Distributed Systems

November 17, 2014 © 2013 Paul Krzyzanowski

Cryptography = Security
Cryptography may be a component of a secure system
Adding cryptography may not make a system secure

November 17, 2014 © 2013 Paul Krzyzanowski

Cryptography: what is it good for?
• Authentication
  – determine origin of message
• Integrity
  – verify that message has not been modified
• Nonrepudiation
  – sender should not be able to falsely deny that a message was sent
• Confidentiality
  – others cannot read contents of the message

November 17, 2014 © 2013 Paul Krzyzanowski

Terms

Plaintext (cleartext) message P
Encryption $E(P)$
Produces Ciphertext, $C = E(P)$
Decryption, $P = D(C)$
Cipher = cryptographic algorithm

November 17, 2014 © 2013 Paul Krzyzanowski

Terms: types of ciphers

• Types
  – restricted cipher
  – symmetric algorithm
  – public key algorithm
• Stream vs. Block
  – Stream cipher
    • Encrypt a message a character at a time
  – Block cipher
    • Encrypt a message a chunk at a time

November 17, 2014 © 2013 Paul Krzyzanowski

Restricted cipher

Secret algorithm
• Vulnerable to:
  – Leaking
  – Reverse engineering
  • HD DVD (Dec 2006) and Blu-Ray (Jan 2007)
  • RC4
  • All digital cellular encryption algorithms
  • DVD and DIVX video compression
  • Firewire
  • Enigma cipher machine
  • Every NATO and Warsaw Pact algorithm during Cold War
• Hard to validate its effectiveness (who will test it?)
• Not a viable approach!
The key

- We understand how it works:
  - Strengths
  - Weaknesses

Based on this understanding, we can assess how much to trust the key & lock.

Symmetric-key algorithm

- Same secret key, $K$, for encryption & decryption
  $$C = E_K(P) \quad P = D_K(C)$$

- Examples: AES, 3DES, IDEA, RC5

- Key length
  - Determines number of possible keys
    - DES: 56-bit key; $2^{56} = 7.2 \times 10^{16}$ keys
    - AES-256: 256-bit key: $2^{256} = 1.1 \times 10^{77}$ keys
  - Brute force attack: try all keys

The power of 2

- Adding one extra bit to a key doubles the search space.
- Suppose it takes 1 second to search through all keys with a 20-bit key

<table>
<thead>
<tr>
<th>key length</th>
<th>number of keys</th>
<th>search time</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 bits</td>
<td>1,048,576</td>
<td>1 second</td>
</tr>
<tr>
<td>21 bits</td>
<td>2,097,152</td>
<td>2 seconds</td>
</tr>
<tr>
<td>32 bits</td>
<td>$4.3 \times 10^9$</td>
<td>1 hour</td>
</tr>
<tr>
<td>56 bits</td>
<td>$7.2 \times 10^{14}$</td>
<td>2,178 years</td>
</tr>
<tr>
<td>64 bits</td>
<td>$1.8 \times 10^{16}$</td>
<td>&gt; 567,000 years</td>
</tr>
<tr>
<td>256 bits</td>
<td>$1.2 \times 10^{17}$</td>
<td>$&gt; 3.5 \times 10^{63}$ years</td>
</tr>
</tbody>
</table>

Distributed & custom hardware efforts typically allow us to search between 1 and >100 billion 64-bit (e.g., RC5) keys per second.
Communicating with symmetric cryptography

- Both parties must agree on a secret key, \( K \)
- Message is encrypted, sent, decrypted at other side

Alice

\[ E_K(P) \]

Bob

\[ D_K(C) \]

- Key distribution must be secret
  - otherwise messages can be decrypted
  - users can be impersonated

Key explosion

Each pair of users needs a separate key for secure communication

- 2 users: 1 key
- 3 users: 3 keys
- 4 users: 6 keys

Alice

Bob

3 users: 3 keys

Charles

100 users: 4,950 keys

6 users: 15 keys

1000 users: 399,900 keys

\[ n \text{ users: }\frac{n(n-1)}{2} \text{ keys} \]

Key distribution

Secure key distribution is the biggest problem with symmetric cryptography

Key exchange

How can you communicate securely with someone you’ve never met?

Whit Diffie: idea for a public key algorithm

Challenge: can this be done securely?

Knowledge of public key should not allow derivation of private key

Diffie-Hellman Key Exchange

Key distribution algorithm
- First algorithm to use public/private “keys”
- Not public key encryption
- Based on difficulty of computing discrete logarithms in a finite field compared with ease of calculating exponentiation

Allows us to negotiate a secret session key without fear of eavesdroppers

Diffie-Hellman Key Exchange

- All arithmetic performed in a field of integers modulo some large number
- Both parties agree on
  - a large prime number \( p \)
  - and a number \( x < p \)
- Each party generates a public/private key pair

Private key for user \( i \): \( X_i \)

Public key for user \( i \): \( Y_i = x^i \mod p \)
Diffie-Hellman exponential key exchange

- Alice has secret key $X_A$
- Alice has public key $Y_A$
- Alice computes $K = Y_A^{X_A} \mod p$

Bob has secret key $X_B$
Bob has public key $Y_B$
Bob computes $K' = Y_B^{X_B} \mod p$

$K = (Bob's\ public\ key) \ (Alice's\ private\ key) \ mod\ p$
$K' = (Alice's\ public\ key) \ (Bob's\ private\ key) \ mod\ p$

RSA Public Key Cryptography

- Ron Rivest, Adi Shamir, Leonard Adleman created a true public key encryption algorithm in 1977
- Each user generates two keys:
  - Private key (kept secret)
  - Public key (can be shared with anyone)
- Difficulty of algorithm based on the difficulty of factoring large numbers
  - keys are functions of a pair of large (~300 digits) prime numbers

RSA algorithm

How to generate keys:
- choose two random large prime numbers $p, q$
- Compute the product $n = pq$
- randomly choose the encryption key, $e$, such that: $e$ and $(p - 1)(q - 1)$ are relatively prime
- Compute a decryption key, $d$, such that: $ed \equiv 1 \pmod{(p - 1)(q - 1)}$
  \[ d = e^{-1} \pmod{(p - 1)(q - 1)} \]
- discard $p, q$

RSA Encryption

- Key pair: $e, d$
- Agreed-upon modulus $n$
- Encrypt:
  - divide data into numerical blocks < $n$
  - encrypt each block:
    \[ c = m^e \mod n \]
- Decrypt:
  \[ m = c^d \mod n \]
Public-key algorithm

- Two related keys:
  \[ C = E_{K_1}(P) \quad P = D_{K_2}(C) \]
  \[ C' = E_{K_2}(P) \quad P = D_{K_1}(C') \]
  \( K_1 \) is a public key
  \( K_2 \) is a private key

- Examples:
  - RSA, Elliptic curve algorithms
  - DSS (digital signature standard)
  - Diffie-Hellman (key exchange only)

- Key length
  - Unlike symmetric cryptography, not every number is a valid key
  - 3072-bit RSA = 256-bit elliptic curve = 128-bit symmetric cipher
  - 15360-bit RSA = 521-bit elliptic curve = 256-bit symmetric cipher

Communication with public key algorithms

- Different keys for encrypting and decrypting
  - No need to worry about key distribution

Hybrid Cryptosystems

- Session key: randomly-generated key for one communication session
- Use a public key algorithm to send the session key
- Use a symmetric algorithm to encrypt data with the session key
- Public key algorithms are almost never used to encrypt messages
  - MUCH slower; vulnerable to chosen-plaintext attacks
  - RSA-2048 approximately 55x slower to encrypt and 2000x slower to decrypt than AES-256.

Communication with a hybrid cryptosystem

Alice

Bob

Pick a random session key, \( K \)

\[ E_K(K) \]

\[ K = D_K(E_K(K)) \]

Bob decrypts \( K \) with his private key

Now Bob knows the secret session key, \( K \)
Communication with a hybrid cryptosystem

Alice

Bob

`E_B(K)`

Bob's public key: `K`

`K = D_B(E_B(K))`

decrypt message using a symmetric algorithm and key `K`

encrypt message using a symmetric algorithm and key `K`

Message Authentication

McCarthy's puzzle (1958)

The setting:

- Two countries are at war
- One country sends spies to the other country
- To return safely, spies must give the border guards a password
- Spies can be trusted
- Guards chat – information given to them may leak

McCarthy's puzzle

Challenge

How can a guard authenticate a person without knowing the password?

Enemies cannot use the guard's knowledge to introduce their own spies

Solution to McCarthy's puzzle:

Michael Rabin, 1958

Use a one-way function, `B = f(A)`

- Guards get `B`
  - Enemy cannot compute `A` if they know `A`
  - Spies give `A`, guards compute `f(A)`
  - If the result is `B`, the password is correct.

One-way functions

- Easy to compute in one direction
- Difficult to compute in the other

Examples:

**Factoring:**

- `pq = N` EASY
- find `p,q` given `N` DIFFICULT

**Discrete Log:**

- `a^b \mod c = N` EASY
- find `b` given `a`, `c`, `N` DIFFICULT
McCarthy's puzzle example

Example with an 18 digit number
A = 289407349786637777
A^2 = 83756614110525308948445338203501729
Middle square, B = 110525308948445338

Given A, it is easy to compute B
Given B, it is difficult to compute A

“Difficult” = no known short-cuts; requires an exhaustive search

Message Integrity: Digital Signatures

• Validate the creator (signer) of the content
• Validate the content has not been modified since it was signed
• The content itself does not have to be encrypted

Digital Signatures: Public Key Cryptography

Encrypting a message with a private key is the same as signing it!

Trusted directory of public keys

Encrypt message with Alice’s private key

Decrypt message with Bob’s public key

But...

• Not quite what we want
  – We don’t want to permute or hide the content
  – We just want Bob to verify that the content came from Alice

• Moreover...
  – Public key cryptography is much slower than symmetric encryption
  – What if Alice sent Bob a multi-GB movie?

Hashes to the rescue!

• Cryptographic hash function (also known as a digest)
  – Input: arbitrary data
  – Output: fixed-length bit string
• Properties
  – One-way function
    • Given H = hash(M), it should be difficult to compute M, given H
  – Collision resistant
    • Given H = hash(M), it should be difficult to find M', such that H = hash(M')
    • For a hash of length L, a perfect hash would take 2^{L/2} attempts
  – Efficient
    • Computing a hash function should be computationally efficient

Popular hash functions

• SHA-2
  – Designed by the NSA; published by NIST
  – SHA-224, SHA-256, SHA-384, SHA-512
    • e.g., Linux passwords used MD5 and now SHA-512
• SHA-3
  – NIST standardization still in progress
• MD5
  – 128 bits (not often used now since weaknesses were found)
• Derivations from ciphers:
  – Blowfish (used for password hashing in OpenBSD)
  – 3DES – used for old Linux password hashes
Digital signatures using hash functions

- You:
  - Create a hash of the message
  - Encrypt the hash with your private key & send it with the message
- Recipient:
  - Decrypts the encrypted hash using your public key
  - Computes the hash of the received message
  - Compares the decrypted hash with the message hash
  - If they're the same then the message has not been modified

Digital signatures: public key cryptography

Alice generates a hash of the message

Alice encrypts the hash with her private key
This is her signature.

Alice sends Bob the message & the encrypted hash

If the hashes match, the signature is valid
– the encrypted hash must have been generated by Alice
Digital signatures: multiple signers

Charles:
- Generates a hash of the message, H(P)
- Decrypts Alice’s signature with Alice’s public key
- Decrypts Bob’s signature with Bob’s public key
- Validates the signature: D_A(S) ≅ H(P)
- Validates the signature: D_B(S') ≅ H(P)

Covert AND authenticated messaging

If we want to keep the message secret
- combine encryption with a digital signature

Use a session key:
- Pick a random key, K, to encrypt the message with a symmetric algorithm
- encrypt K with the public key of each recipient
- for signing, encrypt the hash of the message with sender’s private key

Covert and authenticated messaging

Alice generates a digital signature by encrypting the message with her private key

C=E_K(M)
S=E_a(H(M))

Alice picks a random key, K, and encrypts the message P with it using a symmetric cipher

C=E_K(M)
S=E_a(H(P))

Alice encrypts the session key for each recipient of this message using their public keys

C_1=E_B(K)
C_2=E_C(K)

The aggregate message is sent to Bob & Charles

C_1=E_B(K)
C_2=E_C(K)
Cryptographic toolbox

- Symmetric encryption
- Public key encryption
- One-way hash functions
- Random number generators

The End