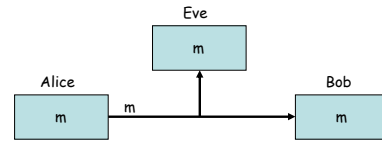


Cryptography: Some References

- David Kahn. *The Codebreakers* (1967).
- Simon Singh. *The Code Book* (1999).
- Niels Ferguson and Bruce Schneier. *Practical Cryptography* (2003).
- Bruce Schneier. *Applied Cryptography, 2nd Edition* (1996).
- Steve Burnett and Stephen Paine. *RSA Security's Official Guide to Cryptography* (2001).
- Brian A. LaMacchia et al. *.NET Framework Security* (2002).
- Michael Howard and David LeBlanc. *Writing Secure Code, 2nd Edition* (2002).
- Bruce Schneier. *Secrets and Lies* (2000).
- Charles Pfleeger and Shari Pfleeger. *Security in Computing, 3rd Edition* (2003).

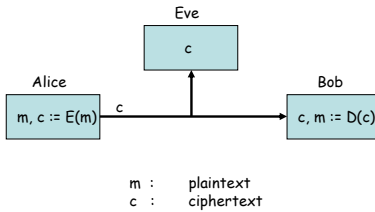
Cryptography

The Basic Problem



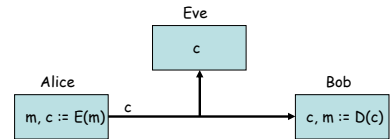
Cryptography

The Basic Solution

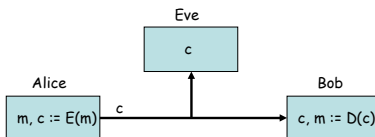


Cryptography

The Basic Solution



- Eve's attacks:
- Break the code (reverse engineering)
 - Replay the message
 - Modify the message
 - Block the message
 - Fabricate a new message

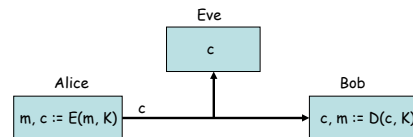


Restricted Algorithm: security is based on keeping the algorithm secret.

- Difficult to use in a large or changing group.
- When someone leaves the group, everyone must change algorithms
 - If someone reveals the secret, everyone must change algorithms

No quality control on the algorithm, so it's difficult to know how secure it is.

It is better to use a known, thoroughly studied algorithm whose security is based on a **key**.



Kerckhoffs' Principle: security is based only on keeping the **key** secret.

Note that in the situation shown, the key is the same for both encryption and decryption. What Bob computes is

$$D(E(m, K), K)$$

Algorithms of this type are called **symmetric**, or **secret key** algorithms. The key is a **shared secret** between Alice and Bob.

Symmetric Algorithms: One-Time Pad

Encode using a random key as long as the plaintext. Then throw away the key.

a
t
t
a
c
k
a
t
d
a
w
n

+

13
2
21
18
14
1
7
12
25
17
16
3

=

n
v
o
s
q
l
h
f
c
r
m
q

(mod 26)

Symmetric Algorithms: One-Time Pad

Encode using a random key as long as the plaintext. Then throw away the key.

a
t
t
a
c
k
a
t
d
a
w
n

+

13
2
21
18
14
1
7
12
25
17
16
3

=

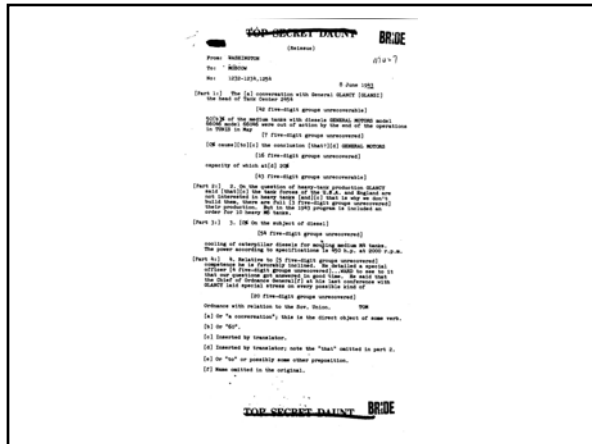
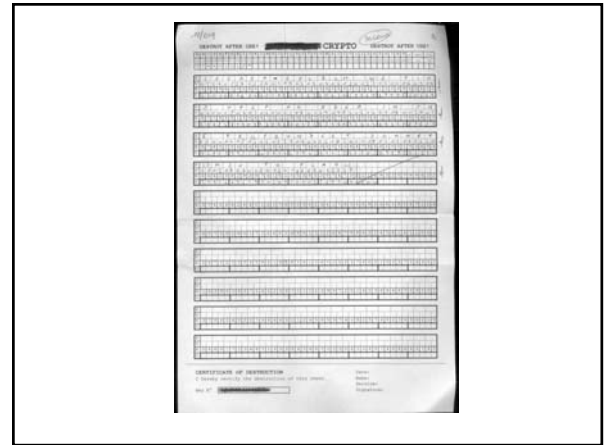
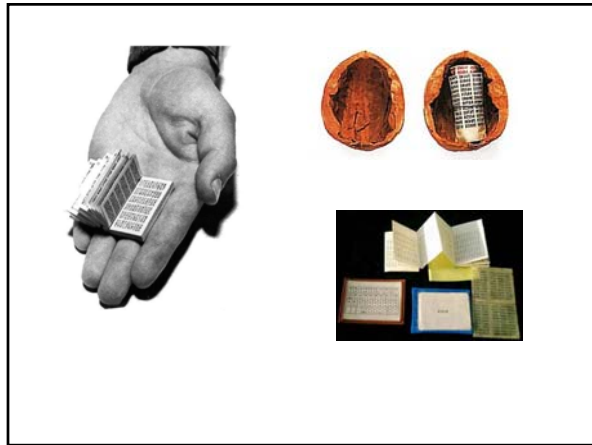
n
v
o
s
q
l
h
f
c
r
m
q

(mod 26)

Decode using the same pad and subtracting.

If the pad is truly random, and keys are never reused, this is a perfect encryption scheme.

Every plaintext message is equally likely, and there is no basis for analysis.



Symmetric Algorithms

Stream cipher

1
0
0
0
1
1
1
0
1
0
0
1
1
0
1
0

⊕

XOR

=

0
1
1
1
1
0
1
0
0
0
1
0
1
1
1
0

keystream

p	q	p ⊕ q
0	0	0
0	1	1
1	0	1
1	1	0

$p \oplus q \oplus q = p$
 $p \oplus q \Leftrightarrow \neg(p \leftrightarrow q)$
 $\Leftrightarrow \neg p \leftrightarrow q$
 $\Leftrightarrow p \leftrightarrow \neg q$

Symmetric Algorithms

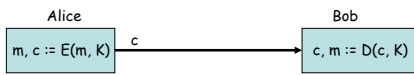
Stream cipher

- Very fast, compact code
- Same algorithm for encrypting and decrypting
- Most widely used algorithm: RC4
- Security is based on the randomness of the keystream
- Must not reuse the same key stream.
- How do you distribute the keystream?

Attacks

- Ciphertext only - all Eve has are encrypted messages
- Known plaintext - Eve knows the plaintext and ciphertext
- Chosen plaintext - Eve can influence the message Alice sends
- Chosen ciphertext - Eve has access to a decryption box
- Replay attack - Eve doesn't break the code, she just replays an encrypted message (such as a login!)
- Side-channel attack - Eve observes the computer hardware carrying out decryption operations, recording the timing, memory accesses, power consumption, heat, noise or other byproducts.
- Purchase-key attack - Eve gets key through bribery or coercion

The Key-Distribution Problem



Encryption and decryption are based on a secret key that Alice and Bob must share. How do they do that in a secure manner?

- Meet face to face
- Send by courier
- Break into parts, sent separately
- Obtain from trusted third party

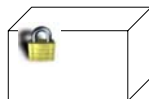
Alice

Bob



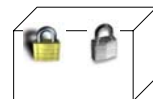
Alice

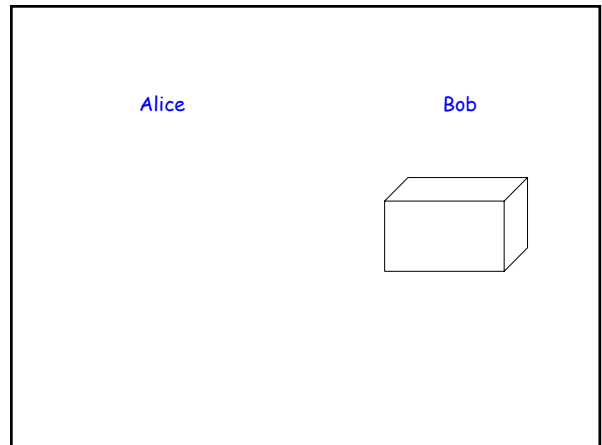
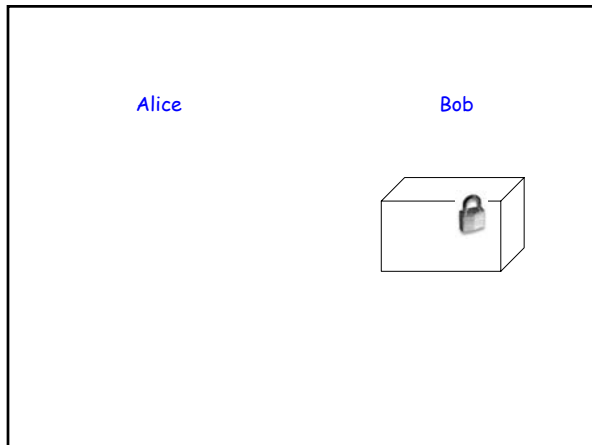
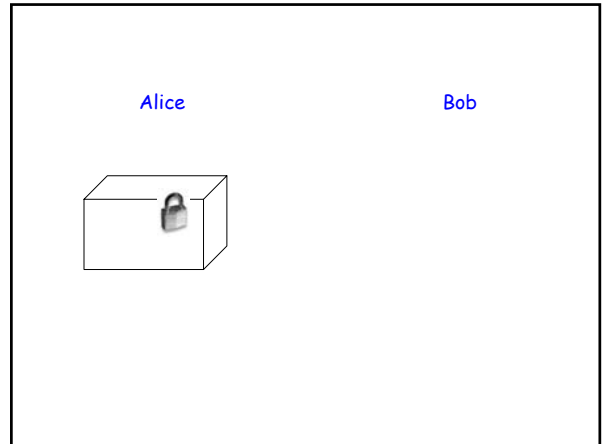
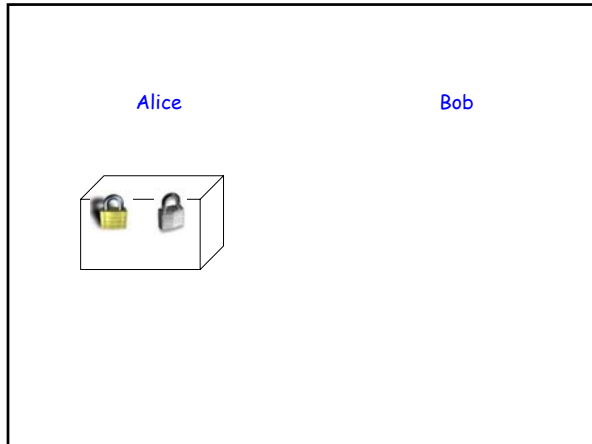
Bob



Alice

Bob





One-Way Functions

Given X, it is easy to compute $F(X) = Z$
 Given Z, it is very difficult to figure out what X was.

$$Y^X \pmod{p}$$

One-Way Functions

Given X, it is easy to compute $F(X) = Z$
 Given Z, it is very difficult to figure out what X was.

$$Y^X \pmod{p}$$

$$453^X \pmod{21997}$$

Suppose the result was 4327
 What was x?

Alice and Bob want to establish a key

They choose numbers Y and p (not secret)

<p>Alice</p> <p>Chooses a secret number A</p> <p>Computes $Y^A \pmod p$</p> <p>Sends this to Bob</p>	<p>Bob</p> <p>Chooses a secret number B</p> <p>Computes $Y^B \pmod p$</p> <p>Sends this to Alice</p>
------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------

Alice and Bob want to establish a key

They choose numbers Y and p (not secret)

<p>Alice</p> <p>Chooses a secret number A</p> <p>Computes $Y^A \pmod p$</p> <p>Sends this to Bob</p> <p>Computes $(Y^B \pmod p)^A \pmod p$</p>	<p>Bob</p> <p>Chooses a secret number B</p> <p>Computes $Y^B \pmod p$</p> <p>Sends this to Alice</p> <p>Computes $(Y^A \pmod p)^B \pmod p$</p>
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Alice and Bob want to establish a key

They choose numbers Y and p (not secret)

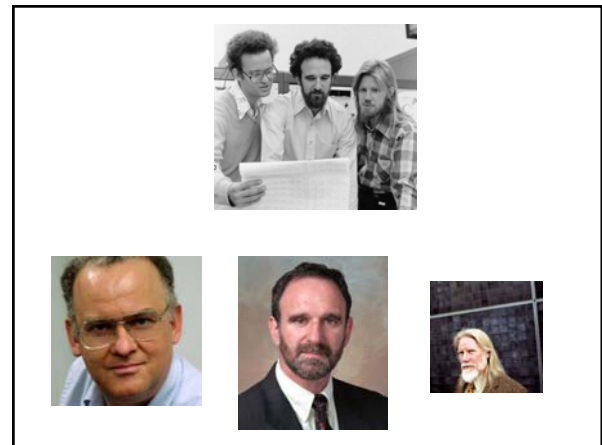
<p>Alice</p> <p>Chooses a secret number A</p> <p>Computes $Y^A \pmod p$</p> <p>Sends this to Bob</p> <p>Computes $(Y^B \pmod p)^A \pmod p$</p>	<p>Bob</p> <p>Chooses a secret number B</p> <p>Computes $Y^B \pmod p$</p> <p>Sends this to Alice</p> <p>Computes $(Y^A \pmod p)^B \pmod p$</p>
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

These two numbers are the same!

$Y^{AB} \pmod p$

The result can be used as a key by Alice and Bob.

Diffie-Hellman-Merkle key exchange



What does Eve know?

Y

p

$Y^A \pmod p$

$Y^B \pmod p$

Not sufficient to compute

$Y^{AB} \pmod p$

Eve needs to solve an equation like this

$$c = Y^X \pmod p$$

for X

If the equation was: $c = Y^X$

the answer would be $X = \log_y c$

Solving $c = Y^X \pmod p$ means finding the **discrete logarithm** (D-LOG)

which is much harder

A problem with DHM key exchange

DHM is subject to a man-in-the-middle attack

Alice $\xleftrightarrow{\text{DHM}}$ Eve $\xleftrightarrow{\text{DHM}}$ Bob

How can Bob be sure he is talking to Alice?

Authentication

Asymmetric algorithms: Public Key Cryptography

Alice: $m, c := E(m, k^E)$ \xrightarrow{c} Bob: $c, m := D(c, k^D)$

k^D allows Bob to decrypt messages encrypted with k^E

Knowing k^E does not allow Eve to compute k^D

Asymmetric algorithms: Public Key Cryptography

Alice: $m, c := E(m, k^E)$ \xrightarrow{c} Bob: $c, m := D(c, k^D)$

k^D allows Bob to decrypt messages encrypted with k^E

Knowing k^E does not allow Eve to compute k^D

k^E can be made public!

Asymmetric algorithms: Public Key Cryptography

Alice: $m, c := E(m, k_B^{\text{Pub}})$ \xrightarrow{c} Bob: $c, m := D(c, k_B^{\text{Pri}})$

Anyone wanting to send a message to Bob simply encrypts it using Bob's public key.

Asymmetric algorithms: Public Key Cryptography

Alice: $m, c := E(m, k_B^{\text{Pub}})$ \xrightarrow{c} Bob: $c, m := D(c, k_B^{\text{Pri}})$

Alice: $c, m := D(c, k_A^{\text{Pri}})$ \xleftarrow{c} Bob: $m, c := E(m, k_A^{\text{Pub}})$

Authentication

With a public-key algorithm, it is possible to encrypt with the **private** key and decrypt with the **public** key. Why do that?

Alice: $c := E(m, k_A^{\text{Pri}})$ Bob: $m := D(c, k_A^{\text{Pub}})$

Authentication

With a public-key algorithm, it is possible to encrypt with the **private** key and decrypt with the **public** key. Why do that?

Alice Bob

$c := E(m, k_A^{pri})$ $m := D(c, k_A^{pub})$

If the decryption is successful, Bob knows that the message **must have come from Alice**, because only Alice has her private key.

This is a **digital signature**, or, as we say, "Alice has 'signed' the message."

Authentication

Since public key encryption is slow, Alice would prefer not to sign the entire message.

Instead, she will sign a smaller **representative** of the message, called a **message digest** or **hash**.

A widely used hash function is SHA-1 (Secure Hash Algorithm), which takes in a message of any length and produces a hash of 160 bits.

So Alice can compute the hash of the message and then sign that.

Bob: receives the message and Alice's signed hash
undoes the signature to recover Alice's hash
computes his own hash of the message
makes sure the two hashes match

Authentication

Digital signatures provide:

Data integrity

If the message has been altered, Bob's new hash won't match the one Alice created.

Non-repudiation

Alice can't later say she didn't send the message, because her public key produces the correct hash.

Properties of a good hash algorithm

- The results appear to be random
- Small changes in the input produce large changes in the digest
- The algorithm is one-way
- You can't find a message that will produce a particular digest
- You can't find two messages that produce the same digest
- Since there are "only" 2^{160} possible digests, and there are infinitely many possible messages, there can obviously be **collisions**, but no one can find a collision on demand.

Crypto support in .NET

<p>Symmetric algorithms:</p> <ul style="list-style-type: none"> DES TripleDES RC2 AES (Rijndael) <p>Asymmetric algorithms:</p> <ul style="list-style-type: none"> RSA DSA 	<p>Hash algorithms:</p> <ul style="list-style-type: none"> MD5 SHA1 SHA256 SHA384 SHA512 <p>Pseudo-random number generator:</p> <ul style="list-style-type: none"> RNGCryptoServiceProvider (Random is not high quality)
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Summary

Symmetric-key encryption provides **privacy**.

One-way functions solve the **key-distribution** problem.

Public-key algorithms are **secure but costly**.

A combined approach can produce an **efficient protocol**.

Digital signatures provide **authentication**.

Digital signatures also provide **non-repudiation**.

Modern programming frameworks provide lots of classes for doing cryptography.