Goals for Today

- Symmetric
- Reminder: Midterm on Friday. (Closed book.)
  - Contents up through the material for today
  - Not as hard as last year's midterm.
  - Make sure you understand the core concepts so far in this course:
    - Threat modeling
    - Software security
    - Problems
    - Defensive approaches
    - Symmetric cryptography
  - Components, definitions, security properties, classic problems

Which Property Do We Need?

- UNIX passwords stored as hash(password)
  - One-wayness: hard to recover password
- Integrity of software distribution
  - Weak collision resistance
  - But software images are not really random... maybe need full collision resistance
- Auction bidding
  - Alice wants to bid B, sends H(B), later reveals B
  - One-wayness: rival bidders should not recover B
  - Collision resistance: Alice should not be able to change her mind to bid B' such that H(B)=H(B')

Common Hash Functions

- MD5
  - 128-bit output
  - Designed by Ron Rivest, used very widely
  - Collision-resistance broken (summer of 2004)
- RIPEMD-160
  - 160-bit variant of MD5
- SHA-1 (Secure Hash Algorithm)
  - 160-bit output
  - US government (NIST) standard as of 1993-95
    - Also the hash algorithm for Digital Signature Standard (DSS)
Basic Structure of SHA-1 (Skip)

Split message into 512-bit blocks

Compression function

• Applied to each 512-bit block
• This is the heart of SHA-1

160-bit buffer (5 registers)

• Initialized with magic values

SHA-1 Compression Function (Skip)

Current message block

Current buffer (five 32-bit registers A,B,C,D,E)

Buffer contains final hash value

Very similar to a block cipher, with message itself used as the key for each round

Four rounds, 20 steps in each

Let's look at each step in more detail…

Fifth round adds the original buffer to the result of 4 rounds

A
E
B
C
D

A
E
B
C
D

+ f t
5 bitwise left-rotate

W t K t

One Step of SHA-1 (80 steps total) (Skip)

Special constant added (same value in each 20-step round, 4 different constants altogether)

Logic function for steps

• \((B \land C) \lor (\neg B \land D)\) 0..19
• \(B \oplus C \oplus D\) 20..39
• \((B \land C) \lor (B \land D) \lor (C \land D)\) 40..59
• \(B \oplus C \oplus D\) 60..79

Current message block mixed in

• For steps 0..15, \(W^{0..15}\) = message block
• For steps 16..79, \(W^t = W^{t-16} \oplus W^{t-14} \oplus W^{t-8} \oplus W^{t-3} + f\)

Multi-level shifting of message blocks

30 bitwise left-rotate

How Strong Is SHA-1?

• Every bit of output depends on every bit of input
  • Very important property for collision-resistance
• Brute-force inversion requires \(2^{160}\) ops, birthday attack on collision resistance requires \(2^{80}\) ops
• Some very recent weaknesses (2005)
  • Collisions can be found in \(2^{63}\) ops
Authentication Without Encryption

Integrity and authentication: only someone who knows KEY can compute MAC for a given message.

HMAC

- Construct MAC by applying a cryptographic hash function to message and key
  - Could also use encryption instead of hashing, but...
  - Hashing is faster than encryption in software
  - Library code for hash functions widely available
  - Can easily replace one hash function with another
  - There used to be US export restrictions on encryption

Invented by Bellare, Canetti, and Krawczyk (1996)
- HMAC strength established by cryptographic analysis
- Mandatory for IP security, also used in SSL/TLS

Structure of HMAC

- Embed hash function (strength of HMAC relies on strength of this hash function)
- “Black box”: can use this HMAC construction with any hash function (why is this important?)
- Block size of embedded hash function
- Secret key padded to block size
- Magic value (flips half of key bits)
- Another magic value (flips different key bits)
- hash(key,hash(key,message))
- “Amplify” key material (get two keys out of one)

Very common problem: given a small secret, how to derive a lot of new keys?

Achieving Both Privacy and Integrity

Authenticated encryption scheme
Recall: Often desire both privacy and integrity. (For SSH, SSL, IPsec, etc.)
Some subtleties! Encrypt-and-MAC

Natural approach for authenticated encryption: Combine an encryption scheme and a MAC.

\[ E_{Ke,Km}(M) \]
\[ D_{Ke,Km}(C) \]

But insecure! \([BN, Kra]\)

Assume Alice sends messages:

\[ E_{Ke}(M_1) \text{ MAC}_{Km}(T_1) \]
\[ E_{Ke}(M_2) \text{ MAC}_{Km}(T_2) \]
\[ E_{Ke}(M_3) \text{ MAC}_{Km}(T_3) \]

If \( T_1 = T_3 \), then \( M_1 = M_3 \)

Adversary learns whether two plaintexts are equal.

Especially problematic when \( M_1, M_2, \ldots \) take on only a small number of possible values.

The Secure Shell (SSH) protocol is designed to provide:

- Secure remote logins.
- Secure file transfers.

Where security includes:

- Protecting the privacy of users' data.
- Protecting the integrity of users' data.

OpenSSH is included in the default installations of OS X and many Linux distributions.

Authenticated encryption in SSH

\[ E_{Ke,Km}(M) \]

Data to be communicated
Assume Alice sends messages $M_1$ and $M_2$ that are the same. Then the tags $T_1$ and $T_2$ will be different with high probability. But if counters repeat, tags may once again leak private information about data.

Results of [BN00,Kra01]

<table>
<thead>
<tr>
<th>Privacy</th>
<th>Strong (CTXT)</th>
<th>Weak (PTXT)</th>
<th>Insecure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integrity</td>
<td>Strong (CIC)</td>
<td>Weak (CPM)</td>
<td>Insecure</td>
</tr>
</tbody>
</table>

Basic Problem

**Given:** Everybody knows Bob’s public key Only Bob knows the corresponding private key

**Goals:**
1. Alice wants to send a secret message to Bob
2. Bob wants to authenticate himself

Applications of Public-Key Crypto

- Encryption for confidentiality
  - Anyone can encrypt a message
  - With symmetric crypto, must know secret key to encrypt
  - Only someone who knows private key can decrypt
  - Key management is simpler (maybe)
  - Secret is stored only at one site: good for open environments

- Digital signatures for authentication
  - Can “sign” a message with your private key
  - Session key establishment
  - Exchange messages to create a secret session key
  - Then switch to symmetric cryptography (why?)
Diffie-Hellman Protocol (1976)

- Alice and Bob never met and share no secrets
- Public info: p and g
  - p is a large prime number, g is a generator of $\mathbb{Z}_p^*$
  - $\mathbb{Z}_p^*$={1, 2, ..., p−1}; i.e. $\mathbb{Z}_p^*$ B- such that $a^m \mod p$
  - Modular arithmetic: numbers "wrap around" after they reach p

Pick secret, random X

Pick secret, random Y

$g^y \mod p$

$g^x \mod p$

Compute k=(g^y)^x = g^{xy} \mod p$

Compute k=(g^x)^y = g^{xy} \mod p

Why Is Diffie-Hellman Secure?

- Discrete Logarithm (DL) problem:
  - given $g^x \mod p$, it's hard to extract x
    - There is no known efficient algorithm for doing this
    - This is not enough for Diffie-Hellman to be secure!

- Computational Diffie-Hellman (CDH) problem:
  - given $g^x$ and $g^y$, it's hard to compute $g^{xy} \mod p$
  - ... unless you know x or y, in which case it's easy

- Decisional Diffie-Hellman (DDH) problem:
  - given $g^x$ and $g^y$, it's hard to tell the difference between $g^{xy} \mod p$ and $g^r \mod p$ where r is random

Properties of Diffie-Hellman

- Assuming DDH problem is hard, Diffie-Hellman protocol is a secure key establishment protocol against passive attackers
  - Eavesdropper can't tell the difference between established key and a random value
  - Can use new key for symmetric cryptography
    - Approx. 1000 times faster than modular exponentiation
- Diffie-Hellman protocol (by itself) does not provide authentication