Most Common Cryptography Mistakes

4/7/2014
## Encrypted credit card numbers

| 76 06 93 93 2b 8f 4b c6 ec e2 b3 d7 a1 09 f7 |
| 76 06 9a 95 27 84 4f c1 ef e2 bb df a5 0a f3 |
| 71 01 9a 93 2b 85 41 ca e2 e9 ba df a0 01 fa 26 |
| 76 05 9d 99 2b 84 4a ca e8 e1 b7 d7 a5 08 f4 |
| 71 04 98 98 22 8b 49 c0 ed e1 b0 d7 a8 08 f6 22 |
| 71 05 93 94 22 8d 4a c7 eb e5 b0 df a8 09 f3 23 |
| 70 02 9d 93 23 8c 4f c4 e2 e8 bb d0 a7 08 f6 20 |
# Encrypted credit card numbers

<p>| | | | | | | | | | | | | | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0x76</td>
<td>0x93</td>
<td>0x93</td>
<td>0x2b</td>
<td>0x8f</td>
<td>0x4b</td>
<td>0xc6</td>
<td>0xec</td>
<td>0xe2</td>
<td>0xb3</td>
<td>0xd7</td>
<td>0xa1</td>
<td>0x09</td>
<td>0xf7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x76</td>
<td>0x9a</td>
<td>0x95</td>
<td>0x27</td>
<td>0x84</td>
<td>0x4f</td>
<td>0xc1</td>
<td>0xef</td>
<td>0xe2</td>
<td>0xbb</td>
<td>0xdf</td>
<td>0xa5</td>
<td>0x0a</td>
<td>0xf3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x71</td>
<td>0xa1</td>
<td>0x9a</td>
<td>0x93</td>
<td>0x2b</td>
<td>0x85</td>
<td>0x41</td>
<td>0xca</td>
<td>0xe2</td>
<td>0xe9</td>
<td>0xba</td>
<td>0xdf</td>
<td>0xa0</td>
<td>0x01</td>
<td>0xfa</td>
<td>0x26</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x76</td>
<td>0xa5</td>
<td>0x9d</td>
<td>0x99</td>
<td>0x2b</td>
<td>0x84</td>
<td>0x4a</td>
<td>0xca</td>
<td>0xe8</td>
<td>0xe1</td>
<td>0xb7</td>
<td>0xd7</td>
<td>0xa5</td>
<td>0x08</td>
<td>0xf4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x71</td>
<td>0xc4</td>
<td>0x98</td>
<td>0x98</td>
<td>0x22</td>
<td>0x8b</td>
<td>0x49</td>
<td>0xc0</td>
<td>0xed</td>
<td>0xe1</td>
<td>0xb0</td>
<td>0xd7</td>
<td>0xa8</td>
<td>0x08</td>
<td>0xf6</td>
<td>0x22</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x71</td>
<td>0xa5</td>
<td>0x93</td>
<td>0x94</td>
<td>0x22</td>
<td>0x8d</td>
<td>0x4a</td>
<td>0xc7</td>
<td>0xeb</td>
<td>0xe5</td>
<td>0xb0</td>
<td>0xdf</td>
<td>0xa8</td>
<td>0x09</td>
<td>0xf3</td>
<td>0x23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x70</td>
<td>0x02</td>
<td>0x9d</td>
<td>0x93</td>
<td>0x23</td>
<td>0x8c</td>
<td>0x4f</td>
<td>0xc4</td>
<td>0xe2</td>
<td>0xe8</td>
<td>0xbb</td>
<td>0xd0</td>
<td>0xa7</td>
<td>0x08</td>
<td>0xf6</td>
<td>0x20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ASCII: ..., ‘3’ = 0x33, ‘4’ = 0x34, ‘5’ = 0x35, ...
Encrypted credit card numbers

ASCII: ‘0’ = 0x30, ..., ‘7’ = 0x37, ‘8’ = 0x38, ‘9’ = 0x39
#7: Don’t re-use nonces/IVs

- Re-using a nonce or IV leads to catastrophic security failure.
Early method for encrypting Wifi: WEP (Wired Equivalent Privacy)
- Share a single cryptographic key among all devices
- Encrypt all packets sent over the air, using the shared key
- Use a checksum to prevent injection of spoofed packets
WEP - A Little More Detail

- WEP uses the RC4 stream cipher to encrypt a TCP/IP packet ($P$) by xor-ing it with keystream ($\text{RC4}(K, IV)$)

\[ \text{IV}, \quad P \oplus \text{RC4}(K, IV) \]
A Risk of Keystream Reuse

- In some implementations, IVs repeat.
  - If we send two ciphertexts \((C, C')\) using the same IV, then the xor of plaintexts leaks \((P \oplus P' = C \oplus C')\), which might reveal both plaintexts.

Lesson: Don’t re-use nonces/IVs
WEP -- Even More Detail

- IV
- Even
- More
- Detail

IV

original unencrypted packet

checksum

RC4

key

encrypted packet

IV
Attack #2: Spoofed Packets

- Attackers can inject forged 802.11 traffic
  - Learn $Z = \text{RC4}(K, IV)$ using previous attack
  - Since the CRC checksum is unkeyed, you can then create valid ciphertexts that will be accepted by the receiver

$$IV, (P, CRC(P)) \oplus Z$$
Attack #3: Packet Modification

- CRC is linear
  \[ \Rightarrow CRC(P \oplus \Delta) = CRC(P) \oplus CRC(\Delta) \]
  \[ \Rightarrow \text{the modified packet } (P \oplus \Delta) \text{ has a valid checksum} \]
- Attacker can tamper with packet \((P)\) without breaking RC4
Attack #4: Inductive Learning

- Learn $Z_{1..n} = \text{RC4}(K, IV)_{1..n}$ using previous attack
- Then guess $Z_{n+1}$; verify guess by sending a ping packet $((P, \text{CRC}(P)))$ of length $n+1$ and watching for a response
- Repeat, for $n=1,2,...$, until all of $\text{RC4}(K, IV)$ is known

Credits: Arbaugh, et al.
Attack #5: Reaction Attacks

- TCP ACKnowledgement returned by recipient
  \[ \iff \text{TCP checksum on modified packet } (P \oplus 0x00010001) \text{ is valid} \]
  \[ \iff \text{wt}(P \& 0x00010001) = 1 \]
- Attacker can recover plaintext \( P \) without breaking RC4
#7: Key Re-use

- Don’t re-use keys for both encryption and authentication.
- Don’t re-use keys for both encryption and signing.
- Don’t use same key for both directions.
#8: Traffic Analysis is Still Possible

- Encryption doesn’t hide sender, recipient, length, or time of message. ("meta-data")
SSH

Client

(handshake; key exchange)

Server

\{l\}_K

\{l\}_{K'}

\{s\}_K

\{s\}_{K'}

\{n\}_K

\{\text{nfoo bar n$}\}_{K'}
SSH

```
{n}_K

{nPassword: }_{K'}
```

- **Client**
  - ```{q}_K```  
  - ```{p}_K```  
  - ```{l}_K```  
  - ```{e}_K```  
  - ```{4}_K```  
  - ```{n}_K```  

- **Server**
  - Reveals the length of the password.
  - Reveals the intervals between keystrokes.
  - This leaks partial information about the password!
Lessons Summarized

• Don’t design your own crypto algorithm.
• Use authenticated encryption (don’t encrypt without authenticating).
• Use crypto-quality random numbers.
• Don’t derive crypto keys from passphrases.
• Be secure by default.
• Be careful with concatenation.
• Don’t re-use nonces/IVs. Don’t re-use keys for multiple purposes.
• Encryption doesn’t prevent traffic analysis (“metadata”).
Meta-Lessons

• Cryptography is hard.
• Hire an expert, or use an existing system (e.g., SSL, SSH, PGP).
• But: Most vulnerabilities are in applications and software, not in crypto algorithms.
Today’s Lecture

• Applying crypto technology in practice
• Goal #1: overview of the most prominent Internet security protocol
  – **SSL/TLS**: transport-level (process-to-process) on top of TCP
    • Secures the web via HTTPS
• Goal #2: cement understanding of crypto building blocks & how they’re used together
Building Secure End-to-End Channels

- **End-to-end** = communication protections achieved all the way from originating client to intended server
  - With no need to trust intermediaries

- Dealing with threats:
  - Eavesdropping?
    - *Encryption* (including session keys)
  - Manipulation (injection, MITM)?
    - *Integrity* (use of a MAC); *replay protection*
  - Impersonation?
    - *Signatures*

(What’s missing? Availability …)
Building A Secure End-to-End Channel: SSL/TLS

- SSL = *Secure Sockets Layer* (predecessor)
- TLS = *Transport Layer Security* (standard)
  - Both terms used interchangeably
- Notion: provide means to secure *any* application that uses TCP
SSL/TLS In Network Layering

7  Application
4   TCP
3    IP
2     Link
1      Physical

7  Application
4   SSL / TLS
3    TCP
2     IP
1      Link
1      Physical
Building A Secure End-to-End Channel: SSL/TLS

• SSL = Secure Sockets Layer (predecessor)
• TLS = Transport Layer Security (standard)
  – Both terms used interchangeably
• Notion: provide means to secure any application that uses TCP
  – Secure = encryption/confidentiality + integrity + authentication (of server, but not of client)
  – E.g., puts the ‘s’ in “https”
Regular web surfing - http://URL

But if we click here ...
Note: Amazon makes sure that all of these images, etc., are now also fetched via https: URLs.

Doing so gives the web page full integrity, in keeping with end-to-end security.

(Browsers do not provide this “promotion” automatically.)
Basic idea

- Browser (client) picks some symmetric keys for encryption + authentication
- Client sends them to server, encrypted using RSA public-key encryption
- Both sides send MACs
- Now they use these keys to encrypt and authenticate all subsequent messages, using symmetric-key crypto