CIS 551 / TCOM 401

Computer and Network Security

Spring 2006

Lecture 10

Announcements

- Project 2 is available on the web.
  - Due: March 14, 2006

- A new mail alias has been set up. Send all course-related e-mail to: cis551staff@seas.upenn.edu

Recap

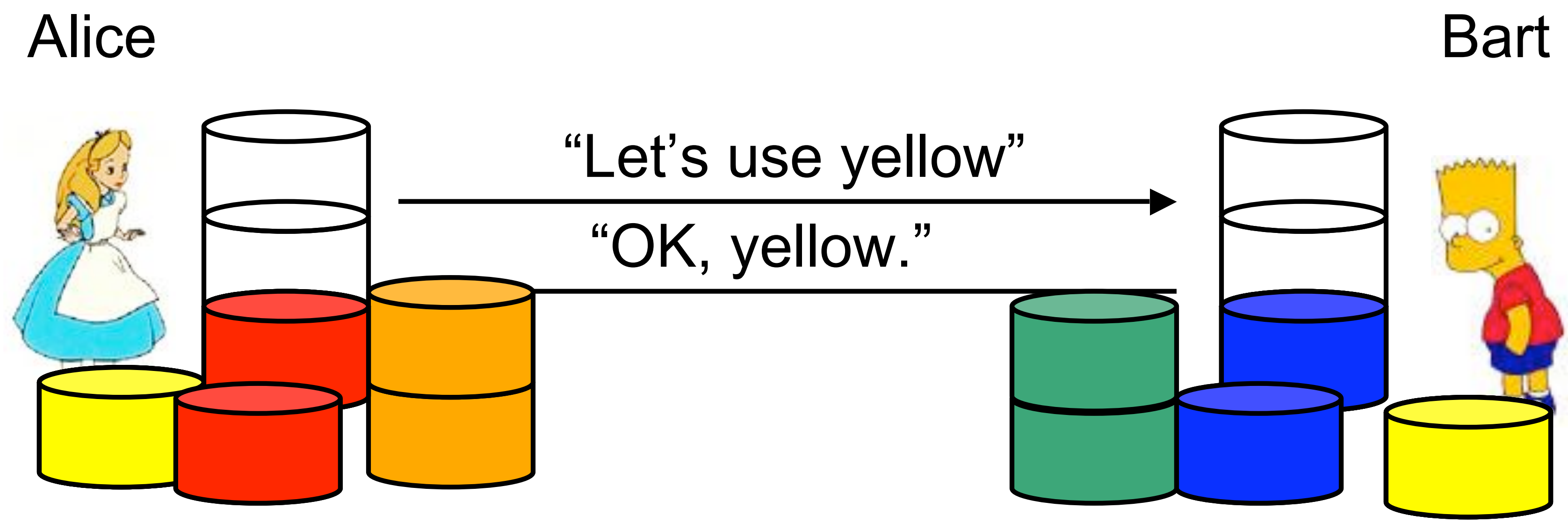
- Last time:
  - RSA
- Today:
  - Diffie Hellman key exchange
  - Cryptographic Hashes
  - Start Digital Signatures

Diffie-Hellman Key Exchange

- Problem with shared-key systems: Distributing the shared key
- Suppose that Alice and Bart want to agree on a secret (i.e. a key)
  - Communication link is public
  - They don't already share any secrets

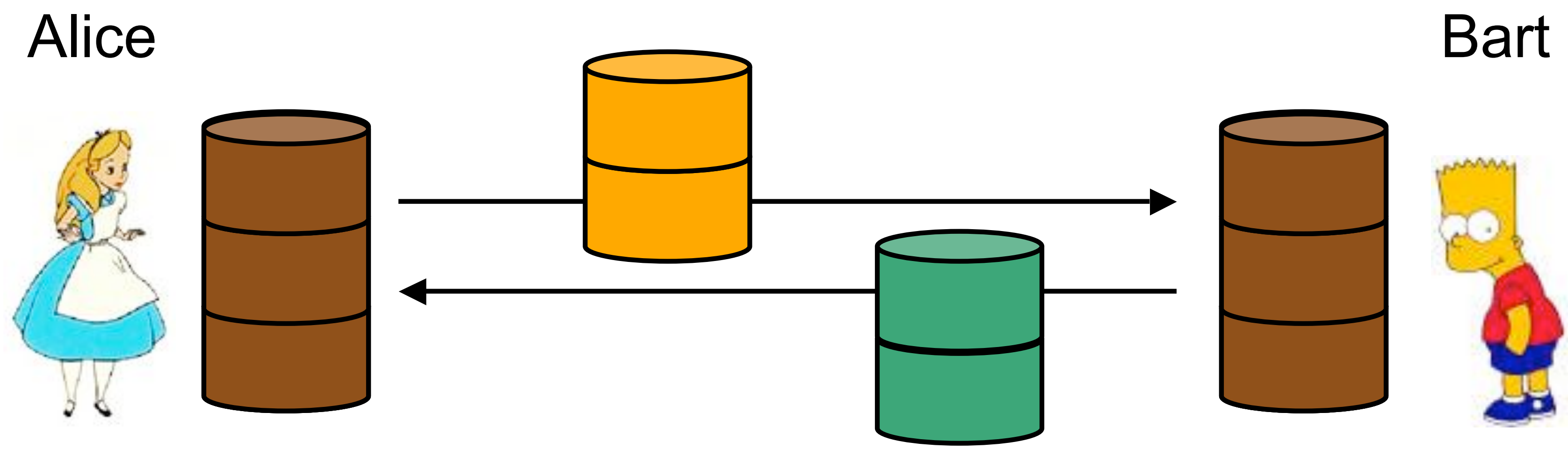
- First public key protocol developed
- Proposed by Whitfield Diffie and Martin Hellman in 1976
  - (Although, again, was known by the British intelligence agency!)

Diffie-Hellman by Analogy: Paint



1. Alice & Bart decide on a public color, and mix one liter of that color.
2. They each choose a random secret color, and mix two liters of their secret color.
3. They keep one liter of their secret color, and mix the other with the public color.

Diffie-Hellman by Analogy: Paint



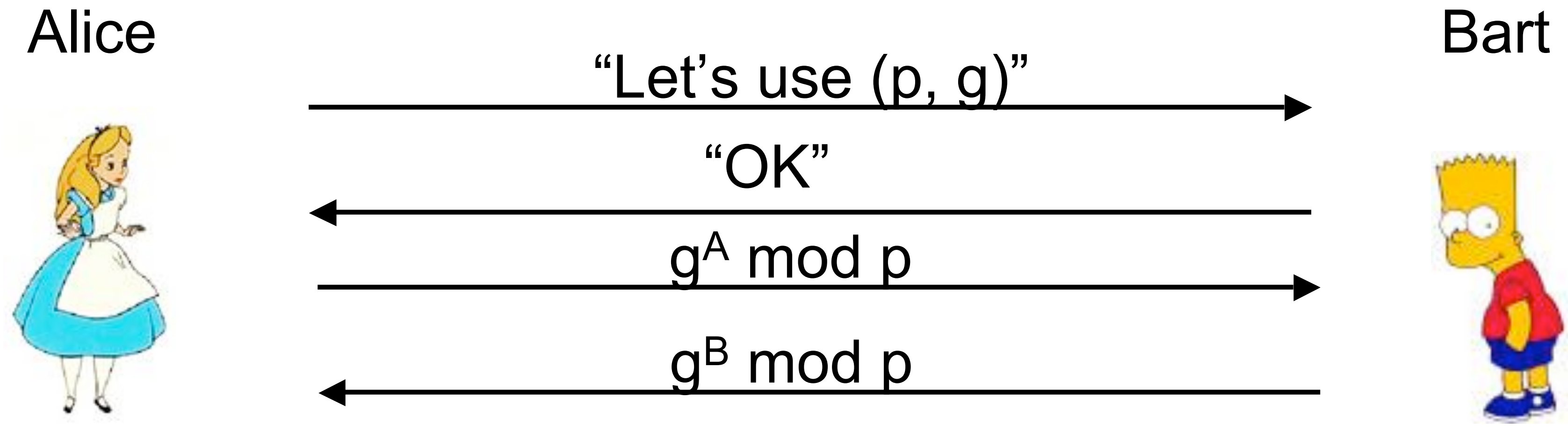
4. They exchange the mixtures over the public channel.
5. When they get the other person’s mixture, they combine it with their retained secret color.
6. The secret is the resulting color: Public + Alice’s + Bart’s

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 $g^{AB} \bmod p$.

Details of Diffie-Hellman

- Alice computes $g^{AB} \bmod p$ because she knows A:
  - $g^{AB} \bmod p = (g^B \bmod p)^A \bmod p$
- An eavesdropper gets $g^A \bmod p$ and $g^B \bmod p$
  - They can easily calculate $g^{A+B} \bmod p$ but that doesn't help.
  - The problem of computing discrete logarithms (to recover A from $g^A \bmod p$ is hard.

Example

- Alice and Bart agree that $q=71$ 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.

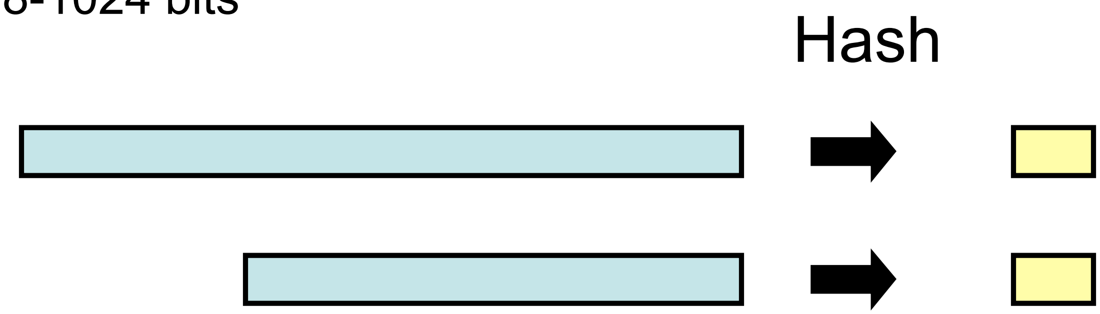
Man-in-the-middle Attack

- As stated, Diffie-Hellman doesn't provide authentication.
- So, an attacker could intercept the messages and impersonate one of the end points.

- (See chalk board)

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)

- Attacks have been found against both SHA-0 and SHA-1

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 shortly)

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
```

Chaining Variables

```
    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
```

Shift A left 5 bits

```
    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
```

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…