Operating Systems Design
21. Cryptography: An Introduction
Paul Krzyzanowski
pxk@cs.rutgers.edu
Cryptography ≠ Security

Cryptography may be a component of a secure system

Adding cryptography may not make a system secure
Terms

**Plaintext** (cleartext), message $M$

**encryption**, $E(M)$

produces **ciphertext**, $C=E(M)$

**decryption**: $M=D(C)$

Cryptographic algorithm, **cipher**
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
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

BTW, the above is a bump key. See http://en.wikipedia.org/wiki/Lock_bumping.
The key

Source: en.wikipedia.org/wiki/Pin_tumbler_lock
The key

Source: en.wikipedia.org/wiki/Pin_tumbler_lock
The key

- We understand how it works:
  - Strengths
  - Weaknesses

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

Source: en.wikipedia.org/wiki/Pin_tumbler_lock
Symmetric-key algorithm

• Same secret key, $K$, for encryption & decryption

\[ C = E_K(M) \quad \text{and} \quad M = D_K(C) \]

• Examples: AES, 3DES, IDEA

• 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
Public key algorithm

{public, private} pair of keys

\[ C_1 = E_{\text{public}}(M) \]
\[ M = D_{\text{private}}(C_1) \]

also:

\[ C_2 = E_{\text{private}}(M) \]
\[ M = D_{\text{public}}(C_2) \]
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 **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.

Example function:

**Middle squares**

- Take a 100-digit number \( A \), and square it
- Let \( B = \) middle 100 digits of 200-digit result
One-way functions

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

Examples:

**Factoring:**

\[ pq = N \quad \text{EASY} \]

find \( p, q \) given \( N \) \quad \text{DIFFICULT}

**Discrete Log:**

\[ a^b \mod c = N \quad \text{EASY} \]

find \( b \) given \( a, c, N \) \quad \text{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 extremely hard to compute \( A \)
Hash functions

• **one-way function**
  – Rabin, 1958: McCarthy’s problem
  – middle squares, exponentiation, ...

• **[one-way] hash function**
  – message digest, fingerprint, cryptographic checksum, integrity check

• **encrypted hash**
  – message authentication code
  – only possessor of key can validate message
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**
  - Under development

• **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
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
Cryptographic toolbox

- Symmetric encryption
- Public key encryption
- One-way hash functions
- Random number generators
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>$7.2 \times 10^{16}$</td>
<td>2,178 years</td>
</tr>
<tr>
<td>64 bits</td>
<td>$1.8 \times 10^{19}$</td>
<td>&gt; 557,000 years</td>
</tr>
<tr>
<td>256 bits</td>
<td>$1.2 \times 10^{77}$</td>
<td>$3.5 \times 10^{63}$ 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.
Secure Communication
Communicating with symmetric cryptography

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

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

- Alice and Bob: 1 key
  - $K_{AB}$
  - 2 users: 1 key

- Alice, Bob, and Charles: 3 keys
  - $K_{AC}$, $K_{AB}$, $K_{BC}$
  - 3 users: 3 keys

- 6 users: 15 keys

- 4 users: 6 keys

100 users: 4,950 keys

1000 users: 399,500 keys

$n$ users: $\frac{n(n-1)}{2}$ keys

© 2014 Paul Krzyzanowski
4/24/2014
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?*

Whitfield Diffie had an 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 two parties to compute a **common key** without fear from eavesdroppers. Then, they can securely transmit a **session key**.
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 \( \alpha < p \)

• Each party generates a public/private key pair

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

  public key for user \( i \): \( Y_i = \alpha^{X_i} \mod p \)
Diffie-Hellman exponential key exchange

- Alice has secret key $X_A$
- Alice has public key $Y_A$
- Alice computes

$$K = Y_B^{X_A} \mod p$$

\[ K = \text{(Bob’s public key)} \ (\text{Alice’s private key}) \mod p \]
Diffie-Hellman exponential key exchange

- Alice has secret key $X_A$
- Alice has public key $Y_A$
- Alice computes

$$K = Y_{B}^{X_{A}} \mod p$$

- Bob has secret key $X_B$
- Bob has public key $Y_B$
- Bob computes

$$K' = Y_{A}^{X_{B}} \mod p$$

$K'$ = *(Alice’s public key)* *(Bob’s private key)* $\mod p$
Diffie-Hellman exponential key exchange

- Alice has secret key $X_A$
- Alice has public key $Y_A$
- Alice computes $K = Y_B^{X_A} \mod p$
  - expanding:
    $K = Y_B^{X_A} \mod p$
    $= (\alpha^{X_B} \mod p)^{X_A} \mod p$
    $= \alpha^{X_BX_A} \mod p$
- Bob has secret key $X_B$
- Bob has public key $Y_B$
- Bob computes $K' = Y_A^{X_B} \mod p$
  - expanding:
    $K' = Y_A^{X_B} \mod p$
    $= (\alpha^{X_B} \mod p)^{X_A} \mod p$
    $= \alpha^{X_BX_A} \mod p$

$K = K'$

$K$ is a **common key**, known only to Bob and Alice
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

– Use the extended Euclidean algorithm to compute the decryption key, $d$:
  
  $ed = 1 \mod ((p - 1)(q - 1))$
  
  $d = e^{-1} \mod ((p - 1)(q - 1))$

– 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 – not encryption!)

• 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

© 2014 Paul Krzyzanowski
Communication with public key algorithms

Different keys for encrypting and decrypting

– no need to worry about key distribution
Communication with public key algorithms

Alice

Alice’s public key: $K_A$

Bob

Bob’s public key: $K_B$

exchange public keys
(or look up in a directory/DB)
**Communication with public key algorithms**

Alice

- Alice’s public key: $K_A$

- Encrypt message with Bob’s public key: $E_B(M)$

Bob

- Bob’s public key: $K_B$

- Decrypt message with Bob’s private key: $D_b(C)$
Communication with public key algorithms

Alice

Alice’s public key: $K_A$

encrypt message with Bob’s public key

$E_B(M)$

decrypt message with Alice’s private key

$D_A(C)$

Bob

Bob’s public key: $K_B$

decrypt message with Bob’s private key

$D_B(C)$

encrypt message with Alice’s public key

$E_A(M)$
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.

© 2014 Paul Krzyzanowski
Communication with a hybrid cryptosystem

Bob’s public key: $K_B$

Get recipient’s public key
(or fetch from directory/database)
Communication with a hybrid cryptosystem

Alice

Bob

Bob’s public key: $K_B$

Pick random session key, $K$

Encrypt session key with Bob’s public key

$E_B(K)$

Bob decrypts $K$ with his private key

$K = D_b(E_B(K))$
Communication with a hybrid cryptosystem

Alice

encrypt message using a symmetric algorithm and key $K$

Bob

Bob’s public key: $K_B$

$E_B(K) \rightarrow K = D_b(E_B(K))$

decrypt message using a symmetric algorithm and key $K$
Communication with a hybrid cryptosystem

Alice

\[ E_B(K) \]

\[ E_K(M) \]

\[ D_K(C') \]

decrypt message using a symmetric algorithm and key \( K \)

Bob

Bob’s public key: \( K_B \)

\[ K = D_b(E_B(K)) \]

\[ D_K(C) \]

\[ E_K(P') \]

crypt message using a symmetric algorithm and key \( K \)

© 2014 Paul Krzyzanowski

4/24/2014
Message Authentication
Digital Signatures

• Validate the creator (signer) of the content
• Validate the content has not been modified since it was signed
• The content does not have to be encrypted
Digital signatures - public key cryptography

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

Encrypt message with Alice’s private key

Decrypt message with Alice’s public key

Trusted directory of public keys

Alice

Bob
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 $H=\text{hash}(M)$, it should be difficult to compute $M$, given $H$

• **Collision resistant**
  - Given $H=\text{hash}(M)$, it should be difficult to find $M'$, such that $H=\text{hash}(M')$
  - For a hash of length $L$, a perfect hash would require you to try $2^{(L/2)}$ 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**
  - Under development

- **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

• What you do:
  – Create a hash of the message
  – Encrypt the hash with your private key & send it with the message

• What the recipient does:
  – 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
Signatures: Hashes to the rescue!

If the hashes don’t match, that means one or more of:

1. The message was modified
2. The encrypted hash was modified
3. The hash was not encrypted by the correct party
4. The wrong public key is used to validate the signature
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
Digital signatures - public key cryptography

Alice sends Bob the message and the encrypted hash

\[ H(M) \xrightarrow{\text{S}} S = E_a(H(M)) \]
1. Bob decrypts the hash using Alice’s public key
2. Bob computes the hash of the message sent by Alice
Digital signatures - public key cryptography

If the hashes match ⇒ the signature is valid
– the encrypted hash *must* have been generated by Alice
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
Secure and authenticated messaging

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

\[ S = E_a(H(M)) \]
Secure and authenticated messaging

Alice picks a random key, $K$, and encrypts the message $(P)$ with it using a symmetric algorithm.
Secure and authenticated messaging

Alice encrypts the session key for each recipient of this message: Bob and Charles using their public keys.

\[ S = E_a(H(M)) \]

\[ C = E_K(M) \]

\[ C_2 = E_B(K) \]  \hspace{1cm} \text{for Bob} \]

\[ C_3 = E_C(K) \]  \hspace{1cm} \text{for Charles} \]
Secure and authenticated messaging

The aggregate message is sent to Bob and Charles
Secure and authenticated messaging

Bob receives the message:
- extracts key by decrypting it with his private key

**Message from Alice**
- Message:
- Signature:
- Key for Bob:
- Key for Charles:

$K = E_b(C_2)$
Secure and authenticated messaging

Bob decrypts the message using $K$

Message from Alice

- Message: 
- Signature: 
- Key for Bob: 
- Key for Charles: 

$M = D_K(C)$

$K = E_b(C_2)$
Secure and authenticated messaging

Message from Alice

- Message:
- Signature:
- Key for Bob:
- Key for Charles:

Bob computes the hash of the message

\[ M = D_K(C) \quad \rightarrow \quad H(M) \]

\[ K = E_b(C_2) \]
Secure and authenticated messaging

Bob looks up Alice’s public key

Message from Alice
- Message:
- Signature:
- Key for Bob:
- Key for Charles:

Directory of public keys

$K = E_b(C_2)$

$M = D_K(C)$

$H(M)$

$K_A$
Secure and authenticated messaging

Bob decrypts Alice’s signature using Alice’s public key.

Message from Alice

- Message: $M = D_K(C)$
- Signature: $H_1 = D_A(C_1)$
- Key for Bob: $K = E_b(C_2)$

Bob decrypts Alice’s signature using Alice’s public key.
Secure and authenticated messaging

Bob validates Alice’s signature

Message from Alice

- Message: $M = D_K(C)$
- Signature: $H_1 = D_A(C_1)$
- Key for Bob: $K = E_b(C_2)$
- Key for Charles: $H_1 = H(M)$?
The End