University of Arizona

CSc 620
Security Through Obscurity

Christian Collberg

January 31, 2002

Cryptography

Terminology

Encryption: Disguising a message to hide its contents.

Plaintext: The message we want to hide.

Ciphertext: The encrypted message.

Encode: Converting the plaintext into a standard alphabet.

Decode: Converting the encoded message back into the plaintext.

Encipher: Converting the plaintext to the ciphertext. Example: `uuencode` converts a binary file into ASCII text. Also, **encrypt**.

Decipher: Converting the ciphertext to the plaintext. Example: `uudecode` reconstitutes a uuencoded file. Also, **decrypt**.

Terminology...

Cipher: A map from the space of the plaintext to the space of the ciphertext.

Stream cipher: A cipher that enciphers the plaintext one character at a time.

Block cipher: A cipher that enciphers the plaintext in chunks of characters.

Mathematical Terminology

P: The plaintext.

M: The plaintext (message).

C: The ciphertext.

E: The encryption function.

D: The decryption function.

$$\begin{aligned} E(M) &= C \\ D(C) &= M \\ D(E(M)) &= M \end{aligned}$$

Slide 11–4

Terminology – Keys

K**:** The key, used by the encryption and decryption functions.

Keyspace: The range of possible values of the key.

$$\begin{aligned} E_K(M) &= C \\ D_K(C) &= M \\ D_K(E_K(M)) &= M \end{aligned}$$

Slide 11–5

Terminology – Keys. . .

- Some algorithms use different keys for encryption and decryption:

$$\begin{aligned} E_{K_1}(M) &= C \\ D_{K_2}(C) &= M \\ D_{K_2}(E_{K_1}(M)) &= M \end{aligned}$$

Cryptosystem: An algorithm, all possible plaintexts, ciphertexts, and keys.

Slide 11–6

Terminology – Cryptanalysis

Ciphertext-only attack

We have: the ciphertext of several messages that have been encrypted with the same key.

We recover: the plaintexts, or the key.

Known-plaintext attack

We have: the ciphertexts and corresponding plaintexts of several messages.

We recover: the key(s) or an algorithm to decrypt new messages encrypted with the same key.

Slide 11–7

Terminology – Cryptanalysis. . .

Chosen-plaintext attack

We have: the ciphertext of several messages that have been encrypted with the same key; *we get to choose the plaintexts.*

We recover: the key(s) or an algorithm to decrypt new messages encrypted with the same key.

Rubber-hose cryptanalysis

We have: access to a person who can be threatened, blackmailed, tortured,. . . AKA, **purchase-key attack.**

We recover: Everything!

Slide 11–8

Substitution Ciphers

- In a **monoalphabetic** cipher each character of the plaintext is mapped to a corresponding character of the ciphertext:

$$A \to 9, B \to 11, \ldots$$

- In a **homophonic** cipher each character of the plaintext is mapped to several characters of the ciphertext:

$$A \to \{9, 10, 11\}, B \to \{3, 1, 8\}, \ldots$$

- In a **polygram** cipher blocks of characters in the plaintext are mapped to blocks of characters in the ciphertext:

$$ARF \to RTW, ING \to PWQ, \ldots$$

Slide 11–9

Substitution Ciphers...

- In a **polyalphabetic** cipher you have several keys, each one used to encrypt one letter of the plaintext. We recycle keys when we run out of them:

K_1	K_2	K_3	K_1	K_2	K_3	K_1	K_2	K_3
a	t	t	a	c	k		a	t
x	v	d	x	t	d	r	p	d

The number of keys is called the *period*.

- In a **running-key** cipher (AKA **book cipher**) one text is used to encrypt another.

Slide 11–10

Monoalphabetic Substitution Ciphers

Caesar Cipher: Add 3 to the ASCII value of each character, mod 26:

$$A \to D, B \to E, X \to A, \ldots$$

ROT13: Unix utility used on usenet. Adds 13 mod 26 to each letter.

$$P = \texttt{ROT13}(\texttt{ROT13}(P))$$

- These methods are simple to break: use the fact that different letters in the English alphabet occur with different frequencies.

Slide 11–11

Transposition Ciphers

- In a **transposition** cipher the original characters of the plaintext are not changed, but simply moved around in the ciphertext. Letter frequencies don't change.
- In a **simple columnar transposition** cipher we write the plaintext horizontally in a fixed width table, and read it off vertically.

  The plaintext `attack␣at␣dawn` could be enciphered into `actwtk␣nt␣f␣aaa␣`, using this table:

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

Slide 11–12

Exclusive-OR

$$
\begin{array}{ccccc|ccccccc}
0 & \oplus & 0 & = & 0 & a & \oplus & a & & & = & 0 \\
0 & \oplus & 1 & = & 1 & a & \oplus & b & \oplus & b & = & a \\
1 & \oplus & 0 & = & 1 & a & \oplus & a & \oplus & a & = & a \\
1 & \oplus & 1 & = & 0 & & & & & & &
\end{array}
$$

- Since xor-ing the same value twice gives us the original, we get a simple symmetric algorithm:

$$
\begin{aligned}
P \oplus K &= C \\
C \oplus K &= P
\end{aligned}
$$

Slide 11–13

One-Time Pads

- The **pad** is a large, non-repeating set of random key letters.
- To encrypt, add each plaintext letter to the next letter on the pad, mod 26:

r	*a*	*n*	*d*	*o*	*m*	*p*	*a*	*d*
a	*t*	*t*	*a*	*c*	*k*	*a*	*t*	*d*
s	*u*	*h*	*e*	*r*	*w*	*q*	*u*	*h*

  Decryption is done the same.
- This is provably secure, provided you have a truly random set of pad letters and never reuse the pad.

Slide 11–14

DES

- *Data Encryption Standard.* Designed by IBM and "certified" by NSA. Widely used in industry. The NSA is rumored to be able to break it in 3–15 minutes.
- A block cipher. Blocks are 64 bits, keys 56 bits:

$$\boxed{\texttt{64-bit plaintext}} \xrightarrow{\texttt{56-bit key}} \boxed{\texttt{64-bit ciphertext}}$$

  DES is a symmetric algorithm; the same key and algorithm is used for encryption and decryption.

Slide 11–15

Confusion and Diffusion

- DES is a combination of two basic principles:

  confusion: Confusion scrambles up the letters of the plaintext so that it has no direct relationship to the ciphertext. Substitution ciphers do this.

  diffusion: Diffusion spreads the plaintext out over the ciphertext. Transposition (AKA permutation) ciphers do this.
- DES has 16 rounds. Each round consists of a substitution followed by a permutation.

DES – The Rounds

1. Split 64-bit input into L_0 and R_0, each 32 bits.
2. Round 1:

$$\begin{aligned} L_1 &= R_0 \\ R_1 &= L_0 \oplus f(R_0, K_1) \end{aligned}$$

3. Round i:

$$\begin{aligned} L_i &= R_{i-1} \\ R_i &= L_{i-1} \oplus f(R_{i-1}, K_i) \end{aligned}$$

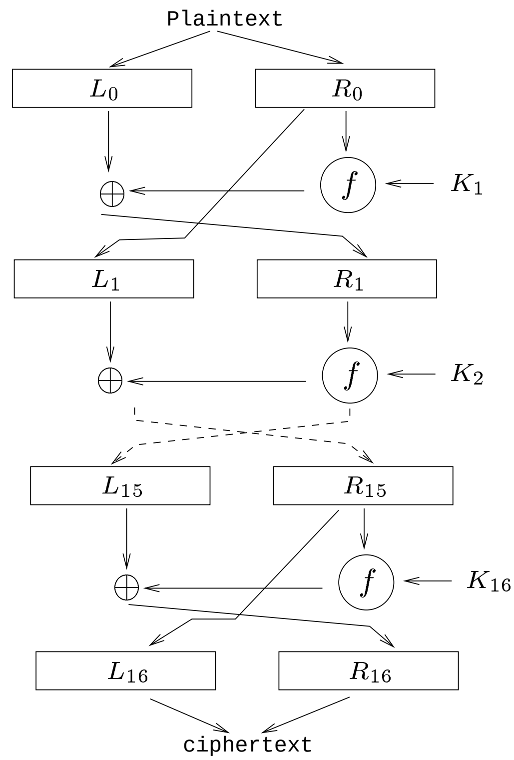
4. Round 16:

$$\begin{aligned} L_{16} &= R_{15} \\ R_6 &= L_{15} \oplus f(R_{15}, K_{16}) \end{aligned}$$

5. Return the 64-bit ciphertext $L_{16}R_{16}$.

Slide 11–16

DES Overview – The Rounds...



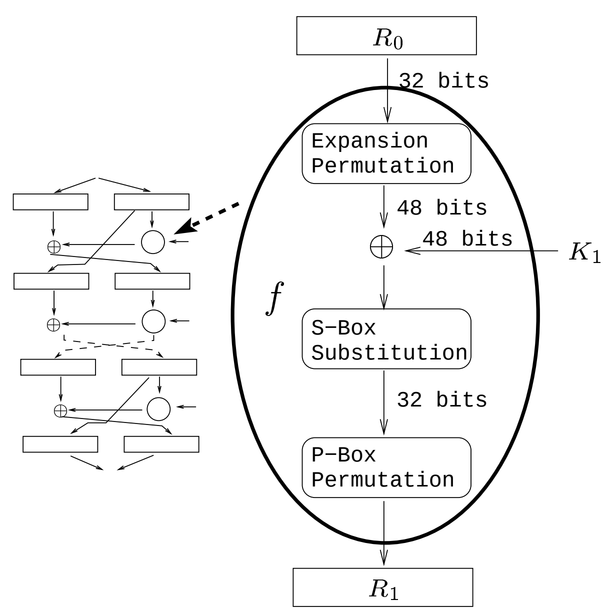
Slide 11–17

DES – The f Function

- f is the function that combines the data (the right 32-bit half, R) with the key during each round:
  1. Get 48 bits of the key,
  2. Expand R to 48 bits using the *expansion permutation*,
  3. XOR the expanded R and the compressed key,
  4. Send the result through 8 S-boxes using the *S-Box Substitution* to get 32 new bits,
  5. Permute the result using the *P-Box Permutation*,
- The result of the f function is XORed with the left half (L) to get the new right half.

Slide 11–18

DES – The f Function...



Slide 11–19

DES – Tables, Tables, Tables,. . .

- The various permutations, shifts, and substitutions used by the algorithm are all specified using tables.
- The shift table expresses by how much (1 or 2 steps) the key should be circularly shifted each round. Because of this shift, a different part of the key will be used in each round.

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Shift	1	1	2	2	2	2	2	2	2	1	2	2	2	2	2	1

Slide 11–22

DES – Expansion Permutation

- The expansion permutation takes the 32 bits of R, permutes these bits, and, by copying some of them, expands R into 48 bits.
- The table below says that bit 32 of R moves into position 1, bit 1 moves to 2, 2 to 3, 4 to 5, 5 to 6, 4 to 7, 5 to 8, etc.

32	1	2	3	4	5	4	5	6	7	8	9
8	9	10	11	12	13	12	13	14	15	16	17
16	17	18	19	20	21	20	21	22	23	24	25
24	25	26	27	28	29	28	29	30	31	32	1

Slide 11–23

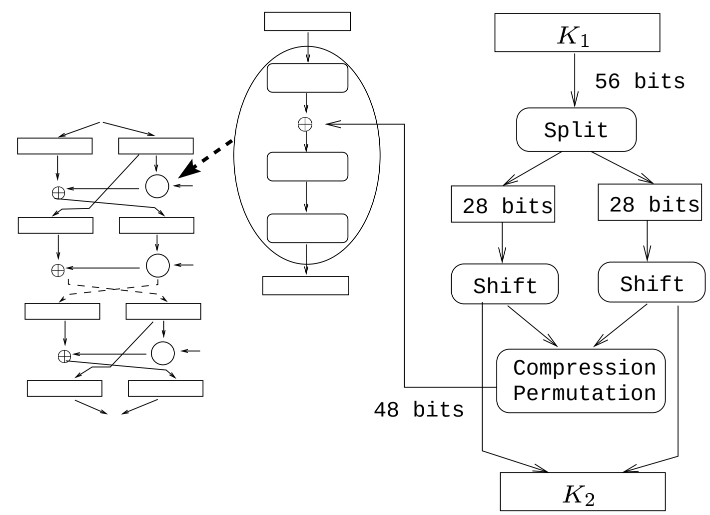
DES – Key Manipulation

- Each round, 48 bits of the 56-bit key is extracted and used by the f function. Also, a new version of the key is generated to be used in the next round:

1. Split the key into two halves, each 28 bits.
2. Shift each half by 1 or 2 steps.
3. Compress the two halves into 48 bits using the *Compression Permutation*,
4. Merge the two shifted halves into the key for the next round.

Slide 11–20

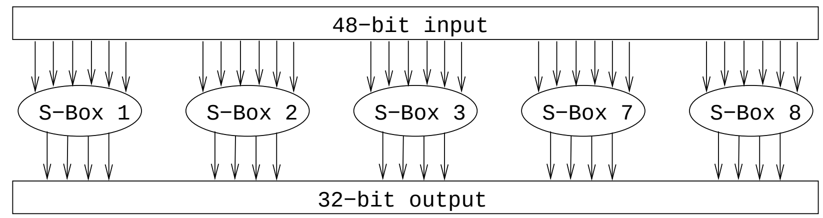
DES – Key Manipulation. . .



Slide 11–21

DES – S-Box Substitution

- The expanded right (R_{i-1}) and the compressed key (K_i) are XORed together. The 48-bit result is then passed through 8 S-Boxes (Substitution Boxes) to form 32 bits.
- Each one of the 8 S-Boxes is different. Each takes 6 bits of input and produces 4 bits of output:



Slide 11–24

DES – S-Box Substitution Tables

- Each S-Box is given by a table consisting of 4 rows of 16 numbers. Each number is a 4-bit quantity.
- The input to each S-Box is 6 bits:

b_1	b_2	b_3	b_4	b_5	b_6

- The S-Box table is indexed by taking the outer 2 bits (b_1 and b_6) and forming a number between 0 and 3. This is used to get the row number, The middle 4 bits ($b_2 \cdots b_5$) get the column number.
- The S-Boxes are what gives DES its security.

Slide 11–25

S-Box 1

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S-Box 2

15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S-Box 3

10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S-Box 4

7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

Slide 11–26

S-Box 5

2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S-Box 6

12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S-Box 7

4	11	2	14	15	0	8	13	3	12	9	7	6	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S-Box 8

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Slide 11–27

Slide 11–28

DES – P-Box Permutation Tables

- The final operation in the f function is the P-Box.
- It's a stright permutation of the 32 bits that come out of the S-Box.
- Bit number 16 moves into position 1, bit 7 into 2, 20 into 3, etc:

16	7	20	21	29	12	28	17	1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9	19	13	30	6	22	11	4	25

Slide 11–29

DES – Decryption

- The same algorithm we just described for encryption can also be used for decryption.
- However, we have to use the keys in the reverse order.
- That is, during decryption we use the keys

$$K_{16}, K_{15}, \cdots, K_2, K_1.$$

Slide 11–30

Symmetric Encryption Protocol

- Now that we have access to a symmetric cryptosystem like DES, how do we *use* it?
- We have to describe a *protocol* that shows how each party uses the cryptosystem to solve a communication/security problem.

1. Alice and Bob agree on a cryptosystem.
2. Alice and Bob agree on a key.
3. Alice encrypts her plaintext, getting a ciphertext.
4. Alice sends the ciphertext to Bob.
5. Bob decrypts the message using the same cryptosystem and key.

Slide 11–31

Symmetric Encryption Protocol – Attacks

- What can an attacker do?
- If Eve listens in on the communication between Alice and Bob she will get a sequence of ciphertext messages. She can use these to launch a ciphertext-only attack.
- Eve could also try to listen in to the first two parts of the protocol, where Alice and Bob decide on a key and cryptosystem to use.
- Eve could also sit in the middle, intercept Alice's messages, and substitute her own messages encrypted with the key she has discovered.

Slide 11–32

Public Key Protocol

- Key-management is the main problem with symmetric algorithms – Bob and Alice have to somehow agree on a key to use.
- In public key cryptosystems there are two keys, a public one used for encryption and and private one for decryption.

1. Alice and Bob agree on a public key cryptosystem.
2. Bob sends Alice his public key, or Alice gets it from a public database.
3. Alice encrypts her plaintext using Bob's public key and sends it to Bob.
4. Bob decrypts the message using his private key.

Slide 11–33

A Hybrid Protocol

- In practice, public key cryptosystems are not used to encrypt messages – they are simply too slow.
- Instead, public key cryptosystems are used to encrypt *keys for symmetric cryptosystems*. These are called *session keys*, and are discarded once the communication session is over.

1. Bob sends Alice his public key.
2. Alice generates a session key K, encrypts it with Bob's public key, and sends it to Bob.
3. Bob decrypts the message using his private key to get the session key K.
4. Both Alice and Bob communicate by encrypting their messages using K.

Slide 11–34

RSA

- RSA is the best know public-key cryptosystem. Its security is based on the (believed) difficulty of factoring large numbers.

1. Select two random primes p and q.
2. Compute $n = pq$.
3. Compute $\phi(n) = (p-1)(q-1)$.
4. Select a small odd integer e relatively prime with $\phi(n)$. 3, 17, or 65537 are good choices.
5. Compute $d = e^{-1} \pmod{\phi(n)}$.
6. $P = (e, n)$ is the RSA public key.
7. $S = (d, n)$ is the RSA private key.

Slide 11–35

RSA...

- To encrypt a message M, do

$$P(M) = M^e \pmod{n}$$

- To decrypt a ciphertext C, do

$$S(C) = C^d \pmod{n}$$

Slide 11–36

Modular Arithmetic

- $a \equiv b \pmod{n}$ if $a = b + kn$ for some integer k.

$$
\begin{aligned}
(a+b) \pmod{n} &= ((a \pmod{n}) + (b \pmod{n})) \pmod{n} \\
(a-b) \pmod{n} &= ((a \pmod{n}) - (b \pmod{n})) \pmod{n} \\
(a*b) \pmod{n} &= ((a \pmod{n}) * (b \pmod{n})) \pmod{n}
\end{aligned}
$$

$$
\begin{aligned}
23 &\equiv 11 \pmod{12} \\
23 &\equiv 2 \pmod{7} \\
&(10+13 \pmod{7}) \\
&= ((10 \pmod{7}) + (13 \pmod{7})) \pmod{7} \\
&= (3+6) \pmod{7} = 2
\end{aligned}
$$

Slide 11–37

Modular Arithmetic...

- Modular exponentiation is an important operation in cryptography:

$$
\begin{aligned}
a^x \pmod{n} &= (a * a * \cdots * a) \pmod{n} \\
a^8 \pmod{n} &= (a*a*a*a*a*a*a*a) \pmod{n} \\
&= ((a^2 \pmod{n})^2 \pmod{n})^2 \pmod{n}
\end{aligned}
$$

Slide 11–38

Modular Arithmetic...

- The inverse of 4 is $\frac{1}{4}$. Modular inverses are harder.
- To find the the inverse of 4 modulo 7 we want to compute:

$$4 * x = 1 \pmod{7}$$

which is the same as finding integers x and k such that

$$4x = 7k + 1$$

- This is also written: $4^{-1} = x \pmod{n}$.
- Sometimes inverses exist, sometimes not:

$$
\begin{aligned}
5^{-1} &= 3 \pmod{14} \\
2^{-1} &= \ ? \pmod{14}
\end{aligned}
$$

Slide 11–39

RSA Example

1. Select two primes: $p = 47$ and $q = 71$.
2. Compute $n = pq = 3337$.
3. Compute $\phi(n) = (p-1)(q-1) = 3220$.
4. Select $e = 79$.
5. Compute

$$
\begin{aligned}
d &= e^{-1} \pmod{\phi(n)} \\
&= 79^{-1} \bmod 3220 \\
&= 1019
\end{aligned}
$$

6. $P = (79, 3337)$ is the RSA public key.
7. $S = (1019, 3337)$ is the RSA private key.

Slide 11–40

Encrypt $M = 6882326879666683$.

1. Break up M into 3-digit blocks:

   $$m = \langle 688, 232, 687, 966, 668, 003 \rangle$$

   Note the padding at the end.

2. Encrypt each block:

   $$\begin{aligned} c_1 &= m_1^e \bmod n \\ &= 688^{79} \bmod 3337 \\ &= 1570 \end{aligned}$$

   We get:

   $$c = \langle 1570, 2756, 2091, 2276, 2423, 158 \rangle$$

3. Decrypt each block:

   $$\begin{aligned} m_1 &= c_1^d \bmod n \\ &= 1570^{1019} \bmod 3337 \\ &= 688 \end{aligned}$$

Slide 11–41

Digital Signatures

- Often, Bob will want to make sure that the document he got from Alice in fact originated with her.
- In the non-digital world, Alice would sign the document. We can do the same with digital signatures.

1. Alice encrypts her document with her private key, thereby signing it.
2. Alice sends the signed document to Bob.
3. Bob decrypts the document using Alice's public key, thereby verifying her signature.

Slide 11–42

One Way Hash Functions

- Public key algorithms are too slow to sign large documents. A better protocol is to use a one way hash function.

1. Alice computes a one-way hash of her document.
2. Alice encrypts the hash with her private key, thereby signing it.
3. Alice sends the encrypted hash and the document to Bob.
4. Bob decrypts the hash Alicec sent him, and compares it against a hash he computes himself of the document. If they are the same, the signature is valid.

Slide 11–43

Software

- An RSA demonstration applet: `http://cisnet.baruch.cuny.edu/holowczak/classes/9444/rsademo/#overview`
- An RSA demonstration applet: `http://www.cs.brandeis.edu/~tim/Packages/jlib2/applets/rsa.html`
- An RSA demonstration applet: `http://www-fs.informatik.uni-tuebingen.de/~reinhard/krypto/English/4.1.applet.e.html`
- A modular exponentiation applet: `http://www.math.umn.edu/~garrett/crypto/a01/FastPow.html`

Slide 11–44

Software – des

```
> cat message
Attack at dawn
> des -k 12345678 -e message ciphertext
> od ciphertext
0000000 036367 147372 026370 003617 051751 024226
132155 011603 0000020
> des -k 12345678 -d ciphertext
Attack at dawn
```

Slide 11–45

Software – PGP

- pgp is a public domain implementation of RSA.
- First you have to generate a public and private key-pair:

```
> pgp -kg
Choose 1, 2, 3, or enter desired number of bits: 1
Generating a 1024-bit DSS/DH key.

Enter a user ID for your public key: Sally Student \
   <sally@student.org>
Enter pass phrase: go 2 Taco Hell
Enter same pass phrase again: go 2 Taco Hell

> pgp -kvv
Type bits/keyID    Date       User ID
sec  1024/68CBA84E 2002/01/31 Sally Student <sally@student.org>
```

Slide 11–46

Software – PGP. . .

- We can extract Sally's public key from the keyfile:

```
> pgp -kx sally sally.pgp
> cat sally.pgp
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: PGP 6.5.8
mQGiBDxZiKMRBADPtM3zA2n0wuQnwSc8bWy...
SEKU5724Fcd8nOLD5/KAQkw0lFo4BzTn8QR...
r5o20x7b+ZnwDhkddFEfrDnWz2vzNL4wGF9...
/ENdU4qStgd3aL82DWQBKw0EAJXjDGcUepG...
XAz6ZCrz7kMC9727t7xG73gVy61MLnexCU1...
....
-----END PGP PUBLIC KEY BLOCK-----
```

Slide 11–47

Software – PGP. . .

- We can encrypt a message using Sally's public key:

```
> pgp -ea message sally
Key for user ID: Sally Student <sally@student.org>
1024-bit key, Key ID 68CBA84E, created 2002/01/31
Transport armor file: message.asc
> cat message.asc
-----BEGIN PGP MESSAGE-----
Version: PGP 6.5.8
qANQR1DBwE4Da/7A1bGYfWQQA/9mrK0...
ChLqlos4oh04+uLMsrJqQnH5Jz+FEZH...
f6I9C+sXQIGSHdjSHSqgCIjC6qK4AyV...
0g+cvAP+K7LA7ke+83LScVCVLr0lujL...
-----END PGP MESSAGE-----
```

Software – PGP...

- Sally can now decrypt the message using her private key:

```
> pgp message.asc
Key for user ID: Sally Student <sally@student.org>
Enter pass phrase: go 2 Taco Hell
Plaintext filename: message
> cat message
Attack at dawn
```

Slide 11–48

Software – PGP...

- Sally can sign her message before sending it to Bob:

```
> pgp -sta message -u sally
Enter pass phrase: go 2 Taco Hell
> cat message.asc
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

Attack at dawn

-----BEGIN PGP SIGNATURE-----
Version: PGP 6.5.8
iQA/AwUBPFmcQL018iJoy6hOEQLmWwCeMoW...
-----END PGP SIGNATURE-----
> pgp message.asc
File is signed.  Good signature from
"Sally Student <sally@student.org>".
Signature made 2002/01/31 19:34 GMT
```

Slide 11–49

Readings and References

- The information in this lecture is taken from *Applied Cryptography* by Bruce Schneier.
- Peter Wayner, *Disappearing Cryptography*.
- More on RSA: `http://www.hobnob.com/mcs225/`

Slide 11–50