Announcements

• No class on Monday (fall break)
Recap

• Last time:
  – RSA discussion
  – Diffie-Hellman key exchange

• Today:
  – Finish Diffie Hellman
  – Cryptographic Hashes
  – Start Digital Signatures
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 \textit{primitive root} of $p$.
  – A primitive root generates the finite field $p$.
  – Every $n$ in $\{1, 2, \ldots, p-1\}$ can be written as $g^k \mod p$
  – Example: 2 is a primitive root of 5
  – $2^0 = 1 \quad 2^1 = 2 \quad 2^2 = 4 \quad 2^3 = 4 \pmod{5}$

  – Intuitively means that it’s hard to take logarithms base $g$ because there are many candidates.
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 $g^A \mod p$.  
4. The shared secret is $g^{AB} \mod p$. 

"Let’s use $(p, g)$"

"OK"

$g^A \mod p$

$g^B \mod p$
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.
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.
Hash Algorithms

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

• (Noncryptographic) Examples we’ve already seen:
  – Parity (or byte-wise XOR)
  – CRC

• 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
• Examples:
  – Secure Hash Algorithm (SHA)
  – Message Digest (MD4, MD5)
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
Desirable Properties

• The probability that a randomly chosen message maps to an n-bit hash should ideally be \((\frac{1}{2})^n\).
  – 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 (i.e. \(h(x) = h(y)\)).
  – **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 $H_0$, $\ldots$, $H_5$ are initialized to special constants
- Result is final contents of $H_0$, $\ldots$, $H_5$
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 \land C) \lor (\neg B \land 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 \oplus C \oplus 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 \land C) \lor (B \land D) \lor (C \land 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 \oplus C \oplus 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
Physical Signatures

- Consider a paper check used to transfer money from one person to another
- Signature confirms authenticity
  - Only legitimate signer can produce signature
- In case of alleged forgery
  - 3rd party can verify authenticity
- Checks are cancelled
  - So they can’t be reused
- Checks are not alterable
  - Or alterations are easily detected
Digital Signatures: Requirements I

- A mark that only one principal can make, but others can easily recognize
- Unforgeable
  - If P signs a message M with signature S\{P,M\} it is impossible for any other principal to produce the pair (M, S\{P,M\}).
- Authentic
  - If R receives the pair (M, S\{P,M\}) purportedly from P, R can check that the signature really is from P.
Digital Signatures: Requirements II

• Not alterable
  – After being transmitted, \((M,S\{P,M\})\) cannot be changed by P, R, or an interceptor.

• Not reusable
  – A duplicate message will be detected by the recipient.
Digital Signatures with Shared Keys

Alice \rightarrow K_{AT}\{msg\} \rightarrow Tom \rightarrow K_{TB}\{Alice, msg, K_{AT}\{msg\}\} \rightarrow Bart

Tom is a trusted 3rd party (or arbiter).

**Authenticity:** Tom verifies Alice’s message, Bart trusts Tom.

**No Forgery:** Bart can keep msg, $K_{AT}\{msg\}$, which only Alice (or Tom, but he’s trusted not to) could produce.
Preventing Reuse and Alteration

• To prevent reuse of the signature
  – Incorporate a *timestamp* (or sequence number)

• Alteration
  – If a block cipher is used, recipient could splice-together new messages from individual blocks.

• To prevent alteration
  – Timestamp must be part of each block
  – Or… use *cipher block chaining*
Digital Signatures with Public Keys

- Assumes the algorithm is commutative:
  \[ D(E(M, K), k) = E(D(M, k), K) \]
- Let \( K_A \) be Alice’s public key
- Let \( k_A \) be her private key
- To sign msg, Alice sends \( D(msg, k_A) \)
- Bart can verify the message with Alice’s public key

- Works! RSA: \( (m^e)^d = m^{ed} = (m^d)^e \)
Digital Signatures with Public Keys

- No trusted 3rd party.
- Simpler algorithm.
- More expensive
- No confidentiality
Variations on Public Key Signatures

• Timestamps again (to prevent replay)
  – Signed certificate valid for only some time.

• Add an extra layer of encryption to guarantee confidentiality
  – Alice sends $K_B{k_A\{msg\}}$ to Bart

• Combined with hashes:
  – Send $(msg, k_A\{MD5(msg)\})$