Overview

Why is Computer Security Important?

- More and more valuable information is held and processed inside computers connected through private and public networks
  - Health records
  - Tax statements
  - Bank accounts
  - Electronic tickets
  - ...
- This information must be protected to ensure that
  - only suitably authorized people can access it
  - the right information is processed
  - it is not created, changed, or deleted without trace
  - it is not revealed in an uncontrolled manner
  - that it is accessible when needed

Some Ways to Cheat

- Gain unauthorized access to information (i.e., violate secrecy or privacy)
- Impersonate another user to either shift responsibility or use the other's privileges
- Disavow responsibility or liability for information
- Claim to have receive some information from somebody that the cheater created
- Claim to have sent information at some time that was either never sent or at another time
- Enlarge the privileges to access more information than authorized
- Modify the privileges of others
- Conceal the presence of information in other information
- Insert oneself actively as *man in the middle* in others' conversation
- Learn who accesses what information when
- Pervert the function of software by adding covert functions
- Cause others to violate a protocol by introducing incorrect information
- Undermine confidence in a protocol by causing apparent system failures
- Prevent legitimate communication by other users

Overview

What is Security?

If I hide my money under the mattress that's **obscurity** but **not security**.

If I put my money into a safe, give you the safe, a **description of its design**, and 100 identical safes with their combinations so you and the worlds most skilled safe crackers can **study** the locking mechanism - and you still can't get to the money - then that's security.

What Will We Cover in this Course?

- Philosophies and Services
- Mechanisms
  - Protocols
  - Cryptographic techniques
  - Protocol analysis
- Applications

Overview

Format of Class

- 6 times, Fridays 10:15 - 14:00 over the course of the summer semester
- dates may change, however this will be announced via my WWW index page
  http://www.zurich.ibm.com/~kai
- ***Schein*** through brief presentation (~15 min) prepared in a group of 2-3 students in the last class
- Topics subject to my approval

Literature (to be found, e.g., at www.amazon.com)

- William Stallings, Cryptography and Network Security: Principles and Practice, Prentice Hall; ISBN: 0138690170
- Bruce Schneier, Applied Cryptography: Protocols, Algorithms, and Source Code in C, 2nd Edition, John Wiley & Sons; ISBN: 0471117099

Real title: *Department of Defense Trusted Computer System Evaluation Criteria.* It is published by the National Computer Security Center (NCSC) a branch of the US National Security Agency (NSA)

Objectives

Three basic objectives for computer security are described there

(1) Policy

- **Security policy** - identifies protection requirements in terms of perceived risks, threats, and the goals of the organization
- **Marking** - systems should mark information with appropriate classification as basis on which to determine accessibility

(2) Accountability

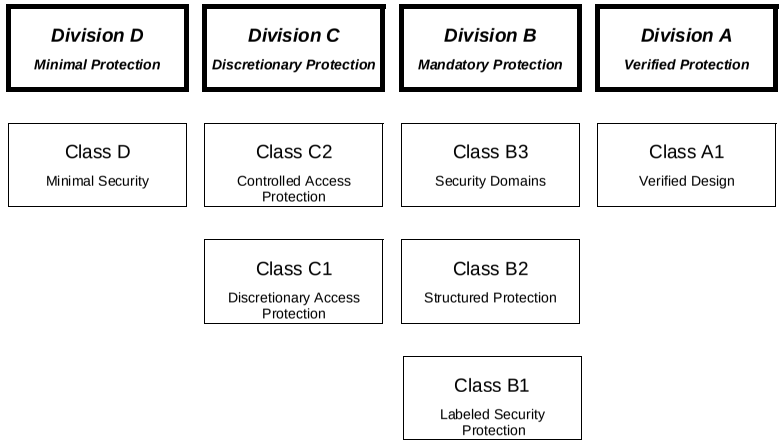
- **Identification** - identify individual users, demand authentication of this identity
- **Auditability** - provide dependable audit capabilities at sufficient granularity to trace auditable events to a specific individual

Objectives (cont.)

(3) Assurance

- Provide guarantees and confidence concerning the implemented security policy
- Rely on the trusted portions of the system to work as intended (Trusted Computing Base)
- Continuous protection with periodic checking that the security policy is "uncircumventably enforced" during system operation

Computer System Evaluation Criteria



Access Control Mechanisms

- **Discretionary Access Control (DAC)**
  Owner of an object may specify at his discretion who may access or manipulate it in what way.
- **Mandatory Access Control (MAC)**
  Objects bear hierarchical sensitivity labels (e.g., internal use only, confidential, top secret) and only users with sufficient access privileges (clearance levels) may access these objects.

Summary

- Relevant for the evaluation of computer security
- Does **not** explicitly address network security services for distributed systems
- Very few commercial operating systems are classified as high as C2
- Even fewer operating systems have been classified as B*x*

Access Control	Discretionary
	Mandatory
Authentication	Origin
	Peer
Confidentiality	Data
	Traffic
Integrity	
Non-Repudiation	Origin
	Destination/Receipt
Availability	
Accountability	

- Address security needs of **distributed systems**
- Specify **network security services**
- Services counter **security attacks**
- Services make use of one or more **security mechanisms** (cryptographic algorithms)

Access Control

- Control access of resources to legitimate parties
- Protect *host* and *network* resources

Authentication

- Verify the claimed identity of agent and use it to allow or deny access
- *Origin* authentication identifies the identity of a sender or originator of a message
- *Peer* authentication allows two or more communicating parties to mutually authenticate each other

Class 0	Direct Disclosure
Class 1	One-Way Functions
Class 2	One-Way Functions with Verifier Identification
Class 3	Cryptographic Functions
Class 4	Cryptographic Functions with Verifier Identification

Confidentiality

- Prevent information disclosure to unauthorized parties
- Ensure *data* confidentiality and *traffic* confidentiality
- Cryptography is key to protection

Integrity

- Ensure that information is *not modified, inserted, reordered, complete, not duplicated*, and *not replayed*
- Distinguish between service with detection only and with recovery

Nonrepudiation

- prevent sender from denying he has sent a message
- prevent receiver from denying he has received a message

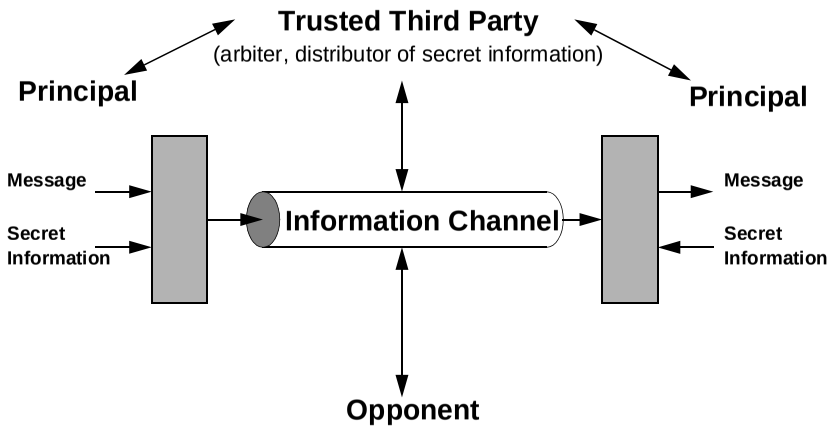
Availability

- Prevent disruption of network or computer services
- Prevent any activities which can compromise the system availability

Accountability

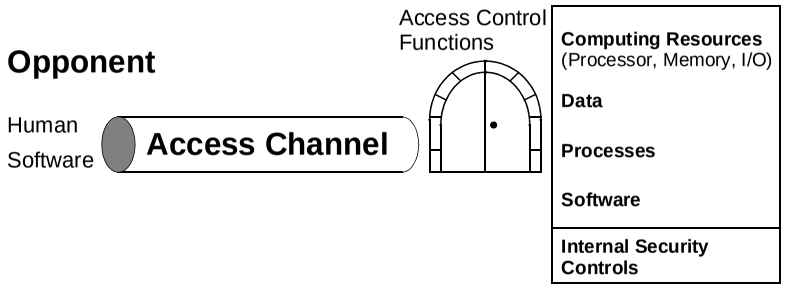
- Hold users accountable for their actions
- Bill correct users for resource usage

Internetwork Security — *Model*



er1.fm: 02.08.2000 10:47

Access Security — *Model*

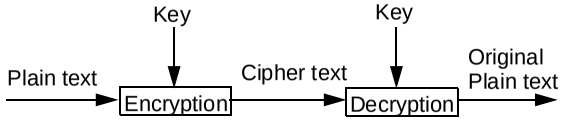


er1.fm: 02.08.2000 10:47

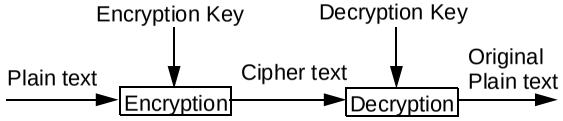
Terminology

Encryption Algorithms (Ciphers) and Keys

- Symmetric Algorithms $E_K(M) = C \qquad D_K(C) = M$



- Asymmetric or Public Key Algorithms $E_{K_1}(M) = C \qquad D_{K_2}(C) = M$



er1.fm: 02.08.2000 10:47

Terminology

Encryption Algorithms and Keys

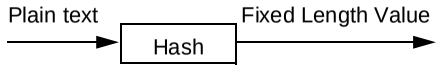
- Symmetric Algorithms
  - encryption and decryption keys are the same or
  - can be easily derived from each other
  - security depends on secrecy of key
- Asymmetric or Public Key Algorithms
  - different keys
  - cannot be derived from each other
  - one of the keys - the **public key** - can be published without fear of compromise of the **private key**

er1.fm: 02.08.2000 10:47

Terminology

Hashes or Cryptographic Checksums

- strong **one-way** functions
- mapping a message of arbitrary length to a fixed size value (e.g., 128 bits)
- computationally infeasible to construct a message that leads to a given hash
- different messages hash to different values
- used in authentication and digital signatures



Terminology

Cryptanalysis

The science of recovering the plain text of a message without access to the key.

Attacks

(1) **Cipher text only**

Given: $C_1 = E_K(M_1), C_2 = E_K(M_2), ...C_n = E_K(M_n)$

Deduce: Either $M_1, M_2, ...M_n$; or an algorithm to infer M_{n+1} from $C_{n+1} = E_K(M_{n+1})$

(2) **Known plain text**

Given: $M_1, C_1 = E_K(M_1), M_2, C_2 = E_K(M_2), ...M_n, C_n = E_K(M_n)$

Deduce: Either Kor an algorithm to infer $M_{n+1}$from $C_{n+1} = E_K(M_{n+1})$

Terminology

Attacks

(3) **Chosen plain text**

Given: $M_1, C_1 = E_K(M_1), M_2, C_2 = E_K(M_2), ...M_n, C_n = E_K(M_n)$
where the crptanalyst gets to choose M_1, M_2, M_n

Deduce: Either Kor an algorithm to infer $M_{n+1}$from $C_{n+1} = E_K(M_{n+1})$

(4) **Adaptive chosen plain text**
Cryptanalyst can not only chose plain text, but he can modify the plain text based on the result of an encryption.

(5) **Chosen cipher text**
Cryptanalyst can chose different cipher texts to be decrypted and gets access to the decrypted plain text

(6) ***Rubber-hose***
Cryptanalyst threatens, blackmails, bribes, or tortures someone until he is given the key. Often the most effective way of breaking an algorithm!

Classical Encryption — *Steganography*

Hide secret messages in other messages

3rd March

Dear George
Greetings to all at Oxford. Many thanks for your letter and for the Summer examination package. All Entry Forms and Fees Forms should be ready for final dispatch to the Syndicate by Friday 10th or at the very latest, I'm told, by the 21st. Admin has improved here, though there's room for improvement still; just give us all two or three more years and we'll really show you! Please don't let these wretched 16+ proposals destroy your basic O and A pattern. Certainly this sort of change, if implemented immediately would bring chaos.

Sincerely yours,

Classical Encryption — *Steganography*

- Invisible ink
- Pin punctures
- Character marking
- Use least significant bits in high resolution images for ***water marking***
  e.g., Photo CD maximum resolution 2048 by 3072 pixel at 24 bits of RGB just using one bit of each color will allow encoding of a 2.3 MB message

Classical Encryption — *Substitution Ciphers*

Caesar Cipher

Replace each letter with the letter three places further down in the alphabet

Generalized

For each plaintext letter m subsitute $C = E(m) = (m + k) \bmod 26$

Very limited key space (25!)

Monoalphabetic Ciphers

Allow an arbitrary permutation of the substitution table. Key space $26! > 4 \times 10^{26}$. Attack still very easy considering the relative frequencies of individual letters, di- and trigraphs.

Classical Encryption — *Substitution Ciphers*

Multiple Letter Encryption

M	**O**	**N**	**A**	**R**
C	**H**	**Y**	B	D
E	F	G	I/J	K
L	P	Q	S	T
U	V	W	X	Z

Encode digraphs using the above table. Repeating plain text letters are separated by a filler letter, e.g., X. Plain text letters that fall into the same row are replaced by the letter to the right. Plain text letters that fall into the same column are replaced by the letter beneath. Otherwise each plain text letter is replaced by the letter that lies in its own row and the column occupied by the other plain text letter.

Thus ***Friday*** is encoded as ***Kokbnb***

Makes frequency analysis slightly more difficult, used in both World Wars.

Classical Encryption — *Substitution Ciphers*

Polyalphabetic Ciphers

For each character of plain text use a different monoalphabetic substitution. The best-known is the Vigenère cipher.

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
b c d e f g h i j k l m n o p q r s t u v w x y z a
. . .
y z a b c d e f g h i j k l m n o p q r s t u v w x
z a b c d e f g h i j k l m n o p q r s t u v w x y
```

- Key is equal to length of plain text. Typically achieved through repeating a fixed size key.
- Key specifies which row is to be used for encryption. Attempts to obscure letter frequency.

```
Key:          deceptivedeceptivedeceptive
Plain text:   wearediscoveredsaveyourself
Cipher text:  zicvtwqngrzgvtwavzhcqyglmgj
```

Repetition of key introduces periods which allow for cryptanalysis. Attempt to discover key length ***n***. Once that is known, we deal with ***n*** monoalphabetic ciphers, which can be solved using character frequency analysis

Vernam Cipher

- Choose keyword that is very long, encrypt with bit-wise ***xor*** $c_i = m_i \oplus k_i$
- Keyword on a tape, very long period. Still vulnerable to cipher text only and plain text attacks.

One Time Pad

- Unbreakable scheme.
- Uses a random key that is as long as the message.
- ***xor*** message with key, and do not reuse key.
- Problem is to provide and protect the random keys at sender and receiver.

Transposition Ciphers

Permutation on the plain text letters instead of substitution

Key: **4312567**
Plain text:. **attackpostponeduntiltomorrow**

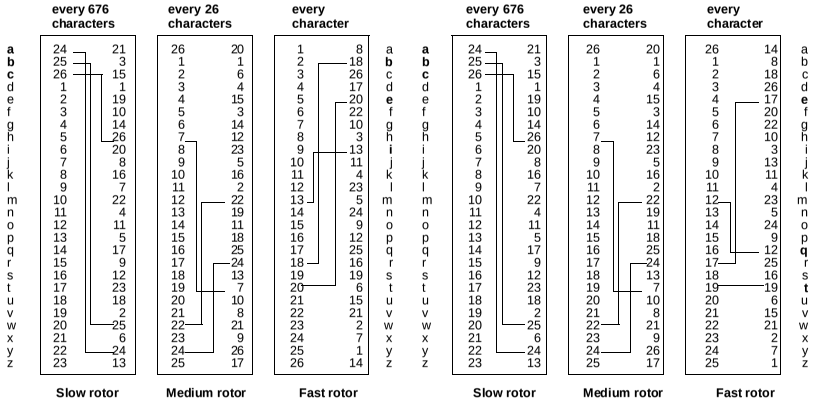
4	3	1	2	5	6	7
a	t	t	a	c	k	p
o	s	t	p	o	n	e
d	u	n	t	i	l	t
o	m	o	r	r	o	w

Cipher text: **ttnoaptrtsumaodocoirknlopetw**

Easily recognizable because letter frequency unchanged. Can be made more secure by feeding cipher text through second stage of transposition:

Cipher text: **nscoouopttoltrdnamieparttokw**

Rotor Machines



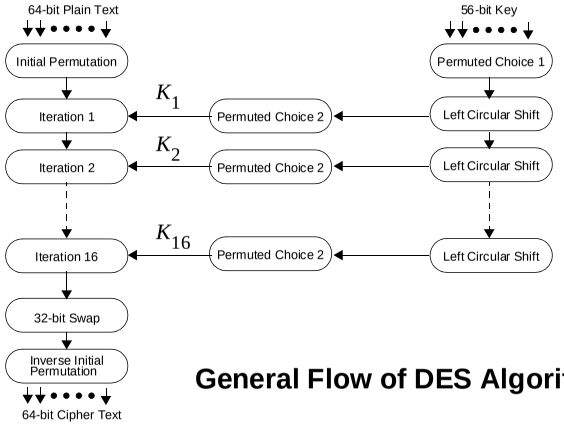
For an *n* rotor machine 26^n substitution alphabets are used before repetition.

Data Encryption Standard (DES)

- Based on an IBM invention, *LUCIFER*
- Submitted 1974 to a a request for standard proposal for cryptographic algorithms to the US National Bureau of Standards (NBS)
- Evaluation of the National Security Agency (NSA) for hardware implementation
- 1977 US federal standard
- 1981 ANSI private sector standard
- Reevaluated every 5 years
- Exportable in full strength for financial sector applications

- Block cipher, encrypts 64-bit blocks
- Key length 56 bits, usually expressed as 64-bit number, every 8th bit used for parity checking

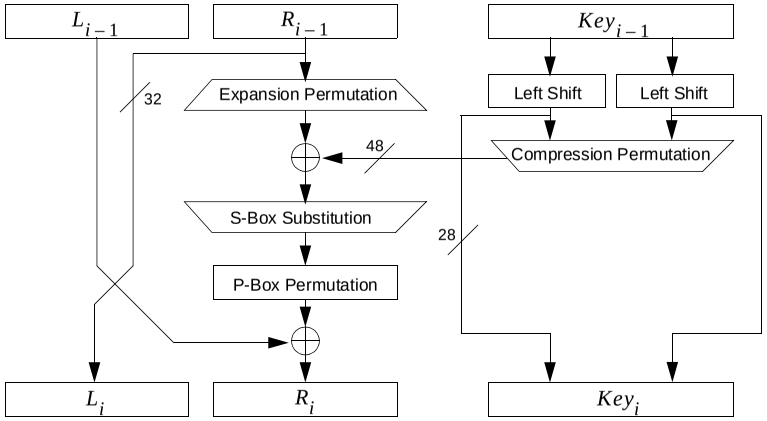
Data Encryption Standard (DES)



General Flow of DES Algorithm

er1.fm: 02.08.2000 10:47

DES Operation



One Round of DES Encryption

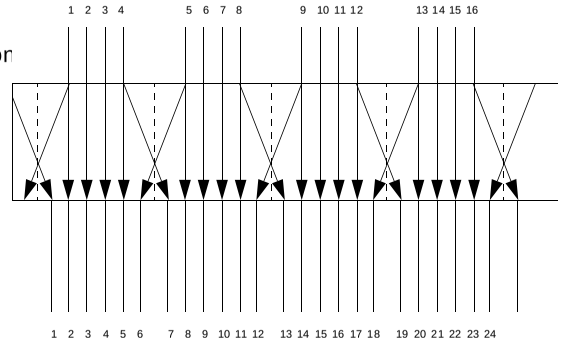
er1.fm: 02.08.2000 10:47

DES Operation

Processing in round i

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

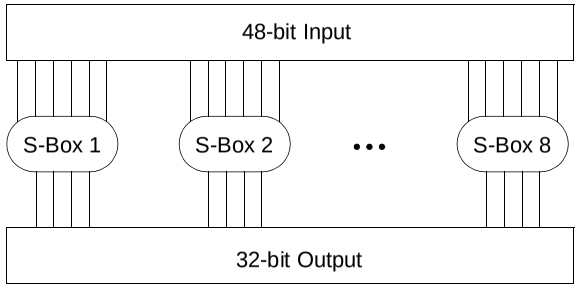
Expansion permutation



er1.fm: 02.08.2000 10:47

DES Operation *Substitution-Box*

- Each S-Box can perform 4 different output substitutions of its input bits
- Which substitution is chosen depends on bit 1 and 6 of its input.



- 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

er1.fm: 02.08.2000 10:47

DES Operation — *Key Generation*

- The 56 bit key is permuted first
- Resulting key is treated as two 28 bit quantities, labeled C_0 and D_0
- At each iteration C and D are rotated left by 1 or 2 bits
- These values serve as input to the next iteration and
- as input to a second permutation which produces a 48-bit output *xor*-ed with the input to the S-Box

DES Operation — *Decryption*

- Uses same algorithm
- Keys are used in inverse order: if K_1, K_2, ..., K_{16} are used in the encryption then K_{16}, K_{15}, ..., K_1 are used in the decryption process
- Key generation algorithm is the inverse of that used in the encryption algorithm
- Key generation schedule:

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Encryption Rotate Left	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1
Decryption Rotate Right	0	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

DES — *Avalanche Effect*

- Every small (1 bit) change in the input or key should create a significant change in the cipher text
- Avalanche effect is one sign for the strength of an encryption algorithm
- Achieved through the design of the S-Boxes and the expansion permutation where one bit affects two substitutions

DES — *S-Box Design Criteria*

- Each S-box has six input bits and four output bits
- No output bit of an S-box should be close to a linear function of the input bits
- Fixing the left and right-most input bits of an S-box and varying the four middle bits should cause each possible four bit output to be attained once
- If two inputs to an S-box differ in exactly one bit, the outputs must differ in at least two bits
- If two inputs to an S-box differ in the two middle bits exactly, the output must differ in at least two bits
- If two inputs to an S-box differ in their first two bits and are identical in their last two bits, the two outputs must not be the same
- For any non-zero six bit difference between inputs, no more than eight of the 32 pairs of inputs exhibiting that difference may result in the same output difference

DES *P-Box Design Criteria*

- The four output bits from each S-box in round *i* are distributed so that two of them affect the middle-bits of S-boxes at round *i*+1 and the other two affect end bits
- The four output bits from each S-box affect six different S-boxes; no two affect the same S-box
- If the output bit from one S-box affects a middle bit of another S-box, then an output bit from that other S-box cannot affect a middle bit of the first S-box

Breaking DES

Brute force

- Search the entire key space of $2^{56} \sim 7.2 \cdot 10^{16}$ possible keys
- Know plain text attack using a proposed key search chip would in 1993 have required 576000 such chips and would have cost $10 Million and taken 21 minutes to retrieve a key
- Additional difficulty is detection of when the correct plain text was decrypted

Differential Cryptanalysis

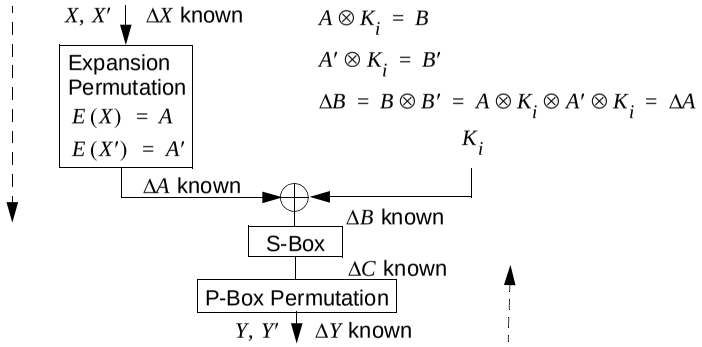
- Trace through rounds of DES
- Based on chosen plain text attack
- Best published results require 2^{47} chosen plain texts

Linear Cryptanalysis

- Requires 2^{43} known plain texts, linear approximation of encryption function

Breaking DES *Differential Cryptanalysis*

- Look at cipher text pairs, i.e., pairs of cipher text whose plain texts have particular differences
- For DES the difference is defined as the *XOR* of the plain texts
- Analyze evolution of these differences as the plain texts propagate through DES rounds

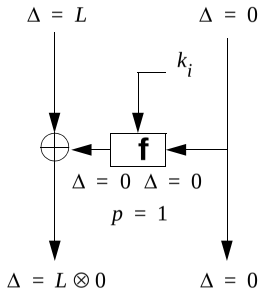


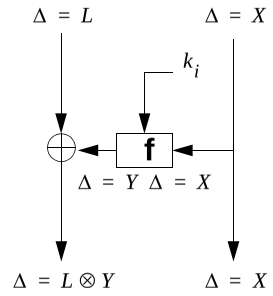
Breaking DES *Differential Cryptanalysis*

- Certain differences in plain text pairs cause certain differences in cipher text pairs
- These differences are called ***characteristics***
- Characteristics extend over a number of rounds
- Characteristics can be calculated for each S-Box

Difference: input / output	0	1	2	3	4	5	...
0	p_{00}	p_{10}	p_{20}	p_{30}	p_{40}	p_{50}	
1	p_{01}	...	...	...	...	...	
2	...	...	...	...	...	...	
3	...	...	...	...	...	...	
...							

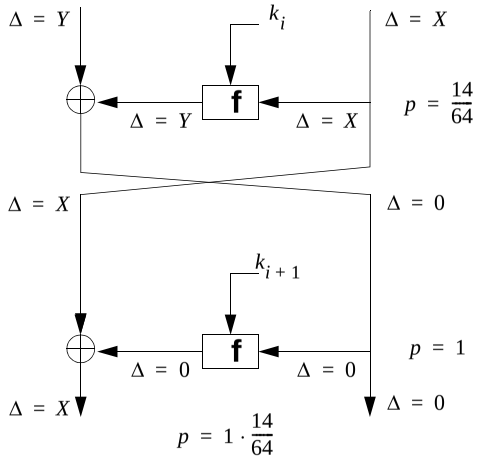
p_{ik} is the probability that from an input difference *i* an output difference *k* results





$p = \frac{14}{64}$ $X = 60000000$ $Y = 00808200$

er1.fm: 02.08.2000 10:47



er1.fm: 02.08.2000 10:47

- Plain text pairs which satisfy the characteristic are *right* pairs
- Right pairs will suggest the correct round key
- Round key found by trying enough guesses so subkey is selected more often than others

er1.fm: 02.08.2000 10:47

Electronic Codebook Mode (ECB)

- For a given key there is a unique cipher text for every 64 bit input block of plain text
- Suitable for communicating single values. E.g., a DES key
- Not suitable for highly structured messages as a cryptanalyst may be able to exploit the regularities to derive some know plain text cipher text pairs

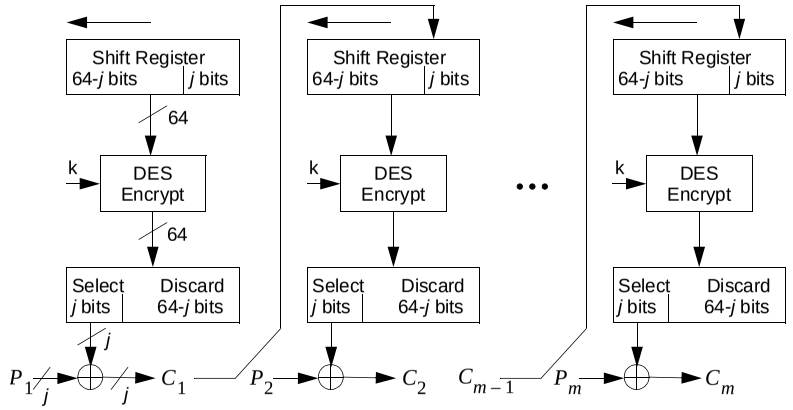
Cipher Block Chaining (CBC)

- Used for general purpose bulk transmissions
- Input to the encryption algorithm is the *XOR* of the next 64 bits of plain text and the preceding cipher text
- A secret initialization vector is used for the *XOR* operation on the first block.

er1.fm: 02.08.2000 10:47

Cipher Feedback (CFB)

- Stream cipher as opposed to the basic DES block (64 bit) cipher
- Input processed *j* bits at a time
- Preceding cipher text is used as input to encryption to generate a pseudo random string
- This output is *xor*-ed with *j* bits of input to produce the cipher text
- Requires a secret and shared initialization vector



Output Feedback Mode

- Stream cipher
- Similar to CFB mode, however the input to the encryption function is solely based on the pseudo random string derived from the secret initialization vector.
- Better resistance against bit errors in transmission as only one *j*-bit piece of the cipher text is hit, others can be recovered
- Vulnerable to attacks that change single bits in the cipher text

Double DES

- Perform two encryptions using two different keys $C = E_{K_2}(E_{K_1}(P))$
- Vulnerable to so-called *Meet-in-the-Middle* Attack
- If $C = E_{K_2}(E_{K_1}(P))$, then $X = E_{K_1}(P) = D_{K_2}(C)$
- Given a known *P* and *C* encrypt *P* for all 2^{56} possible values of K_1
- Store these values in a table sorted by *X*
- Decrypt *C* with all 2^{56} possible values of K_2
- Check the result against the table for a match, if a match occurs test the two resulting keys against a known plain text cipher text pair
- A known plain text attack against double DES (112 bit key size) will succeed with an effort on the order of 2^{56} operations

Making DES More Secure

Triple DES

- Use three stages of encryption to counter the meet-in-the-middle attack
- However instead of using a key of 168 bits, perform an encrypt-decrypt-encrypt cycle using only 112 bit keys
- Cipher text is produced through $C = E_{K_1}(D_{K_2}(E_{K_1}(P)))$
- Compatibility is maintained for older plain DES implementations
- Currently there are no known practical cryptanalytic attacks on triple DES
- Brute force key search is on the order of $2^{112} = 5 \cdot 10^{35}$ operations

er1.fm: 02.08.2000 10:47

Making DES More Secure

DES with Key-Dependent S-Boxes

- Linear and differential cryptanalysis work only if the composition of the S-boxes is known
- To counter these attacks make the order of the S-boxes dependent on an additional 56 bit key:
  (1) Rearrange the DES S-boxes in the order: 2 4 6 7 3 1 5 8
  (2) Select 16 key bits. If the first bit is 1, swap the first two rows of S-box 1 with its last two rows. If the second bit is 1, swap the first 8 columns of S-box 1 with its second 8 columns. Do the same with S-box *i* for bits 2*i*-1 and 2*i.*
  (3) Take the remaining 32 key bits, *XOR* the first four with every entry of S-box 1, the second four with every entry of S-box 2, etc.
- Complexity of a differential cryptanalysis attack against this system is 2^{51}; linear cryptanalysis attack is 2^{53}; brute force 2^{112}
- Easy to implement with current hardware, as many DES chips are sold with loadable S-boxes

er1.fm: 02.08.2000 10:47

Some Number Theory

Divisors

We say that $b \neq 0$ divides a if $a = m \cdot b$ for some m . $b|a$ means b divides a

The following relations hold:

(1) If $a|1$, then $a = \pm 1$
(2) If $a|b$ and $b|a$, then $a = \pm b$
(3) Any $b \neq 0$ divides 0
(4) If $b|g$ and $b|h$ then $b|(mg + nh)$ for arbitrary integers m and n

Prime Numbers

An integer $p > 1$ is a prime if its only divisors are ± 1 and $\pm p$. Any positive integer a can be factored uniquely as $a = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot \ldots \cdot p_t^{\alpha_t}$ where $p_1 > p_2 > \ldots > p_t$ are prime numbers and each $\alpha_i \geq 0$

er2.fm: 01.08.2000 13:15

Some Number Theory

If P is the set of all primes then any positive integer can be expressed as

$a = \prod_{P} p^{a_p}$ where each $(a_p \geq 0)$. If we can say $a|b \Rightarrow a_p \leq b_p$ for all p .

Relatively Prime Numbers

Using the above notation the greatest common divisor (*gcd*) of two numbers a and b is defined as follows: $k = \gcd(a, b)$ if $k_p = \min(a_p, b_p) \ \forall p$

Two numbers are said to be relatively prime, if their greatest common divisor is 1.

Modular Arithmetic

Given any integer *a* and any quotient *n*, *a* can be expressed as:

$$a = q \cdot n + r \qquad 0 \leq r < n; \quad q = \left\lfloor \frac{a}{n} \right\rfloor$$

er2.fm: 01.08.2000 13:15

We define $a \bmod n$ to be the remainder when *a* is divided by *n*. Thus for any *a* we can write $a = \left\lfloor \frac{a}{n} \right\rfloor \cdot n + (a \bmod n)$

Two integers *a* and *b* are **congruent modulo *n***, if $a \bmod n = b \bmod n$. This is written as $a \equiv b \bmod n$. The modulo operator has the following properties:

(1) $a \equiv b \bmod n$ if $n \mid (a - b)$

(2) $a \equiv b \bmod n \Rightarrow b \equiv a \bmod n$

(3) $a \equiv b \bmod n$ and $b \equiv c \bmod n \Rightarrow a \equiv c \bmod n$

Modular arithmetic has the following properties:

(1) $[(a \bmod n) + (b \bmod n)] \bmod n = (a + b) \bmod n$

(2) $[(a \bmod n) - (b \bmod n)] \bmod n = (a - b) \bmod n$

(3) $[(a \bmod n) \cdot (b \bmod n)] \bmod n = (a \cdot b) \bmod n$

(4) If $a \cdot b \equiv a \cdot c \bmod n$ then $b \equiv c \bmod n$ if *a* relatively prime to *n*

Euler's Totient Function

Euler's Totient function $\Phi(n)$ is the number of positive integers less than some *n* which are relatively prime to *n*. For any prime p the following holds: $\Phi(p) = p - 1$

If we have two prime numbers p and q with $(p \neq q)$ then for $n = p \cdot q$

$$\Phi(n) = \Phi(p \cdot q) = \Phi(p) \cdot \Phi(q) = (p-1) \cdot (q-1)$$

The set of reminders in Z_n is $\{0, 1, 2, \ldots, p \cdot q - 1\}$. The reminders that are not relatively prime to n are all multiples of q, namely $\{q, 2 \cdot q, \ldots, (p-1) \cdot q\}$, all multiples of p and 0.

$$\begin{aligned} \Phi(n) &= p \cdot q - [(q-1) + (p-1) + 1] \\ &= p \cdot q - (p + q) + 1 \\ &= (p-1) \cdot (q-1) \\ &= \Phi(p) \cdot \Phi(q) \end{aligned}$$

Fermat's theorem states:

$a^{n-1} \equiv 1 \bmod n$ if *a* and *n* are relatively prime and *n* is prime

Lemma:

Consider the sequence . It consists of all the integers from 1 to $n - 1$.

$$a \bmod n, 2a \bmod n, \ldots, (n-1) \cdot a \bmod n$$

If this were not the case then two different integers $x, y < n$ would have to exist for which the relation $x \cdot a \equiv y \cdot a \bmod n$ would have to hold. Because, however, a is relatively prime to *n* we can cancel a and get $x \equiv y \bmod n$ and thus $x = y$ which is a contradiction.

Proof:

$$\begin{aligned} a \times 2a \times 3a \times \ldots \times (n-1) \cdot a &\equiv [(a \bmod n) \times (2a \bmod n) \times \ldots \times ((n-1)\, a \bmod n)] \bmod n \\ &\equiv (n-1)! \bmod n \end{aligned}$$

However

$$a \times 2a \times \ldots \times (n-1) \cdot a = (n-1)! \cdot a^{n-1}$$

and therefore

$$(n-1)! \cdot a^{n-1} \equiv (n-1)! \bmod n$$

Because n is prime, $(n-1)!$ is relatively prime to n.

We therefore can cancel $(n-1)!$ from the congruence which yields Fermat's theorem:

$a^{n-1} \equiv 1 \bmod n$ or $a^n \equiv a \bmod n$, if a, n are relatively prime and n is prime.

Euler's theorem states for any *a* and *n* that are relatively prime to each other that:

$$a^{\Phi(n)} \equiv 1 \bmod n$$

Proof:

This is true by Fermat's Theorem if *n* is prime. Consider $R = \{x_1, x_2, \ldots, x_{\Phi(n)}\}$

Now multiply each element by $a \bmod n$:

$$S = \{(ax_1 \bmod n), (ax_2 \bmod n), \ldots, (ax_{\Phi(n)} \bmod n)\} \quad .$$

This set is a permutation of R because:

(1) a is relatively prime to n and x_i is relatively prime to n, thus $a \cdot x_i$ is also relatively prime to n. Because of the modulo operator these numbers are also all less than n.

(2) All members of S are different, the proof is analog the lemma for Fermat's theorem.

er2.fm: 01.08.2000 13:15

Therefore:

$$\prod_{i=1}^{\Phi(n)} (a \cdot x_i \bmod n) = \prod_{i=1}^{\Phi(n)} x_i$$

$$\prod_{i=1}^{\Phi(n)} a \cdot x_i \equiv \prod_{i=1}^{\Phi(n)} x_i \bmod n$$

$$a^{\Phi(n)} \cdot \prod_{i=1}^{\Phi(n)} x_i \equiv \prod_{i=1}^{\Phi(n)} x_i \bmod n$$

Because all x_i are relatively prime to *n*, they can be cancelled out, thus yielding:

$$a^{\Phi(n)} \equiv 1 \bmod n \text{ or } \quad a^{\Phi(n)+1} \equiv a \bmod n$$

er2.fm: 01.08.2000 13:15

Consider:

$$a^m \equiv 1 \bmod n$$

If *a* and *n* are relatively prime to each other, then at least one integer exists that fulfills the congruence namely $m = \Phi(n)$. The smallest m that satisfies the above equation is called the length of the period generated by a.

For example:

$$7^1 \bmod 19 = 7$$
$$7^2 \bmod 19 = 11$$
$$7^3 \bmod 19 = 1$$
$$7^4 \bmod 19 = 7$$

The period generated by 7 for 19 is 3. The sequence repeats because $7^3 \equiv 1 \bmod 19$ and therefore $7^{3+j} = 7^3 \cdot 7^j \equiv 7^j \bmod 19$.

er2.fm: 01.08.2000 13:15

We can observe:

(1) All sequences end in 1

(2) The length of the sequence divides $\Phi(n)$

(3) Only for integers of the form $2, 4, p^{\alpha}, 2p^{\alpha}$ where p is an odd prime some a can produce sequences of length $\Phi(n)$

A sequence of length $\Phi(n)$ is called a **primitive root** of n; e.g. 3 is a primitive root of 19.

Recall that ordinary logarithms $y = x^{\log(y)}$ have the following properties (*log* is base *x)*:

(1) $\log(1) = 0$

(2) $\log(x) = 1$

(3) $\log(y \cdot z) = \log(y) + \log(z)$

(4) $\log(y^r) = r \cdot \log(y)$

er2.fm: 01.08.2000 13:15

Some Number Theory *Discrete Logarithms*

Any integer *b* can be expressed in the form

$$b = r \bmod p \qquad 0 \le r \le p-1$$

For a primitive root *a* of some integer *p* one can find a unique exponent such that:

$$b = a^i \bmod p \qquad 0 \le i \le p-1$$

This exponent i is called the index of b for base $a \bmod p$, we denote it as $\underset{a,p}{\text{ind}}(b)$

Note:

$\underset{a,p}{\text{ind}}(1) = 0$ because $a^0 \bmod p = 1 \bmod p = 1$

$\underset{a,p}{\text{ind}}(a) = 1$ because $a^1 \bmod p = a$

Some Number Theory *Discrete Logarithms*

We recall:

$$a^{\Phi(n)} \equiv 1 \bmod n$$

Any positive integer z can be expressed as $z = q + k \cdot \Phi(n)$.

Therefore by Euler's theorem:

$$a^z = a^q \bmod n \qquad \text{if } z = q \bmod \Phi(n)$$

We can use this relation for:

$$\text{ind}(xy) = [\text{ind}(x) + \text{ind}(y)] \bmod \Phi(p)$$

and in general

$$\text{ind}(x^r) = [r \cdot \text{ind}(x)] \bmod p$$

Some Number Theory *Discrete Logarithms*

For $y = g^x \bmod p$ given x, g and p, it is very easy to calculate y, however the inverse given y, g and p it is very hard to calculate the discrete logarithm x.

The fastest known algorithm for discrete logarithms is: $O(e^{(\ln p)^{1/3} \ln(\ln p)^{2/3}})$ which is not feasible for large primes p.

Diffie-Hellman Key Exchange

- First published public-key algorithm, 1976
- Security based on difficulty of calculating discrete logarithms in a finite field
- Useful for key distribution to generate a secret key
- Cannot be used for encryption

Protocol

(1) A large prime n and g, such that g is a primitive root of n are published

(2) A chooses a random large integer x and sends B: $X = g^x \bmod n$

(3) B chooses a random large integer y and sends A: $Y = g^y \bmod n$

(4) A computes $k = Y^x \bmod n$

(5) B computes $k' = X^y \bmod n$

Both k and k' are equal to $g^{xy} \bmod n$

An attacker would have to calculate, for example, $\underset{g,n}{\text{ind}}(X)$ to attack A's secret key

Diffie-Hellman with Three or More Parties

The key-exchange protocol can easily be extended to work between three or more parties:

(1) A chooses a random large integer x and computes: $X = g^x \bmod n$

(2) B chooses a random large integer y and sends: $Y = g^y \bmod n$ to C

(3) C chooses a random large integer z and sends: $Z = g^z \bmod n$ to A

(4) A sends B: $Z' = Z^x \bmod n$

(5) B sends C: $X' = X^y \bmod n$

(6) C sends A: $Y' = Y^z \bmod n$

(7) A computes $k = Y'^x \bmod n$

(8) B computes $k = Z'^y \bmod n$

(9) C computes $k = X'^z \bmod n$

The secret key is then $k = g^{xyz} \bmod n$. Only the participants of that conversation can compute this value.

Diffie-Hellman Key-Exchange *Variation*

Hughes describes a variation of the key-exchange protocol that allows one user to generate the key before any interaction and thus message encryption is possible without contacting the recipient(s) first.

(1) A chooses a random large integer x and computes $k = g^x \bmod n$

(2) B chooses a random large integer y and sends A: $Y = g^y \bmod n$

(3) A sends B: $X = Y^x \bmod n$

(4) B computes

$$z = y^{-1}$$

$$k' = X^z \bmod n$$

Key-Exchange without Exchanging Keys

- Each user i publishes a public key $X_i = g^{x_i} \bmod n$
- If one user wants to communicate with another, he retrieves the key and can generate the shared secret key
- The recipient could then also retrieve his partner's public key to generate the secret key
- Each pair of users would thus have a unique shared secret key, without requiring any prior communication by the users
- The public keys must be certified to prevent spoofing attacks and should be changed regularly

RSA Public-Key Encryption

Background

- Named after its inventors Rivest, Shamir, and Adleman
- Published in 1978 as response to a 1976 paper by Diffie and Hellman challenging cryptologists to come up with cryptographic algorithms that meet the requirements of public-key systems

(1) It is easy for a party to generate a pair of p**u**blic (*KU*) and p**r**ivate (*KR*) keys

(2) It is easy for a sender A knowing B's public key to encrypt a message

$$C = E_{KU_B}(M)$$

(3) It is easy for B to decrypt the cipher text using the private key

$$M = D_{KR_B}(C) = D_{KR_B}(E_{KU_B}(M))$$

(4) It is infeasible for an opponent knowing a public key *KU* to compute the private key *KR*

RSA Public-Key Encryption

Background cont.

(5) It is infeasible for an opponent knowing the public key *KU* and the cipher text *C* to reconstruct the plain text message *M*

(6) The encryption and decryption functions can be applied in either order:

$$M = E_{KU}(D_{KR}(M)) = D_{KR}(E_{KU}(M))$$

- RSA is the only widely accepted approach to public-key cryptography
- Implemented and used in many applications, e.g., Lotus Notes, PGP, SET, PEM, SSL,...

RSA Algorithm

- Block cipher where plain/cipher text is an integer between 0 and $n-1$
- Encryption of a message *M* and decryption of cipher text *C* work as follows:

$$C = M^e \bmod n$$

$$M = C^d \bmod n = (M^e)^d \bmod n = M^{ed} \bmod n$$

- The public key *KU* is $\{e, n\}$, the private key *KR* is $\{d, n\}$

The problem is now to find a relationship of the form: $M^{ed} \equiv M \bmod n$

RSA Algorithm

We recall Euler's theorem which stated:

$$m^{\Phi(n)} \equiv 1 \bmod n \qquad \text{if } m \text{ and } n \text{ are relatively prime}$$

We need to develop a corollary to Euler's theorem to derive the RSA algorithm, because we cannot guarantee that messages are relatively prime to n.

Given two primes p, q and **integers** $n = pq$ and m, with $(0 < m < n)$ the relationship $m^{\Phi(n)} = m^{(p-1)\cdot(q-1)} \equiv 1 \bmod n$ or $m^{\Phi(n)+1} = m \bmod n$ holds

Proof

- if $\gcd(m, n) = 1$, i.e., m and n are relatively prime, then Euler's theorem holds
- if $\gcd(m, n) \neq 1$ then m must be a multiple of p or q.
- if m is a multiple of p, then $\exists c$ with $m = c \cdot p$ and $\gcd(m, q) = 1$. The reason is that $m < pq$ and q is prime . Thus $m^{\Phi(q)} \equiv 1 \bmod q$ holds

RSA Algorithm

Proof cont.

- following the rules of modular arithmetic, we can then state

$$(m^{\Phi(q)})^{\Phi(p)} \equiv 1 \bmod q \qquad m^{\Phi(n)} \equiv 1 \bmod q \qquad \text{and } m^{k\Phi(n)} \equiv 1 \bmod q$$

- Then there must be some integer j such that $m^{\Phi(n)} = 1 + j \cdot q$
- Multiplying each side with $m = c \cdot p$ the gives

$$m^{\Phi(n)+1} = m + jcpq = m + jcn$$

$$m^{\Phi(n)+1} \equiv m \bmod n \quad \text{and equivalently} \quad m^{k\Phi(n)+1} \equiv m \bmod n$$

- The same reasoning can be applied for m being a multiple of q.
- This proves the corollary, which can also be expressed in the following way:

$$m^{k\Phi(n)+1} = m^{k(p-1)(q-1)+1} \equiv m \bmod n$$

RSA Algorithm

We can now solve $M^{ed} \equiv M \bmod n$

$$ed = k(p-1)(q-1) + 1$$
$$ed \equiv 1 \bmod (p-1)(q-1)$$
$$e \equiv d^{-1} \bmod \Phi(n)$$ provided e and d are relatively prime to $\Phi(n)$

For the RSA Algorithm we therefore need:

(1) p, q two primes, secretly chosen

(2) $n = pq$ public

(3) e with $\gcd(\Phi(n), e) = 1$ and $(0 < e < \Phi(n))$ public

(4) d with $\gcd(\Phi(n), d) = 1$ and $(0 < d < \Phi(n))$ and $ed \equiv 1 \bmod \Phi(n)$ privately calculated and kept secret

RSA Algorithm *Example*

First we chose the parameters

$$p = 13 \qquad q = 2$$
$$n = 13 \cdot 2 = 26 \qquad \Phi(26) = 12$$
$$e = 5$$
$$d \cdot 5 \bmod 12 = 1 \qquad d = 5$$

Then we encrypt a message

$$m = 7 \qquad 7^5 = 16807 \qquad c = 16807 \bmod 26 = 11$$
$$c = 11 \qquad 11^5 = 161051 \qquad m = 161051 \bmod 26 = 7$$

Breaking the RSA Crypto System

Factoring n and computing d

Factor $n = pq$ and compute $(p-1)(q-1)$. Since we know e we can the calculate d.

The security of the RSA system thus relies on the assumption that factoring a large number is difficult. Factoring n would allow a cryptanalyst to break the scheme.

The fastest known factoring algorithms can factor a number n in

$O(e^{\sqrt{\ln n \times \ln(\ln n)}})$ time which makes a brute force attack infeasible.

Breaking the RSA Crypto System

Computing (*p*-1)(*q*-1) without Factoring

If we could compute $(p-1) \cdot (q-1) = \Phi(n)$ we would also know how to factor n.

$$(p-1) \cdot (q-1) = n - (p+q) + 1 = \Phi(n)$$
$$(p+q) = n + 1 - \Phi(n)$$
$$(p-q) = \sqrt{(p+q)^2 - 4n}$$

As we know n we can solve for p and q:

$$q = \frac{n + 1 - \Phi(n) - \sqrt{(n + 1 - \Phi(n))^2 - 4n}}{2}$$

Breaking the RSA Crypto System

Computing d without factoring n nor knowing $\Phi(n)$

$$ed - 1 = k(p-1)(q-1)$$

Again we could factor n.

Computing $D(c)$ directly from $E(m)$

$$m^e \equiv c \bmod n$$

Doing this reduces to the question whether we can take the e-th root modulo n quickly.

Breaking the RSA Crypto System

Chosen Cipher Text Attack Against RSA (1)

Eve listening on Alice's communications collects a cipher text message c. She wants to read the message, i.e., $m = c^d \bmod n$. To recover m she chooses a random number $r, r < n$. She takes Alice's public key e and computes

$$x = r^e \bmod n$$
$$y = x \cdot c \bmod n$$
$$t = r^{-1} \bmod n$$

If $x = r^e \bmod n$ then $r = x^d \bmod n$. Eve gets Alice to sign y with her private key d, i.e., $u = y^d \bmod n$. Now she can compute

$$t \cdot u \bmod n = r^{-1} \cdot y^d \bmod n = r^{-1} \cdot x^d \cdot c^d \bmod n = c^d \bmod n = m$$

Breaking the RSA Crypto System

Chosen Cipher Text Attack Against RSA (2)

Trent is a computer notary public. If Alice wants a document notarized she sends it to Trent. He signs it with his private RSA key and sends it back.

Mallory wants Trent to sign a message m', Trent never wouldn't sign. Mallory chooses an arbitrary value x and computes $y = x^e \bmod n$ with Trent's public key e. He then computes $m = y \cdot m' \bmod n$, which he sends on to Trent for signature. Trent returns $m^d \bmod n$. Now Mallory can compute

$$(m^d \bmod n) \cdot (x^{-1} \bmod n) = m'^d \bmod n$$

Breaking the RSA Crypto System

Chosen Cipher Text Attack Against RSA (3)

Eve wants Alice to sign m_3. She generates two messages such that $m_3 \equiv m_1 \cdot m_2 \bmod n$. She now gets Alice to sign m_1 and m_2. She now can calculate herself the signed m_3, namely $m_3^d = (m_1^d \bmod n) \cdot (m_2^d \bmod n)$

Breaking the RSA Crypto System

Common Modulus Attack Against RSA

Assume we don't reveal p and q and assign everyone the same modulus n and different encryption/decryption keys. The attack works as follows, if the same message is encrypted with two different encryption keys, it can be recovered without knowing the decryption keys:

$$c_1 = m^{e_1} \bmod n$$

$$c_2 = m^{e_2} \bmod n$$

The cryptanalyst knows c_1, c_2, e_1, e_2 and n. As e_1 and e_2 are generally relatively prime, he computes $r \cdot e_1 + s \cdot e_2 = 1$ using the extended Euclid algorithm to find inverses modulo n. One of the two factors r, s is negative, assume r. To recover m the extended Euclid algorithm is used again to compute c_1^{-1}. Then

$$m = ((c_1^{-1})^{-r} \cdot c_2^s) \bmod n$$

Breaking the RSA Crypto System

Encrypting a Message Before Signing

Alice wants to send a message to Bob, instead of following the usual practice of signing the message first and then encrypting it, she first encrypts it with Bob's public key and then signs it with her private key $(m^{e_B} \bmod n_B)^{d_A} \bmod n_A$.

Bob may now claim that Alice sent him m' instead of m. Since Bob knows the factorization of $n_B = p_B \cdot q_B$ he can calculate discrete logarithms with respect to n_B. He just has to find an x such that $m'^x = m \bmod n_B$. He then could publish $x \cdot e$ as his new public exponent without having to change n_B and claim that Alice sent him m' instead of m.

RSA Crypto System *Implementation Issues*

Finding Large Prime Numbers p and q

(1) Pick an odd integer n at random (use a pseudo random number generator)

(2) Test if n is prime

(3) If n is composite, then $n = n + 2$, go to step 2

The chances of finding a prime number are: $P(\text{integer} \le n \text{ is prime}) \sim \frac{1}{\ln n}$.

With very high probability one will thus find an integer in the interval $[n, n + \ln n]$ and will at most only need to test $\frac{\ln n}{2}$ different integers.

RSA Crypto System *Implementation Issues*

Testing for Primality (Miller-Rabin)

As factoring a large number is infeasible, the following two laws are used:

(1) Fermat: If p is prime $\forall x \ (x < p) \quad x^{p-1} \equiv 1 \bmod p$

(2) Miller: If p is prime $\forall x \ (x < p)$ and $\forall i$ such that $\frac{p-1}{2^i} = m$ then

$\gcd(x^{m-1}, p) = 1$ or p

Testing for Primality - cont.

Generate a random number x such that $2 < x < n$ and test:

(1) Is $x^{p-1} \equiv 1 \bmod p$?

(2) Choose i such that $m = \frac{p-1}{2^i}$ is integral and test if $\gcd(x^{m-1}, p) = 1$ or p ?

If either test returns false, p is not a prime. Otherwise we don't know anything and generate a new x. After k positive tests, the probability that p is composite is $\leq \frac{1}{2^k}$

The reason is Rabin's theorem: if p is composite at least half of the integers $x < n$ will act as witnesses in test 1 or 2.

Exponentiation and Modulo Operation

```
int exp(int m, int e, int n)
{
        int c=1, M=m;
        while(e){
                if(e&1)c=(c*M)%n;
                e >>= 1;
                M = (M*M)%n;
        }
        return c;
}
```

requires at most $2 \cdot \mathrm{ld}(e)$ operations

Popular encryption key values are 3, 17, and 65537. Either number has in its binary representation just two ones. X.509 recommends using 65537, PEM recommends 3, and PKCS #1 3 or 65537.

Euclid's Algorithm to Determine the gcd (1)

For any non-negative integer a and any positive integer b:

$$\gcd(a, b) = \gcd(b, a \bmod b)$$

To prove this consider $d = \gcd(a, b)$, then $d|a$ and $d|b$. For any positive integer b, a can be expressed as $a = kb + r = kb + a \bmod b$. Thus $a \bmod b = a - kb$ for some k. But since $d|b$, d also divides kb. Per our definition $d|a$, so we can conclude $d|a \bmod b$. So d is a common divisor of b and $a \bmod b$.

For the right side of the equation, if $d|b$, then $d|kb$ and because $d|a \bmod b$, then $d|[kb + a \bmod b]$ which is equivalent to $d|a$.

Using this equation we can derive a fast algorithm to determine the *gcd* of two numbers.

Euclid's Algorithm to Determine the gcd (2)

```
int euclid(int a, int b)
{
        int x, y, r;
        x=a;
        y=b;
        while(y!=0){
                r=x%y;
                x=y;
                y=r;
        }
        return x;
}
```

Determining the Multiplicative Inverse (1)

If $\gcd(a, n) = 1$ then a has a multiplicative inverse a^{-1} modulo n such that $a \cdot a^{-1} \equiv 1 \bmod n$. We can extend Euclid's algorithm to yield the multiplicative inverse.

```
int inverse(int a, int n)
{
    int x1=1, x2=0, x3=a, y1=0, y2=1, y3=n, t1, t2, t3, q;
    for(){
        if(y3==0) return 0;  // x3 == gcd(a,n) != 1
        if(y3==1) return y1; // y3 == gcd(a,n) == 1
        q=x3/y3;
        t1=x1-q*y1, t2=x2-q*y2, t3=x3-q*y3;
        x1=y1, x2=y2, x3=y3;
        y1=t1, y2=t2, y3=t3;
    }
}
```


Determining the Multiplicative Inverse (2)

Throughout the computation the following relationships hold:

$$ax_1 + nx_2 = x_3 \qquad ay_1 + ny_2 = y_3 \qquad at_1 + nt_2 = t_3$$

In each iteration x_3 is assigned the previous value of y_3 and y_3 the previous value of $x_3 \bmod y_3$. If $\gcd(a, n) = 1$ then we have in the last iteration $y_3 = 1$ and can state:

$$ay_1 + ny_2 = 1$$
$$ay_1 = 1 + (-n)y_2$$
$$ay_1 \equiv 1 \bmod n$$

Thus $y_1 = a^{-1}$ is the multiplicative inverse of $a \bmod n$.

A one-way hash function $H(m) = h$ operates on an arbitrarily long message and returns a fixed-length hash value h. One way hash functions must fullfil the following

(1) Given m it must be easy to compute h

(2) Given h it must be hard to compute m such that $H(m) = h$

(3) Given m it must be hard to find another message m' such that $H(m) = H(m')$

Hash functions provide a *fingerprint* that is unique for a given message. If Alice signs the hash of a message, and Mallory could create a different message that had the same hash, he could claim that Alice had signed that message.

A hash function should however also protect against the so-called birthday attack, i.e., an adversary would like to find two random messages m and m' such that $H(m) = H(m')$. This is an easier attack than the above one. It can easily be mounted by generating enough variations of a legal message and an equivalent number of a fraudulent message. Of each set the hashes are computed and if a match is found, the appropriate fraudulent message can replace the legal one.

How many people must be in a room such that the probability p that one other person has his birthday on the same day as you is $p > 0.5$. The probability that one other person has his birthday on the same day as you is $\frac{1}{365}$, for n persons it is then $\frac{n}{365}$ which means for $n = 183$ that $p > 0.5$.

How many people must be in a room such that the probability p that two share the same birthday is $p > 0.5$.

We define $P(n, k) =$ Probability (at least one duplicate in k items) where each item takes on equally likely values from 1 to n.

We will first compute the inverse probability. The number of different ways N we can have k values with no duplicates is:

$$N = n \cdot (n-1) \cdot (n-2) \cdot (n-\ldots) \cdot (n-k+1) = \frac{n!}{(n-k)!}$$

Birthday Paradox (2)

On the other hand, we have n^k different ways we can select k items out of a range from 1 to n if we allow multiple items with the same value. So the probability that there are no duplicates when we select k items from the same range is

$$Q(n,k) = \frac{n!}{(n-k)! \cdot n^k}$$

So now we can solve for $P(n,k)$ and obtain:

$$P(n,k) = 1 - Q(n,k) = 1 - \frac{n!}{(n-k)! \cdot n^k}$$

Applied to the birthday question, we can compute $P(365, 23) = 0.5073$, i.e., with already 23 people in the room there is a probability better than 0.5 that two persons share the same birthday.

The reason is that there are $\binom{23}{2} = \frac{23!}{2! \cdot (23-2)!} = 253$ different pairs one can construct from these 23 people.

Birthday Paradox *Generalization* (1)

To represent $P(n,k)$ in closed form we make use of the inequality $1 - x \le e^{-x}$, so we can now express:

$$P(n,k) > 1 - \left[e^{\frac{-1}{n}} \cdot e^{\frac{-2}{n}} \cdot \ldots \cdot e^{\frac{-(k-1)}{n}} \right] = 1 - e^{-\left[\frac{1}{n} + \frac{2}{n} + \ldots + \frac{k-1}{n}\right]} = 1 - e^{\frac{-k(k-1)}{2n}}$$

So, if we now want to know what value of k is required for $P(n,k) > 0.5$, we can solve

$$\frac{1}{2} = 1 - e^{\frac{-k(k-1)}{2n}}$$

$$2 = e^{\frac{k(k-1)}{2n}}$$

$$\ln(2) = \frac{k(k-1)}{2n}$$

Birthday Paradox *Generalization* (2)

For large k we can replace $k(k-1)$ with k^2 and thus

$$k = \sqrt{2(\ln 2)n} = 1.17\sqrt{n} \approx \sqrt{n}$$

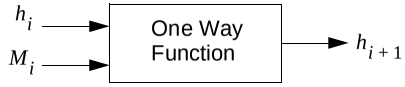
For hash functions we can thus state: suppose a function H with 2^m possible outputs. If H is applied to k random inputs, what must be the value of k so that the probability of a collision, i.e., $H(x) = H(y)$ for two random inputs x and y is greater than 0.5?

$$k = \sqrt{2^m} = 2^{m/2}$$

The consequence is that one should choose a hash-value that is twice as long as you might otherwise think you need. If the odds of breaking the system should be 1 in 2^{80} then use a 160 bit hash function.

Hash Functions

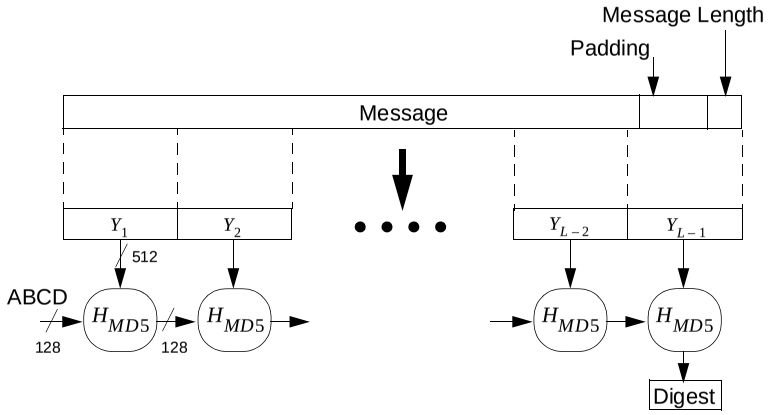
A hash function takes typically two inputs, the hash of the previous block of data and a new block of data.



There are a number of different hash functions, the most common ones are Message Digest 2 and 5 (*MD5, MD2)* and the Secure Hash Algorithm (*SHA*).

MD5 Message Digest Algorithm

MD5 takes as input a message of arbitrary length and produces a 128-bit message digest. Input is processed in 512-bit blocks.



MD5 Message Digest Algorithm

(1) The message is padded so that its bit length is $length \equiv 448 \bmod 512$. Messages are always padded with 1 to 512 bits. The padding consists of a single 1 bit followed by 0-bits.

(2) The length of the message (as a 64 -bit number) is appended to the result of step 1. If the message is longer than 2^{64} then the low order bits of the length are used.

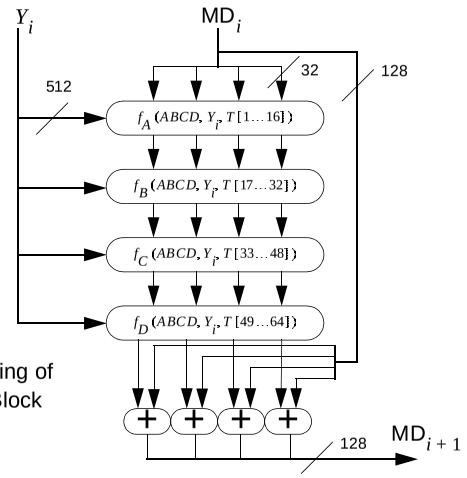
(3) The MD buffer is initialized. It consists of 128 bits represented as four 32-bit registers A, B, C, D. These registers are initialized as

```
A=0x01234567, B=0x89abcdef, C=0xfedcba98, D=0x76543210
```

(4) The message is processed in 512-bit, 16 32-bit word blocks. Processing occurs in four rounds each using a different primitive logical functions referred to as f_A, f_B, f_C and f_D

(5) Finally after all blocks have been processed, the output from the last stage is the 128-bit digest

MD5 Message Digest Algorithm



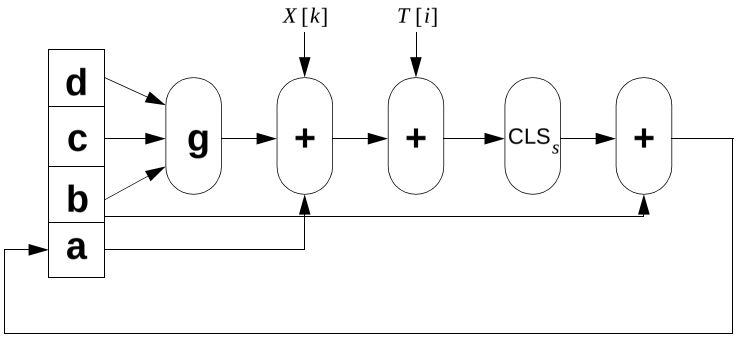
MD5 Processing of One 512-bit Block

MD5 Message Digest Algorithm

Processing in each round is a sequence of 16 operations operating on the buffer ABCD. The steps are of the form: $a \leftarrow b + \mathrm{CLS}_s(a + g(b, c, d) + X[k] + T[i])$.

a, b, c, d	the four words of buffer ABCD in a different order for each operation
g	one of the primitive functions F, G, H, I
CLS_s	left rotation of the 32-bit argument by s bits
$X[k]$	the k-th word of the i-th 512-bit block
$T[i]$	the i-th 32-bit word in array T, which contains for $1 \le i \le 64$ $T[i] = 2^{32} \cdot \mathrm{abs}(\sin i)$, where i is expressed in radians (requires 256 byte of storage, often already too much for smart cards)
+	addition modulo 2^{32}

MD5 Message Digest Algorithm



MD5 Message Digest Algorithm

Primitive Function g	$g(b,c,d)$
f_A	$(b \wedge c) \vee (\bar{b} \wedge d)$ if b then c else d
f_B	$(b \wedge d) \vee (c \wedge \bar{d})$ if d then b else c
f_C	$b \oplus c \oplus d$ parity bit
f_D	$c \oplus (b \wedge \bar{d})$

- For each operation the four registers ABCD are used in a different sequence: ABCD, DABC, CDAB, BCDA,...
- The input words, indexed with k are accessed in a different sequence for each round.
- The rotation count is different for every operation in every round.

MD5 Message Digest Algorithm *Strength*

- MD5 is an extension of MD4 through addition of a fourth round
- Differential cryptanalysis has successfully been applied to a single round of MD5
- A technique has been developed on how to find a message block x and two related digest values that will yield the same output state.
- Neither of these attacks have been successful on a full version of MD5
- Alternative, assumed to be more secure, hashes are MD2 and the *Secure Hash Algorithm* (SHA), which is a US federal standard and creates hashes of 160 bits.

Basic Uses of Hash Functions

Hash functions are typically used to protect the integrity of a message M.

(1) $A \rightarrow B \qquad E_K[M \| H(M)]$

- Confidentiality because only A and B share the key
- Authentication because hash is protected via encryption

(2) $A \rightarrow B \qquad M \| E_K[H(M)]$

- Authentication because hash is protected via encryption

(3) $A \rightarrow B \qquad M \| E_{KR_a}[H(M)]$

- Authentication and signature because hash is protected via public key encryption, and only A could have signed the hash

(4) $A \rightarrow B \qquad E_K[M \| E_{KR_a}[H(M)]]$

- Authentication and digital signature
- Confidentiality

Basic Uses of Hash Functions

(5) $A \rightarrow B \qquad M \,\|\, H(M \,\|\, S)$

- Authentication because only A and B share the secret S

(6) $A \rightarrow B \qquad E_K[M \,\|\, H(M \,\|\, S)]$

- Authentication because only A and B share the secret S
- Confidentiality because only A and B share the encryption key

Message Authentication Code *MAC*

Message Authentication Codes or ***Cryptographic Checksums*** are key-dependent one-way hash functions:

- Provide message authenticity without requiring secrecy
- Hash can only be verified by someone who knows the key
- Simplest realization is use of an encryption function in CBC or CFB mode
- The *Data Authentication Code* is an example for a CBC-based MAC:

$$C_1 = E_K(M_1 \oplus 0)$$
$$C_2 = E_K(C_1 \oplus M_2)$$
$$\ldots$$
$$C_n = E_K(C_{n-1} \oplus M_n)$$

C_n is the final output and is used either in its entirety or with its i leftmost bits $16 \le i \le 64$ as the message authenticator.

Message Authentication Code *MAC*

MACs can also be built based on one-way hash functions. Assuming Alice and Bob share a key K then Alice can concatenate K and M and compute the hash $H(K \,\|\, M)$. The problem with this approach is that Mallory can always add new blocks to the message and compute a valid MAC.

If the message length is put at the start of the message or the key at its end, then this attack can be avoided. Other examples for MAC computation are

$H(K_1, H(K_2, M))$

$H(K, H(K, M))$

$H(K, p, M, K)$ where p pads K to a full message block

Authentication Protocols *Introduction*

Authentication protocols enable the communicating parties to satisfy themselves (mutually) about each other's identity and optionally to exchange session keys to allow for a confidential exchange of messages.

We distinguish:

- **One-way authentication**
  where a user authenticates to a computer system, application or network
- **Mutual authentication**
  where two principals (users, applications, computer systems) authenticate each other

Central to any authentication protocol are two issues:

- **Confidentiality**
  to prevent masquerade and compromise of session keys
- **Timeliness**
  to protect against the threat of message replays. Such replays could allow an adversary to compromise a session key or impersonate another party.

The following types of replay attacks can be distinguished:

- **Simple replay**
  Opponent copies message and replays it later
- **Logable repetition**
  Opponent replays a timestamped message within valid time window
- **Undectable repetition**
  Opponent suppresses original message before replay
- **Backward replay without modification**
  Opponent returns message to sender who cannot, because of symmetric encryption, detect that this is his own message

To cope with replay attacks sequence numbers, timestamps, or challenge/response exchanges are used.

Timestamps

Alice accepts a message as ***fresh*** only if the message contains a timestamp which in Alice's judgement is close enough to the current time.

For timestamps to work, clocks throughout the system must be synchronized.

Challenge/Response

Alice expecting a ***fresh*** message from Bob first sends him a ***nonce*** (challenge) which Bob then must return in his subsequent message.

While no clocks need to be synchronized, challenge/response protocols require an extra handshake before actual communication may take place.

Key Distribution Using Symmetric Encryption

The following possibilities exists, if Alice and Bob want to communicate in secret with each other:

(1) Alice selects a key and delivers it out-of-band to Bob

(2) Trent selects a key and delivers it out-of-band to both Alice and Bob

(3) If Alice and Bob have recently used a secret key for communication with each other, Alice may encrypt the new key with this secret key and deliver it in-band to Bob

(4) If Alice and Bob each have a secret key they share with Trent, they can as Trent to deliver a secret key to each of them.

Options (1) and (2) are not really suitable for a distributed system, typically option (4) is implemented via a so-called ***Key Distribution Center*** (KDC)

Key Distribution Using Public-Key Encryption

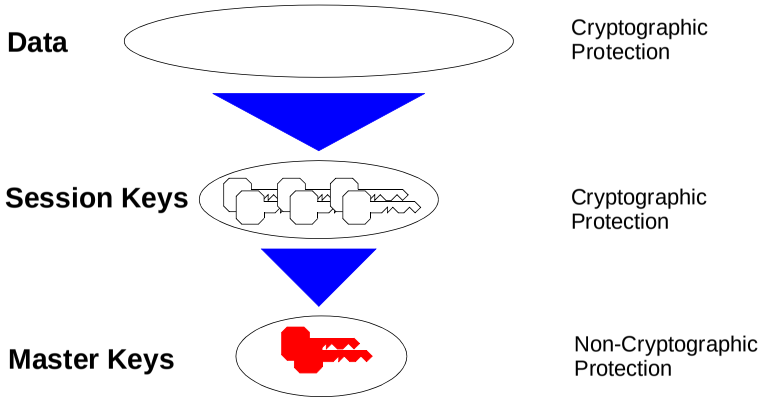
A major problem with the use of symmetric encryption in key distribution protocols is that scalability is more difficult to achieve than with the use of public-key encryption.

Key distribution based on public-key encryption relies on key certificates and/or key directories that can be trusted to only hand out verified keys.

Another advantage of public-key encryption is the fact that it allows electronic signatures which would be far more difficult to accomplish using symmetric encryption.

Authentication Protocols — *Key Distribution*

The efficient operation of a KDC depends on a hierarchy of keys. At least two levels of keys are used.



Authentication Protocols — *Key Hierarchy*

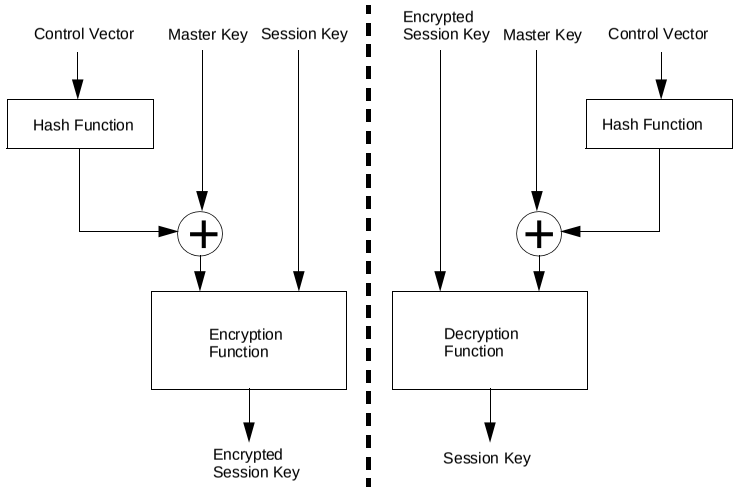
Hierarchical Key Control

For large systems a hierarchy of KDCs can be operated, the protocol then extends to these KDCs. Limits damage when lower level master keys become compromised.

Session Keys

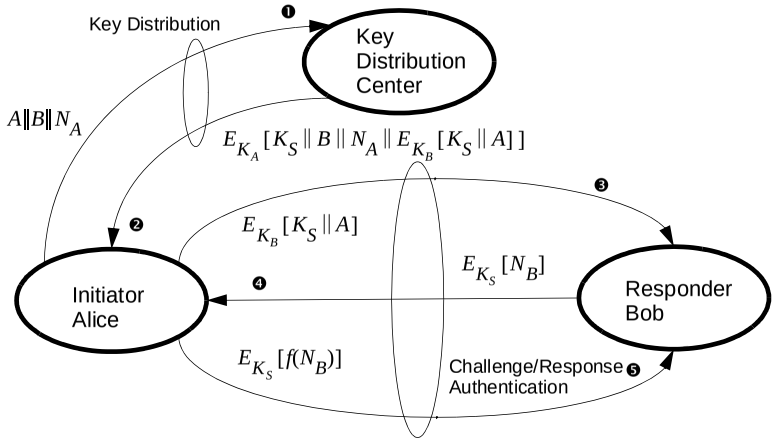
- Lifetime
  Typically the lifetime of session keys is limited, so the amount of accessible cipher text is limited too. Often, for connection-oriented protocols, a session key is only valid while the connection lasts.
- Usage
  Session keys are marked for their intended use, e.g.:
  - Data encrypting key
  - PIN encrypting key
  - File encrypting key

Authentication Protocols — *Key Usage*

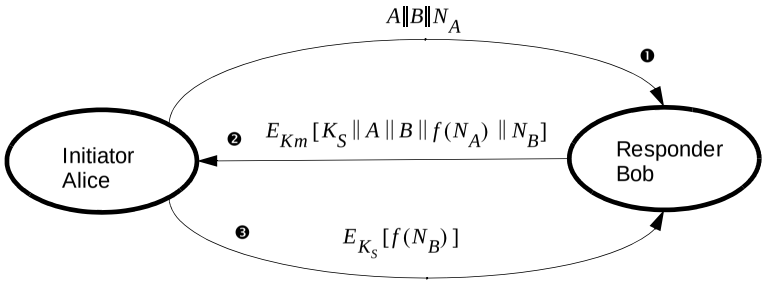


Authentication Protocols — *Key Distribution*

Needham-Schroeder Key Distribution Protocol, 1978



Alice and Bob Share a Common Master Key Km



er4.fm: 03.08.2000 16:02

Problem with the Needham-Schroeder Protocol

The problem centers around a compromised session key K_S

(1) Mallory could replay message 3 to Bob.
(2) Bob extracts the session key and sends his nonce encrypted under K_S
(3) Mallory intercepts message 4, decrypts it and returns Bob his nonce.
(4) Bob then verifies his nonce and now treats Mallory as if he were Alice.

er4.fm: 03.08.2000 16:02

Otway-Rees Authentication Protocol

Otway-Rees propose a protocol that is immune to replay attacks:

(1) $A \rightarrow B$ $\quad I \| A \| B \| E_{K_A}[N_A \| I \| A \| B]$

(2) $B \rightarrow KDC$
$I \| A \| B \| E_{K_A}[N_A \| I \| A \| B] \| E_{K_B}[N_B \| I \| A \| B]$

(3) $KDC \rightarrow B$ $\quad I \| E_{K_A}[N_A \| K_S] \| E_{K_B}[N_B \| K_S]$

(4) $B \rightarrow A$ $\quad I \| E_{K_A}[N_A \| K_S]$

Index may be used in a known plaintext attack.

er4.fm: 03.08.2000 16:02

Modification of the Needham-Schroeder Protocol

To fix the replay/suppress problem, Denning suggests 1981/82 a modification of the Needham-Schroeder protocol through adding a timestamp to step 2 and 3:

(1) $A \rightarrow KDC$ $\quad A \| B$

(2) $KDC \rightarrow A$ $\quad E_{K_A}[K_S \| B \| T \| E_{K_B}[K_S \| A \| T]]$

(3) $A \rightarrow B$ $\quad E_{K_B}[K_S \| A \| T]$

(4) $B \rightarrow A$ $\quad E_{K_S}[N_B]$

(5) $A \rightarrow B$ $\quad E_{K_S}[f(N_B)]$

The timestamp T serves to assure freshness of the session key generated by the KDC. Both Alice and Bob can check that $|Clock - T| < \Delta t_1 + \Delta t_2$, where Δt_1 is the clock skew between the KDC and Alice or Bob and Δt_2 is the network delay.

er4.fm: 03.08.2000 16:02

Neuman-Stubblebine Protocol (1)

While the Denning modification requires synchronized clocks, Neuman and Stubblebine suggest in 1993 an alternative protocol to protect against suppress/ replay attacks, without requiring synchronized clocks:

(1) $A \rightarrow B$ $\quad A \| N_A$

(2) $B \rightarrow KDC$ $\quad B \| N_B \| E_{K_B}[A \| N_A \| T_B]$

(3) $KDC \rightarrow A$ $\quad E_{K_A}[B \| N_A \| K_S] \| E_{K_B}[A \| K_S \| T_B] \| N_B$

(4) $A \rightarrow B$ $\quad E_{K_B}[A \| K_S \| T_B] \| E_{K_S}[N_B]$

The timestamp T_B is only relative to Bob's clock and can therefore be checked without requiring cooperation from others.

Neumann-Stubblebine Protocol (2)

The Neumann-Stubblebine protocol allows Alice to establish subsequent sessions with Bob without going back to the KDC, if this happens before the ticket Alice received from the KDC expires:

(1) $A \rightarrow B$ $\quad E_{K_B}[A \| K_S \| T_B] \| N_A'$

(2) $B \rightarrow A$ $\quad N_B' \| E_{K_S}[N_A']$

(3) $A \rightarrow B$ $\quad E_{K_S}[N_B']$

Distributed Authentication Security Service (DASS)

Developed by DEC, based on public-key encryption, incorporated in SPX product.

(1) $A \rightarrow DIR$ $\quad B$

(2) $DIR \rightarrow A$ $\quad E_{KR_{DIR}}[B, KU_B]$

(3) $A \rightarrow B$ $\quad E_{K_S}[T_A] \| E_{KR_A}[L \| A \| K_{KU_S}] \| E_{KR_S}[E_{KU_B}[K_S]]$

(4) $B \rightarrow DIR$ $\quad A$

(5) $DIR \rightarrow B$ $\quad E_{KR_{DIR}}[A, KU_A]$

At this stage Bob can verify that the message is from Alice and recover the public session key Alice generated in round 3. He uses this key to check the signature on the shared and encrypted session key K_S. Decrypting T_A allows Bob to check that the message is current. If mutual authentication is required, Bob can send:

(6) $B \rightarrow A$ $\quad E_{K_S}[T_B]$ and Alice can verify that the message is current.

Denning-Sacco Protocol

(1) $A \rightarrow DIR$ $\quad A \| B$

(2) $DIR \rightarrow A$ $\quad E_{KR_{DIR}}[B \| KU_B] \| E_{KR_{DIR}}[A \| KU_A]$

(3) $A \rightarrow B$

$$E_{KU_B}[E_{KR_A}[K_S \| T_A]] \| E_{KR_{DIR}}[B \| KU_B] \| E_{KR_{DIR}}[A \| KU_A]$$

This problem has a flaw, which allows Bob to masquerade as Alice in a communication with Carol:

(1) $B \rightarrow DIR$ $\quad B \| C$

(2) $DIR \rightarrow B$ $\quad E_{KR_{DIR}}[C \| KU_C] \| E_{KR_{DIR}}[B \| KU_B]$

(3) $B \rightarrow C$

$$E_{KU_C}[E_{KR_A}[K_S \| T_A]] \| E_{KR_{DIR}}[A \| KU_A] \| E_{KR_{DIR}}[C \| KU_C]$$

A fix to this flaw is to add the names inside the encrypted messages of step 3:

Woo-Lam Protocol

This protocol has the advantage over the Denning-Sacco protocol that it does not require synchronized clocks but rather can live with nonces.

(1) $A \rightarrow KDC \quad A \,||\, B$

(2) $KDC \rightarrow A \quad E_{KR_{KDC}}[B \,||\, KU_B]$

(3) $A \rightarrow B \quad E_{KU_B}[N_A \,||\, A]$

(4) $B \rightarrow KDC \quad B \,||\, A \,||\, E_{KU_{KDC}}[N_A]$

(5) $KDC \rightarrow B \quad E_{KR_{KDC}}[A \,||\, KU_A] \,||\, E_{KU_B}[E_{KR_{KDC}}[N_A \,||\, K_S \,||\, A \,||\, B]]$

(6) $B \rightarrow A \quad E_{KU_A}[E_{KR_{KDC}}[N_A \,||\, K_S \,||\, A \,||\, B] \,||\, N_B]$

(7) $A \rightarrow B \quad E_{K_S}[N_B]$

In its first version the protocol did not include the identity of Alice in message 5.

Kerberos is probably the best known authentication service, based on symmetric encryption, designed for distributed systems, where users at untrusted workstations can access services on servers distributed throughout an open network.

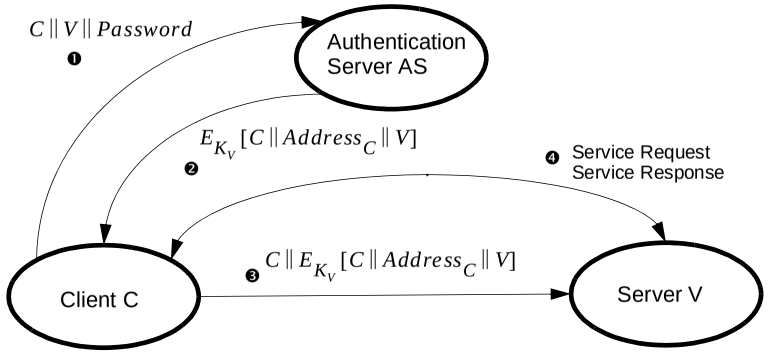
Kerberos is the security service in the OSF Distributed Computing Environment.

Requirements

- *Secure:* an eavesdropper should not be able to obtain the necessary information to impersonate a user.
- *Reliable:* as many services will rely on Kerberos as access control mechanism, Kerberos must be highly reliable and available to ensure continued access to these services
- *Transparent:* the user should only need to enter a *single* password to obtain network services, other than that he should not be aware of the underlying authentication protocols
- *Scalable:* the system should be scalable to support thousands of users and hundreds of servers, as such it must be follow a modular and distributed architecture

Kerberos Version 4 was the first released version, previous versions were internal test versions. Version 5 is the current version of the protocol correcting some of the deficiencies of the previous version.

A Simple Authentication Dialogue

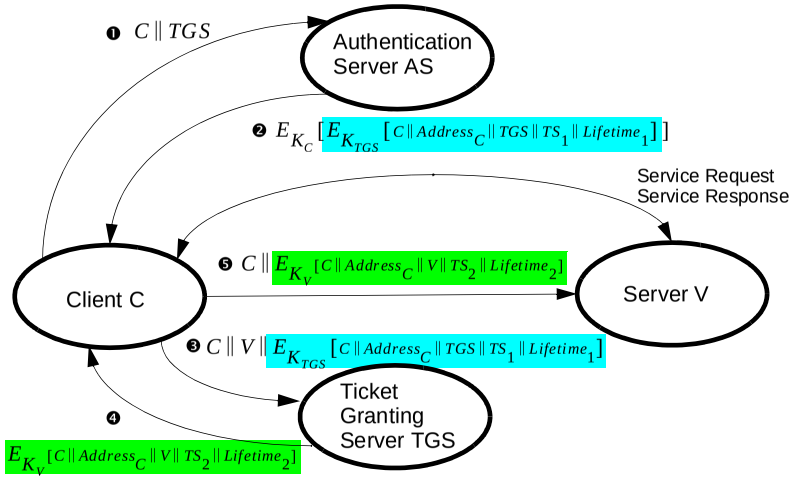


Problems with this Simple Protocol

- Password is exchanged in the clear
- If ticket can only be used once, the user would have to reenter his password again
- Ticket $E_{K_V}[C \,||\, Address_C \,||\, V]$ is only good for a particular server, if user wants to access a different service, he would have to go back to the authentication server again

The solution to these problems is to never transmit the password in the clear and to introduce a second server, the so-called ticket granting server (TGS), from which tickets for other services can be requested.

Authentication via Ticket Granting Server



Authentication via TGS

The ticket sent from the authentication server (*Ticket-Granting Ticket* TGT) is encrypted with a DES key derived from the user's password, thus it can only be recovered correctly if the user enters his password correctly at the workstation. The password never leaves the workstation.

Problems with the Solution

- The TGT has a limited lifetime to protect against an opponent from stealing the ticket and replaying it. The lifetime, however cannot be minutes only, otherwise the user would have to reenter his password too often.
- The service-granting ticket also has a limited lifetime, if an opponent were to capture it, he would be able to obtain services illegitimately.
- Servers don't authenticate themselves vis-a-vis the users, as such a fake server could be set up which captures user information that otherwise would not be divulged.

Kerberos V4 Protocol

To cope with the problems of the suggested protocol, each message from the client to the TGS and to individual servers bears a one-time use authenticator encrypted with a special session key. Thus stealing only the tickets will no longer work for an attacker.

Authentication Service Exchange: Obtain a TGT
Once per login session

(1) $C \rightarrow AS \qquad C \| TGS$

(2) $AS \rightarrow C \qquad E_{K_C}[K_{C,TGS} \| TGS \| TS_2 \| Lifetime_2 \| Ticket_{TGS}]$

$$Ticket_{TGS} = E_{K_{TGS}}[K_{C,TGS} \| C \| Address_C \| TGS \| TS_2 \| Lifetime_2]$$

Kerberos V4 Protocol (cont)

Ticket-Granting Service Exchange: Obtain a Service-Granting Ticket
Once per type of service

(3) $C \rightarrow TGS \qquad V \| Ticket_{TGS} \| E_{K_{C,TGS}}[C \| Address_C \| TS_3]$

(4) $TGS \rightarrow C \qquad E_{K_{C,TGS}}[K_{C,V} \| V \| TS_4 \| Lifetime_4 \| Ticket_V]$

$$Ticket_V = E_{K_V}[K_{C,V} \| C \| Address_C \| V \| TS_4 \| Lifetime_4]$$

Client/Server Authentication Exchange: Obtain Basic Service
Once per service session

(5) $C \rightarrow V \qquad Ticket_V \| E_{K_{C,V}}[C \| Address_C \| TS_5]$

(6) $V \rightarrow C \qquad E_{K_{C,V}}[TS_5 + 1]$

Kerberos Realms

Kerberos authentication server (AS) and ticket-granting server (TGS) must be physically protected because all the security centers around their correct operation.

The AS/TGS within a Kerberos realm must store:

(1) User identification and hashed password of all registered users
(2) Secret shared keys for all registered servers

To allow inter-realm authentication:

(3) a Kerberos AS/TGS in an interoperating realm must share a key with the AS/TGS in the other realm.

To obtain service in a foreign realm a user applies to his local TGS to obtain a ticket-granting ticket for the remote TGS. The client can then apply to the remote TGS for service-granting tickets in that TGS's realm.

For many realms, the number of shared keys between realms becomes unwieldy large as it grows $O(n^2)$, if full connectivity is desired.

Kerberos Version 5

Version 5 addresses a number limitations of the previous version.

Environmental Improvements

(1) lift dependency on DES encryption
(2) lift dependency on IP style addresses
(3) use ASN.1/BER for message encoding
(4) ticket lifetimes expressed as interval instead of maximum lifetime
(5) authentication forwarding: a server can act on behalf of a client
(6) improved inter-realm authentication

Technical Improvements

(1) Remove double encryption in messages 2 and 4
(2) Hardened encryption: CBC instead of *plain-and-cipher block chaining*
(3) Renegotiation of sub-session keys between clients and servers
(4) Improved protection against password guessing attacks through preauthentication

Kerberos V5 Protocol (1)

Authentication Service Exchange: Obtain a TGT
Once per login session

(1) $C \rightarrow AS \qquad Options \| C \| Realm_C \| TGS \| Times \| N_1$

(2) $AS \rightarrow C$

$Realm_C \| C \| Ticket_{TGS} \| E_{K_C}[K_{C,TGS} \| Times \| N_1 \| Realm_{TGS} \| TGS]$

$Ticket_{TGS} = E_{K_{TGS}}[Flags \| K_{C,TGS} \| Realm_C \| C \| Address_C \| Times]$

Kerberos V5 Protocol (2)

Ticket-Granting Service Exchange: Obtain a Service-Granting Ticket
Once per type of service

(3) $C \rightarrow TGS$

$Options \| C \| Times \| V \| N_2 \| Ticket_{TGS} \| E_{K_{C,TGS}}[C \| Realm_C \| TS_1]$

(4) $TGS \rightarrow C$

$Realm_C \| C \| Ticket_V \| E_{K_{C,TGS}}[K_{C,V} \| Times \| N_2 \| Realm_V \| V]$

$Ticket_V = E_{K_V}[Flags \| K_{C,V} \| Realm_C \| C \| Address_C \| Times]$

Kerberos V5 Protocol (3)

Client/Server Authentication Exchange: Obtain Basic Service
Once per service session

(5) $C \rightarrow V$
$Options \| Ticket_V \| E_{K_{C,V}}[C \| Realm_C \| Address_C \| TS_5 \| Subkey \| Seq\#]$

(6) $V \rightarrow C$ $E_{K_{C,V}}[TS_5 \| Subkey \| Seq\#]$

$Realm$	the user's realm
$Times$	used by the client to request time settings in the ticket: from, to, and renew until time
$Subkey$	client's choice of an encryption key for use in the client/server session, if this field is omitted $K_{C,V}$ is used instead
$Seq\#$	optional to specify a numbered message exchange to protect from message replay

Kerberos V5 Protocol (4)

Options and Flags

A client may request in the options certain flags to be set in the returned tickets.

INITIAL	The ticket was issued using the AS protocol and not based on a TGT
PRE-AUTHENT	During initial authentication, the client was authenticated by the AS before a ticket was issued
HW-AUTHENT	The protocol for initial authentication requires the user to use a hardware device (smart card)
RENEWABLE	Tells the TGS that the ticket can be used to acquire a replacement ticket that expires at a later date
MAY-POSTDATE	Tells the TGS that a post-dated ticket may be issued based on this TGT
POSTDATE	Indicates that the ticket has been postdated
INVALID	The ticket is invalid and must be validated by the TGS before use
PROXIABLE	Tells the TGS that a new service-granting ticket with a different network address may be issued for the presented TGT
PROXY	Indicates that the ticket is a proxy
FORWARDABLE	Tells TGS that a new TGT with a different network address may be issued for the presented TGT
FORWARDED	Indicates that this ticket has either been forwarded or was issued based on authentication involving a forwarding TGT

Protocol Verification

For the verification of authentication protocols and key-exchange protocols a number of different approaches exist:

(1) Model and verify the protocol using specification languages and verification tools
(2) Develop expert systems to investigate different scenarios
(3) Model the requirements of a protocol family using a logic for the analysis of knowledge and belief
(4) Formal methods based on algebraic term-rewriting properties of cryptographic systems

The third approach has gained the widest acceptance. Two popular such logics are:

- Burrows, Abadi, and Needham (BAN)
- Gong, Needham, and Yahalom (GNY)

BAN was presented first in 1989, GNY which extends BAN in 1990.

BAN Logic of Authentication

The BAN logic tries to answer the following questions:

- Does the protocol work?
- Can it be made to work?
- What does the protocol achieve?
- What assumptions does the protocol build on?
- Does the protocol do anything unnecessary?

Symbols

P, Q	principals, specifically instantiated as A, B, S
K_{ab}, K_{as}, K_{bs}	shared key
$K_a, K_b, K_s, K_a^{-1}, K_b^{-1}, K_s^{-1}$	public and corresponding private keys
N_a, N_b, N_c	nonces or statements

Constructs (1)

P **believes** X	P *believes* X, he may act as if X were true
P **sees** X	P *sees* X, somebody has sent a message containing X to P
P **said** X	P *once said* X, he at some time sent a message which included X. It is not known when this message was sent, but it is known that P believed X then

er4.fm: 03.08.2000 16:02

Constructs (2)

P **controls** X	P has *jurisdiction over* X, the principal P is an authority on X and should be trusted on this matter, e.g., a server will generate encryption keys correctly
fresh (X)	The formula X is fresh, i.e., it has not been sent at any time before the current run of the protocol.
$P \overset{K}{\leftrightarrow} Q$	P and Q may use the shared key K to communicate. The key is good, that it will never be discovered by any principal except P or Q or a principal authorized by them
$\overset{K}{\mapsto} P$	P has the *public key* K. The matching secret key K^{-1} will never be discovered by any other principal except P or one trusted by P

er4.fm: 03.08.2000 16:02

Constructs (3)

$P \overset{X}{\rightleftharpoons} Q$	P and Q share the secret X, it is only known to them or possibly other principals trusted by them. Using X they may prove their identities to each other
$\{X\}_K$	is the formula X encrypted under K
$[X]_Y$	is the formula X combined with secret Y, its presence proves the identity of the sender of X

er4.fm: 03.08.2000 16:02

Logic Postulates (1)

(1) Message Meaning Rule

- shared keys (P must be able to recognize his own messages)

$$\frac{P \textbf{ believes } Q \overset{K}{\leftrightarrow} P,\ P \textbf{ sees } \{X\}_K}{P \textbf{ believes } Q \textbf{ said } X}$$

- public keys $\dfrac{P \textbf{ believes } \overset{K}{\mapsto} Q,\ P \textbf{ sees } \{X\}_{K^{-1}}}{P \textbf{ believes } Q \textbf{ said } X}$
- shared secrets (again P must be able to recognize his own messages)

$$\frac{P \textbf{ believes } Q \overset{K}{\rightleftharpoons} P,\ P \textbf{ sees } [X]_K}{P \textbf{ believes } Q \textbf{ said } X}$$

er4.fm: 03.08.2000 16:02

Logic Postulates (2)

(2) Nonce-Verification Rule

$$\frac{P \textbf{ believes fresh } (X), P \textbf{ believes } Q \textbf{ said } X}{P \textbf{ believes } Q \textbf{ believes } X}$$

The receiver of a message can verify that the message is recent and belongs to the current protocol run.

(3) Jurisdiction Rule

$$\frac{P \textbf{ believes } Q \textbf{ controls } X, P \textbf{ believes } Q \textbf{ believes } X}{P \textbf{ believes } X}$$

If P believes that Q has jurisdiction over X, then P trusts Q on the truth of X

Logic Postulates (3)

(4) Conjunction and Decryption Rule

$$\frac{P \textbf{ sees } (X, Y)}{P \textbf{ sees } X} \qquad \frac{P \textbf{ sees } [X]_Y}{P \textbf{ sees } X} \qquad \frac{P \textbf{ believes } Q \overset{K}{\leftrightarrow} P, P \textbf{ sees } \{X\}_K}{P \textbf{ sees } X}$$

$$\frac{P \textbf{ believes } \overset{K}{\mapsto} Q, P \textbf{ sees } \{X\}_{K^{-1}}}{P \textbf{ sees } X} \qquad \frac{P \textbf{ believes } Q \overset{K}{\rightleftharpoons} P, P \textbf{ sees } [X]_K}{P \textbf{ sees } X}$$

If a principal sees a formula he also sees its components, likewise, if he knows the key he can also decrypt the received message

(5) Freshness Rule

$$\frac{P \textbf{ believes fresh } (X)}{P \textbf{ believes fresh } (X, Y)}$$

If one part of a message is fresh then the entire message must also be fresh

Protocol Analysis

Analyzed authentication protocols are annotated with logical formulas similar to a proof using Hoare's program verification logic. Two main rules guide us in deriving legal annotations:

(1) If X holds before the message $P \rightarrow Q$**:** Y then both X and Q **sees** Y hold afterwards

(2) If Y can be derived from X by the logical postulates, then Y holds whenever X holds

The annotation is similar to a sequence of comments about the beliefs of the principals and what they see in the course of the authentication protocol.

At the end of the protocol analysis, we want to be able to say that some formulas hold as conclusions.

For example for a key exchange protocol:

$$A \textbf{ believes } A \overset{K}{\leftrightarrow} B \qquad B \textbf{ believes } A \overset{K}{\leftrightarrow} B$$

or even stronger:

$$A \textbf{ believes } B \textbf{ believes } A \overset{K}{\leftrightarrow} B \qquad B \textbf{ believes } A \textbf{ believes } A \overset{K}{\leftrightarrow} B$$

Kerberos Protocol

Goal

Authenticate two principals with each other and establish a shared session key.

Protocol

(1) $A \rightarrow AS \qquad A, B$

(2) $AS \rightarrow A \qquad \{T_s, L, K_{ab}, B, \{T_s, L, K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$

The authentication server returns an encrypted message containing the session key for the two parties, a timestamp, a lifetime, and a ticket that only B can read.

(3) $A \rightarrow B \qquad \{T_s, L, K_{ab}, A\}_{K_{bs}}, \{A, T_a\}_{K_{ab}}$

(4) $B \rightarrow A \qquad \{T_a + 1\}_{K_{ab}}$

Kerberos Protocol

Idealized Protocol

(2) $AS \rightarrow A \qquad \left\{T_s, A \overset{K_{ab}}{\leftrightarrow} B, \left\{T_s, A \overset{K_{ab}}{\leftrightarrow} B\right\}_{K_{bs}}\right\}_{K_{as}}$

(3) $A \rightarrow B \qquad \left\{T_s, A \overset{K_{ab}}{\leftrightarrow} B\right\}_{K_{bs}}, \left\{T_a, A \overset{K_{ab}}{\leftrightarrow} B\right\}_{K_{ab}}$

(4) $B \rightarrow A \qquad \left\{T_a, A \overset{K_{ab}}{\leftrightarrow} B\right\}_{K_{ab}}$

Kerberos Protocol

Assumptions

A **believes** $A \overset{K_{as}}{\leftrightarrow} S$

S **believes** $A \overset{K_{as}}{\leftrightarrow} S$

S **believes** $A \overset{K_{ab}}{\leftrightarrow} B$

A **believes** $\left(S \textbf{ controls } A \overset{K}{\leftrightarrow} B\right)$

A **believes fresh** (T_s)

B **believes** $B \overset{K_{bs}}{\leftrightarrow} S$

S **believes** $B \overset{K_{bs}}{\leftrightarrow} S$

B **believes** $\left(S \textbf{ controls } A \overset{K}{\leftrightarrow} B\right)$

B **believes fresh** (T_s)

B **believes fresh** (T_a)

Kerberos Protocol

Protocol Analysis (1)

A receives message 2, thus: A **sees** $\left\{T_s, A \overset{K_{ab}}{\leftrightarrow} B, \left\{T_s, A \overset{K_{ab}}{\leftrightarrow} B\right\}_{K_{bs}}\right\}_{K_{as}}$

Combined with the assumption A **believes** $A \overset{K_{as}}{\leftrightarrow} S$, we apply the message meaning rule for shared keys which yields A **believes** S **said** $\left(T_s, A \overset{K_{ab}}{\leftrightarrow} B, \left\{T_s, A \overset{K_{ab}}{\leftrightarrow} B\right\}_{K_{bs}}\right)$.

Applying the conjunction rule yields A **believes** S **said** $\left(T_s, A \overset{K_{ab}}{\leftrightarrow} B\right)$. Together with the hypothesis A **believes fresh** (T_s), we can apply the nonce-verification rule.

Kerberos Protocol

Protocol Analysis (2)

A **believes** S **believes** $\left(T_s, A \overset{K_{ab}}{\leftrightarrow} B \right)$, if we break the conjunction and instantiate the assumption about S A **believes** $\left(S \textbf{ controls } A \overset{K_{ab}}{\leftrightarrow} B \right)$ lets us derive the jurisdiction rule: A **believes** $A \overset{K_{ab}}{\leftrightarrow} B$. When A passes the ticket with his nonce to B, B can initially only decrypt the ticket, so B **believes** $A \overset{K_{ab}}{\leftrightarrow} B$. Knowing the new key K_{ab} allows B to decrypt the rest of the message B **believes** A **believes** $A \overset{K_{ab}}{\leftrightarrow} B$.

er4.fm: 03.08.2000 16:02

Kerberos Protocol

Protocol Analysis (3)

The fourth message assures A that B believes the key and has received A's last message. After applying again the message-meaning and nonce-verification rules yields the final result:

A **believes** $A \overset{K_{ab}}{\leftrightarrow} B$ and B **believes** $A \overset{K_{ab}}{\leftrightarrow} B$, as well as

B **believes** A **believes** $A \overset{K_{ab}}{\leftrightarrow} B$ and A **believes** B **believes** $A \overset{K_{ab}}{\leftrightarrow} B$.

If only the first three messages were sent, we would not obtain the last assertion. A would thus no be convinced of B's existence, he could observe the same messages regardless if B were running or not. We note the dependency on clock synchronization to verify freshness and observe that S's encryption of the ticket in message 2 is unnecessary and adds overhead.

er4.fm: 03.08.2000 16:02

Needham-Schroeder Public-Key Protocol

Goal

Allow two principals A and B to exchange two secret numbers using a server S as certification authority handing out the public keys.

Protocol

(1) $A \rightarrow S$: A, B

(2) $S \rightarrow A$: $\{K_b, B\}_{K_s^{-1}}$

(3) $A \rightarrow B$: $\{N_a, A\}_{K_b}$

(4) $B \rightarrow S$: B, A

(5) $S \rightarrow B$: $\{K_a, A\}_{K_s^{-1}}$

(6) $B \rightarrow A$: $\{N_a, N_b\}_{K_a}$

(7) $A \rightarrow B$: $\{N_b\}_{K_b}$

The protocol expects that initially A and B hold S's public key K_s. Therefore both principals can get the other's public key from S. This is done in messages 1, 2, 4, and 5. In messages 3, 6, and 7, the principals communicate the secret nonces to be used later for signing further messages.

er4.fm: 03.08.2000 16:02

Needham-Schroeder Public-Key Protocol

Idealized Protocol

For protocol analysis, we can idealize the protocol by leaving out messages that do not contribute to the logical properties of the protocol.

(2) $S \rightarrow A$: $\left\{ K_b \overset{}{\mapsto} B \right\}_{K_s^{-1}}$

(3) $A \rightarrow B$: $\{N_a\}_{K_b}$

(5) $S \rightarrow B$: $\left\{ K_a \overset{}{\mapsto} A \right\}_{K_s^{-1}}$

(6) $B \rightarrow A$: $\left\{ \left[A \overset{N_b}{\rightleftharpoons} B \right]_{N_a} \right\}_{K_a}$

(7) $A \rightarrow B$: $\left\{ \left[A \overset{N_a}{\rightleftharpoons} B \right]_{N_b} \right\}_{K_b}$

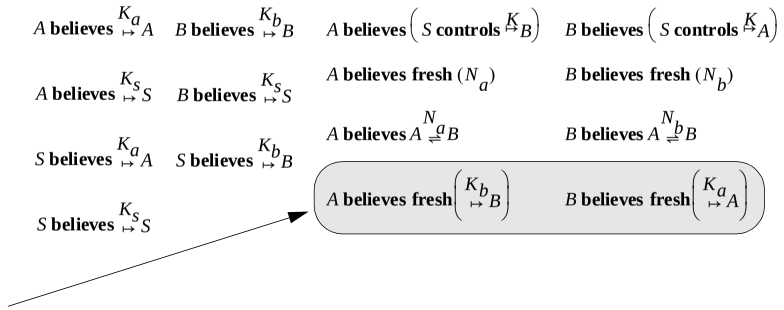
- Message 1 and 4 can be omitted as they do not contribute to the protocol
- In message 3, N_a is still unknown to B and thus not used to prove the identity of A
- In messages 6 and 7, however, N_a and N_b are used as secrets

er4.fm: 03.08.2000 16:02

Needham-Schroeder Public-Key Protocol

Protocol Analysis

We assume the partners hold initially the following beliefs:



These two assumptions are problematic as they represent a weakness of the protocol. Without timestamps, the principals **cannot** verify the freshness of the other's public key.

Needham-Schroeder Public-Key Protocol

Result of the Protocol Analysis

Carrying out the protocol analysis leads to the following final beliefs:

A believes $\overset{K_b}{\mapsto} B$ — B believes $\overset{K_a}{\mapsto} A$

A believes B believes $A \overset{N_b}{\rightleftharpoons} B$ — B believes A believes $A \overset{N_a}{\rightleftharpoons} B$

Each principal knows the public key of the other and has knowledge of a shared secret that he believes the other will accept as being shared only by the two principals. Now the two principals can exchange messages using the nonces and public-key encryption to transfer data and other keys securely.

Secure Andrew RPC Handshake

Goal

Do a handshake between a client A and a server B to provide a new session key K'_{ab} when both principals already share a key K_{ab}

Protocol

(1) $A \rightarrow B$ $\quad A, \{N_a\}_{K_{ab}}$

(2) $B \rightarrow A$ $\quad \{N_a + 1, N_b\}_{K_{ab}}$

(3) $A \rightarrow B$ $\quad \{N_b + 1\}_{K_{ab}}$

(4) $B \rightarrow A$ $\quad \{K'_{ab}, N'_b\}_{K_{ab}}$

N'_b is the initial sequence number used in subsequent communications.

Secure Andrew RPC Handshake

Idealized Protocol

(1) $A \rightarrow B$ $\quad \{N_a\}_{K_{ab}}$

(2) $B \rightarrow A$ $\quad \{N_a, N_b\}_{K_{ab}}$

(3) $A \rightarrow B$ $\quad \{N_b\}_{K_{ab}}$

(4) $B \rightarrow A$ $\quad \left\{A \overset{K'_{ab}}{\longleftrightarrow} B, N'_b\right\}_{K_{ab}}$

Secure Andrew RPC Handshake

Assumptions

A **believes** $A \overset{K_{ab}}{\leftrightarrow} B$

A **believes** B **controls** $A \overset{K}{\leftrightarrow} B$

A **believes fresh** (N_a)

B **believes fresh** (N_b)

B **believes** $A \overset{K_{ab}}{\leftrightarrow} B$

B **believes** $A \overset{K'_{ab}}{\leftrightarrow} B$

B **believes fresh** (N'_b)

Results

B **believes** $A \overset{K'_{ab}}{\leftrightarrow} B$

B **believes** A **believes** N_a

A **believes** B **said** $\left(A \overset{K'_{ab}}{\leftrightarrow} B, N'_b\right)$

A **believes** B **believes** (N_a, N_b)

er4.fm: 03.08.2000 16:02

Secure Andrew RPC Handshake

Solution

The problem can be solved if B were to add the nonce N_a to the last message.

Changing the protocol even more can reduce the amount of encryption needed:

(2) $B \rightarrow A$ $\left\{N_a, A \overset{K'_{ab}}{\leftrightarrow} B\right\}_{K_{ab}}$

(3) $A \rightarrow B$ $\{N_a\}_{K'_{ab}}$

B would thus know this message is good because K'_{ab} is fresh.

Analysis

A **believes** $A \overset{K'_{ab}}{\leftrightarrow} B$

A **believes** B **believes** $A \overset{K'_{ab}}{\leftrightarrow} B$

B **believes** $A \overset{K'_{ab}}{\leftrightarrow} B$

B **believes** A **believes** $A \overset{K'_{ab}}{\leftrightarrow} B$

er4.fm: 03.08.2000 16:02

Secure Andrew RPC Handshake

Modified and Improved Protocol

The modified protocol yields thus a much stronger result. Its concrete representation is then:

(1) $A \rightarrow B$ A, N_a

(2) $B \rightarrow A$ $\{N_a, K'_{ab}\}_{K_{ab}}$

(3) $A \rightarrow B$ $\{N_a\}_{K'_{ab}}$

(4) $B \rightarrow A$ N'_b

er4.fm: 03.08.2000 16:02

CCITT X.509 Protocol

Goals

Provide signed secure communication between two principals where each knows the other's public key.

Protocol

(1) $A \rightarrow B$ $A, \{T_a, N_a, B, X_a, \{Y_a\}_{K_b}\}_{K_a^{-1}}$

(2) $B \rightarrow A$ $B, \{T_b, N_b, A, N_a, X_b, \{Y_b\}_{K_a}\}_{K_b^{-1}}$

(3) $A \rightarrow B$ $A, \{N_b\}_{K_a^{-1}}$

X and Y are user data. The protocols ensures the non-repudiation, integrity and authenticity of X and the confidentiality of Y to either partner.

er4.fm: 03.08.2000 16:02

CCITT X.509 Protocol

Idealized Protocol

(1) $A \rightarrow B \qquad \{T_a, N_a, X_a, \{Y_a\}_{K_b}\}_{K_a^{-1}}$

(2) $B \rightarrow A \qquad \{T_b, N_b, N_a, X_b, \{Y_b\}_{K_a}\}_{K_b^{-1}}$

(3) $A \rightarrow B \qquad \{N_b\}_{K_a^{-1}}$

Assumptions

A **believes** $\overset{K_a}{\mapsto} A$ $\qquad$ A **believes fresh** (N_a)

A **believes** $\overset{K_b}{\mapsto} B$ $\qquad$ A **believes fresh** (T_b)

B **believes** $\overset{K_b}{\mapsto} B$ $\qquad$ B **believes fresh** (N_b)

B **believes** $\overset{K_a}{\mapsto} A$ $\qquad$ B **believes fresh** (T_a)

CCITT X.509 Protocol

Analysis (1)

The analysis yields a rather weak result, namely:

A **believes** B **believes** X_b and B **believes** A **believes** X_a

In particular we do not obtain:

A **believes** B **believes** Y_b nor B **believes** A **believes** Y_a

While Y has been transferred in signed messages, neither sender is actually aware of the data he sent in the private part of the message. This corresponds to a scenario, where some third party intercepts a message and removes the signature while adding his own, blindly copying the encrypted section within the signed message. The simple fix for this problem is to sign the secret data before it is encrypted for privacy.

In the second message either T_b or N_a can be used to ensure timeliness of the message.

CCITT X.509 Protocol

Analysis (2)

The draft recommendation for the standard also suggests that T_a need not be checked. This can cause a problem, because it only ensures timeliness of the first message. An intruder could thus perform the following message exchanges:

(1) $C \rightarrow B \qquad A, \{T_a, N_a, B, X_a, \{Y_a\}_{K_b}\}_{K_a^{-1}}$ where C sends an old message he had intercepted from A

(2) $B \rightarrow C \qquad B, \{T_b, N_b, A, N_a, X_b, \{Y_b\}_{K_a}\}_{K_b^{-1}}$ is a normal reply.

C somehow causes A to initiate authentication with C

(3) $A \rightarrow C \qquad A, \{T'_a, N'_a, C, X'_a, \{Y'_a\}_{K_c}\}_{K_a^{-1}}$

(4) $C \rightarrow A \qquad C, \{T_c, N_b, A, N'_a, X_c, \{Y_c\}_{K_a}\}_{K_c^{-1}}$

C supplied B's nonce N_b which is not secret

CCITT X.509 Protocol

Analysis (3)

(5) $A \rightarrow C \qquad A, \{N_b\}_{K_a^{-1}}$

A has now replied to C having signed the message such than C can now use it to impersonate A when conversing with B.

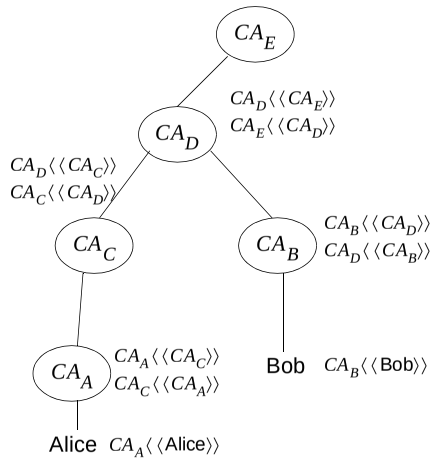
The solution to this problem is to include the recipient's name in the last message, which assures the creator of the nonces that the message is in fact linked to the current instance of the protocol.

X.509 Certificates

- Part of the ISO authentication framework
- Authentication and key exchange across networks
- Initially issued in 1988, revised in 1993
- Defines public key certificates
- Each user has a distinct name
- A trusted **Certification Authority** (CA) assigns a unique name to each user
- CA issues a signed certificate for the user with his name and public key
- CA's are arranged in tree like structures where CAs certify each other
- Certificates have a limited lifetime
- Certificates can be revoked before they expire, therefore each CA must also maintain a signed list of revoked certificates it had previously issued

X.509 Certificates

Version
Serial Number
Algorithm Id: - Algorithm - Parameters
Issuer
Validity - Not Before Date - Not After Date
Subject
Subject's Public Key - Algorithm - Parameters - Public Key
Signature



X.509 Certificates *Certification Hierarchy*

- Alice's certificate is signed by CA_A
- Bob's certificate is signed by CA_B
- Alice knows CA_A's public key
- CA_C has a certificate signed by CA_A
- Alice can thus verify CA_C's public key
- CA_D has a certificate signed by CA_C
- CA_B has a certificate signed by CA_D

By moving up the certificate tree to a common point, here CA_D, and then down to Bob, Alice can verify Bob's certificate and vice-versa.

X.509 Authentication Procedures

Three different protocols:

(1) One-way authentication
- Ensure identity of sender
- Ensure that message was for recipient
- Ensure integrity and originality of message

$A \rightarrow B \qquad E_{KR_A}[T_A \| N_A \| B \| E_{KU_B}[K_{AB}]]$

(2) Two-way authentication
- Ensure identity of receiver and that the reply was sent by receiver
- Ensure that the reply was received by sender
- Ensure integrity and originality of reply

$B \rightarrow A \quad E_{KR_B}[T_B \| N_B \| A \| N_A \| E_{KU_A}[K_{AB}]]$

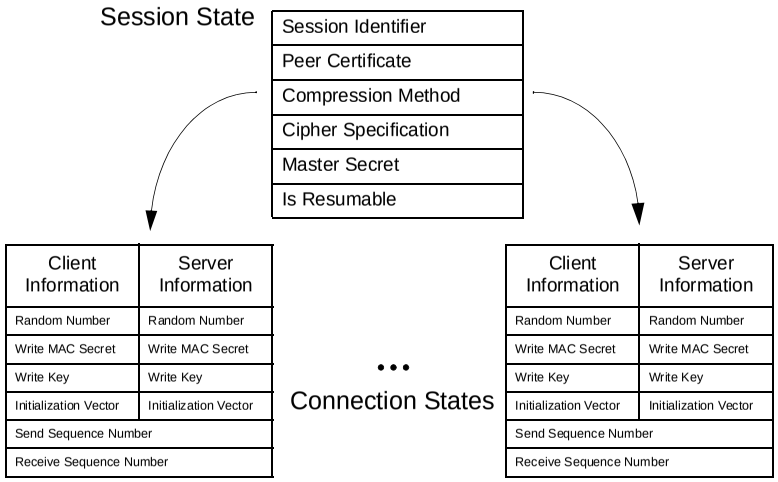
(3) Three-way authentication (avoids synchronized clocks)

$A \rightarrow B \quad E_{KR_A}[N_B]$

Secure Socket Layer (SSL) Version 3.0

- Internet Draft for a security protocol, authored and implemented by Netscape
- Competitor is Microsoft's *Private Communication Technology* (PCT) Protocol
- Both define a framework into which cryptographic algorithms can be placed to provide for authenticated and confidential message exchange
- Layered on top of a reliable transport protocol such as TCP
- SSL itself consists of two layers:
  (1) SSL Record Protocol for encapsulation of higher level protocols
  (2) SSL Handshake Protocol for mutual authentication and negotiation of encryption algorithms and cryptographic keys
- SSL provides connection security
  (1) Connection is private, encryption is used after an initial handshake to define a secret key. Symmetric cryptography is used for data encryption (e.g., DES, RC4, IDEA, ...)
  (2) The peer's identity can be authenticated using asymmetric cryptography (e.g., RSA, DSS, Diffie-Hellman)
  (3) The connection is reliable, message transport includes a message integrity check via a keyed MAC based on secure hash functions

SSL Details *Sessions/Connections*



SSL Details *Record Layer*

- The record layer fragments information blocks into records of up to 16KB.
- Client message boundaries are not preserved in the record layer. It may coalesce multiple client messages of the same type into a single SSL record.
- All messages sent via the record layer are first compressed and then encrypted using the compression method and cipher specification found in the session state.
- The cipher specification not only specifies the encryption method but also contains a MAC specification to be used to guarantee message integrity.
- Initially compression method and cipher specification are set to null.

```
enum { change_cipher_spec, alert, handshake,
       application_data } ContentType;

struct {
       ContentType           type;
       ProtocolVersion       version;      // 3.0
       uint16                length;
       opaque                fragment[length];
} SSLRecord;
```


SSL Details *Cipher Messages and MACs*

- SSL distinguishes generic stream and CBC-based block ciphers
- For block ciphers the initialization vector obtained in the initial handshake is used
- In either case the MAC is computed before encryption

```
stream-ciphered struct {
        opaque content[length];
        opaque    MAC[hash_size];
} GenericStreamCipher;
```

```
block-ciphered struct {
        opaque content[length];
        opaque    MAC[hash_size];
        uint8     padding[padding_length];
        unit8     padding_length;
} GenericBlockCipher;
```

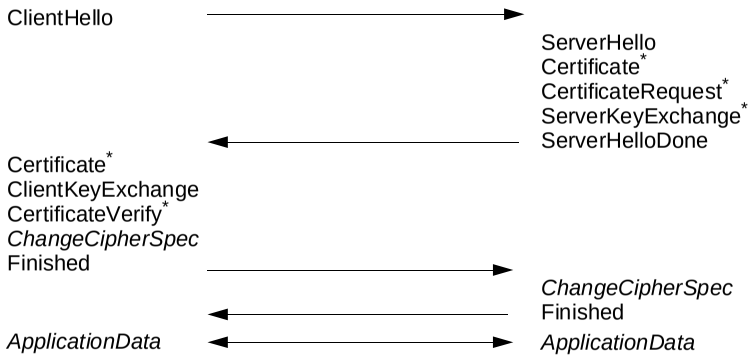
- The MAC is computed according to the following scheme:

```
hash(Write_MAC_Secret+pad_2+
        hash(Write_MAC_Secret+pad_1+
                Sequence_Number+length+content))
```

- *pad_1* the character 0x36 repeated 48 times for MD5 or 40 times for SHA
  pad_2 the character 0x5c repeated like *pad_1*
  hash the hash algorithm(s) for this cipher specification
  Sequence_Number the sequence number for this message

SSL Details — *Handshake Protocol*

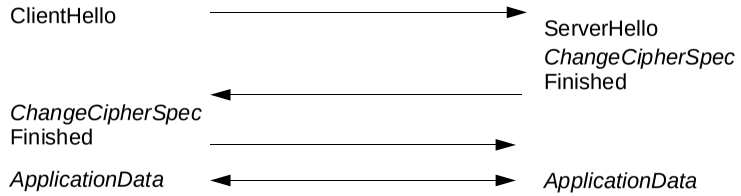
Before starting actual data transfer the client and server execute a handshake protocol in which they negotiate the cipher specification, exchange a secret key and, optionally, mutually authenticate each other.



* indicates optional messages depending on the cipher specification

SSL Details — *Handshake Protocol*

When the client and server decide to resume a previous session or duplicate the current session parameters on a new connection:



The client sends in his *Hello* message the previously used session identifier, if the server still has the session identifier in his cache and is willing to reuse these session parameters, the above protocol is executed, otherwise the server generates a new session identifier and the full handshake is executed.

SSL Details — *Hello Messages*

Client and server hello messages are very similar, except that the client offers a list of supported compression algorithms and cipher specifications - each of which of course may only contain one element - whereas the server responds with a single compression and cipher specification.

```
struct {
   ProtocolVersion    version;
   Random             random;
   SessionID          session_id;
   CipherSuite        cipher_suites<2..2^16-1>;
   CompressionMethod  compression_methods<1..2^8-1>;
} ClientHello;
struct {
   ProtocolVersion    version;
   Random             random;
   SessionID          session_id;
   CipherSuite        cipher_suite;
   CompressionMethod  compression_method;
} ServerHello;
```

The cipher suites are encoded in two bytes specifying encryption algorithm and MAC. The session identifier is a variable length vector of up to 32 bytes and is assigned by the server.

SSL Details — *Hello Messages*

The *random* structure is used to salt the generation of the shared master secret.

```
struct {
       uint32         gmt_unix_time;
       opaque         random_bytes[28];
} Random;
```

The time is derived from the internal clock in standard Unix 32 bit format. Clocks do not need to be synchronized for the SSL protocol.

The 28 random bytes must be generated by a cryptographically secure random number generator.

SSL Details *Certificate Messages*

The certificate messages are identical between client and server, in general X.509.v3 certificates are exchanged.

Usually the server will be expected to follow its hello message with a certificate for the client.

Only a non-anonymous server may — if appropriate for the selected cipher suite — request also a certificate from the client. It then specifies which types of certificate it is willing to accept and from which certificate authorities.

SSL Details *Server Key Exchange Message*

The server key exchange message is sent, if the server has no certificate or has a certificate only usable for signing (e.g., DSS, signing-only RSA). If it has a Diffie-Hellman certificate containing the Diffie-Hellman parameters then this message is not sent.

For key exchange SSL V3 supports the following algorithms:

- **RSA**
  the server sends a temporary public RSA key (for export reasons 512 bits)
- **Diffie-Hellman**
  the server sends his parameters: g, p, X where $X = g^x \bmod p$
- **Fortezza Key Exchange Algorithm**
  a proprietary algorithm using a hardware device for key generation and storage

SSL Details *Server Key Exchange Message*

Signature of Public Key Parameters

The key information may be signed by the server using the previously sent certificate or it may be sent anonymously.

If the server had a DSS certificate, it will sign with it:

```
SHA(ClientHello.random+ServerHello.random+
        key_parameters)
```

If the server had an RSA signature certificate it will, in addition to the above hash, also sign:

```
MD5(ClientHello.random+ServerHello.random+
        key_parameters)
```


SSL Details *Client Key Exchange Message*

The message the client will send depends on which public key algorithm has been selected:

- If RSA is used for key agreement and authentication, the client generates a 48-byte pre-master secret, encrypts it under the server's public key which is either derived from the server's key certificate or based on the RSA key from the server key exchange message.
- For Diffie-Hellman, the client sends Y, where $Y = g^y \bmod p$ is computed from the server's Diffie-Hellman parameters, either derived from the server's certificate or from the server key exchange message.
  The public key is not repeated in this message, if the client had already sent a certificate with his public key.

SSL Details *CertificateVerify*

This message provides explicit verification of a client certificate. The message is sent following all client certificates that have signing capability (i.e. not Diffie-Hellman).

```
MD5(master_secret+pad2+
        MD5(handshake_messages+master_secret+pad1))
SHA(master_secret+pad2+
        SHA(handshake_messages+master_secret+pad1))
```

The hashes are computed over all handshake messages starting from the *ClientHello* up to but not including this message. They are then signed with the signature algorithm in the client's certificate.

SSL Details *ChangeCipherSpec/Finished*

The *ChangeCipherSpecification* message indicates to the receiver that the handshake protocol is over and that the newly arbitrated keys should be used. It is sent both by the client and server.

The *Finish* message is the first message that is encoded under the new keys, it is sent by both partners. It consists of two hashes:

```
MD5(master_secret+pad2+
        MD5(handshake_messages+Sender+master_secret+pad1))
SHA(master_secret+pad2+
        SHA(handshake_messages+Sender+master_secret+pad1))

Sender = 0x53525652   // for the server
Sender = 0x434C4E54   // for the client
```

The hashes are computed over all handshake messages starting from the *ClientHello* up to but not including the *ChangeCipherSpec* nor the *Finished* messages.

SSL Details *Cryptographic Computations*

SSL uses for efficiency symmetric encryption for the data exchange, the asymmetric algorithms are only used to authenticate the parties and to generate the shared keys.

A *pre_master_secret* is generated for RSA by the client as a 48-bit random number and encrypted with the server's public key. For Diffie-Hellman, the negotiated key is used as *pre_master_secret*.

The *master_secret* is computed as follows:

```
master_secret =
        MD5(pre_master_secret+SHA('A'+pre_master_secret +
                ClientHello.random+ServerHello.random))+
        MD5(pre_master_secret+SHA('BB'+pre_master_secret +
                ClientHello.random+ServerHello.random))+
        MD5(pre_master_secret+SHA('CCC'+pre_master_secret +
                ClientHello.random+ServerHello.random))+
```

After that computation, the *pre_master_secret* should be purged from memory.

SSL Details *Master Secret*

The *master_secret* is used to generate a sequence of secure bytes which are assigned to the MAC secrets, keys, and non-export initialization vectors for CBC mode encryption. The key block is computed as follows:

```
key_block=
        MD5(master_secret+SHA('A'+master_secret+
                ServerHello.random+ClientHello.random))+
        MD5(master_secret+SHA('BB'+master_secret+
                ServerHello.random+ClientHello.random))+
        MD5(master_secret+SHA('CCC'+master_secret+
                ServerHello.random+ClientHello.random))+
        MD5(...)
```

The key block is then partitioned into the various keys and IVs. For exportable encryption the *write_key*s are computed from the *write_key*s derived from the key block through an additional hash operation:

```
export_write_key=MD5(write_key+
        ClientHello.random+ServerHello.random)
```

The export IVs do not contain the *master_secret:*

```
IV=MD5(ClientHello.random+ServerHello.random)
```

SSL Details — *Cipher Suites*

SSL provides for a number of cipher suites that are combinations of the selected public-key handshake algorithm, the symmetric encryption algorithm, the MAC, and whether export restrictions must be met or not.

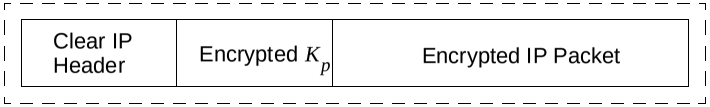
	NULL	RSA	RSA_EXP	DH_DSS	DH_DSS_EXP	DH_anon	DH_anon_EXP
NULL		MD5 or SHA					
DES_CBC		SHA		SHA		SHA	
DES40_CBC			SHA		SHA		SHA
3DES_EDE_CBC		SHA		SHA		SHA	
IDEA_CBC		SHA					
RC2_CBC_40			MD5				
RC4_40			MD5				MD5
RC4_128		MD5 or SHA				MD5	

Temporary RSA keys of 512 bits are considered relatively insecure and should be changed at least daily or every 500 transactions.

er5.fm: 03.08.2000 12:36

SKIP — *SImple-Key-Management for Internet Protocols*

- Other than SSL or PCT, SKIP is based on top of the connectionless IP layer
- Diffie-Hellman Public Key Algorithm used for key exchange
- Each node (IP address) holds a Diffie-Hellman Public/Private Key Pair X, x where $X = g^x \bmod p$
- Public keys may be authenticated either via X.509 certificates, PGP certificates, or secure DNS
- The mutually authenticated long-term secret for two nodes is thus $g^{xy} \bmod p$ of which the low-order *key-size* (256) bits are used
- An individual packet is encrypted and/or authenticated using a randomly generated traffic key K_p, which is encrypted with the long term key



Encrypted Unicast Packet

er5.fm: 03.08.2000 12:36

SKIP — *Master Key Update*

The master key can be updated through issuing new Diffie-Hellman key certificates to one of both of the involved parties.

As this is an expensive operation, the master key can also be updated through using a counter n and making the master key a function of this number:

$$K_{xyn} = \mathbf{hash}\,(K_{xy} \| n \| 1) \,\|\, \mathbf{hash}\,(K_{xy} \| n \| 0) \quad .$$

n must only be incremented and never be decremented.

n can, for example, be implemented in a stateless way using the hours since January 1, 1995 0:00.

Updating the master key in this way allows to:

- minimize the exposure of the key-encrypting key
- avoid sending of forged traffic should the traffic key ever be compromised
- avoid acceptance of packet replays, if the packets are too old

er5.fm: 03.08.2000 12:36

SKIP — *Key Separation*

SKIP uses different keys for message encryption and authentication. These keys are derived from the traffic key K_p as follows:

$$E_{K_p} = \mathbf{hash}\,(K_p \| \text{CryptAlg} \| 2) \,\|\, \mathbf{hash}\,(K_p \| \text{CryptAlg} \| 0)$$

$$A_{K_p} = \mathbf{hash}\,(K_p \| \text{MACAlg} \| 3) \,\|\, \mathbf{hash}\,(K_p \| \text{MACAlg} \| 1)$$

er5.fm: 03.08.2000 12:36

SKIP *Multicast Extension*

SKIP can be extended to multicast communication by choosing a group owner who distributes a *Group Interchange Key* (GIK), which is then used as a key encrypting key similarly to the uni-cast master key.

Using the GIK allows each participant in the group to use different traffic encrypting keys and thus stream ciphers like RC4, which otherwise would not be possible, if the same traffic encryption key were used by all group members.

WWW Authentication *HTTP*

The HyperText Transfer Protocol (HTTP) is the main protocol used in the World WIde Web to exchange information between a client's browser (*user agent*) and a server.

HTTP is stateless, it is built directly on top of TCP. In HTTP V1.0 for every single information exchange between client and server a new TCP connection is established.

In the HTTP header a so-called *Universal Resource Identifier* specifies the information to be accessed. For the actual access, HTTP V1.0 provides for three different methods:

- GET to retrieve information
- POST to request that the server accept the entity in the HTTP packet
- HEAD which is a GET but only returns an HTTP header

Authentication uses a simple challenge-response mechanism.

HTTP Basic Authentication

A server which requires clients to authenticate themselves replies to a client's HTTP method invocation with an *Unauthorized* (401) error code and an authentication challenge that contains an indication of the required authentication method.

The client may then resubmit the request including the authentication information. The authentication challenge is always tied to a server-defined *protection space*, which is a combination of the server's root *Universal Resource Locator* (URL) and a server-specified *realm.*

The only authentication method specified for HTTP V1.0 is so called *Basic Authentication* (BA) where a browser prompts the user for user id and password and then sends this information as *base-64* encoded strings.

- Challenge

```
WWW-Authenticate: Basic realm="x-rays"
```

- Response

```
Authorization: Basic YWh10mFodQ==
```

The browser can then cache this authentication information and reuse it for future accesses.

HTTP Authentication *Alternatives*

Clearly the basic authentication scheme is not secure as anybody can trap the user credentials as they are tansmitted.

To address this shortcoming of basic authentication, two proposals exist:

- *Digest Authentication* (DA) and
- *Mediated Digest Authentication* (MDA)

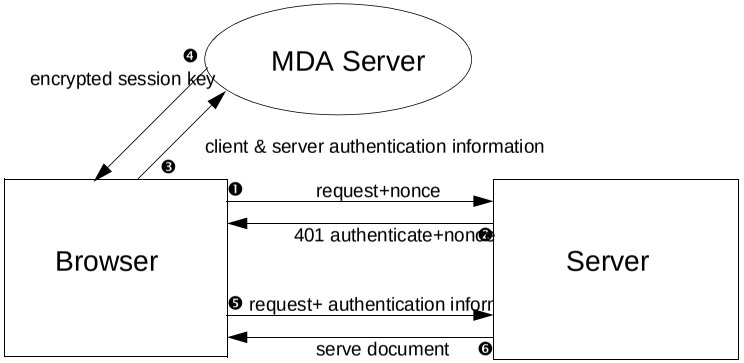
Digest Authentication

The authentication failure message contains a server nonce. The browser then constructs a digest from the user's password, the accessed realm and the nonce using MD5. This digest is then included with the user identification as the authenticator in the user's request message. The server recalculates the digest and makes an authentication decision based on whether is matches the user's digest.

To prevent replay of captured digests, the server may at any time rechallenge the client with a new nonce.

Mediated Digest Authentication

MDA builds on the concept of DA, it introduces however an authentication server which shares secrets with both the users and the servers and allows mutual authentication between the two parties.



The MDA protocol works as follows:

(1) $C \rightarrow S$ $\quad N_c \| Realm$

(2) $S \rightarrow C$ $\quad MDAServer \| S \| N_s \| MAC_s$

$MAC_s = \mathbf{hash}(S \| Realm \| N_c \| N_s \| K_s)$

(3) $C \rightarrow MDAServer$ $\quad C \| Realm \| S \| N_s \| N_c \| MAC_s \| MAC_c$

$MAC_c = \mathbf{hash}(C \| Realm \| N_c \| N_s \| MAC_s \| K_c)$

(4) $MDAServer \rightarrow C$ $\quad N_c \| K_s' \| K_c' \| MAC_s' \| MAC_c'$

$K_s' = K_{cs} \oplus \mathbf{hash}(C \| Realm \| N_s \| K_s)$

$K_c' = K_{cs} \oplus \mathbf{hash}(S \| Realm \| N_c \| K_c)$

$MAC_s' = \mathbf{hash}(S \| K_s' \| N_s \| K_s \| K_{cs})$

$MAC_c' = \mathbf{hash}(N_c \| K_s' \| K_c' \| MAC_s' \| K_c \| K_{cs})$

The last two messages in the MDA protocol are then:

(5) $C \rightarrow S$ $\quad Realm \| \mathbf{hash}(K_{cs} \| Realm \| N_s) \| K_s' \| MAC_s'$

(6) $S \rightarrow C$ $\quad response \| \mathbf{hash}(response \| K_{cs})$

As the HTTP authentication information is cached inside the client's browser and is only valid for a single root URL and a server specified realm, clients must resubmit authentication information when they want to access a different server via a different root URL.

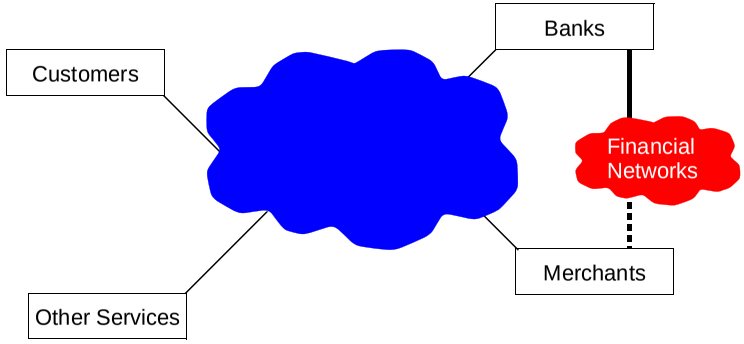
If we assume these servers define a virtual server group, say *med_image_grp* and they elect one of the servers to give the group its root name, e.g., *hundshorn.zurich.ibm.com*, then we can extend the authentication header in the challenge as follows:

- Challenge for authentication

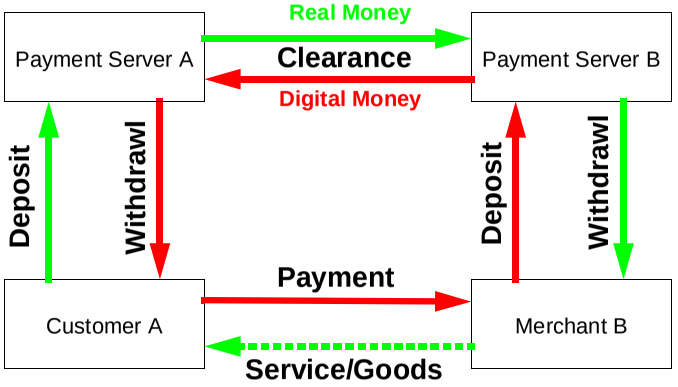
```
WWW-Authenticate: Basic realm="x-rays"
servgrp="hundshorn.zurich.ibm.com:med_image_grp"
```

Combining this group definition with *Mediated Digest Authentication* then allows authentication to arbitrary groups of servers with a single sign-on from the user.

Internet as Electronic Marketplace

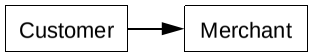


Flow of Digital and Real Money



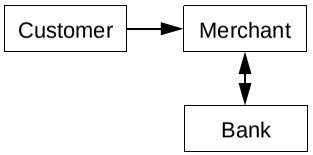
- Digital money can only be generated by the payment server
- It is transferable from customer to merchant or even to other customers

Off-line vs. On-line Payments



Off-line Payments

- Less communication
- Lower transaction costs
- Assure via *tamperproof security modules* that digital money is not spent multiple times



On-line Payments

- High transaction costs
- More communication
- Central authorization assures that money is not spent multiple times

E-Cash and Electronic Funds Transfers

	E-Cash	Electronic Funds Transfers
Amount	Change ~DM 400	All quantities
Payment	Off-line	On-line
Customer Anonymity	Similar to cash	Depending on the amount
Reimbursement	Batched	Individual

Ideal Properties of Digital Cash

(1) **Independence**. The security of the digital cash is independent of physical location. The cash can be transferred through any open computer network

(2) **Security.** Digital cash cannot be copied and spent twice

(3) **Privacy (Untraceability).** The user's privacy is protected; the relationship between the user and his purchases is untraceable

(4) **Off-line Payment.** When a user pays for a purchase with electronic cash, the transaction may be executed off-line

(5) **Transferability.** The digital cash can be transferred to other users

(6) **Divisibility.** A piece of digital cash in a given amount can be subdivided into smaller amounts to allow for *micro-payments*

Two Main Security Requirements

Integrity of Payment Server

- Total amount of money does not grow without involvement of payment server
- If amount grows, the cheater can be identified

Anonymity of Customer

- Transactions cannot be observed by outsiders
- Payments cannot be traced
  - through merchant
  - through merchant and payment server

Proposals and Projects

	On-line Payment	Off-line Payment

Not anonymous except for use of Pseudonyms	Credit Card via Internet
	• Plain *MOTO*
	• Alternate account number Authorization via e-mail *First Virtual*
	• Secure
	  • Cyber Cash
	  • Open Market
	  • NetMarket
	  • CommerceNet
	  • SET
	Payment Server
	• NetBill
	• NetCash (1)
		Smartcard based Purses
		• Danmont, Proton
		• Mondex
		• ...

Anonymous	Reliable Changer • NetCash • Anonymous Credit Card Blind Signatures • DigiCash E-Cash	Blind Signatures • CAFE

Electronic Cash Protocols

Protocol #1:

(1) Alice prepares 100 anonymous money orders for $1000 each

(2) Alice puts each and a piece of carbon paper into 100 different sealed envelopes, which she all gives to the bank

(3) The bank opens 99 envelopes and convinces itself that each is a money order for $1000

(4) The bank signs the one remaining unopened envelope. The signature goes through the carbon paper to the money order. The bank deducts $1000 from Alice's account and gives Alice back the envelope

(5) Alice opens the envelope and spends the money order with a merchant

(6) The merchant checks for the bank's signature to make sure that the money order is legitimate

(7) The merchant takes the money order to the bank

(8) The bank verifies its signature and credits $1000 to the merchant

Electronic Cash Protocols

Protocol #1 prevents Alice from writing a money order for more than she claims, it does however not protect the bank from Alice making a copy of the money order and spending it twice. To solve the **double-spending** problem we augment the protocol.

Protocol #2

(1) Alice prepares 100 anonymous money orders for $1000 each, in each she includes a different random uniqueness string long enough to make the chances of somebody else using it negligible.

(8) The bank verifies its signature, checks its database to make sure that a money order with the same uniqueness string hasn't been deposited before. If it hasn't is credits $1000 to the merchant and records the uniqueness string

(9) Otherwise it does not accept the money order

Protocol #2 will protect against Alice or the merchant photocopying the money order and submitting it twice.

Electronic Cash Protocols

Protocol #2 prevented double-spending but did not identify who did it. Protocol #3 corrects this by identifying the cheater.

Protocol #3

(7) The merchant asks Alice to write a random identity string on the money order when accepting it

(8) Alice complies

(9) The merchant takes the money order to the bank

(10) The bank verifies its signature, checks its database to make sure that a money order with the same uniqueness string hasn't been deposited before. If it hasn't is credits $1000 to the merchant and records the uniqueness string

(11) If the uniqueness string is in the database, it compares the identity string with the one stored in the database. If it is the same then the bank knows that the merchant copied the money order. If it is different, then the bank knows that the person who had bought the money order had copied it

Electronic Cash Protocols

Protocol #3 allowed identification of the cheater, however, if it was the person spending the money, that person remained anonymous. The next protocol, will allow identification of the cheater, while preserving the anonymity of the honest spender.

Before we can discuss this new algorithm we need to look at three cryptographic techniques first: **Secret Splitting**, **Blind Signatures**, and, **Bit Commitment**.

Secret Splitting

(1) Trent creates a random bit string R, the same length as the plain text M

(2) Trent XORs M with R to generate S: $S = M \oplus R$

(3) Trent gives S to Bob and R to Alice

To reconstruct the message Bob and Alice have to cooperate:

(4) $R \oplus S = M$

This protocol can easily be extended to multiple parties, just generate a random string for each participant.

Electronic Cash Protocols

Blind Signatures

Assume Bob has a public key e, public modulus n, and a private key d. Alice wants Bob to sign a message m blindly.

(1) Alice chooses a random value k between 1 and n. She then blinds m by computing $t = mk^e \bmod n$

(2) Bob signs t $t^d \equiv (mk^e)^d \bmod n$

(3) Alice unblinds t^d through computing $s = \frac{t^d}{k} \bmod n$ which is $m^d \bmod n$

$$t^d \equiv (mk^e)^d \equiv m^d k^{ed} \equiv m^d k \bmod n \quad \text{so} \quad \frac{t^d}{k} \equiv \frac{m^d k}{k} \equiv m^d \bmod n$$

Electronic Cash Protocols

Bit Commitment

Alice wants to commit to a bit or strings of bits however does not want Bob to learn what she committed to just now.

(1) Alice generates two random strings R_1 and R_2

(2) Alice generates a message consisting of her random strings and the bit string M she wants to commit to: $[R_1 \| R_2 \| M]$. Note M could be a single bit.

(3) Alice computes a one-way hash of the message and sends the result to as well as the one of the random strings to Bob $\mathbf{hash}\,(R_1 \| R_2 \| M) \| R_1$

The transmission from Alice is evidence of commitment. Bob cannot find out what she committed to. When it is time for Bob to find out the commitment

(4) Alice sends Bob the original message $[R_1 \| R_2 \| M]$

(5) Bob recomputes the hash and compares it and the random string he received in step 3. If they match Alice's commitment was valid.

Electronic Cash Protocols

Protocol #4 (1)

(1) Alice prepares n money orders for a given amount. Each of the money orders contains a different random uniqueness string X. On each money order, there are also n pairs of identity strings $I_1, I_2, \ldots, I_n$. They are generated from Alice's name, address, and other information the bank may require. Each string is split into two pieces using the secret splitting algorithm. Alice commits to each piece using a bit commitment protocol. E.g., I_8 consists of two parts I_{8_L} and I_{8_R}.

Each part is a bit-committed packet that Alice can be asked to open and whose proper opening can be verified instantaneously. Any pair I_{8_L} and I_{8_R} but not I_{8_L} and I_{9_R} reveals Alice's identity. Each of the money orders then looks like

```
Amount
Uniqueness String: X
Identity Strings: I1=(I1L,I1R), ..., In
```


Electronic Cash Protocols

Protocol #4 (2)

(2) Alice blinds all n money orders and gives them all to the bank

(3) The bank asks Alice to unblind $n-1$ of the money orders at random and confirms that they are well formed, i.e., it checks the amount, the uniqueness string and asks Alice to reveal all of the identity strings

(4) The bank then signs the one remaining blinded money order, deducts the amount from Alice's account and hands it back to Alice

(5) Alice unblinds the money order and spends it

(6) The merchant verifies the banks signature

(7) The merchant also asks Alice to randomly reveal either the left or the right half of each identity string on the money order via a random bit selector $b_1, b_2, \ldots, b_n$. Alice opens either half depending on whether $b_i = 0$ or 1

(8) Alice complies

(9) The merchant takes the money order to the bank

Electronic Cash Protocols

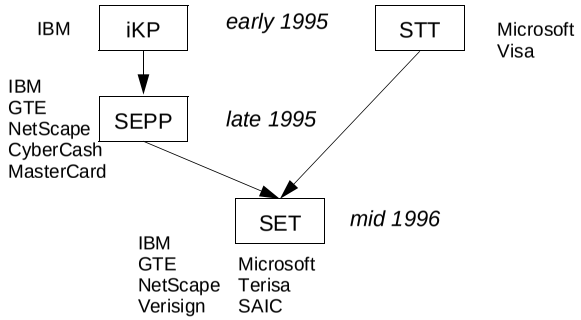
Protocol #4 (3)

(10) The bank verifies its signature, checks its database to make sure that a money order with the same uniqueness string hasn't been deposited previously. If it hasn't it credits the merchant's account. The bank records the uniqueness string and all of the identity information in its database.

(11) If the uniqueness string is in the database, the bank refuses to accept the money order. It then compares the identity information on the money order with the one stored in the database. If it is the same, the bank knows that the merchant cheated. If it is different, the bank knows that the person who bought the money order cheated. Since the second merchant who accepted the money order handed Alice a different selector string, the bank finds a bit position where one merchant had Alice open the left and the other the right half. XORing the two halves reveals Alice's identity.

Secure Electronic Transaction Protocol *SET*

- Joint standard by Visa and MasterCard for secure payment transactions over open networks



- Goals
  - Confidentiality of payment data
  - Payment integrity
  - Authenticate merchants, cardholders, and acquirers

er6.fm: 03.08.2000 15:53

SET *Participants*

Cardholder

The customer holds a credit card which has been issued by an issuer. SET ensures that the interactions the cardholder has with the merchant keep the payment card account information confidential.

Issuer

An issuer is a financial institution that establishes an account for a cardholder and issues the payment card. The issuer guarantees payment for authorized transactions using the payment card in accordance with payment card brand rules.

Merchant

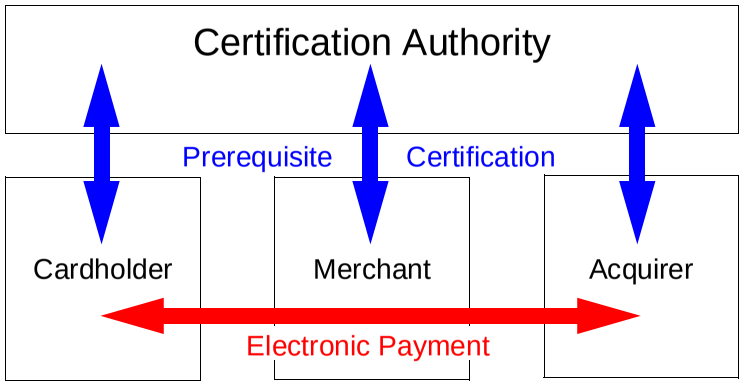
A merchant offers goods for sale or provides services in exchange for payment. SET allows the merchant to offer secure electronic interactions with cardholders. A merchant who accepts payment cards must have a relationship with an acquirer.

Acquirer

An acquirer is a financial institution that establishes an account with a merchant and processes payment card authorizations and actual payments.

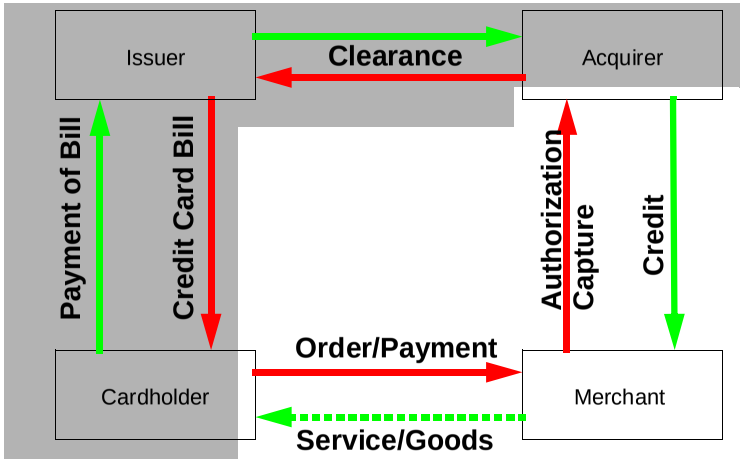
er6.fm: 03.08.2000 15:53

SET *Standardized Interactions*



er6.fm: 03.08.2000 15:53

SET *Flow of Payment Information*



er6.fm: 03.08.2000 15:53

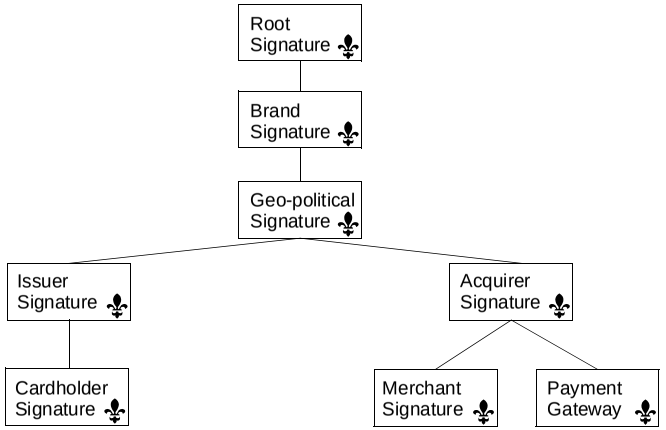
SET *Dual Signatures*

SET introduces a new application of digital signatures, so called ***Dual Signatures*** which it uses to link order messages sent to the merchant with account information sent to the acquirer.

Bob wants to send to Alice an offer to purchase a piece of property and an authorization to his bank to transfer the money only if Alice accepts his offer. Bob wants neither the bank to see his offer nor does he want Alice to see his account information. He accomplishes this by digitally signing both messages with a single signature operation that creates a dual signature.

The dual signature is generated by computing the message digest of both messages and computing a third digest of the concatenation of the first two digests. This digest is then signed (encrypted with the private key). The signer must include the digest of the other message for the recipient to verify the dual signature.

SET *Certification Authority Hierarchy*



SET *Cryptographic Operators* (1)

SET defines a number of cryptographic operators used in the protocol

H(*t*)	*Hash*	160-bit SHA-1 hash of ***t***
HMAC(*t, k*)	*Keyed Hash*	160-bit keyed hash of ***t*** *using* key ***k*** based on HMAC SHA-1
L(*t1, t2*)	*Linkage*	A reference or *link* to ***t2*** is included with ***t1***; equivalent to ***{t1, H(t2)}***
S(*s, t*)	*Signed Message*	The signature of entity ***s*** on ***t*** including the plaintext of ***t***. The default signature algorithm is RSA with SHA-1 hash.
SDO(*s, t1, t2*)	*Dual Signature Only*	The signature of ***s*** on the pair ***t1*** and ***t2***. Does not include the plaintext messages nor does it link them.
E(*r, t*)	*Asymmetric Encryption*	First encrypt ***t*** with a fresh symmetric key ***k***, then insert ***k*** into an RSA envelope for entity ***r*** under *Optimal Asymmetric Encryption Padding* (OAEP) using the public key of ***r.***
EH(*r, t*)	*Integrity Encryption*	Like **E** except that the RSA envelope contains also **H(*t*)** for a guarantee of message integrity when a signature of the sender is not available.
EX(*r, t, p*)	*Extra Encryption*	Like **E** except that ***t*** and ***p*** are the parts of a two-part message; ***t*** is to be liked with ***p*** and subject to ordinary encryption, while ***p*** is a *parameter* subject to extra processing. Both ***p*** and ***k*** are inserted into the RSA envelope.
EXH(*r, t, p*)	*Extra Encryption with Integrity*	Like **EX**, except that the RSA envelope contains also **{H(*t*), p}** for a guarantee of message integrity when a signature of the sender is not available.
EK(*k, t*)	*Symmetric Encryption with a provided key*	Symmetric encryption (DES) of the message ***t*** using the secret key ***k***.

SET *Cryptographic Operators* (2)

Based on the cryptographic operators, SET provides for a number of message encapsulation operators

Enc(*s, r, t*)	*Simple Encryption with Signature*	Signed, then encrypted message
EncK(*k, s, t*)	*Simple Encapsulation with Signature and a Provided Key*	Signed messages encrypted with a known secret key
EncX(*s, r, t, p*)	*Extra Encapsulation with Signature*	Two-part messages encrypted with the first part of the message in the ordinary (symmetric) encryption slot of **EX** and the second part in the extra slot of **EX**
EncB(*s, r, t, b*)	*Simple Encapsulation with Signature and Baggage*	Signed, encrypted messages with external; baggage
EncBX(*s, r, t, b, p*)	*Extra Encapsulation with Signature and Baggage*	Signed **EX**-encrypted, two-part messages with baggage
EncDXL2(*s, r, t1, t2, p2*)	*Linked, Signed, Extra Encapsulation with Dual Signature*	Models the treatment of the purchase request with order information from the cardholder to merchant. the receiver ***r*** is the payment gateway, it receives the order information (***t1***) and the payment information with parts ***t2*** and ***p2***.
{A, B, C}	*Tuple*	Tuple containing ***A, B, C***
A|B|C	*Ordered Concatenation*	Explicit ordering of the components
[A]	*Optional*	The item is optional
<A, B, C>	*Selection*	Exactly one of the elements must appear

SET *Features*

Cryptography

SET uses:

- RSA encryption for digital signatures and digital envelopes
- Signatures are based on PKCS#7 RSA with SHA-1
- DES in CBC mode is used for symmetric encryption with 56 bit keys
- *Commercial Data Masking Facility* (CDMF) for messages from the acquirer to the cardholder tunneled through the merchant
- SHA-1 is used for all hashes in SET

Because SET encrypts only financial data (except for acquirer - cardholder messages), strong encryption is permitted even for export. All public keys, except for the Root CA, are 1024-bits long. The Root CA's key is 2048 bits.

Protocol

All protocol messages are designed to be idempotent, a request may thus be repeated without harm until a response is received.

SET *Cardholder Registration* (1)

Cardholders in the current specification of SET do not necessarily require a certificate. To obtain a certificate, however, the cardholder (CH) requests it via the following six messages exchanged with his certification authority (CCA).

Cardholder Certificate Initiation Request/Response

(1) CH → CCA: {LID_EE, Chall_EE, BrandID, [Thumbs]}

(2) CCA → CH: S(CCA,{LID_EE, Chall_EE, CAKThumb, [BrandCRLIdentifier],[Thumbs]})

LID_EE	local ID to track protocol
Chall_EE	fresh challenge to prevent message replay
BrandID	BrandID of requested certificate
Thumbs	Thumbprints of Certs, CRLs, and BrandCRLIds currently held by CH
CAKThumb	Thumbprint of CA Key-exchange certificate that the card holder should use to encrypt the registration from request message
BrandCRLIdentifier	List of CRLs that the CH should screen received certificates against

SET *Cardholder Registration* (2)

Cardholder Registration Form Request/Response

(3) CH → CCA: EXH(CCA, {LID_EE, Chall_EE2, RequestType, Language}, PANOnly)

(4) CCA → CH: S(CCA,{LID_EE, Chall_EE2, RequestType <{LID_CA, Chall_CA, RegTemplate, Policy}, {Reason, [ReferralLoc]}>})

LID_EE	local ID to track protocol
Chall_EE2	fresh challenge to prevent message replay
RequestType	indicates whether the cardholder issues a request for a new or renewed signature and/or encryption certificate
Language	desired language for rest of exchange
PANOnly	{PAN, EXNonce}
PAN	Primary Account Number, the creditcard number
EXNonce	A fresh nonce to foil dictionary attacks on the PAN
LID_CA	CA's ID to track protocol
Chall_CA	Fresh challenge by the CA to prevent message replay
RegTemplate	URLs referring to a form to be filled out for registration
Policy	To be displayed along with registration template on cardholders system
Reason/ReferralLoc	In case the operation failed, this may give an explanation

SET *Cardholder Registration* (3)

Certificate Request and Response

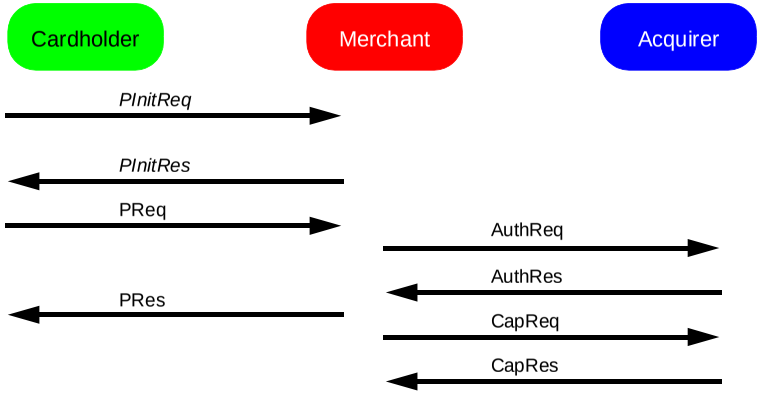
(5) CH → CCA: EncX(CH, CCA, {LID_EE, Chall_EE3, LID_CA, Chall_CA, RegForm, [CaBackKeyData], PK_S}, AcctInfo})

(6) CCA → CH: <S(CCA,{LID_EE, Chall_EE3, LID_CA, Chall_CA, CertStatusCode, [EEMessage], [CertThumbs]}) **or** EncK(CaBackKey, CCA, {LID_EE, Chall_EE3, LID_CA, Chall_CA, CertStatusCode, [EEMessage], Nonce_CCA, [CaMsg], CertThumbs]})>)

Chall_EE3	fresh challenge to prevent message replay
RegForm	filled out request form
CaBackKeyData	{CaAlgId, CaKey} symmetric algorithm and corresponding secret key that the CA may use to protect the CaMsg
AcctInfo	{PAN, ExpiryDate, CardNonce, EXNonce} CardNonce is the cardholder's proposed half of a shared secret, PANSecret which will be shared between him and the CCA. The CCA generates the other half (Nonce_CCA) and both parties will XOR the two halves to calculate the shared secret.
PK_S	Proposed public key to be certified
CertStatusCode	Status of the certification process
EEMessage	Message to be displayed on the CH screen
Nonce_CCA	The other half of a shared secret between cardholder and CCA.
CertThumbs	Thumbprint of the enclosed, newly generated signature certificate

SET *Payment Flow*

The main flow for SET is payment processing. It involves the cardholder (CH), the merchant (M), and the acquirer (payment gateway) (P).



er6.fm: 03.08.2000 15:53

SET *Payment Protocol* (1)

Initialization Request/Response

(1) CH → M {Language, LID_C, Chall_C, BrandID, [Thumbs]})

(2) M → CH S(M, {TransIDs, Chall_C, Chall_M, [BrandCRLIdentifier], GKThumb})

Language	Cardholder requests this language for the transaction
LID_C	Local ID; generated by the cardholder system
Chall_C	Cardholder's signature freshness challenge to merchant
BrandID	Cardholder's chosen creditcard brand
Thumbs	List of certificates, CRL, and BrandCRLIdentifier thumbprints in the cardholder's cache
TransIDs	Composed from several identifiers to uniquely identify this transaction, contains among other information LID_C
Chall_M	Merchant's challenge to cardholder's signature freshness
BrandCRLIdentifier	List of current CRLs for all CAs under a brand CA
GKThumb	Identifies the encryption certificate to be used in PReq to encrypt data for the payment gateway

er6.fm: 03.08.2000 15:53

SET *Payment Protocol* (2)

Purchase Order Request

The purchase request consists of an order instruction (OI) for the merchant and a payment instruction tunneled through the merchant to the acquirer payment gateway.

(3) CH → M <PReqDualSigned, PReqUnsigned>

PReqUnsigned is sent for cardholders who hold no certificates

PReqDualSigned	EncDXL2(C, P, OIData, PIHead, PANData) which is equivalent to {PIDualSigned, OIDualSigned}
PIDualSigned	{SDO(C, L(OIData, PIData), L(PIData, OIData)), EX(P, L(PIHead, OIData), PANData)}
OIDualSigned	L(OIData, PIData)
PIData	{PIHead, PANData}
PIHead	{TransIDs, PINonce, MerchantID}
PANData	{PAN, CardExpiry, PANSecret, EXNonce}, PANSecret is a secret nonce shared between cardholder, acquirer, and cardholder's CA
OIData	{TransIDs, CHALL_c, H({OD, PurchAmt, ODSalt}), ODSalt, Chall_M, BrandID}
OD	Order description
ODSalt	Fresh nonce to prevent dictionary attacks on H(OD, PurchAmt, ODSalt)

er6.fm: 03.08.2000 15:53

SET *Payment Protocol* (3)

Authorization Request/Response

After receiving the purchase order, the merchant requests payment authorization from the payment gateway.

(4) M → P EncB(M, P, AuthReqData, PIDualSigned)

(5) P → M EncB(P, M, AuthResData, CapToken)

AuthReqData	{TransIDs, AuthRetNum, H(OIData), H({OD, PurchAmt, ODSalt})
AuthRetNum	Identification of the authorization request used within the financial network 0 on its way to P, filled in on its return
PIDualSigned	taken from Purchase Order Request Message
AuthResData	{TransIDs, AuthRetNum, AuthResPayload}
AuthResPayload	{AuthCode, AuthAmt, RespCode, ResponseData} essentially the authorization information
CapToken	EncX(P, P, CapTokenData, PANToken)
CapTokenData	{AuthAmt, TokenOpaque}
AuthAmt	Actual authorized amount
PANToken	{PAN, CardExpiry, EXNonce}

er6.fm: 03.08.2000 15:53

SET *Payment Protocol* (4)

Purchase Order Response

The purchase response follows as the result of an authorization request/response exchange between the merchant and his acquirer gateway.

(6) M → CH S(M, {TransIDs, Chall_C, [BrandCRLIdentifier], AuthCode})

AuthCode	enumerated authorization code

SET *Payment Protocol* (5)

Capture Request/Response

After having received payment authorization and shipped the goods, the merchant may now capture the payment and get credit from his acquirer.

(7) M → P EncB(M, P, CapReqData, CapToken)

(8) P → M Enc(P, M, CapResData)

CapReqData	{TransIds, CapReqAmt}
CapReqAmt	Amount requested for capture
CapResData	{TransIds, CapCode, CapAmt}
CapCode	Completion status
CapAmt	Amount merchant asked for in his capture request, if it matched the authorization amount