SQL Injection Attacks

Enter last name [ ]

Search

Your code:

```
string sql = SELECT * FROM Employees WHERE LastName LIKE '"
                 + TextBox1.Text.Trim() + "%'";
cmd.CommandText = sql;
```

User input:

```
%'; UPDATE Employees SET LastName = 'you have been hacked'; --
```

Result:

If the user has the required permissions, all last names are changed.

Preventing SQL Injection Attacks

- Validate user input
- Change ' to '' (two single quotes are the escape sequence for ')
- Limit the privileges of the account running the query
- Never show debugging information to the user
- Use parameterized commands
- Use stored procedures with specific permissions

Search the internet for more information

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).

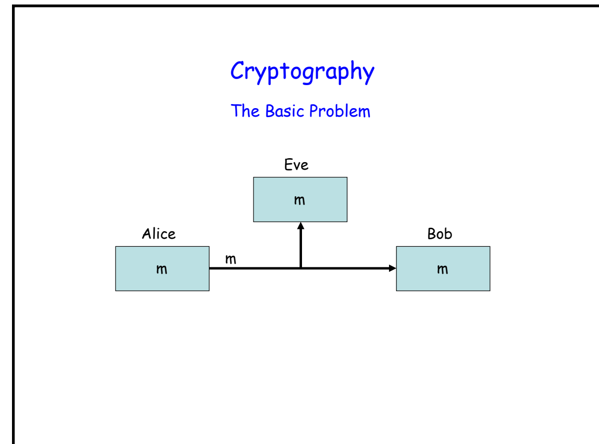
Steve Burnett and Stephen Paine. *RSA Security's Official Guide to Cryptography* (2001).

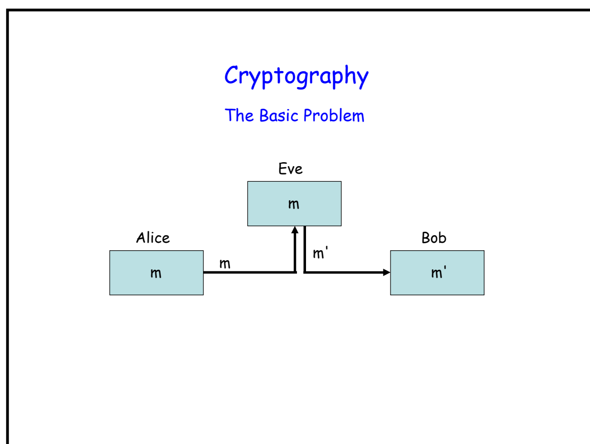
Brian A. LaMacchia et al. *.NET Framework Security* (2002).

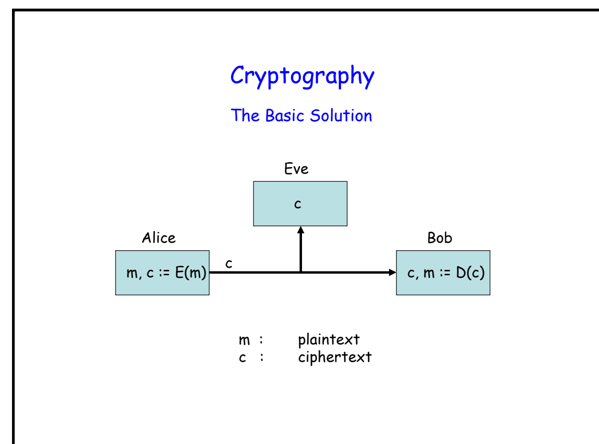
Michael Howard and David LeBlanc. *Writing Secure Code, 2nd Edition* (2002).

Bruce Schneier. *Secrets and Lies* (2000).

Charles Pfleeger and Shari Pfleeger. *Security in Computing, 3rd Edition* (2003).







Cryptography

The Basic Solution

Eve
c
Alice
m, c := E(m)
c
Bob
c, m := D(c)

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

Eve
c
Alice
m, c := E(m)
c
Bob
c, m := D(c)

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**.

Eve
c
Alice
m, c := E(m, K)
c
Bob
c, m := D(c, K)

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.

plaintext		key		ciphertext
a		13		n
t		2		v
t		21		o
a		18		s
c	+	14	=	q
k		1		l
a		7		h
t		12		f
d		25		c
a		17		r
w		16		m
n		3		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

plaintext		keystream		ciphertext
1		0		1
0		1		1
0		1		1
1		0		1
1	⊕ XOR	1	=	0
1		0		1
0		0		0
1		1		0
0		0		0
0		1		1
1		1		0
1		0		1

keystream

Symmetric Algorithms

Stream cipher

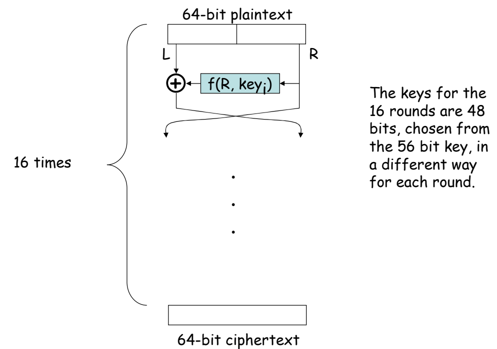
- Very fast, compact code
- Same algorithm for encrypting and decrypting
- Most widely used algorithm: RC4
- Security is based on the randomness of the keystream
- Must not reuse the same key stream. If you do:
  --If Eve has a plaintext/ciphertext pair, she has the keystream
  --XORing two ciphertexts gives the XOR of two plaintexts, which can be broken

Symmetric Algorithms

Block cipher – encrypts plaintext in blocks, usually 64 bits long

- Most widely used algorithms are DES, TripleDES, AES
- DES was developed in 1974-6 by IBM for the NBS, as a public encryption standard
- The key is 56 bits (plus 8 bits used as check bits)
- In 1997, researchers using 3,500 machines in parallel found a DES key in four months
- In 1999, a "DES cracker" found a DES key in 24 hours
- TripleDES effectively doubles the key length

The basic idea of DES



The keys for the 16 rounds are 48 bits, chosen from the 56 bit key, in a different way for each round.

AES

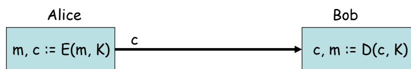
- In 1997, NIST called for a new algorithm
  - -- unclassified
  - -- publicly disclosed
  - -- available royalty-free worldwide
  - -- symmetric block cipher, for 128-bit blocks
  - -- usable with keys sizes of 128, 192, and 256 bits
- In 1998, 15 of the submitted algorithms were chosen
- In 1999, field narrowed to 5, which underwent extensive public and private analysis
- The winner: **Rijndael**, by Vincent Rijmen and Joan Daemen
- 9, 11, or 13 cycles, depending on key size

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!)
- Purchase-key attack – Eve gets key through bribery or coercion

The Key-Distribution Problem

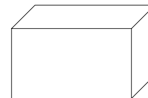
Alice Bob



Alice and Bob must share the key. 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



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 (mod p)

453^X (mod 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)

Alice	Bob
Chooses a secret number A	Chooses a secret number B
Computes Y^A (mod p)	Computes Y^B (mod p)
Sends this to Bob	Sends this to Alice

Alice and Bob want to establish a key

They choose numbers Y and p (not secret)

Alice	Bob
Chooses a secret number A	Chooses a secret number B
Computes Y^A (mod p)	Computes Y^B (mod p)
Sends this to Bob	Sends this to Alice
Computes $(Y^B \text{ (mod p)})^A$ (mod p)	Computes $(Y^A \text{ (mod p)})^B$ (mod p)

These two numbers are the same: Y^{AB} (mod p)

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

Diffe-Hellman-Merkle key exchange

What does Eve know?

Y

P

Y^A (mod p)

Y^B (mod p)

Not sufficient to compute

Y^{AB} (mod p)

Eve needs to solve an equation like this

$c = Y^A$ (mod p)

for A

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

the answer would be $A = \log_Y c$

Solving $c = Y^A$ (mod p) means finding the discrete logarithm (D-LOG)

which is much harder

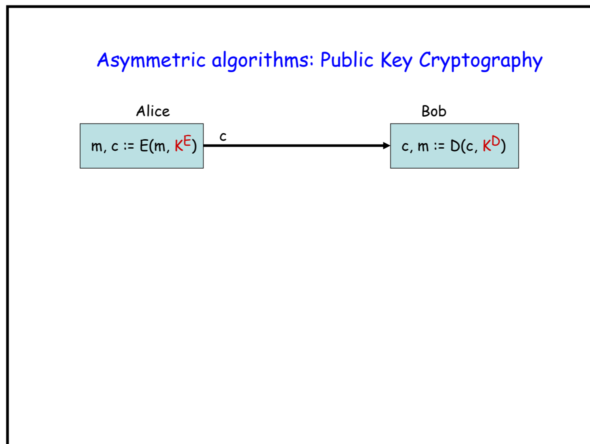
A problem with DHM key exchange

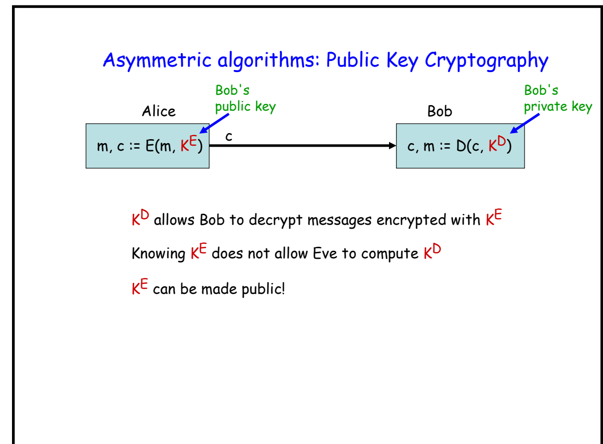
The man-in-the-middle attack

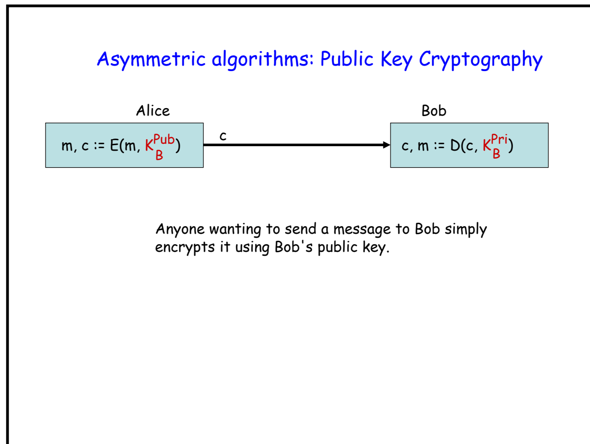
Alice ⟷ DHM ⟷ Eve ⟷ DHM ⟷ Bob

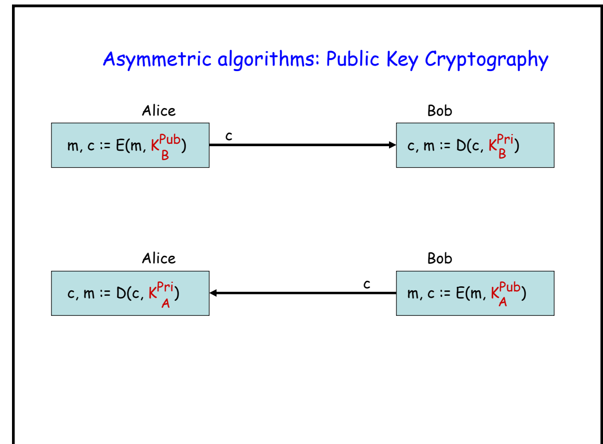
How can Bob be sure he is talking to Alice?

Authentication









Asymmetric algorithms: RSA

(Rivest, Shamir, Adleman)

Public key

Exponent e
Modulus n (product of two large primes p and q)

Private key

Exponent d (computed from e, p, and q)
Modulus n (the same one)

Typical modulus 1024 bits

Asymmetric algorithms: RSA

(Rivest, Shamir, Adleman)

Public key

Exponent e
Modulus n (product of two large primes p and q)

Private key

Exponent d (computed from e, p, and q)
Modulus n (the same one)

Typical modulus 1024 bits

Eve's problem Find p and q such that pq = n (prime factorization)

The prime factors of 18,206,927 are 9,419 and 1,933

RSA

RSA calculates the ciphertext from the plaintext like this:

$C = M^e \bmod n$

So e and n are public.

Decryption is done the same way, using a private number d:

$M = C^d \bmod n$

So we need it to be the case that

$M^{ed} \bmod n = M \bmod n$

and that no one can figure out d from e and n.

Finding an efficient protocol

Asymmetric algorithms are 500-1000 times slower than symmetric algorithms

So......

Alice picks a session key

Alice uses RSA to send the session key to Bob

The session key is used with AES to encrypt the messages

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

Symmetric algorithms:	Hash algorithms:
DES	MD5
TripleDES	SHA1
RC2	SHA256
Rijndael	SHA384
	SHA512

Asymmetric algorithms:	Pseudo-random number generator:
RSA	RNGCryptoServiceProvider
DSA	(Random is not high quality)

Encryption/Decryption uses CryptoStreams

Backing store in memory

CryptoStream

plaintext → Encryptor → ciphertext

MemoryStream

```
CryptoStream encryptionStream =
        new CryptoStream(
          memoryStream,          ← Target stream
          des.CreateEncryptor(), ← Transform
          CryptoStreamMode.Write ← Mode
                 );
```

Encryption/Decryption uses CryptoStreams

Backing store in memory

CryptoStream

ciphertext → Decryptor → plaintext

MemoryStream

```
CryptoStream decryptionStream =
        new CryptoStream(
          memoryStream,          ← Target stream
          des.CreateDecryptor(), ← Transform
          CryptoStreamMode.Write ← Mode
                 );
```

Encryption/Decryption uses CryptoStreams

Backing store in memory

CryptoStream

ciphertext → Decryptor → plaintext

MemoryStream

Memory streams are usually created to use empty, expandable buffers as their backing stores:

```
MemoryStream ms = new MemoryStream();
```

After writing to the stream, the buffer is available:

```
byte[] buffer = ms.GetBuffer();
```

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 (e.g., .NET) provide lots of classes for doing cryptography.