CS 475: Authentication

Rachel Greenstadt
February 19, 2013
Reminders

- Quiz 2 next Tuesday
- Project 2 due Thursday
Authenticating Messages

- Hashes
- MAC
- HMAC
- Passwords
- SSL / public key authentication
Introduction to Hash Functions

- If $H$ is a hash function, $m$ is an input bit string, and $h$ is the output of $H$ applied to the input $m$, then we write $h = H(m)$. Some common and useful terminology:
  - If $h = H(m)$ then
    - $h$ is called the "hash" of $m$,
    - $m$ is called a "preimage" of $h$,
  - for a given input $m$, a "second preimage" of $m$ is a different input $m'$ such that $H(m) = H(m')$,
  - if $m$ and $m'$ are different inputs such that $H(m) = H(m')$ then the pair \{m,m'\} is called a "collision" for $H$. 
Collision Resistance

- Strong collision resistance
  - Hard to find *any* \( x, y \) such that \( h(x) = h(y) \)

- Weak collision resistance / preimage attacks
  - First preimage attack: given hash \( h_1 \), find \( m \) such that \( h(m) = h_1 \)
  - Second preimage attack: Given message \( m_1 \) find message \( m_2 \) such that \( h(m_1) = h(m_2) \)
Birthday Attacks on Collision Resistance

• Given function $h$, goal is find two inputs $x, y$ such that $h(x) = h(y)$

• Based on the birthday paradox: A group of 23 or more people will have the same probability $> 50$

• $H$ different outputs, then expected $1.25 \times \sqrt{H}$ to find a match

• so $2^{160}$ outputs for SHA-1, leads to approx $2^{80}$ tries
Types of Hash Functions

- **MD5**
  - 128-bit output
  - Designed by Ron Rivest, used very widely
  - Collision-resistance broken (summer of 2004 and it keeps getting worse)

- **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)
Group exercise

- Hash functions are reasonably fast, but here's a much faster function to compute. Take your message, divide it into 128-bit chunks, and xor all the chunks together to get a 128-bit result. Do the standard hash function on the result. Is this a good hash function? Why or why not?
SHA-1

Split message into 512-bit blocks

160-bit buffer (5 registers) initialized with magic values

Compression function
- Applied to each 512-bit block and current 160-bit buffer
- This is the heart of SHA-1

Message

Padding
(1 to 512 bits)

Message length
(K mod 2^64)

Against padding attacks
SHA-1 Compression Function

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

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

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

Buffer contains final hash value
How Strong is SHA-1?

- Every bit of output depends on every bit of input
  - Very important for collision resistance
- Brute-force inversion requires $2^{160}$ ops, birthday attack on collision resistance requires $2^{80}$
- Recent weaknesses (2005)
  - Collisions can be found in $2^{63}$ ops
How to get authentication?
Given: Everybody knows Bob’s public key
Only Bob knows the corresponding private key

Goal: Bob sends a “digitally signed” message
1. To compute a signature, must know the private key
2. To verify a signature, enough to know the public key
Digital Signature Properties

- Authentication - “It’s really Bob that sent this”
- Nonrepudiation - “Bob can’t later claim he didn’t mean this”
- Integrity - “This is the thing Bob meant to send”
RSA Signatures

- Public key is \((n,e)\), private key is \(d\)
- To sign message \(m\): \(s = m^d \mod n\)
  - Signing and decryption are the same operation in RSA (not true for all schemes)
- It’s infeasible to compute \(s\) on \(m\) if you don’t know \(d\)
- To verify signature \(s\) on message \(m\): \(s^e \mod n = (m^d)^e \mod n = m\)
  - Just like encryption
  - Anyone who knows \(n\) and \(e\) (public key) can verify signatures produced with \(d\) (private key)
- In practice, also need padding & hashing (why?)
More on Signing

• Decryption not always signature
• Sign a hash not the message
• Signing a hash image with size equal to modulus is provably secure
Digital Signature Attacks

• Attack models (GMR)
  - key only (only public key)
  - known message (have some messages)
  - adaptive chosen message (can get chosen messages before attack)

• Attack Results
  - total break (recovery of signing key)
  - universal forgery (forge signatures in all messages)
  - selective forgery (adversary can create and sign some messages)
  - existential forgery (some valid but unchosen msg/signature pair created)

• Provably secure - No existential forgery under adaptive chosen message attack
Authentication without Encryption

Integrity and Authentication: only someone who knows key can compute MAC for a given message
How to hash the key and message?

• Seems easy, just compute $h(\text{key} | \text{message})$

• Problems?

  • Assume $h$ is SHA-1

  • Recall that in SHA-1, the message is hashed from left to right in 512 bit chunks
Enter Carol

- Bob is Carol’s boss, and Alice is Bob’s boss
- Carol appends “P.S. Give Carol a promotion and triple her salary” to Alice’s message to Bob

Carol can take the original message, add some padding, then add her postscript and pass it into SHA-1
HMAC

• MAC that is “as secure as underlying hash”

• Strong collision resistance

• attacker that doesn’t know key K cannot compute digest(K,x) for data x even if the attacker can see digest(K,y) for arbitrary y not equal to x

• Result slow but provable
HMAC

- Construct MAC by applying 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
How HMAC Works

- If key > 512 bits, digest(key) and pad to 512 else if key < 512 bits, pad to 512
- result1 = digest ((Const1 XOR padded key) . message)
- result2 = digest((Const2 XOR padded key) . result1)
- HMAC(message, key) = result2
HMAC

Secret key padded to block size

Another magic value (flips different key bits)

“Amplify” key material (get two keys out of one)

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

Magic value (flips half of key bits)

Block size of embedded hash function

Embedded 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?)

Hash: $\text{Hash}(\text{key}, \text{hash(key, message)})$
Combine encryption and MAC for confidentiality and integrity
Encrypt-and-MAC

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

\[ E_{K_e, K_m}(M) \]
- \( E_{K_e} \)
- \( MAC_{K_m} \)
- \( C' \)
- \( T \)

\[ D_{K_e, K_m}(C, T) \]
- \( D_{K_e} \)
- \( Verify_{K_m} \)
- \( M \)
- \( valid/invalid \)
- \( valid \)

Return \( M \) if valid
Insecure!

Assume Alice sends messages:

- FIRE
  - Encrypt\text{\textsubscript{ke}}
    - C'\text{\textsubscript{1}}
    - T\text{\textsubscript{1}}
  - MAC\text{\textsubscript{Km}}

- DON'T FIRE
  - Encrypt\text{\textsubscript{ke}}
    - C'\text{\textsubscript{2}}
    - T\text{\textsubscript{2}}
  - MAC\text{\textsubscript{Km}}

- FIRE
  - Encrypt\text{\textsubscript{ke}}
    - C'\text{\textsubscript{3}}
    - T\text{\textsubscript{3}}
  - MAC\text{\textsubscript{Km}}

If T\text{\textsubscript{i}} = T\text{\textsubscript{j}} then M\text{\textsubscript{i}} = M\text{\textsubscript{j}}
Adversary learns whether two plaintexts are equal.

Especially problematic when M\textsubscript{1}, M\textsubscript{2}, \ldots take on only a small number of possible values.
Attacks

• Confidentiality considers indistinguishability under...
  • Chosen Plaintext Attack (CPA) An attacker can obtain the ciphertext for any provided plaintext (but does not have the key).
  • Chosen Ciphertext Attack (CCA) An attacker can obtain the plaintext for any provided ciphertext (but does not have the key).

• Integrity
  • PTXT - Integrity of Plaintext - computationally infeasible to produce a ciphertext decrypting to a new plaintext message.
  • CTXT - Integrity of Ciphertext - computationally infeasible to produce a new, valid ciphertext
Results of [BN00,Kra01]

<table>
<thead>
<tr>
<th>Method</th>
<th>Privacy</th>
<th>Integrity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encrypt-then-MAC</td>
<td>Strong (CCA)</td>
<td>Strong (CTXT)</td>
</tr>
<tr>
<td>MAC-then-Encrypt</td>
<td>Weak (CPA)</td>
<td>Weak (PTXT)</td>
</tr>
<tr>
<td>Encrypt-and-MAC</td>
<td>Insecure</td>
<td>Weak (PTXT)</td>
</tr>
</tbody>
</table>

- M: Message
- C': Ciphertext C
- T: Tag
- EncryptKe: Encrypt
- MACKm: MAC

Thursday, February 21, 2013
Authenticating Users

“Something you forget, something you lose, and something you used to be”

• Passwords
• Alternatives
• Multi-factor Authentication
Password Security Review

• Summarize system
  • Identify assets: What do you wish to protect
  • Identify adversaries and threats
  • Identify vulnerabilities
  • Calculate the risks
  • Evaluate controls/mitigation strategies
• Iterate
Assets
Adversaries
Vulnerabilities
Vulnerabilities

- Online guessing/dictionary attack
- Offline guessing/dictionary attack
- Shared passwords
- Password fallback schemes
Risks
Mitigation Strategies
Mitigation Strategies

• Limited number of attempts
• Salts
• Encrypted Storage
• Challenge/Response
Alternatives to Passwords

- Graphical passwords, phrases
- Tokens/dongles
- Biometrics
Multifactor Authentication
Public Key Authentication
Authenticity of Public Keys

Problem: How does Alice know that the public key she received is really Bob’s public key?
Distribution of Public Keys

• Public announcement or public directory
  • Risks: forgery and tampering

• Public-key certificate
  • Signed statement specifying the key and identity
    \[ \text{sig}_{\text{Alice}}(\text{"Bob"}, \text{PK}_B) \]

• Common approach: certificate authority (CA)
  • Single agency responsible for certifying public keys
  • After generating a private/public key pair, user proves his identity and knowledge of the private key to obtain CA’s certificate for the public key (offline)
  • Every computer is pre-configured with CA’s public key
Hierarchical Approach

- Single CA certifying every public key is impractical
- Instead, use a trusted root authority
  - For example, Verisign
  - Everybody must know the public key for verifying root authority’s signatures
- Root authority signs certificates for lower-level authorities, lower-level authorities sign certificates for individual networks, and so on
  - Instead of a single certificate, use a certificate chain
    - $\text{sig}_{\text{Verisign}}(\text{“UW”, } \text{PK}_{\text{UW}})$, $\text{sig}_{\text{UW}}(\text{“Alice”, } \text{PK}_A)$
  - What happens if root authority is ever compromised?
X509 Certificates

[Diagram of X509 certificate structure]

Added in X.509 versions 2 and 3 to address usability and security problems.
Bad Certificates

- What to do if a bad certificate is issued?
  - In practice...wait for it to expire
  - In theory
    - Revocation Services
    - Revocation Lists