CSE 484 (Winter 2010)

Symmetric Cryptography
+
Web Security

Tadayoshi Kohno

Thanks to Dan Boneh, Dieter Gollmann, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...
Goals for Today

- CELT -- Confidential course feedback opportunity
- Finish hash functions and MACs
- Combining Encryption and MACs
- Web security
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 recently broken! (Theoretically -- not practical.)

- **SHA-256, SHA-512, SHA-224, SHA-384**

- **SHA-3**: Forthcoming.
International Criminal Tribunal for Rwanda
(Which Properties of Hash Functions?)


Credits: Alexei Czeskis, Karl Koscher, Batya Friedman
Basic Structure of SHA-1 (Not Required)

- **Split message into 512-bit blocks**
- **Compression function**
  - Applied to each 512-bit block and current 160-bit buffer
  - This is the heart of SHA-1
- **160-bit buffer** (5 registers) initialized with magic values
- **Against padding attacks**
- **Message length** ($K \mod 2^{64}$)
- **Padding** (1 to 512 bits)
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

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

\[
\begin{align*}
\text{Encrypt}_{Ke} & \quad \text{MAC}_{Km} \\
M & \quad \text{Ciphertext} \\
\text{Ciphertext} & \quad \text{valid/invalid} \\
\text{Return } M \text{ if valid} & \\
\end{align*}
\]
But insecure!  [BN, Kra]

Assume Alice sends messages:

If $T_i = T_j$ then $M_i = M_j$

Adversary learns whether two plaintexts are equal.

Especially problematic when $M_1, M_2, \ldots$ take on only a small number of possible values.
Results of [BN00,Kra01]

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

\[ E_{K_e, K_m}(M) \]

Data to be communicated

Maintained internally; not transmitted

ctr 4 bytes

pl 4 bytes

pdl 1 byte

M

Encrypt_{K_e}

MAC_{K_m}

C'  T

Ciphertext packet
What’s different about SSH?

Assume Alice sends messages $M_1$ and $M_2$ that are the same. Then the tags $T_1$ and $T_2$ may be different with high probability.

But if counters repeat, tags may once again leak private information about data.

Then the tags $T_1$ and $T_2$ will be different with high probability.
Browser and Network

Browser

OS

Hardware

Network

website

request

reply
Security and Browsers ...  

IE zero-day used in Chinese cyber assault on 34 firms

Updated Hackers who breached the defenses of Google, Adobe Systems and at least 32 other companies used a potent vulnerability in all versions of Internet Explorer to carry out at least some of the attacks, researchers from McAfee said Thursday.

...  

"In our investigation we discovered that one of the malware samples involved in this broad attack exploits a new, not publicly known vulnerability in Microsoft Internet Explorer," Kurtz wrote. "Our investigation has shown that Internet explorer is vulnerable on all of Microsoft's most recent operating system releases, including Windows 7."
Example Questions

- How does website know who you are?
- How do you know who the website is?
- Can someone intercepting traffic?
- Related: How can you better control flow of information?

Our focus: High-level principles (Lab focuses on pragmatics)
HTTP: HyperText Transfer Protocol

Used to request and return data
- Methods: GET, POST, HEAD, ...

Stateless request/response protocol
- Each request is independent of previous requests
- Statelessness has a significant impact on design and implementation of applications

Evolution
- HTTP 1.0: simple
- HTTP 1.1: more complex
- ... Still evolving ...
# HTTP Request

<table>
<thead>
<tr>
<th>Method</th>
<th>File</th>
<th>HTTP version</th>
<th>Headers</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>/default.asp</td>
<td>HTTP/1.0</td>
<td>Accept: image/gif, image/x-bitmap, image/jpeg, <em>/</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Accept-Language: en</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Connection: Keep-Alive</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If-Modified-Since: Sunday, 17-Apr-96 04:32:58 GMT</td>
</tr>
</tbody>
</table>

Data – none for GET

Blank line
HTTP Response

HTTP version  Status code  Reason phrase

HTTP/1.0  200  OK
Date:  Sun, 21 Apr 1996 02:20:42 GMT
Server:  Microsoft-Internet-Information-Server/5.0
Connection:  keep-alive
Content-Type:  text/html
Last-Modified:  Thu, 18 Apr 1996 17:39:05 GMT
Content-Length:  2543

<HTML>  Some data...  blah, blah, blah  </HTML>
Primitive Browser Session

Store session information in URL; easily read on network
FatBrain.com circa 1999  [due to Fu et al.]

◆ User logs into website with his password, authenticator is generated, user is given special URL containing the authenticator

https://www.fatbrain.com/HelpAccount.asp?t=0&p1=me@me.com&p2=540555758

• With special URL, user doesn’t need to re-authenticate
  – Reasoning: user could not have not known the special URL without authenticating first. That’s true, BUT...

◆ Authenticators are global sequence numbers

• It’s easy to guess sequence number for another user

https://www.fatbrain.com/HelpAccount.asp?t=0&p1=SomeoneElse&p2=540555752

• Partial fix: use random authenticators
Bad Idea: Encoding State in URL

- Unstable, frequently changing URLs
- Vulnerable to eavesdropping
- There is no guarantee that URL is private
  - Early versions of Opera used to send entire browsing history, including all visited URLs, to Google
Cookies
A **cookie** is a file created by an Internet site to store information on your computer.

HTTP is a stateless protocol; cookies add state.
What Are Cookies Used For?

- **Authentication**
  - Use the fact that the user authenticated correctly in the past to make future authentication quicker

- **Personalization**
  - Recognize the user from a previous visit

- **Tracking**
  - Follow the user from site to site; learn his/her browsing behavior, preferences, and so on
Cookie Management

◆ Cookie ownership
  • Once a cookie is saved on your computer, only the website that created the cookie can read it (supposedly)

◆ Variations
  • Temporary cookies
    – Stored until you quit your browser
  • Persistent cookies
    – Remain until deleted or expire
  • Third-party cookies
    – Originates on or sent to another website
Privacy Issues with Cookies

◆ Cookie may include any information about you known by the website that created it
  • Browsing activity, account information, etc.
◆ Sites can share this information
  • Advertising networks
  • 2o7.net tracking cookie
◆ Browser attacks could invade your privacy

November 8, 2001:
Users of Microsoft's browser and e-mail programs could be vulnerable to having their browser cookies stolen or modified due to a new security bug in Internet Explorer (IE), the company warned today
The Weather Channel

The website "twci.coremetrics.com" has requested to save a file on your computer called a "cookie." This file may be used to track usage information...
The website “insightexpressai.com” has requested to save a file on your computer called a “cookie”...
Let’s Take a Closer Look…
Storing State in Browser

❖ Dansie Shopping Cart (2006)
  • “A premium, comprehensive, Perl shopping cart. Increase your web sales by making it easier for your web store customers to order.”

```html
<form method="post" action="http://www.dansie.net/cgi-bin/scripts/cart.pl">
  Black Leather purse with leather straps<br>Price: $20.00
  <input type="hidden" name="name" value="Black leather purse">
  <input type="hidden" name="price" value="20.00">
  <input type="hidden" name="sh" value="1">
  <input type="hidden" name="img" value="purse.jpg">
  <input type="hidden" name="custom1" value="Black leather purse with leather straps">
  <input type="submit" name="add" value="Put in Shopping Cart">
</form>
```

Change this to 2.00
Shopping Cart Form Tampering

Many Web-based shopping cart applications use hidden fields in HTML forms to hold parameters for items in an online store. These parameters can include the item's name, weight, quantity, product ID, and price. Any application that bases price on a hidden field in an HTML form is vulnerable to price changing by a remote user. **A remote user can change the price of a particular item they intend to buy, by changing the value for the hidden HTML tag that specifies the price, to purchase products at any price they choose.**

**Platforms Affected:**

- 3D3.COM Pty Ltd: ShopFactory 5.8 and earlier
- @Retail Corporation: @Retail Any version
- Adgrafix: Check It Out Any version
- Baron Consulting Group: WebSite Tool Any version
- ComCity Corporation: SalesCart Any version
- Crested Butte Software: EasyCart Any version
- Dansie.net: Dansie Shopping Cart Any version
- Intelligent Vending Systems: Intellivend Any version
- Make-a-Store: Make-a-Store OrderPage Any version
- McMurtrey/Whitaker & Associates: Cart32 2.6
- McMurtrey/Whitaker & Associates: Cart32 3.0
- knutsen@nethut.no: CartMan 1.04
- Rich Media Technologies: JustAddCommerce 5.0
- SmartCart: SmartCart Any version
- Web Express: Shoptron 1.2
Storing State in Browser Cookies

- Set-cookie: price=299.99
- User edits the cookie...  cookie: price=29.99
- What’s the solution?
- Add a MAC to every cookie, computed with the server’s secret key
  - Price=299.99; MAC(ServerKey, 299.99)
- Is this the solution?
Storing State in Browser

❖ Dansie Shopping Cart (2006)
   • “A premium, comprehensive, Perl shopping cart. Increase your web sales by making it easier for your web store customers to order.”

```html
<FORM METHOD=POST
ACTION="http://www.dansie.net/cgi-bin/scripts/cart.pl">
  Black Leather purse with leather straps<br>Price: $20.00<br>
  <INPUT TYPE=HIDDEN NAME=name VALUE="Black leather purse">
  <INPUT TYPE=HIDDEN NAME=price VALUE="F13A3....B2">
  <INPUT TYPE=HIDDEN NAME=sh VALUE="1">
  <INPUT TYPE=HIDDEN NAME=img VALUE="purse.jpg">
  <INPUT TYPE=HIDDEN NAME=custom1 VALUE="Black leather purse with leather straps">
  <INPUT TYPE=SUBMIT NAME="add" VALUE="Put in Shopping Cart">
</FORM>

Better: MAC(K, “$20,Black leather purse, product number 12345, ...”)
```
Web Authentication via Cookies

Need authentication system that works over HTTP and does not require servers to store session data

Servers can use cookies to store state on client
  
  - When session starts, server computes an authenticator and gives it back to browser in the form of a cookie
    - Authenticator is a value that client cannot forge on his own
    - Example: MAC(server’s secret key, session id)
  
  - With each request, browser presents the cookie
  
  - Server recomputes and verifies the authenticator
    - Server does not need to remember the authenticator
Typical Session with Cookies

- **client**
  - POST /login.cgi
  - Set-Cookie: authenticator
  - GET /restricted.html
  - Cookie: authenticator
  - Restricted content

- **server**
  - Verify that this client is authorized
  - Check validity of authenticator (e.g., recompute hash(key, sessId))

Authenticators must be **unforgeable and tamper-proof**

(malicious client shouldn’t be able to compute his own or modify an existing authenticator)
WSJ.com circa 1999  [due to Fu et al.]

◆ Idea: use `user,hash(user||key)` as authenticator
  - Key is secret and known only to the server. Without the key, clients can’t forge authenticators.
  - `||` is string concatenation

◆ Implementation: `user,crypt(user||key)`
  - `crypt()` is UNIX hash function for passwords
  - `crypt()` truncates its input at 8 characters
  - Usernames matching first 8 characters end up with the same authenticator
  - No expiration or revocation

◆ It gets worse... This scheme can be exploited to extract the server’s secret key
## Attack

<table>
<thead>
<tr>
<th>username</th>
<th>crypt(username,key,&quot;00&quot;)</th>
<th>authenticator cookie</th>
</tr>
</thead>
<tbody>
<tr>
<td>AliceBob1</td>
<td>008H8LRfzUXvk</td>
<td>AliceBob1008H8LRfzUXvk</td>
</tr>
<tr>
<td>AliceBob2</td>
<td>008H8LRfzUXvk</td>
<td>AliceBob2008H8LRfzUXvk</td>
</tr>
</tbody>
</table>

"Create" an account with a 7-letter user name...

- AliceBoA 0073UYEre5rBQ  Try logging in: access refused
- AliceBoB 00bkHcfOXBKno  Access refused
- AliceBoC 00ofSJV6An1QE  Login successful! 1st key symbol is C

Now a 6-letter user name...

- AliceBCA 001mBnBErXRuc  Access refused
- AliceBCB 00T3JLLfuspdo  Access refused... and so on

- Only need 128 x 8 queries instead of intended $128^8$
- Minutes with a simple Perl script vs. billions of years