Current Concepts and Issues in Cryptography and Security: An Overview

Assistant Prof. Sarah Mocas
Portland State University

Background

- Underlying security is cryptography. Secure cryptographic algorithms do not imply secure implementations using these algorithms!

- Security often rests in:

  (1) the security in communication protocols using cryptographic algorithms,

  (2) the security in key distribution protocols.

Cryptographic Services

The following properties are "services" which may be provided by a set of cryptographic functions used in conjunction with protocols and keys.

1 *Confidentiality* is the property of communicating such that the intended recipients know what was sent but unintended parties can not determine what was sent.

  Encryption is a mechanism used to provide confidentiality.

  Steganography is the art of hiding a secret message within a larger public message. This provides a service closely related to confidentiality which is usually associated with digital watermarking or covert channels.

Cryptographic Services

2 *Authentication* is the property of knowing that the data received is the same as the data that was sent and that the claimed sender is in fact the actual sender.

3 *Integrity* is the property of ensuring that the data is transmitted from source to destination without undetected alteration. (This is a "cryptographic checksum".)

4 *Digital Signatures*, **like Authentication, provide the property of knowing that the data received is the same as the data that was sent and that the claimed sender is in fact the actual sender. In addition the signed message can only have been created by the signer and the signer can not disavow a signature.**

In authentication:
If Alice and Bob share a secret key K and Alice sends Bob a message M that has been "authenticated" using K then the message could have been created by either Alice or Bob (since they both own K).

For a signed message:
If Alice sends Bob a message M that has been "signed" by Alice using a key K_A then the message could only have been created by Alice.

Confidentiality

1. *Encryption* is the process of disguising a plain-text message P and producing ciphertext C, (using a secret key K):
   $E_K(P) = C$.

2. *Decryption* is the process of transforming ciphertext into plaintext:
   $D_K(C) = P$.

3. $D_{K2}(E_{K1}(P)) = P$, where $K1, K2$ are keys.

Plaintext	$\longrightarrow$	Ciphertext	$\longrightarrow$	Plaintext
	Encryption $K1$		Decryption $K2$	
	$E_{K1}(P) = C$		$D_{K2}(C) = P$	

Confidentiality

There are two standard types of cryptosystems.

(1) **Symmetric or Secret Key Cryptosystems:**

$K1$ and $K2$ are secret.
Usually $K1 = K2$.

(2) **Public-key Cryptosystems:**

$K1$ is public.
$K2$ is secret or private.
$K1 \neq K2$.

Plaintext	$\longrightarrow$	Ciphertext	$\longrightarrow$	Plaintext
	Encryption $K1$		Decryption $K2$	
	$E_{K1}(P) = C$		$D_{K2}(C) = P$	

Confidentiality

Symmetric Key Cryptography

1. The decryption key can be calculated from the encryption key. Often the same key.
2. Key distribution is harder.
3. Encryption is usually fast.
4. Example: Block encryption algorithms: a fixed key used to encrypt/decrypt a block of bits, usually 64 bits or 128 bits. DES

Confidentiality

Public Key Cryptography (1975)

1. The key used for decryption is not the same as the key used for encryption, $Ke \neq Kd$.

2. The decryption key can NOT be calculated from the encryption key in a *reasonable* amount of time.

3. The encryption key can be made public.
   encryption key $\equiv$ public key
   decryption key $\equiv$ private key
   It is still true that $D_{Kd}(E_{Ke}(P)) = P$.

4. Key distribution is easier.

5. Encryption is usually slower.

Notions of Security

Computational security - can not be broken with current or future computational resources. BIG CLAIM.

Typically can not prove computationally secure but show *provably secure*. A cryptosystem is provably secure if it is proved that breaking it is as hard as solving another problem which is believed to be hard. Typically a problem in NP (class of problems solvable in nondeterministic polynomial time).

Example: Factoring large numbers is in NP and is used as the base of some public key systems.

Unconditional security (Shannon) - can not be broken given infinite resource. Example: One-time Pad.

A "strong" cryptosystem may still be cracked if poorly used!

Notions of Security

How long should it take to decrypt a message if all possible keys are checked (*brute force attack*)?

How much time is too much time?

2^{14} **years**	**the next Ice Age**
2^{30} **years**	**age of the planet**
2^{170}	**atoms in the planet**
2^{190}	**atoms in the sun**

Most cryptosystems use keys that are length 64 or 128. If it requires 2^{128} operations to break an algorithm and 1 million operations per second can be done on a computer and you use 1 million parallel processors then the time is greater then 10^{19}. But 2^{30} is only about 10^{9}!

Symmetric Key - Classical Cryptography

Simple Substitution

Let $\mathcal{P}$ be the plaintext alphabet; $\mathcal{K}$ be the key alphabet and $\mathcal{C}$ be the cipher text alphabet. One character in $\mathcal{P}$ is replaced via a fixed algorithm (and key) by another character in $\mathcal{C}$.

Example: Shift Cipher / Caesar Cipher

$\mathcal{P} = \mathcal{C} = \mathcal{K} = \{$ **A, B, ... Z** $\}$

Let letters be represented as numbers so

A is 0, B is 1, ..., Z is 25.

Let x be the numeric representation of a letter.

$E_K(x) = (x + K) \bmod 26$

$D_K(y) = (y - K) \bmod 26$

So we are shifting K letters to the right to encrypt and reversing the shift to decrypt.

How hard is it to break this cipher?

Symmetric Key - Classical Cryptography

Example: **Caesar Cipher**

Encrypt CRYPTO with $K = D$ **(so** $K = 3$**)**

$E_3(C) = (2 + 3) \bmod 26 = 5 = F$

$E_3(R) = (17 + 3) \bmod 26 = 20 = U$

$E_3(Y) = (24 + 3) \bmod 26$

$\quad = 27 \bmod 26$

$\quad = 1$ **which is** B

$E_3(P) = S$, $E(T) = W$, $E(O) = R$

The cipher text is FUBSWR.

Decrypt FUBSWR with $K = 3$

$D_3(F) = (5 - 3) \bmod 26 = 2 = C$

$D_3(U) = (20 - 3) \bmod 26 = 17 = R$

$D_3(Y) = (1 - 3) \bmod 26$

so we need a rule for negative numbers

$(1 - 3) \bmod 26 = 26 - 2 = 24$ **which is** Y

etc.

If $c < k$ **and** $D_K(c)$ **then** $D_K(c) = 26 - (k - c)$**.**

Symmetric Key - Classical Cryptography

Example: One-time Pad (1917)

The key is a large non-repeating random string. The sender and the receiver both have key pads. Keys are used only once.

If the message and the key are both binary strings then the ciphertext is the XOR of the key bits and the message bits.

$$
\begin{aligned}
P \quad &= HIDE \\
H \text{ is } 7,\ I \text{ is } 8,\ D &\text{ is } 3,\ E \text{ is } 4 \\
&= 7834 \\
&= 0111100000110100 \text{ in binary}
\end{aligned}
$$

$$
\begin{aligned}
KEY &= 1011011110010101 1101\ldots \\
P &= 0111100000110100 \\
C &= 1100111110100001
\end{aligned}
$$

To decrypt just XOR K and C.

Symmetric Key - Classical Cryptography

One-time Pad

If the key is generated randomly then the ciphertext will also be a random string! This means NO information about the plaintext can be retrieved from the ciphertext! All possible plaintext messages are equally likely.

This is the ONLY cryptosystem that is commonly known to be *unconditionally secure*.

Drawbacks:
(1) distributing keys is hard
(2) keys are long
(3) on computers - synchronization

Uses:
Ultrasecure low-bandwidth channels. Maybe the Soviet Union/US hot line.

Symmetric Key - Modern Cryptography

The most commonly used symmetric system is DES (Data Encryption Standard) which encrypts 64 bit blocks of plaintext at a time.

A Little History on DES

- DES evolved from work done at IBM in the early 1970s (an earlier related algorithm is called Lucifer). At the request of what is now the NIST (National Institute of Standards), the NSA evaluated/modified the algorithm developed by IBM. This is DES.

  The NSA (National Security Agency) is the official security agency of the US. They cryptanalyze world wide communications and develop cryptosystems.

Symmetric Key - Modern Cryptography

More History on DES

- NIST published DES in 1975 (20 years ago)!! NIST made DES the federal standard for all unclassified government communications (1977).

- ANSI (American National Standards Institute) approved DES as the private sector standard (1981). EVERYONE GOT ON BOARD.

- DES is reviewed every 5 years. It is generally agreed that DES has exceeded it's useful lifetime. The problem is that there has been no good alternative cipher proposed that is agreeable to both the NSA and the public. A recent cipher proposed by the NSA is the Skipjack algorithm which is used in the Clipper Chip. Skipjack is no longer in consideration.

Symmetric Key - Modern Cryptography

Cracking DES

To prove the insecurity of DES, EFF built the first unclassified hardware for cracking messages encrypted with it. In July 1998 the EFF DES Cracker, which was built for less than $250,000, easily won RSA Laboratory's "DES Challenge II" contest and a $10,000 cash prize. It took the machine less than 3 days to complete the challenge, shattering the previous record of 39 days set by a massive network of tens of thousands of computers.

More information is available online at:

http://www.eff.org/descracker/

Symmetric Key - Modern Cryptography

A DES Replacement

- **A process to develop a Federal Information Processing Standard (FIPS) for Advanced Encryption Standard (AES) specifying an Advanced Encryption Algorithm (AEA) has been initiated by NIST. It is intended that the AES will specify an unclassified, publicly-disclosed encryption algorithm available royalty-free worldwide, that is capable of protecting sensitive government information well into the next century. It is hoped that this standard will be as widely accepted as the Data Encryption Standard (DES) in the private and public sectors.**

- **Advanced Encryption Standard (AES) Development Effort:**

  http://csrc.nist.gov/encryption/aes/aes_home.htm

- **The winning algorithm will be announced August 2000.**

Symmetric Key - Modern Cryptography

AES Candidate Algorithms

CAST-256 (Entrust Technologies, Inc.),
CRYPTON (Future Systems, Inc.),
DEAL (Richard Outerbridge, Lars Knudsen),
DFC (CNRS Centre National pour la Recherche Scientifique),
E2 (NTT Nippon Telegraph and Telephone Corp.),
FROG (TecApro Internacional S.A),
HPC (Rich Schroeppel),
LOKI97 (Lawrie Brown, Josef Pieprzyk, Jennifer Seberry),
MAGENTA (Deutsche Telekom AG),
MARS (IBM),
RC6 (RSA Inc.),
RIJNDAEL (Joan Daemen, Vincent Rijmen),
SAFER (Cylink Corporation),
SERPENT (Ross Anderson, Eli Biham, Lars Knudsen),
TWOFISH (Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson)

Triple DES or EDE

* Two keys are used $K1$ and $K2$. EDE is $E_{K1}(D_{K2}(E_{K1}(P))) = C$ and $D_{K1}(E_{K2}(D_{K1}(C))) = P$.
* Current proposed secure use of DES is as Triple DES. Advantage: no new crypto system is needed to upgrade.
* Only a brute force attack, which requires finding both secret keys or searching over 2^{112} possible strings, is know against EDE.
* EDE is often used in CBC mode to get a chained encrypted message. CBC mode is used on the outside to preserve it's error correcting properties - *self-synchronizing*.
* Applications for DES
  Banking transactions,
  PINs - transactions over the ATM machines,
  Department of Energy, Dept of Justice, Federal Reserve System

Symmetric Key - Modern Cryptography

DES (and many of the AES candidates) is a combination of substitutions and permutations.

- A *substitution* specifies for each of the 2^k possible values of the input, the k output bits. This is assuming that k bits are encrypted at a time and that a binary alphabet is used.

  In the Ceaser cipher the size of the alphabet is 26 and one character was encrypted at a time so there are 26^1 substitutions to specify.

  DES encrypts 64 bit blocks of binary data so just considering substitution gives 2^{64} substitutions - too many to check.

- A *permutation* specifies, for each of the k input bits, the output position to which it goes. Also called *transposition*.

Symmetric Key - Modern Cryptography

Overview of DES

* Symmetric block cipher which encrypts 64 bit blocks to produce 64 bits of ciphertext.
* The key length is 56 bits (that's funny). The key is used to generate sixteen 48-bit per-round keys one key for each round.
* Encrypt and decrypt use the same algorithm (with per-round keys reversed). So it is no harder to decrypt then it is to encrypt.
* The security rests with the secret key.
* DES uses substitution and permutation of the text combined with the key. One set of substitutions and permutations is called a *round*. DES has 16 rounds that are repeated on 1 block.
* DES is easy/fast to implement.

Symmetric Key - Modern Cryptography

Overview of the DES Algorithm

Input: 64 bit plaintext, x; 64 bit key, k.

1. **Perform an** *initial permutation*, $IP(x) = x_0$. **Then x_0 is divided into two halves, 32 bit left half L_0 and 32 bit right half R_0.**

2. **There is an encryption function that is repeated 16 times. Each time that it is repeated the left 32 bits are XORed with the output of a function $f(R_0, K_1)$ (inputs are the right 32 bits and a key) and become the right 32 bits of the output. The right 32 bits from the start of the round become the left 32 bits of the output. So** $round(L_0R_0) = R_0(L_0 \oplus f(R_0, K_1)) = L_1R_1$. **The key, K_1 is a key generated from the secret key k. K_1 is 48 bits long.**

3. **Perform an** *inverse permutation*, $IP^{-1}(R_{16}L_{16}) = y$. **This is the inverse of the permutation in step 1. y is the ciphertext.**

Initial and Final Permutations (IP and IP^{-1})

These permutations do not increase security! Anyone can reverse the permutations since they do not depend on the secret key.

The table means that bit 58 of the input goes to position 1 in the output, bit 50 of the input goes to position 2 in the output, bit 42 of the input goes to position 3 in the output, etc.

Initial Permutation (IP)

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

The inverse permutation IP^{-1} is done as the very last step of DES. It reverses the effect of this permutation.

Example Inverse Permutation

The following gives an example of a much smaller permutation (over strings length 9) and the inverse permutation.

The string abcdefghi permutes to fighdecba.
The string fighdecba inverts to abcdefghi.

Permutation:	Inverse:
9 8 7	6 9 7
5 6 1	8 4 5
3 4 2	3 2 1

Generating Per-round Keys

The key is given as a 64 bit quantity but every 8^{th} bit is not used and can be a parity check on the other 7 bits. So the real length of the key is 56 bits.

First the key is divided into two 28 bit halves C_0 and D_0 with a fixed permutation PC-1.

C_0	D_0
57 49 41 33 25 17 09	63 55 47 39 31 23 15
01 58 50 42 34 26 18	07 62 54 46 38 30 22
etc.	etc.

(No bit 8, 16 , 24, ..., 64)

This permutation and the final permutation on the key also have no security value.

So the secret key is now in two pieces C_0 and D_0. Each piece is now left shifted.

Generating Per-round Keys

Let $C_i = LeftShift(C_{i-1})$, $D_i = LeftShift(D_{i-1})$, The shifting is done 16 times . There are 16 rounds of DES to encrypt one 64 bit block. Each round uses a different per-round key. Each per-round key is gotten after a shift of C_i and D_i. The shift is not the same every time.

Left Shift Table

Round Number:	1	2	3	4	5	6	7	8
Amount Shifted:	1	1	2	2	2	2	2	2

Round Number:	9	10	11	12	13	14	15	16
Amount Shifted:	1	2	2	2	2	2	2	1

After a left shift C_i and D_i are permuted to create one per-round key of length 48 bits. The 2 pieces (before permutation) are also used to make the next per-round key.

Generating Per-round Keys

Note: $|C_i D_i| = 56$ **but** $|K_i| = 48$ **so some bits are not used. No bit 9, 18, 22 and some more. There is a second permutation** PC**-2 which reduces the size of the key. This is called a** *compression permutation* **(on page 76).**

Because of the shifting, a different subset of key bits is used in each per-round key. Each bit of the secret key K **is used in about 14 of the per-round keys.**

Pages 76-78 list which bit from the initial key k **are used in each per-round key.**

Some initial secret keys are called *weak keys* **because they do not produce good per-round keys. Can you give 3 weak keys?**

Size of the keyspace is 2^{56}**.**

Mangler Function $f(R_i, K_{i+1}))$ - The Guts

The input to this function is the right side R_i, 32 bits, and one per-round key, 48 bits.

First R is expanded to 48 bits via an expansion permutation of R that uses some bits more then once. By allowing one bit to affect two substitutions, the dependency of the output bits on the input bits spreads faster. This is called an *avalanche effect*.

For every 4 bits the first and fourth bits are used twice. Input bit 3 becomes output bit 4.

Expansion Permutation:
32 01 02 03 04 05 04 05 06 07 08 09
08 09 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 01

Next the per-round subkey is XORed with the result of the expansion permutation. This string is divided into 8 pieces each 6 bits long.

S Boxes

Each of the 6 bit length strings is input to a S-box (Substitution box). There are 8 S-boxes and $48/6 = 8$ strings of 6 bits. Each S-box gets a 6 bit input and gives a 4 bit output. These are compression substitutions. There are exactly 4 different 6 bit inputs that map to each 4 bit output.

S Box 1

14	04	13	01	02	15	11	08	03	19	06	12	05	09	0	07
00	15	07	04	14	02	13	01	10	06	12	11	09	05	3	08
04	01	14	08	13	06	02	11	15	12	09	07	03	10	5	00
15	12	08	02	04	09	01	07	05	11	03	14	10	00	6	13

Each entry in the S-box is read as 4 bits, for example 1 is 0001.
Rows and column are numbered starting at 0.

S Boxes

The 6 input bits specify which row and column gives the output.

Idea:
Say that the input is $b_1b_2b_3b_4b_5b_6$.
Concatenate b_1b_6 to get a 2 bit number. This is the row number.
Concatenate $b_2b_3b_4b_5$ to get a 4 bit number. This is the column number.

Example: Given input 110010 pick row 10 which is the second row and column 1001 which is column 9. The output is 12 (1100 in binary).

Guts continued

The 4 bits output form each of the 8 eight S-boxes are concatenated together to make a 32 bit string. This string then gets one more permutation (permutation P page 75). This helps security because the output bits from one S-box of the current round will then affect the output of a different S-box on the next round (remember 16 rounds). This permutation gets 32 bits as input and gives 32 bits as output.

The S-boxes are nonlinear and give DES it's cryptographic strength.

The 16 rounds are used as a product cipher. What cipher have we seen that would not be any more secure if it was repeated a number of times then if it is done once?

Security

* S-boxes give DES it's security but S-boxes are mysterious! Swapping the order of S-boxes has been shown to make DES less secure.

* Properties designed into S-boxes:
  (0) Each row of an S-box permutes the integers $1 \ldots 15$.
  (1) No S-box is a linear or affine function.
  (2) Each input bit to an S-box affects 2 output bits of the S-box.
  (3) There are on average as many 1's as 0's in the output of an S-box, assuming a random key.

* There are some weak keys (all 1's; all 0's; 1/2 1's and 1/2 0's) but the list is small, only 16 such keys where C and D are all 1's; all 0's or alternating 1's and 0's. The probability of generating one of these is 16 in 2^{56}.

Public Key Cryptography

Diffie-Hellman Key Agreement Protocol 1976

Diffie-Hellman protocol predates RSA. Not a cryptosystem but a key exchange protocol. Two people agree on a secret key by the exchange of public messages. Cleaver!

A and B publicly pick 2 large integers p, g, $g < p$.

1 **A chooses random number x and computes**

$$X = g^x \bmod p.$$

2 **B chooses random number y and computes**

$$Y = g^y \bmod p.$$

3 **A sends X to B.**
B sends Y to A (x and y are secret).

4 **A computes $K = Y^x \bmod p$.**

5 **B computes $K' = X^y \bmod p$.**

Claim: $K = K' = g^{xy} \bmod p$. **K is the secret key.**

Proof: $K = Y^x \bmod p$

$$= (g^y \bmod p)^x \bmod p$$

$$= g^{yx} \bmod p.$$ **Same idea works for K'.**

What about authentication?

Public Key Cryptography

Diffie-Hellman Key Agreement

Not just any g and p will work.

* **p needs to be a prime number and $(p-1)/2$ should also be prime. This type of prime is called a *strong prime*.**

* **g should be a primitive element of p. This means that $g^x \bmod p$ will cycle over all of the numbers from 0 to $p-1$.**

* **p, x and y should be large, at least 512 bits.**

* **The security is based on the notion that computing *discrete logarithms* in a finite field is hard. So computing $\log_g X \bmod p$ to find x is assumed to be hard. But computing modular exponentiation is easy.**

* **$f(x) = g^x \bmod n$ is a CANDIDATE for a one-way function.**

Public Key Cryptography

Attack Against Diffie-Hellman

A and B publicly pick 2 large integers p, g, $g < p$.

1	**A chooses x and computes $X = g^x \bmod p$.**
2	**B chooses y and computes $Y = g^y \bmod p$.**
3	**A sends X to B.**
	M intercepts X.
	M chooses z and computes $Z = g^z \bmod p$.
	M sends Z to B.
	B sends Y to A.
	M intercepts Y.
	M sends Z to A.
4	**A computes $K = Z^x \bmod p$ (g^{zx}).**
5	**B computes $K' = Z^y \bmod p$ (g^{zy}).**
	M computes $K = Y^z \bmod p$ (g^{xz}).
	M computes $K' = X^z \bmod p$ (g^{xz}).

Now if M intercepts ALL of the communications between A and B he can read everything without A and B even knowing. This occurs because there is no authentication in Diffie-Hellman. This is called a *man-in-the-middle attack*. To fix, authenticate messages.

Public Key Cryptography

Encryption using Public Key Cryptography

RSA (Rivest, Shamir, Adleman)

Key Generation:

Creating a public/private key pair.

1. Choose random prime numbers p and q of length 200 digits (more like 512 bits).
2. Compute $n = p \cdot q$.
3. Choose a random numbers e such that $gcd(e, (p-1)(q-1)) = 1$.
4. Compute $d = e^{-1} \bmod (p-1)(q-1)$. (Find d such that $e \cdot d \equiv 1 \bmod (p-1)(q-1)$).
5. Publish n and e. Keep d private. Throw out p, q.

The public and private keys are a function of 2 large primes. In order to break the system (find d given n and e) it would be good enough to be able to factor n and find the primes.

Public Key Cryptography

RSA - Confidentiality

Public key (e, n) is in a public directory.

Encryption

1 **Divide the message m into blocks such that each block represents a 200-digit number. m_i is a block.**

2 **Compute and send $c_i = m_i^e \bmod n$.**

Decryption

1 **Compute and send $m_i = c_i^d \bmod n$.**

If a is a constant then this works since
$c^d \equiv (m^e)^d \equiv m^{ed} \equiv m^{a(p-1)(q-1)}m \equiv 1 \cdot m \equiv m \bmod n$

Public Key Cryptography

RSA - Digital Signature

Public key (e, n) is in a public directory.

Sign a message:

1 Divide the message m into blocks such that the each block represents a 200-digit number. m_i is a block.

2 Compute $s_i = m_i^d \bmod n$ and send (s_i, m_i).

Verify a signature.:

Get the sender's public key from the directory.

1 Compute $n_i = s_i^e \bmod n$.

2 Check that $m_i = n_i$.

A common use of a digital signature is to sign the message digest (hash) of a message (128 bits). This is faster then signing the whole message. Signatures are not as fast as algorithms like MD5 and DES.

Integrity

Cryptographic Hash Functions

A *cryptographic hash function* **or** *message digest function* **is typically used to supply integrity; as a part of a scheme which supplies authentication or in a signature scheme. In a signature scheme a hash or digest of a message is signed.**

A *cryptographic hash function* **is a type of one-way function.**

A message digest function is *cryptographically secure* **if it is computationally infeasible to**
(1) find the message given a hash and
(2) find two messages with the same hash.

What is the problem with breaking a message up into blocks and signing each block?

Integrity

Uses of Message Digests

- *Fingerprint* a document. Keep a hash of a document so that changes can be detected.
- Authentication: Assume that A and B share a secret key K and messages digest $MD()$.
  A generates r_A (random) and sends it to B.
  B sends $MD(K|r_A)$ to A
  A computes $MD(K|r_A)$ and check that this is what B sent.
  A now believes that the communications is with B. This is also repeated in the reverse direction.
- Generating a MIC (Message Integrity Code): Two keyed message digests.
  Solution 1: A sends $MD(P|K), P$, to B.
  Solution 2: A sends half of the bits of $MD(P|K)$ plus P to B.
  Either way B can compute $MD(P|K)$ and check to see that P was not changed.

Integrity

MD5 - Rivset

MD5 is a commonly used cryptographic has algorithm.

In practice hash algorithms are used in a more complex way. This is an example of how MD5 is used in network security protocols (IPSEC).

$pad1 = 0x36$ **repeated 48 times.**
$pad2 = 0x5C$ **repeated 48 times.**

HMAC-MD5 is $MD5(K, pad2, MD5(K, pad1, text))$**.**

K **is a 16 byte random key string.**

The NIST proposed message digest function is SHA (Secure Hash Function) is based on MD4 but is considered to be more secure then both MD4 and MD5.

Digital Signature Schemes

A digital signature scheme is a method of electronically signing a message.

A signature scheme has 2 poly-time algorithms: one to sign messages (private - owned by signer), $sig(m) = s$
one to verify signatures (public),
$ver(m, s) =$ True or False.

- A message x is sent in the clear with the signature s. The goal is NOT confidentiality of the message but is to associate the message with the senders identity.

- The cryptographic service provided by a Signature is stronger then that provided by Authentication. A signed message can only be produced by one entity.

- Needed for Internet commerce.

Digital Signature Schemes

A signature scheme is $(\mathcal{P}, \mathcal{A}, \mathcal{K}, \mathcal{S}.\mathcal{V})$ so that

1. $\mathcal{P}$ **is a finite set of possible messages.**
2. $\mathcal{A}$ **is a finite set of possible signatures,** $sig(x) = s$.
3. $\mathcal{K}$ **is a finite set of possible keys,** $sig_K(x) = s$.
4. For each $K \in \mathcal{K}$ **there is a signing algorithm** $sig_K : \mathcal{P} \to \mathcal{A}$ **and a verification algorithm** $ver_K : \mathcal{P} \times \mathcal{A} \to (True, False)$ **and for every message** $x \in \mathcal{P}$ **and signature** $s \in \mathcal{A}$**:**

$$ver(x, s) = \begin{cases} True & \text{if } s = sig_K(x) \\ False & \text{if } s \neq sig_K(x) \end{cases}$$

Digital Signature Schemes

Signatures - Security

A signature is secure under the following conditions.

* Signing something does not give away the secret/private key.

* Nobody can generate a signature for a message unless they own the private key. It is computationally not possible to do.

* Nobody can generate a message that matches a given signature if then don't own the private key. Forgery is not possible.

* Nobody should be able to modify a signed message without wreaking the signature. Every bit of the message should be "covered" by the signature.

Digital Signature Schemes

RSA Digital Signature

Anyone can get the public key (e, n) from the public directory. In this case, (e, n) is used for the verification algorithm.

Sign a message:

1 Divide the message m into blocks such that the each block represents a 200-digit number. m_i is a block.

2 Compute $s_i = m_i^d \bmod n = sig_d(m_i)$ send (s_i, m_i).

Verify a signature.:

Get the sender's public key from the directory.

1 Compute $n_i = s_i^e \bmod n$.

2 Check that $m_i = n_i$.

The value of this comparison is $ver_e(s_i, m_i)$.

Digital Signature Schemes

Signing Encrypted Messages

A has public-key a_v for verifying signatures.

B has public-key b_e for encrypting messages.

Protocol

1 A signs message x, $s = sig_{a_s}(x)$
A encrypts (x, s) with B's public-key,
$y = E_{b_e}((x, s))$
A sends B y

2 B decrypts y as $(i, j) = D_{b_d}(y)$
B verifies A's signature using A's public-key,
$ver_{a_v}(i, j)$. If $ver_{a_v}(i, j) = True$ then
B knows that y MUST be from A.

What if A first encrypts and then signs?

A encrypts x with B's public-key, $y = E_{b_e}(x)$

A signs message y, $s = sig_{a_s}(y)$

A sends B (y, s)

Then M could intercept (y, s) and instead compute $z = sig_{m_s}(y)$ to sign y. Now M sends (y, z) to B. B may believe that M sent the message y. M can sign y even if he does not know the plaintext.

Timestamps

A timestamp provides proof that a message was signed at a certain time.

If the key used to sign messages is gotten then all messages before the key was known are okay. Idea: If your credit card is lost today, the credit card company assumes that all charges before today were made by you.

Protocol:
Let TSS be a trusted Timestamp Service.
B computes $z = h(x)$ and $y = sig_{K_B}(z)$ and sends (z, y) to TSS.
TSS signs (z, y, D), D is the date so $sig_{K_{TSS}}(z, y, D)$.
TSS can send this to B or can hold it.

This is good if the TSS is trusted.

Key Distribution

- *Key distribution* is a method that lets 1 party pick a secret key and then transmit it to 1 (or more) other users.
  Example: A Key Distribution Center (KDC).

- *Key agreement* is a protocol that lets 2 (or more) users establish a shared secret key via the communication of public values. The final key is determined by both users.
  Example: Diffie-Hellman protocol.

- *Trusted Authority* (TA) is a mutually trusted party that may be part of a protocol or key distribution scheme.
  Example: Certificate Authority or KDC.

Key Distribution

So far we have seen secret key (symmetric) ciphers and public key ciphers and we have fudged over the issue of how users got their keys (with the exception of Diffie-Hellman). Now we will look at methods for getting keys over a network.

* Let's divide keys into two kinds:
  (1) *Data Keys* or *Session Keys* which are used to encrypt/decrypt large amounts of data (say with DES) and
  (2) *Key-Encryption Keys* which are used to encrypt/decrypt other keys (such as data keys) for distribution .

* Data keys are used to encrypt lots of information and so they are changed often. Key-encryption keys are only used to encrypt and distribute other keys so they are changed less often.

Key Distribution

* The issue is how to distribute key-encryption keys. Possibilities: (1) manually (2) distributed - split and send over several different channels.

* Do NOT send a new key-encryption key over a network encrypted via the key that it is going to replace.

* Scalability
  In a network, for n users of a symmetric key system each user would need $n-1$ key-encryption keys and there would be a total of $n(n-1)/2$ key exchanges required for everyone to get data keys with every other user. Too much. Public-key systems are scalable when symmetric systems are not.

Key Distribution

Active Adversary

An active adversary can modify a protocol by adding, deleting or modifying messages.

Goals of an active adversary are:

- To fool users A and B into accepting an "invalid" key as valid. An invalid key might be an old key (that the adversary can decrypt) or a key picked by the adversary (basic man-in-the-middle against Diffie-Hellman).
- To make A and B believe that they have exchanged a key when they have not.

Key Distribution

KDCs

Solution: Set up one trusted node that shares a key-exchange key with each user of the system. This is called a *Key Distribution Center* or KDC. The KDC can then give out, on request, data keys which are encrypted with key-distribution keys. This model is used for symmetric crypto systems.

* Advantages:
  (1) when a new node is added only the KDC needs to get a new key for it;
  (2) users only need one key-distribution key.

* Disadvantages:
  (1) if the KDC is compromised the whole network is not secure;
  (2) if the KDC goes down then nobody can start a new secure session;
  (3) the KDC is a performance bottleneck.

Key Distribution

Kerberos

Kerberos is a secret key based service for providing authentication over a network (designed at MIT). It's used to give a user legal access to remote resources securely.

Assume that DES keys are shared between each user and a TA. DES will use CBC mode.

The first steps transmit a secret key.

The last steps provide confirmation. U and V are convinced that they share the same key. This is done by using the new session key K to encrypt a known quantity.

Timestamps & Lifetimes prevent an adversary from replaying messages - replay prevention. Kerberos needs synchronized clocks for times.

There is no formal proof that Kerberos is secure - this is common.

Key Distribution

Diffie-Hellman Predistribution of Keys

This system is computationally secure (depends on Discrete Log Problem).

- (α, p) are public, α is a primitive element for prime p.
- Each User:
  1. $ID(U)$ is a users ID for U.
  2. a_u is a secret exponent for U
  3. $b_u = \alpha^{a_u} \bmod p$ is a public value for U
- TA (Trusted Authority):
  1. private signing algorithm sig_{TA}
  2. public verification algorithm ver_{TA} - each user has this
  3. Assume data is hashed before signing so $sig_{TA}(ID(U), b_u)$ might be $sig_{TA}(MD5(ID(U), b_u))$
- Certificate for each user:
  $C(U) = (ID(U), b_u, sig_{TA}(ID(U), b_u))$.
  Certificates are stored in a public directory. TA doesn't know a_u.

Key Distribution

Diffie-Hellman Predistribution of Keys

If U wants to send V a message m, U needs to get V's certificate from the public directory, compute $K_{U,V} = (b_v)^{a_u} \bmod p$,
encrypt $E_{K_{U,V}}(m) = c$,
send V either (c, U) or $(c, C(U))$.
V can use U's certificate to compute
$K_{U,V} = (b_u)^{a_v} \bmod p$ and
decrypt by $D_{K_{U,V}}(c) = m$.

If U sends $(c, C(U))$ then V does not have to get U's certificate.

Typically, certificates also need to include information like, an expiration time, the algorithms to use for the values like b_u, etc.

This system is predistribution because there is no need for an exchange between U and V before encryption can be done.

Key Distribution

Security

Diffie-Hellman Problem

Problem: Given $I = (p, \alpha, \beta, \gamma)$

p **is a prime**

α **is a primitive element for** p

$\beta, \gamma \in Z_p$

Objective:

Compute $\beta^{\log_\alpha \gamma} \bmod p = \gamma^{\log_\alpha \beta} \bmod p$

$\log_\alpha \gamma$ **and** $\log_\alpha \beta$ **are the secret exponents.**

Fact:

Diffie-Hellman Key Predistribution is secure against passive attach iff the Diffie-Hellman Problem is intractible.

Key Distribution

Certificate Authorities (CAs)

* **In a symmetric system a Key Distribution Center, KDC, is used to help pass session keys but in a public key system the trusted node is called a** *Certification Authority* **(CA). A CA generates** *certificates* **which are signed messages that contain a name and a corresponding public key, example: $E_{KCAd}(A, KAe)$, where $KCAd$ is the private (signing) key of CA and KAe is A's public (encoding) key.**

* **So a signed message, $E_{KCAd}(M)$, from the CA can be read by anyone using CA's public key, $KCAe$, and since every other user of the system trusts CA all users will believe that the message is correct.**

* **All nodes must have the CA'c public key in advance. So this is one key that must be passed securely off line.**

Key Distribution

CAs

* T is going to be the CA. His private key, used for signing messages is $KCAd$. His public key, $KCAe$, which everyone has, can be used to verify his signature and read any message that T signed.

* The users are A, B and C. Each user has a public, private key pair $\langle KAe, KAd \rangle$ where T keeps the public key KAe for each user. In these protocols the private key is general being used to sign a message (which gives authentication) rather then to decrypt a message. The public key is used to verify a signature not to encrypt.

* Any signed message sent over a network can be read and verified by anyone who can get the public key of the person who signed the message.

Key Distribution

CAs

* Advantages:
  (1) when a new node is added, only the CA needs to get a new key.

* Advantages over Key Distribution Center:
  (1) the CA does not need to be online, so it is less prone to attack over the network;
  (2) it can be simpler then a KDC;
  (3) if the CA goes down then only new users can not be added;
  (4) certificates, if used correctly, are less security-sensitive (they can be deleted but not modified or replaced);
  (5) A dishonest CA can not decrypt conversations (it can impersonate another user but can not decrypt messages between two legal users).

* Disadvantages:
  (1) if the CA is compromised, the whole network is not secure.

Key Distribution

CAs

Basic key exchange protocol with public-key cryptography.

1 A sends T a request for B's public key A, B.
2 T sends A $D_{KTd}(KBe, B)$ the key signed.
3 A decrypts $E_{KTe}(D_{KTd}(KBe, B))$.
A generates K a session key.
A sends B, $E_{KBe}(K)$.
4 B decrypts $D_{KBd}(E_{KBe}(K)) = K$.

A and B then can use K to encrypt a session.

The certificate authority, T, is signing (using a private key) the public key for B in step (2). Because A trusts T's signature then A believes that this is B's public key. In step (3) A can decrypt or verify that B's key was signed by T because every user has T's public key.

Key Distribution

CAs

Denning-Sacco Protocol

T keeps a database of everyones public key.

t_A **is a timestamp.**

1 **A sends T a request for B's public key** A, B.

2 **T sends A** $D_{KTd}(KBe, B)$ **the key signed.**

3 **A decrypts** $E_{KTe}(D_{KTd}(KBe, B))$.

A generates K **a session key.**

A sends B, $(E_{KBe}(D_{KAd}(K, t_A)), D_{KTd}(KBe, B), D_{KTd}(KAe, A))$.

4 **B decrypts** $D_{KBd}(E_{KBe}(D_{KAd}(K, t_A)))$

B decrypts $E_{KTe}(D_{KTd}(KBe, B))$

B decrypts $E_{KTe}(D_{KTd}(KAe, A))$.

B decrypts $E_{KAe}(D_{KAd}(K, t_A))$

B verifies A's signature.

B checks for a valid timestamp t_A.

A and B then can use K **to encrypt a session. Problem! The Denning-Sacco protocol has a problem because after completing the protocol B can now masquerade as A. Many protocols fail because the designer tries to optimize.**

Key Distribution

CAs Now B can replay the old message because he has A's signature.

Denning-Sacco Protocol

1 B sends T a request for C's public key B, C.

2 T sends B $D_{KTd}(KCe, C)$ the key signed.
Assume B has $D_{KTd}(KBe, B)$, his key signed.

3 B decrypts $E_{KTe}(D_{KTd}(KCe, C))$.
B uses the signed session key from A.
B sends C, $(E_{KCe}(D_{KAd}(K, t_A)), D_{KTd}(KAe, A),$
$D_{KTd}(KCe, C))$.

4 C decrypts $D_{KCd}(E_{KCe}(D_{KAd}(K, t_A)))$
C decrypts $E_{KTe}(D_{KTd}(KAe, A))$
C decrypts $E_{KTe}(D_{KTd}(KCe, C))$.
C decrypts $E_{KAe}(D_{KAd}(K, t_A))$
C verifies A's signature.
C checks for a valid timestamp t_A.

B has fooded C into thinking that she is talking with A. This works until the timestamp expires.
To fix this in step (3) send
$E_{KBe}(D_{KAd}(A, B, K, t_A)), D_{KTd}(KBe, B), D_{KTd}(KAe, A)$.

Key Distribution

Certificate Revocation

Suppose that there is a reason for a user to loose his right to encrypt/decrypt in a system.

If the system uses a Key Distribution Center then revoking the users access to shared keys is easy - just don't issue any.

If the system uses a CA then revoking a certificate is harder. It is standard to put an expiration date on any certificate but this could be a year.

Solution is similar to that used with credit cards. Publish a book of revoked certificates.

Key Distribution

Certificate Revocation

X.509 (ISO88) is a standard for security services within X.500 directory services frameworks. The X.509 encoding of public key certificates is widely used, other protocols of X.509 are not.

X.509 uses a list called CRL which has serial numbers of certificates that are revoked. A new CRL is posted periodically and lists revoked, unexpired certificates. Certificates include: an expire time, user ID, user public key, serial number and a CA signature over the entire certificate. The CRL has an issue time and is signed by the CA.

A certificate is valid if has a legal CA's signature and has not expired and is not on the revoked list on the most recent CRL.

Key Distribution

Session Key Establishment

Usually you want to use one key to encrypt data or session keys and another key to encrypt the data in a session. The key which is used to encrypt the session keys is often also used to establish authentication.

* Keys wear out. The more encrypted or signed data that goes over the network the higher the odds of breaking the cipher (finding the key). Since it is hard to give out authentication/ key exchange keys then don't use them to encrypt lots of data. The "bootstrap" is hard to do online.

* Changing session keys frequently prevents adding, deleting and reordering old messages (block replay). Packet transmission might not include dated information that would prevent replay of a packet.

IP Layer Security

* Two new security headers are proposed (IETF working group IPSEC) for use on Internet packets. These headers go before IP headers and after TCP/UDP headers.

* Authentication Header , AH, supplies authentication and integrity over a datagram. Authentication is of the IP address, and a "trust relationship".

* Encapsulation Security Payload header, ESP, supplies confidentiality and optionally, integrity and authentication.

* A Security Association , SA, must exist between the users. A SA is a negotiated set of algorithms, keys and usage modes that will be used to process a header. SAs define trust relationships and relate to a given network connection or set of connections.. The tricky part is negotiating keys.

IP Layer Security

Security Associations

* An IP Destination Address and SPI value identify a SA uniquely.

* A SA typically includes:
  (1) Authentication algorithms and modes.
  (2) Keys for use with the algorithms.
  (3) Encryption algorithms and modes of use.
  (4) Keys for use with the encryption algorithm.
  (5) The size of a cryptographic synchronization IV if required by an algorithm.
  (6) Lifetime of keys.
  (7) Lifetime of the SA.
  (8) Source Address of the SA.
  (9) Sensitivity level (classified, unclassified, etc.).

* An SA is shared by two or more hosts and/or gateways.

IP Layer Security

AH Header - Authentication

Provides connectionless integrity and data origin authentication for IP datagrams. Also replay protection.

AH Header Format

Next Header (8) | Payload Len (8) | Reserved (16)

Security Parameter Index or SPI (32)

Sequence Number Field (32)

Authentication Data (variable number of 32-bit words)

Transport Mode for IPv4

IP header | AH | TCP header | Data

$\longleftarrow$ **authentication** $\longrightarrow$

The SPI is a random value used, with the destination IP address, to index a SA. The SA contains the keys, algorithms, etc. needed to process the header.

IP Layer Security

AH Header - Authentication

The Authenticated Data contains the integrity check value. All fields of the IP header which are not modified during transit are authenticated; i.e, covered by the integrity check.

AH Header Format

Next Header (8) | Payload Len (8) | Reserved (16)

Security Parameter Index or SPI (32)

Sequence Number Field (32)

Authentication Data (variable number of 32-bit words)

IP Layer Security

ESP Header - Confidentiality

Provides confidentiality, connectionless integrity, data origin authentication, anti-replay services and limited flow control confidentiality. The services depend on the SA and tunnel or transport modes.

ESP Header Format

Security Parameter Index or SPI (32)

Sequence Number Field (32)

Payload data (variable length)

Padding (0-255 bytes) opt.

Pad Length | Next Header (8)

Authentication Data (variable - max 96 bits) opt.

Transport Mode for IPv4

IP header | ESP | TCP | Data | ESP trailer | ESP Auth.

$\longleftarrow$ **encryption** $\longrightarrow$

$\longleftarrow$ **authentication** $\longrightarrow$

Encryption is performed before authentication on outbound packets.
ESP assumes a symmetric encryption system.

IP Layer Security

ESP Header - Confidentiality

For ESP Transport Mode:

The authenticated data includes all of the ESP fields except the Authentication Data field. (Not the IP header.)

Encryption covers the Payload Data, Padding, Pad Length and Next Header fields.

ESP Header Format

Tunnel Mode for IPv4

NEW IP | ESP | Old IP | TCP | Data | ESP trailer | ESP Auth.

$\longleftarrow$ encryption $\longrightarrow$

$\longleftarrow$ authentication $\longrightarrow$

The “inner” IP packet carries the ultimate source and destination addresses. The “outer” IP header may contain different IP addresses.

Encryption is now over the Old IP header, TCP, Data and ESP trailer fields. Authentication is over all of these plus the ESP field.

IP Layer Security

Key Negotiation

* ISAKMP, Internet Security Association and Key Management Protocols, is a framework for security association management and cryptographic key management protocols.

* ISAKMP assumes that all users have access to public keys. BIG assumption. At the least there needs to be an Internet-standard scalable key management protocol.

* It is assumed that key management data is carried by higher layer protocols like TCP/UDP.

* Host-oriented vra. user-oriented keying.

* A few public-key options:
  (1) DNS - signed public keys added to the Domain Name Server.
  (2) DNSSEC - Secure Domain Name Servers.
  (3) Certificate Infrastructure.

Chaffing and Winnowing

From the paper *Chaffing and Winnowing: Confidentiality without Encryption*, by R. L. Rivest (MIT)

- Winnowing can be used to develop a new confidentiality scheme that does not fall under the same definition as cryptography.

- No encryption so no decryption key needed. Circumvents export laws on cryptography. The regulation of cryptography does not cover regulation of steganography and winnowing.

- To winnow is to “separate out or eliminate (the poor or useless parts)”.

Chaffing and Winnowing

Chaffing Scheme

Two Parts to sending a message:

- Authentication
  (add a MAC - for example HMAC or SHA1).

- Adding chaffing.

Note: Authentication algorithms are all approved for export even if they are keyed with a shared secret key. Authenticated Diffie-Hellman can be used by two users to get a shared secret key.

Chaffing and Winnowing

Chaffing - Sender

- Break the message into packets.

- Authenticate each packet using a secret authentication key.
  Compute $h(K, Packet) = MAC$
  to get $Packet, MAC$).
  The packet is still "in the clear".

- In general, a packet will also carry a unique sequence number which is used to delete duplicate packets, detect missing packets, reorder received packets. The MAC for a packet is computed as a functions of the packet sequence number also.

$$h((number, data), K) = MAC$$

- Add chaff to the packet stream.

Chaffing and Winnowing

Chaffing - Sender

- **Add chaff to the packet stream. Add fake packets with bogus MACs.**

- **Construct a series of extra, reasonable packets and sequence numbers. The sequence numbers can duplicate already existing sequence numbers.**

- **Add fake MACs to the extra packets. Include these packets in the packet stream.**

$h((1, data), K) = MAC_1$ **send** $((1, data), MAC_1)$

send $((1, fake-data), Random-String_1)$

$h((2, data), K) = MAC_2$ **send** $((2, data), MAC_2)$

send $((2, fake-data), Random-String_2)$

Chaffing and Winnowing

Winnowing - Receiver

- Compute the MAC of every packet that is received using the agreeded on secret key.

- Discard any packet for which the MAC is incorrectly computed, the “chaff”’.

- Order the remaining packets - these are the “wheat”.

- Note: this is the typical behavior on receiving a packet with an incorrect MAC.

- An adversary must determine what is chaff and what is wheat which might be independent of finding the shared secret key.

Chaffing and Winnowing

How to make the Adversary's job hard

- Break the message into packets containing 1 bit each.

- Authenticate each packet using a secret authentication key.
  Compute a MAC over the data and packet sequence number.

- Add chaff to the packet stream.
  If a packet with sequence number s contains a 0 as the data then create a new packet with the same sequence number but with a 1 as the data.

- Add a fake MAC to this packet. Include these packets in the packet stream.

Chaffing and Winnowing

Claim: This scheme provides confidentiality without using encryption - the message is in the clear.

Claim: Used with 1 bit messages, separating the wheat from the chaff requires either breaking the MAC algorithm or knowing the secret authentication key.