transform arbitrarily long message *m* into (short) fixed-length message *h(m)*
  - Must be easy to compute *h(m)*
  - Given *h(m)*, hard to find (an) *m*
  - Hard to find m_1 and m_2 such that $h(m_1)=h(m_2)$
- Goal: *h(m)* should appear random
  - Non-trivial to define "appear random"

(Strange) Hash Uses

- Authentication
  - A sends challenge r_A
  - B responds with $h(k|r_A)$ and r_B
  - A responds with $h(k|r_B)$
- Integrity / Message Authentication Code
  - $h(m \mid k)$
- Generate a one-time pad
  - $h(k \mid r)$ gives first block, then $h(k \mid b_{i-1})$ gives b_i
- Can also generate a hash using symmetric encryption

Hashing (MD5): How it Works

- Basic idea: Continuously update hash value with 512 bit blocks of message
  - 128 bit initial value for hash
  - Bit operations to "compress"
- Compression function: Update 128 bit hash with 512 bit block
  - Pass 1: Based on bits in first word, select bits in second or third word
  - Pass 2: Repeat, selecting based on last word
  - Pass 3: xor bits in words
  - Pass 4: y xor (x or ~z)

Public Key Cryptography

- Public key d, private key e
  - m = e(d(m)) = d(e(m))
- Given d, d(m), hard to find m
  - same for e, e(m)
- Given d, hard to find e
  - same for e, d
- Most based on modular arithmetic
  - Modular exponentiation

Algorithms: Diffie-Hellman

- Goal: Two parties agree on common number
  - E.g., learn shared key
- Initial: large prime p, g < p
  - publicly known
- Each chooses secret
- $T = g^s \bmod p$
- Exchange and repeat
  - Result is the same

Diffie-Helman: Problems

- Authentication
  - Am I talking to the right person?
- Man in the middle
  - Sets up session with either end

Algorithms: RSA (Rivest, Shamir, Adleman)

- Key generation
  - Choose primes p,q
  - Choose e relatively prime to (p-1)(q-1)
  - Public key <e,n>
  - Private key <d,n> where d = 1/(e mod (p-1)(q-1))
- Encrypt: $c = m^e \bmod n$
  - Decrypt: $m = c^d \bmod n$
- $de = 1 \bmod (p-1)(q-1)$, so $m = (m^e)^d \bmod n$
- Breakable if we can factor (why?)

Problems with RSA

- Probing
  - If I get e(m), I can check if m=m'
  - Solution: random pad
- Efficiency: Key concepts
  - $x^e \bmod n = (x * x) \bmod n * x^{e-1} \bmod n$
  - $x^{2(e/2)}$ = left shift of $x^{(e/2)}$
- Generating keys expensive
  - Select large primes
  - Find e relatively prime to (p-1)(q-1)
    - In practice, e=65537
- Any x<n is a valid signature
  - Also, given a signatures for m1, m2; can compute signature for (some) other messages

Public-Key Cryptography Standard

- Encryption Format
  - Octal: 0 2 (eight random values) 0 data
  - Data is typically a "session key"
- Signature Format
  - 0 1 (64 bits of ones) 0 hash

Digital Signature Standard

- ElGamal-based algorithm
  - Diffie-Helman style