CIS551: Computer and Network Security

Jonathan M. Smith

jms@cis.upenn.edu

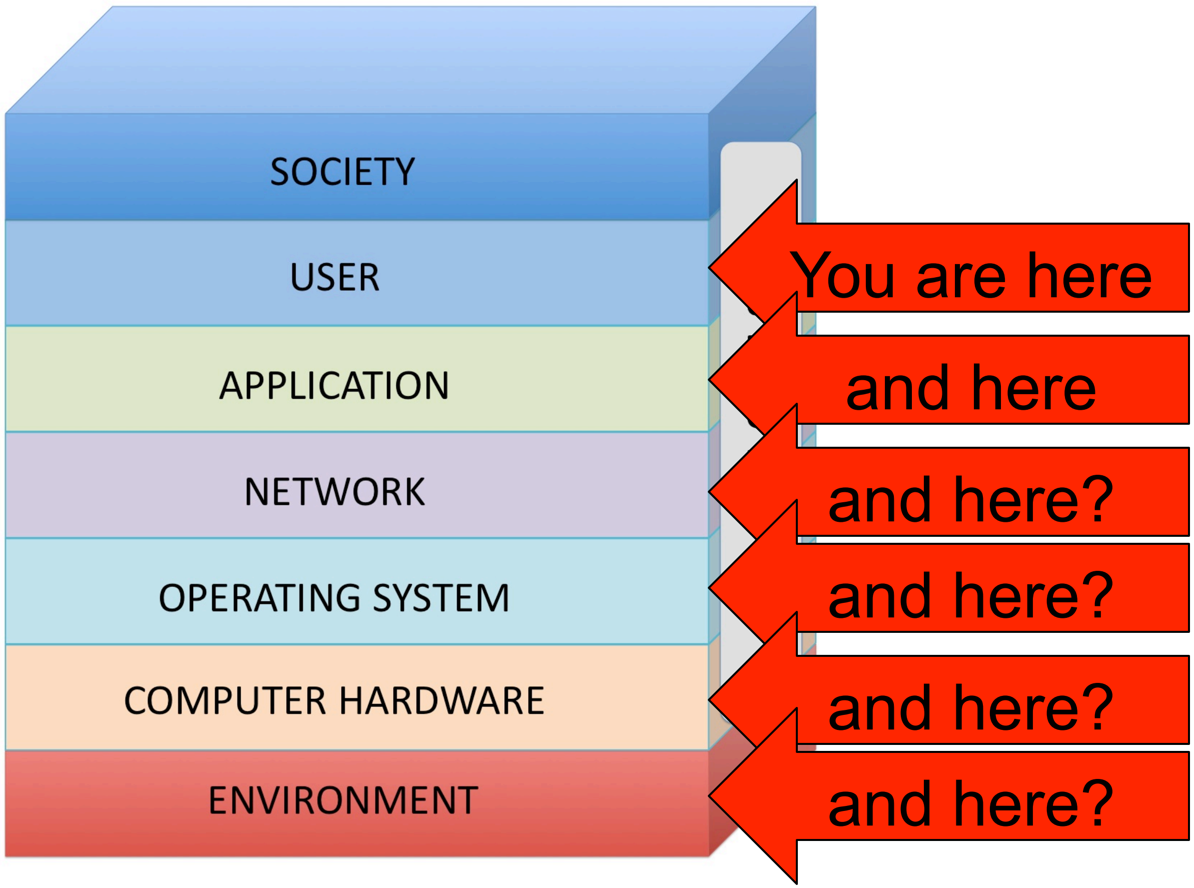
03/19/2014

Uses material from S. Zdancewic/C. Gunter

CIS551 Topics

- Computer Security
  - Software/Languages, Computer Arch.
  - Access Control, Operating Systems
  - Threats: Vulnerabilities, Viruses
- Computer Networks
  - Physical layers, Internet, WWW, Applications
  - Cryptography in several forms
  - Threats: Confidentiality, Integrity, Availability
- Systems Viewpoint
  - Users, social engineering, insider threats

Sincoskie NIS model

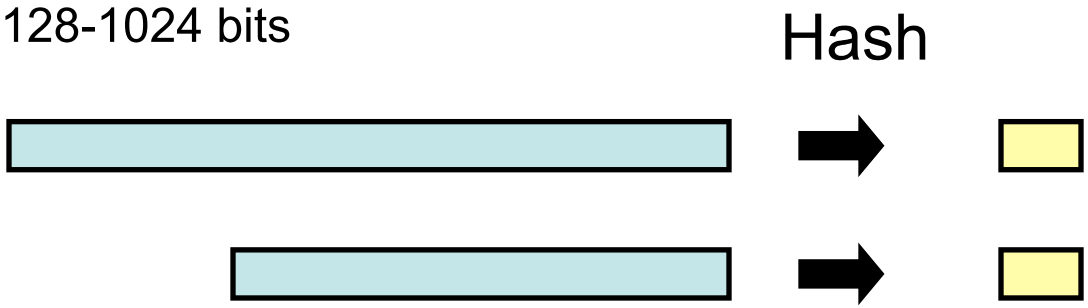


W.D. Sincoskie, *et al.* “Layer Dissonance and Closure in Networked Information Security” (white paper)

Uses material from S. Zdancewic/C. Gunter

Hash Algorithms

- Take a variable length string
- Produce a fixed length digest
  - Typically 128-1024 bits



- (Noncryptographic) Examples:
  - Parity (or byte-wise XOR)
  - CRC (cyclic redundancy check) used in communications
  - Ad hoc hashes used for hash tables
- Realistic Example
  - The NIST Secure Hash Algorithm (SHA) takes a message of less than 2^{64} bits and produces a digest of 160 bits

Cryptographic Hashes

- Create a hard-to-invert summary of input data
- Useful for integrity properties
  - Sender computes the hash of the data, transmits data and hash
  - Receiver uses the same hash algorithm, checks the result
- Like a check-sum or error detection code
  - Uses a cryptographic algorithm internally
  - More expensive to compute
- Sometimes called a Message Digest
- History:
  - Message Digest (MD4 -- invented by Rivest, MD5)
  - Secure Hash Algorithm - 1993 - (SHA-0)
  - Secure Hash Algorithm (SHA-1)
  - SHA-2 (actually a family of hash algorithms with varying output sizes)
  - SHA-3 - 2012 winner of competition, not yet standardized by NIST

- Attacks on SHA-0 + SHA-1 exist, but not SHA-2 (yet)

Uses material from S. Zdancewic/C. Gunter

Uses of Hash Algorithms

- Hashes are used to protect *integrity* of data
  - Virus Scanners
  - Program fingerprinting in general
  - Modification Detection Codes (MDC)
- Message Authenticity Code (MAC)
  - Includes a cryptographic component
  - Send (msg, hash(msg, key))
  - Attacker who doesn't know the key can't modify msg (or the hash)
  - Receiver who knows key can verify origin of message
- Make digital signatures more efficient (we'll see this later)

Desirable Properties

- The probability that a randomly chosen message maps to an n-bit hash should ideally be $(½)^n$.
  - Attacker must spend a lot of effort to be able to modify the source message without altering the hash value

- Hash functions *h* for cryptographic use as MDC's fall in one or both of the following classes.
  - *Collision Resistant Hash Function:* It should be computationally infeasible to find two distinct inputs that hash to a common value (ie. $h(x) = h(y)$).
  - *One Way Hash Function:* Given a specific hash value *y*, it should be computationally infeasible to find an input x such that $h(x)=y$.

Secure Hash Algorithm (SHA)

- Pad message so it can be divided into 512-bit blocks, including a 64 bit value giving the length of the original message.
- Process each block as 16 32-bit words called *W(t)* for *t* from 0 to 15.
- Expand from these 16 words to 80 words by defining as follows for each t from 16 to 79:
  - $W(t) := W(t-3) \oplus W(t-8) \oplus W(t-14) \oplus W(t-16)$
- Constants H0, …, H5 are initialized to special constants
- Result is final contents of H0, … , H5

```
for each 16-word block begin
    A := H0; B := H1; C := H2; D := H3; E := H4
    for I := 0 to 19 begin
        TEMP := S(5,A) + ((B ∧ C) ∨ (¬ B ∧ D)) + E + W(I) + 5A827999;
        E := D; D := C; C := S(30,B); B := A; A := TEMP
    end
    for I := 20 to 39 begin
        TEMP := S(5,A) + (B ⊕ C ⊕ D) + E + W(I) + 6ED9EBA1;
        E := D; D := C; C := S(30,B); B := A; A := TEMP
    end
    for I := 40 to 59 begin
        TEMP := S(5,A) + ((B ∧ C) ∨ (B ∧ D) ∨ (C ∧ D)) + E + W(I) + 8F1BBCDC;
        E := D; D := C; C := S(30,B); B := A; A := TEMP
    end
    for I := 60 to 79 begin
        TEMP := S(5,A) + (B ⊕ C ⊕ D) + E + W(I) + CA62C1D6;
        E := D; D := C; C := S(30,B); B := A; A := TEMP
    end
    H0 := H0+A; H1 := H1+B; H2 := H2+C; H3 := H3+D; H4 := H4+E
end
```

Chaining Variables

Shift A left 5 bits

Uses material from S. Zdancewic/C. Gunter

Attacks against SHA-1

- In early 2005, Rijmen and Oswald published an attack on a reduced version of SHA-1 (53 out of 80 rounds) which finds collisions with a complexity of fewer than 2^{80} operations.
- In February 2005, an attack by Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu was announced. The attacks can find collisions in the full version of SHA-1, requiring fewer than 2^{69} operations (brute force would require 2^{80}.)
- In August 2005, same group lowered the threshold to 2^{63}.
- May lead to more attacks…

Problems with Shared Key Crypto

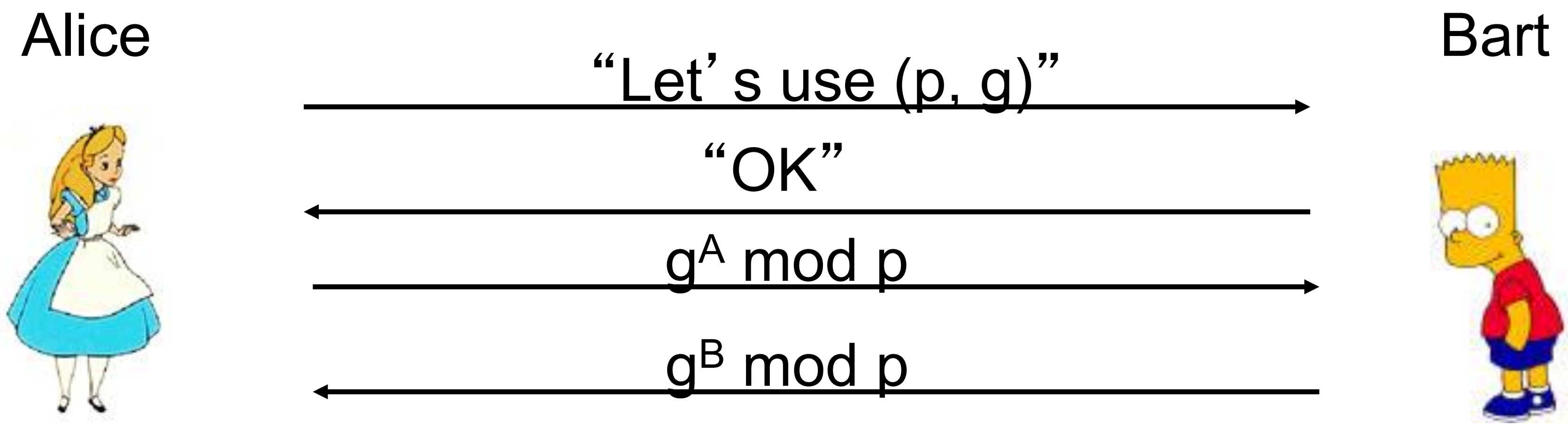
- Compromised key means interceptors can decrypt any ciphertext they've acquired.
  - Change keys frequently to limit damage
- Distribution of keys is problematic
  - Keys must be transmitted securely
  - Use couriers?
  - Distribute in pieces over separate channels?
- Number of keys is $O(n^2)$ where n is # of participants
- Potentially easier to break?

Diffie-Hellman Key Exchange

- Choose a prime p (publicly known)
  - Should be about 512 bits or more
- Pick g < p (also public)
  - g must be a *primitive root* of p.
  - A primitive root *generates* the finite field p.
  - Every n in {1, 2, …, p-1} can be written as g^k mod p
  - Example: 2 is a primitive root of 5
  - $2^0 = 1$ $2^1 = 2$ $2^2 = 4$ $2^3 = 3$ (mod 5)

  - Intuitively means that it's hard to take logarithms base g because there are many candidates.

Diffie-Hellman



1. Alice & Bart decide on a public prime p and primitive root g.
2. Alice chooses secret number A. Bart chooses secret number B
3. Alice sends Bart $g^A \bmod p$.
4. The shared secret is $g^{AB} \bmod p$.

Uses material from S. Zdancewic/C. Gunter

Details of Diffie-Hellman

- Alice computes g^{AB} mod p because she knows A:
  - g^{AB} mod p = $(g^{B}$ mod p$)^{A}$ mod p

- An eavesdropper gets g^{A} mod p and g^{B} mod p
  - They can easily calculate g^{A+B} mod p but that doesn't help.
  - The problem of computing discrete logarithms (to recover A from g^{A} mod p) is hard.

Uses material from S. Zdancewic/C. Gunter

Example

- Alice and Bart agree that $p=71$ and $g=7$.
- Alice selects a private key $A=5$ and calculates a public key $g^A \equiv 7^5 \equiv 51 \pmod{71}$. She sends this to Bart.
- Bart selects a private key $B=12$ and calculates a public key $g^B \equiv 7^{12} \equiv 4 \pmod{71}$. He sends this to Alice.
- Alice calculates the shared secret: $S \equiv (g^B)^A \equiv 4^5 \equiv 30 \pmod{71}$
- Bart calculates the shared secret $S \equiv (g^A)^B \equiv 51^{12} \equiv 30 \pmod{71}$

Why Does it Work?

- Security is provided by the difficulty of calculating discrete logarithms.
- Feasibility is provided by
  - The ability to find large primes.
  - The ability to find primitive roots for large primes.
  - The ability to do efficient modular arithmetic.
- Correctness is an immediate consequence of basic facts about modular arithmetic.

One-way Functions

- A function is one-way if it’s
  - Easy to compute
  - Hard to invert (in the average case)

- Examples
  - Exponentiation vs. Discrete Log
  - Multiplication vs. Factoring
  - Knapsack Packing
    - Given a set of numbers {1, 3, 6, 8, 12} find the sum of a subset
    - Given a target sum, find a subset that adds to it

- Trapdoor functions
  - Easy to invert given some extra information
  - E.g. factoring p*q given q

Uses material from S. Zdancewic/C. Gunter

Public Key Cryptography

- Sender encrypts using a *public key*
- Receiver decrypts using a *private key*
- Only the private key must be kept secret
  - Public key can be distributed at will
- Also called *asymmetric* cryptography
- Can be used for *digital signatures*
- Examples: RSA, El Gamal, DSA, various algorithms based on elliptic curves
- Used in SSL, ssh, PGP, …

Public Key Notation

- Encryption algorithm

  E : keyPub x plain → cipher

  Notation: K{msg} = E(K, msg)

- Decryption algorithm

  D : keyPriv x cipher → plain

  Notation: k{msg} = D(k,msg)

- D inverts E

  D(k, E(K, msg)) = msg

- Use *capital* "K" for public keys
- Use *lower case* "k" for private keys

- Sometimes E is the same algorithm as D

Uses material from S. Zdancewic/C. Gunter

Secure Channel

Alice

Bart

$K_B\{Hello!\}$ →

← $K_A\{Hi!\}$

K_A, K_B
k_A

K_A, K_B
k_B

Uses material from S. Zdancewic/C. Gunter

Trade-offs for Public Key Crypto

- More computationally expensive than shared key crypto
  - Algorithms are harder to implement
  - Require more complex machinery
- More formal justification of difficulty
  - Hardness based on complexity-theoretic results
- A principal needs one private key and one public key
  - Number of keys for pair-wise communication is O(n)

RSA Algorithm

- Ron Rivest, Adi Shamir, Leonard Adleman
  - Proposed in 1979
  - They won the 2002 Turing award for this work
- Has withstood years of cryptanalysis
  - Not a guarantee of security!
  - But a strong vote of confidence.
- Hardware implementations: 1000 x slower than DES

RSA at a High Level (more later)

- Public and private key are derived from secret prime numbers
  - Keys are typically ≥ 1024 bits
- Plaintext message (a sequence of bits)
  - Treated as a (large!) binary number
- Encryption is modular exponentiation
- To break the encryption, conjectured that one must be able to factor large numbers
  - Not known to be in P (polynomial time algorithms)
  - Is known to be in BQP (bounded-error, quantum polynomial time – Shor's algorithm)