CSE508  Network Security (PhD Section)

2/24/2015  Encrypted Communication

Michalis Polychronakis
Stony Brook University
Cryptography
Goals

Confidentiality
   Keep content secret from all but authorized entities

Integrity
   Protect content from unauthorized alteration

Authentication
   Identification of data or communicating entities

Non-repudiation
   Prevent entities from denying previous commitments or actions
Basic Terminology

**Plaintext**: the original message

**Ciphertext**: the coded message

**Cipher**: algorithm for transforming plaintext to ciphertext (encryption) and back (decryption)

**Key**: info used in cipher known to sender and receiver

**Cryptanalysis (codebreaking)**: the study of methods of deciphering ciphertext without knowing the key

**Cryptology**: the field of both cryptography and cryptanalysis
Plaintext vs. Ciphertext

Encryption

Plaintext in

Encryption algorithm

Encryption key

P → E → C = E(P, K_E) → D → P

Ciphertext

Decryption

Decryption algorithm

Decryption key

Plaintext out

P → D → P = D(C, K_D) → E → C
Kerckhoffs's Principle

A cryptosystem should be secure even if everything about the system, except the key, is public knowledge.

The security of the system must rest entirely on the secrecy of the key:
- Only brute force attacks are possible
- Otherwise the algorithm is broken

Contrast with security by obscurity: every secret creates a potential failure point.
Caesar Cipher

Ciphertext: WKH TXLFN EURZQ IRA MXPSV RYHU WKH ODCB GRJ
Plaintext: the quick brown fox jumps over the lazy dog

Shift by X (e.g., ROT-13)

Monoalphabetic substitution
Easy to break using frequency analysis

Distribution of letters in a typical sample of English language text
Vigenère Cipher

Plaintext: ATTACKATDAWN
Key: LEMONLEMONLE
Ciphertext: LXFOPVEFRNHR

Successive Ceasar ciphers with different shift values

Polyalphabetic substitution

Defeats simple frequency analysis, but still breakable
Properties of a Good Cryptosystem

Given the ciphertext, an adversary should not be able to recover the original message

Enumerating all possible keys must be infeasible
There should be no way to produce plaintext from ciphertext without the key

The ciphertext must be indistinguishable from true random values

Given a ciphertext, the probability of any possible plaintext being encrypted should be the same
Symmetric Key Cryptography

Pros:
- Fast
- Short keys
- Well known
- Simple key generation

Cons:
- Secrecy of keys
- Management of keys
- Number of keys
One-time Pad

XOR plaintext with a keystream
  1882 Frank Miller [Bellovin ’11]
  1917 Vernam/Mauborgne cipher

“Information-theoretically secure” against ciphertext-only attacks
(Shannon 1949)

The keystream must be
  Truly random
  As long as the plaintext
  Used only once
  Kept completely secret
Block Ciphers

- Process one block at a time
- Substitution and transposition (permutation) techniques
- Examples: DES, AES, …

Stream Ciphers

- Process one bit or byte at a time
- Plaintext is combined (XOR) with a pseudorandom keystream (NOT the same as one-time pad)
- Synchronous vs. asynchronous (self-synchronizing)
- Examples: RC4, any block cipher in OFB or CTR mode, …
Block Ciphers

Multiple rounds of substitution, permutation, …

*Confusion:* each character of the ciphertext should depend on several parts of the key

*Diffusion:* changing a plaintext character should result in several changed ciphertext characters

<table>
<thead>
<tr>
<th></th>
<th>DES</th>
<th>AES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key length</td>
<td>56 bits</td>
<td>128, 192, 256 bits</td>
</tr>
<tr>
<td>Block size</td>
<td>64 bits</td>
<td>128 bits</td>
</tr>
<tr>
<td>Rounds</td>
<td>16</td>
<td>10, 12, 14</td>
</tr>
<tr>
<td>Construction</td>
<td>Substitution, permutation</td>
<td>Substitution, permutation, mixing, addition</td>
</tr>
<tr>
<td>Developed</td>
<td>1977</td>
<td>1998</td>
</tr>
<tr>
<td>Status</td>
<td>Broken!</td>
<td>OK (for now)</td>
</tr>
</tbody>
</table>
Modes of Operation

Direct use of block ciphers is not very useful

- Enemy can build a “code book” of plaintext/ciphertext equivalents
- Message length should be multiple of the cipher block size

How to repeatedly apply a block cipher to securely encrypt/decrypt arbitrary inputs?

Five standard modes

- ECB: Electronic Code Book
- CBC: Cipher Block Chaining
- CFB: Cipher Feedback
- OFB: Output Feedback
- CTR: Counter
ECB: Electronic Code Book Mode

Direct use of the block cipher

Each block is encrypted independently
No chaining, no error propagation

Problem: if $m_i = m_j$ then $c_i = c_j$
ECB: Electronic Code Book Mode

Data patterns may remain visible
Susceptible to replay attacks, block insertion/deletion
CBC: Cipher Block Chaining Mode

Each plaintext block is XORed with the previous ciphertext block before being encrypted

 Initialization Vector (IV)

Cipher Block Chaining (CBC) mode encryption

Must be random!
Must never be reused!
CTR: Counter Mode

Turns a block cipher into a stream cipher

Next keystream block is generated by encrypting successive values of a counter combined with a nonce (IV)
Diffie–Hellman Key Exchange

Allows two parties to jointly establish a shared secret key over an insecure communication channel

The established key can then be used to encrypt subsequent communication using a symmetric key cipher

“New Directions in Cryptography” by Whitfield Diffie and Martin Hellman, 1976

Based on the discrete logarithm problem

\[ 3^{29} \mod 17 \quad \text{easy} \quad ?? \]

\[ 3^{??} \mod 17 \quad \text{hard} \quad 12 \]
Diffie–Hellman Key Exchange

Alice and Bob agree on a large (at least 1024 bit) prime number \( p \) and a base \( g \)

- \( p \) is usually of the form \( 2q+1 \) where \( q \) is also prime
- \( g \) is a generator of the multiplicative group of integers modulo \( p \)
  (for every \( x \) coprime to \( p \) there is a \( k \) such that \( g^k \equiv x \mod p \))

Alice picks a secret (private) large random number \( a \) and sends to Bob \( g^a \mod p \)

Bob picks a secret large random number \( b \) and sends to Alice \( g^b \mod p \)

Alice calculates \( s = (g^b \mod p)^a = g^{ba} \mod p \)
Bob calculates \( s = (g^a \mod p)^b = g^{ab} \mod p \)
Alice

\[ p = 23, \ g = 5 \]

\[ a = 6 \]

\[ 5^6 \mod 23 = 8 \]

\[ 19^6 \mod 23 = 2 \]

Bob

\[ p = 23, \ g = 5 \]

\[ b = 15 \]

\[ 5^{15} \mod 23 = 19 \]

\[ 8^{15} \mod 23 = 2 \]
Man-in-the-Middle Attack

Alice and Bob share no secrets

Mallory actively decrypts and re-encrypts all traffic

Alice and Bob think they communicate directly

General problem: need for a root of trust
Public Key Cryptography

Pros:
- No shared secrets
- Easier key management
- Provides secrecy and authenticity

Cons:
- Slow
- Large keys
- Key generation is more difficult
RSA

Named after its inventors Rivest, Shamir, Adleman

Based on the problem of factoring large numbers

Choose two distinct large prime numbers \( p \) and \( q \)
Let \( n = pq \) (modulus)
Select \( e \) as a relative prime to \( (p - 1)(q - 1) \)
Calculate \( d \) such that \( ed \equiv 1 \mod (p - 1)(q - 1) \)
Public key = \((e, n)\)
Private key = \((d, n)\)
To encrypt \( m \), calculate \( c = m^e \mod n \)
To decrypt \( c \), calculate \( m = c^d \mod n \)
RSA in Practice

RSA calculations are computationally expensive

Use RSA in combination with a symmetric key

Send an encrypted message:
- Encrypt message with a random symmetric key
- Encrypt symmetric key with recipient’s public key
- Transmit both the encrypted message and the encrypted key

Set up an encrypted communication channel:
- Negotiate a symmetric key using RSA
- Use the symmetric key for subsequent communication
Forward Secrecy

Threat: capture encrypted traffic now, use in the future
   Private keys may be compromised (e.g., infiltrate system)
   Cryptanalytic breakthrough

FS: Ensure that even if current keys are compromised, past encrypted traffic cannot be compromised
   Cannot read old messages
   Cannot forge a message and claim that it was sent in the past

Support
   IPsec, SSH, Off-the-Record messaging (OTR)
   TLS (Diffie–Hellman instead of RSA key exchange)

Note a panacea
   Session keys might be kept in memory for hours
   Server could be forced to record all session keys
   TLS Session tickets need careful treatment
Cryptographic Hash Functions

Hash functions that are considered practically impossible to invert

Arbitrary length input → Cryptographic hash function → Fixed length output

Properties of an ideal cryptographic hash function

- Easy to compute the hash value for any given message
- Infeasible to generate a message that has a given hash
- Infeasible to modify a message without changing the hash
- Infeasible to find two different messages with the same hash

Many-to-one function: *collisions can happen*
Cryptographic Hash Function Properties

Pre-image resistance

Given a hash value $h$ it should be computationally infeasible to find any input $m$ such that $h = \text{hash}(m)$

Example: break a hashed password

Second pre-image resistance

Given $m_1$ it should be computationally infeasible to find $m_2$ such that $m_1 \neq m_2$ and $\text{hash}(m_1) = \text{hash}(m_2)$

Example: forge an existing certificate

Collision Resistance

It should be computationally infeasible to find two different inputs $m_1$ and $m_2$ such that $\text{hash}(m_1) = \text{hash}(m_2)$ (collision)

Example: prepare two contradicting versions of a contract
Birthday Paradox

How many people does it take before the odds are 50% or better of having...

...another person with the same birthday as you? 253
  Second pre-image resistance

...two people with the same birthday? 23
  Collision resistance
Uses of Cryptographic Hash Functions

Data integrity
Digital signatures
Message authentication
User authentication
Timestamping
Certificate revocation management
Common Hash Functions

**MD5:** 128-bit output

- 1993: Boer and Bosselaers, “pseudo-collision” of the MD5 compression function: 2 different IVs which produce an identical digest
- 1996: Dobbertin, collision of the MD5 compression function
- 2004: Wang, Feng, Lai, and Yu, collisions for the full MD5
- 2005: Lenstra, Wang, and de Weger, construction of two X.509 certificates with different public keys but same hash
- 2008: Sotirov, Stevens, Appelbaum, Lenstra, Molnar, Osvik, de Wege, creating rogue CA certificates

*Use it? NO, it’s unsafe*

**SHA-1:** 160-bit output

- 2005: Rijmen and Oswald, attack on a reduced version of SHA1 (53 out of 80 rounds)
- 2005: Wang, Yao, and Yao, an improvement, lowering complexity for finding a collision to $2^{63}$
- 2006: Rechberger, attack with $2^{35}$ compression function evaluations

*Use it? Use SHA-256 or better instead*
Since our last MSRC blog post, we’ve received questions on the nature of the cryptographic attack we saw in the complex, targeted malware known as Flame. This blog summarizes what our research revealed and why we made the decision to release Security Advisory 2718704 on Sunday night PDT. In short, by default the attacker’s certificate would not work on Windows Vista or more recent versions of Windows. They had to perform a collision attack to forge a certificate that would be valid for code signing on Windows Vista or more recent versions of Windows. On systems that pre-date Windows Vista, an attack is possible without an MD5 hash collision. This certificate and all certificates from the involved certificate authorities were invalidated in Security Advisory 2718704. We continue to encourage all customers who are not installing updates automatically to do so immediately.

**Mysterious Missing Extensions**

When we first examined the Flame malware, we saw a file that had a valid digital signature that chained up to a Microsoft Root authority. As we reviewed this certificate, we noticed several irregularities. First, it had no X.509 extension fields, which was not consistent with the certificates we issued from the Terminal Server licensing infrastructure. We expected to find a Certificate Revocation List (CRL) Distribution Point (CDP) extension, an Authority Information Access (AIA) extension, and a “Microsoft Hydra” critical extension.
Message Authentication Codes (MACs)

Verify both message integrity and authenticity

Keyed-hash message authentication code (HMAC)

For a cryptographic hash function H:

\[ \text{HMAC}(K, m) = H( (K \oplus \text{opad}) \ || \ H(K \oplus \text{ipad} \ || \ m) ) \]

**opad/ipad**: outer/inner padding

\(||\)** denotes concatenation

Impossible to generate the HMAC of a message without knowing the secret key
Order of Encryption and MACing

Encrypted data usually must be protected with a MAC

Encryption alone protects only against passive adversaries

Different options:

Encrypt-and-MAC $E(P) \ || \ M(P)$

No integrity of the ciphertext

MAC-then-Encrypt $E(P \ || \ M(P))$

No integrity of the ciphertext (have to decrypt it first)

Encrypt-then-MAC $E(P) \ || \ M(E(P))$

Preferable option – *always MAC the ciphertext*
Digital Signatures

Use RSA backwards:

Sign (encrypt) with the private key
Verify (decrypt) with the public key

Ownership of a private key turns it into a digital signature

Anyone can verify that a message was signed by its owner
What if a private key was stolen or deliberately leaked?

Non-repudiation

Again, too expensive to sign the whole message

Calculate a cryptographic hash of the message and sign the hash
Digital Signatures

**Sender**
- **Message**
  - To be, or not to be, that is the question, whether 'tis nobler in the...
- **Message Digest Algorithm**
- **Sender's Private Key**
  - Message Digest

**Receiver**
- **Message Digest Algorithm**
- **Sender's Public Key**
  - Message Digest
  - Encrypted Message Digest

**Equal?**
- yes: Message transmitted correctly
- no: Message has been modified!
## Hashes vs. MACs vs. Digital Signatures

<table>
<thead>
<tr>
<th></th>
<th>Hash</th>
<th>MAC</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integrity</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Authentication</td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Non-repudiation</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Keys</td>
<td>None</td>
<td>Symmetric</td>
<td>Asymmetric</td>
</tr>
</tbody>
</table>
Public Key Authenticity

Authentication without confidence in the keys used is pointless

Need to gain confidence or proof that a particular public key is authentic

- It is correct and belongs to the person or entity claimed
- Has not been tampered with or replaced by an attacker

Different ways to establish trust

- TOFU: trust on first use (e.g., SSH)
- Web of trust – decentralized (P2P) trust model (e.g., PGP)
- PKI: public key infrastructure (e.g., SSL)

(subject of future lecture)
Shamir: Crypto is usually not broken, but bypassed