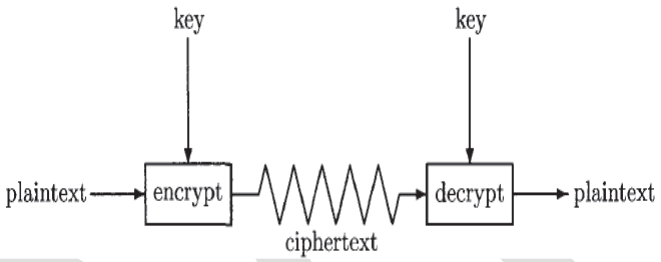
Crypto Basics

The basic terminology of crypto includes the following:

• *Cryptology* — the art and science of making and breaking "secret codes."
• *Cryptography*— the making of "secret codes."
• *Cryptanalysis*— the breaking of "secret codes."
• *Crypto*— a synonym for any or all of the above and more.

- A *cipher* or *cryptosystem* is used to *encrypt* data. The original unencrypted data is known as *plaintext*, and the result of encryption is *ciphertext.Decrypting* the ciphertext to recover the original plaintext.
- A *key* is used toconfigure a cryptosystem for encryption and decryption.

In a ***symmetric* cipher**, the same key is used to encrypt and to decrypt, as illustrated by the black box cryptosystem in below Figure.



Crypto as a Black Box

In **public key crypto**, the encryption key is known as the *public key*, whereas thedecryption key, which must remain secret, is the *private key*.

[Kerckhoffs's principle goes as follows: A cryptographic system should be secure even if everything about the system, except the key, is public knowledge].

Classic Crypto:

- Four classic ciphers:
  - Simple Substitution Cipher.
  - Double Transposition Cipher
  - Codebook Cipher
  - One-Time Pad

Simple Substitution Cipher:

The message is encrypted by substitutingthe letter of the alphabet *n* places ahead of the current letter. For example,with *n* = 3, the substitution—which acts as the key—is

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

The convention that the plaintext is lowercase,and the ciphertext is uppercase. Using the key 3, encrypt the plaintext message:

fourscoreandsevenyearsago

to the resulting ciphertext is:

IRXUVFRUHDAGVHYHABHDUVDIR

To decrypt this simple substitution, look up the ciphertext letter in the ciphertext row and replace it with the corresponding letter in the plaintext row, or shift each ciphertext letter backward by three. The simplesubstitution with a shift of three is known as the **Caesar's cipher**.

- If we limit the simple substitution to shifts of the alphabet, then the possible keys are $n \in \{0,1,2,..., 25\}$.
- Attacker can suspect that received text was encrypted with a simple substitution cipher usinga shift by *n.* Then he can try each of the 26 possible keys, decrypt the message, this **Brute force approach** is known as***Exhaustive key search.***
- It's necessary that the number of possible keys be too large for the attacker to simply try them all in any reasonable amount of time.

Keyspace: Suppose attacker has a fast computer that's able to test 2^{40} keys each second. Then a keyspace of size 2^{56} can be exhausted in 2^{16} seconds, or about 18 hours, whereas a keyspace of size 2^{64} would take more than half a year for an exhaustive key search, and a keyspace of size 2^{128} would require more than nine quintillion years.

.

Permutation: Anypermutation of the 26 letters willserve as a key.

For example, the following permutation, gives us a key for a simple substitution cipher:

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

- A simple substitution cipher can employ any permutation of the alphabet as a key, which implies that there are **26! = 2^{88} possible keys**.
- With attacker's superfast computer that tests 2^{40} keys per second, trying all possible keys for the simple substitution would take more than 8900 millennia.

- Attacker would expect to find the correct key half that time, or just 4450millennia. Since 2^{88} keys is far more than attacker can try in any reasonable amount of time. The keyspace is big enough so that an **exhaustive key search** is infeasible.

Cryptanalysis of a Simple Substitution:

Suppose attacker intercepts the following ciphertext, which he suspects was produced by a simple substitution cipher, where the key could be any permutation of the alphabet:

```
PBFPVYFBQXZTYFPBFEQJHDXXQVAPTPQJKTOYQWIPBVWLXTOXBTFXQWA
XBVCXQWAXFQJVWLEQNTOZQGGQLFXQWAKVWLXQWAEBIPBFXFQVXGTVJV
WLBTPQWAEBFPBFHCVLXBQUFEVWLXGDPEQVPQGVPPBFTIXPFHXZHVFAG
FOTHFEFBQUFTDHZBQPOTHXTYFTODXQHFTDPTOGHFQPBQWAQJJTODXQH
FOQPWTBDHHIXQVAPBFZQHCFWPFHPBFIPBQWKFABVYYDZBOTHPBQPQJT
QOTOGHFQAPBFEQJHDXXQVAVXEBQPEFZBVFOJIWFFACFCCFHQWAUVWFL
QHGFXVAFXQHFUFHILTTAVWAFFAWTEVOITDHFHFQAITIXPFHXAFQHEFZ
QWGFLVWPTOFFA
```

Assuming the plaintext is English,attacker can make use of the English letter frequency counts in Figure 2.2 together with the frequency counts for the ciphertext in (2.2), which appear in Figure 2.3.

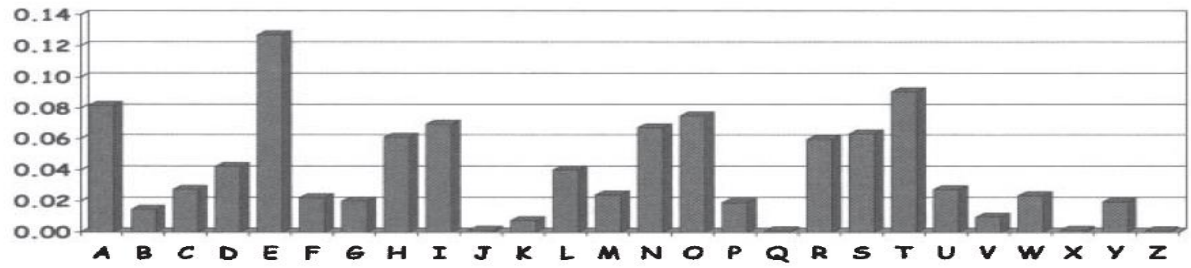


Figure 2.2: English Letter Frequency Counts

From the ciphertext frequency counts in Figure 2.3

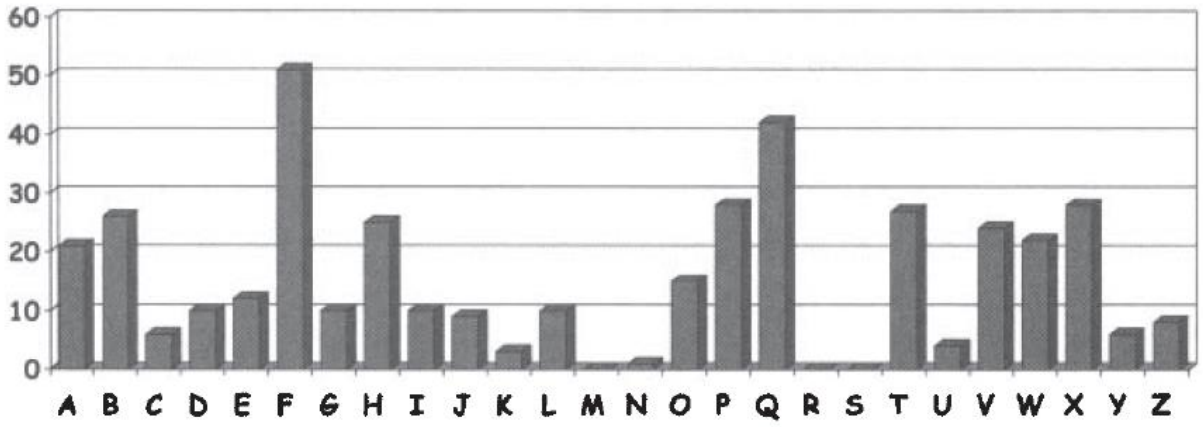


Figure 2.3: Ciphertext Frequency Counts

- "F" is the most common letter in the encrypted message and, according to Figure 2.2, "E" is the most common letter in the English language.
- Attacker therefore conclude that it's likely that "F" has been substituted for "E." Continuing in this manner, attacker can try likely substitutions until he recognizes words, at which point he can be confident in her guesses.

Conclusion: Above discussed attack on the simple substitution shows that a large keyspace is notsufficient to ensure security.

Double Transposition Cipher:

- Write the plaintext into an array of a given size.
- Then permute the rows and columns according to specified permutations.
- For example, suppose consider the plaintext:

attackatdawn into a 3 x 4 array

$$\begin{bmatrix} a & t & t & a \\ c & k & a & t \\ d & a & w & n \end{bmatrix}$$

- Transpose (or permute) the rows according to (1,2,3) —>(3,2,1) and then transpose the columns according to (1,2,3,4) —> (4,2,1,3), we obtain

$$\begin{bmatrix} a & t & t & a \\ c & k & a & t \\ d & a & w & n \end{bmatrix} \longrightarrow \begin{bmatrix} d & a & w & n \\ c & k & a & t \\ a & t & t & a \end{bmatrix} \longrightarrow \begin{bmatrix} n & a & d & w \\ t & k & c & a \\ a & t & a & t \end{bmatrix}$$

- The ciphertext is then read from the final array:

NADWTKCAATAT

- The key consists of the size of the matrix and the row and column permutations.
- **Attackatdawn:**If anyone who knows the key can put the ciphertext into the appropriate sized matrix and undo the permutations to recover the plaintext.

  For example, to decrypt the ciphertext is first put into a 3 x 4 array. Then the columns are numbered as (4,2,1,3) and rearranged to (1,2,3,4), and the rows are numbered (3,2,1) and rearrangedinto (1,2,3), and we have recovered the plaintext.

$$\begin{bmatrix} N & A & D & W \\ T & K & C & A \\ A & T & A & T \end{bmatrix} \longrightarrow \begin{bmatrix} D & A & W & N \\ C & K & A & T \\ A & T & T & A \end{bmatrix} \longrightarrow \begin{bmatrix} A & T & T & A \\ C & K & A & T \\ D & A & W & N \end{bmatrix}$$

Conclusion: The double transposition appears to thwart an attack that relies on the statistical information contained in the plaintext, since the plaintext statistics are disbursed throughout the ciphertext.

One-Time Pad:

- The one-time pad, which is also known as the Vernam cipher, is a provably secure cryptosystem.
- Let's consider an alphabet with only eight letters and the corresponding binary representation of letters appear in the below table.

  Note: **It's important to note that the mapping between letters and bits is not secret.**

letter	e	h	i	k	l	r	s	t
binary	000	001	010	011	100	101	110	111

- Onetime pad is used to encrypt the plaintext message

  h e i l h i t l e r.

  convert the plaintext letters to the **bit string** using above table.

  001 000 010 100 001 010 111 100 000 101

- The one-time pad key consists of a randomly selected string of bits that is the same length as the message.
- The key is then XORed with the plaintext to yield the ciphertext.
- Decryption is accomplished by XOR-ing the same key with the ciphertext.

Example:

Consider that Alice has the key :

111 101 110 101 111 100 000 101 110 000

which is of the proper length to encrypt her message above. Then to encrypt, Alice computes the ciphertext as:

	h	e	i	l	h	i	t	l	e	r
plaintext:	001	000	010	100	001	010	111	100	000	101
key:	111	101	110	101	111	100	000	101	110	000
ciphertext:	110	101	100	001	110	110	111	001	110	101
	s	r	l	h	s	s	t	h	s	r

Converting these ciphertext bits back into letters, the ciphertext message to be transmitted is

srlhssthsr.

Bob, receives Alice's message, he decrypts it using the same shared key and thereby recovers the plaintext:

	s	r	l	h	s	s	t	h	s	r
ciphertext:	110	101	100	001	110	110	111	001	110	101
key:	111	101	110	101	111	100	000	101	110	000
plaintext:	001	000	010	100	001	010	111	100	000	101
	h	e	i	l	h	i	t	l	e	r

Different scenarios examples:

1) Suppose that Alice has an enemy, Charlie, within her spy organization. Charlie claims that the actual key used to encrypt Alice's message is

   101 111 000 101 111 100 000 101 110 000.

   Bob decrypts the ciphertext using the key given to him by Charlie and obtains **killhitler** which is a wrong message.

	s	r	l	h	s	s	t	h	s	r
ciphertext:	110	101	100	001	110	110	111	001	110	101
“key”:	101	111	000	101	111	100	000	101	110	000
“plaintext”:	011	010	100	100	001	010	111	100	000	101
	k	i	l	l	h	i	t	l	e	r

2) Suppose that Alice is captured by her enemies, who have also intercepted the ciphertext. The captors ask Alice is to provide the key for this super-secret message. Alice claims that she is actually a double agent and to prove it she provides the "key".

   111 101 000 011 101 110 001 011 101 101.

When Alice's captors "decrypt" the ciphertext using this "key," they find it as **helikesike and** Alice's captors will release her.

	s	r	l	h	s	s	t	h	s	r
ciphertext:	110	101	100	001	110	110	111	001	110	101
"key":	111	101	000	011	101	110	001	011	101	101
"plaintext":	001	000	100	010	011	000	110	010	011	000
	h	e	l	i	k	e	s	i	k	e

Advantage:If the key is chosen at random, and used only once, then an attacker who sees the ciphertext provides no meaningful information at all about the plaintext.

Why that the one-time pad can only be used once?

Scenario 1: Suppose we have two plaintext messages P1 and *P2* and encrypted these as as $\mathbf{C_1} = \boldsymbol{P_1} \oplus \boldsymbol{K}$ **and** $\mathbf{C_2} = \boldsymbol{P_2} \oplus \boldsymbol{K}$, i.e two messages encrypted with the same "one-time" pad ***K***. In the cryptanalysis , this is known as a ***depth***. With one-time pad ciphertexts in depth,

$$C_1 \oplus C_2 = P_1 \oplus K \oplus P_2 \oplus K = P_1 \oplus P_2$$

and the **key has disappeared** from the problem. In this case, the ciphertext does yield some information about the underlying plaintext.

Scenario 2: Another way is considering an exhaustive key search. If the pad is only used once, thenthe attacker has no way to know whether the guessed key is correct or not. But if two messages are in depth, for the correct key, both putative plaintexts must make sense.
Let's consider an example of one-time pad encryptions **that are in depth**.Using the same bit encoding as in Table. Suppose

$$P_1 = \texttt{like} = 100\,010\,011\,000 \quad \text{and} \quad P_2 = \texttt{kite} = 011\,010\,111\,000$$

and both are encrypted with the same key ***K*** **= 110 011 101 111**. Then

	l	i	k	e
P_1:	100	010	011	000
K:	110	011	101	111
C_1:	010	001	110	111
	i	h	s	t

and

	k	i	t	e
P_2:	011	010	111	000
K:	110	011	101	111
C_2:	101	001	010	111
	r	h	i	t

- If the attacker knows that the messages are in depth, immediately he sees that the **second** and **fourth** letters of P_1 and P_2 are the same, since the corresponding ciphertext letters are identical.
- Now attacker can guess a putative message $P_1$and check her results using P_2.
- Suppose that attacker (who only has $C_1$and C_2) suspects that **P_1 = k i l l = 011010100100**. Then he can find the correspondingputative key:

	k	i	l	l
putative P_1:	011	010	100	100
C_1:	010	001	110	111
putative K:	001	011	010	011

and he can then use this ***K* to "decrypt" C_2** and obtain

C_2:	101	001	010	111
putative K:	001	011	010	011
putative P_2:	100	010	000	100
	l	i	e	l

- Since this *K* does not yield a sensible decryption for P_2, attacker can safely assume that his guess for P_1 was incorrect.
- Eventually attacker guesses P_1= **like** he will obtain the correct key *K* and decrypt to find P_2= **kite**, thereby confirming the correctness of the key therefore, the correctness of both decryptions.

Project VENONA:

- The VENONA project is an example of a real-world use of the one-time pad. In the 1930s and 1940s, spies from the Soviet Union who entered the United States brought with them one-time pad keys.
- When it was time to report back to their handlers in Moscow, these spies used their one-time pads to encrypt their messages, which could then be safely sent back to Moscow.

- These spies were extremely successful, and their messages dealt with the most sensitive U.S. government secrets of the time.
- In particular, the development of the first atomic bomb was a focus of much of the espionage. The Rosenbergs, Alger Hiss, and many other wellknowntraitorsfigure prominently in VENONA messages.
- The Soviet spies were well trained and never reused the key, yet many of the intercepted ciphertext messages were eventually decrypted by American cryptanalysts.
- There was a flaw in the method used to generate the pads, so that, long stretches of the keys were repeated. As a result, many messages were in depth, which enabled the successful cryptanalysis of much VENONA traffic.

Codebook Cipher:

- A classic codebook cipher is a dictionary-like book containing (plaintext) words and their corresponding (ciphertext) codewords.
- To encrypt a given word, the cipher clerk would simply look up the word in the codebookand replace it with the corresponding codeword.
- Decryption, using the inverse codebook, was equally straightforward.

Example: Below table contains a famous codebook used by Germany during World War I(used to encrypt the famous Zimmermann telegram.)

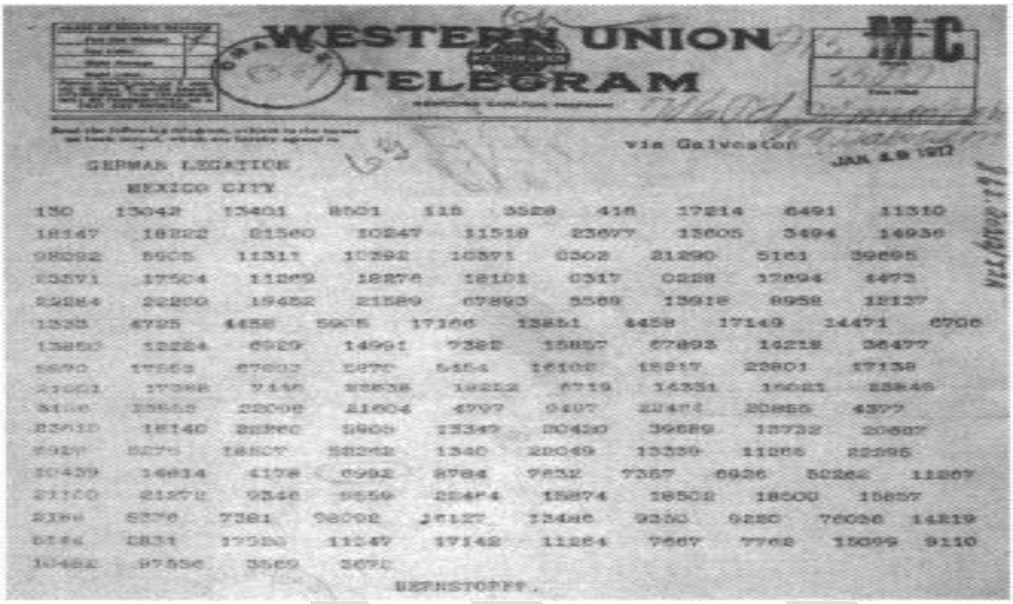
Table 2.3: Excerpt from a German Codebook

Plaintext	Ciphertext
Februar	13605
fest	13732
finanzielle	13850
folgender	13918
Frieden	17142
Friedenschluss	17149
⋮	⋮

This codebook was used for encryption, while the corresponding inverse codebook, arranged with the 5-digit codewords in numerical order, was used for decryption. A codebook is a form of a substitution cipher.

- The German Foreign Minister, Arthur Zimmermann, sent an encrypted telegram to the German ambassador in Mexico City.

- The ciphertext message, which appears in below figure was intercepted by the British. At the time, the British and French were at war with Germany, but the U.S. was neutral.



- The Russians had recovered a damaged version of the German codebook, and the partial codebook had been passed on to the British.
- Through analyses, the British were able to fill in the gaps in the codebook so that by the time they obtained the Zimmermann telegram, they could decrypt it.
- The telegram stated that the German government was planning to begin "**unrestricted submarine warfare**" and had concluded that this would likely lead to war with the United States.
- As a result, Zimmermann told his ambassador that Germany should try to recruit Mexico as an ally to fight against the United States.
- The incentive for Mexico was that it would "reconquer the lost territory in Texas, New Mexico and Arizona."
- When the Zimmermann telegram was released in the U.S., public opinion turned against Germany and, after the sinking of the Lusitania, the U.S. declared war.

Security:

- The security of a classic codebook cipher depends primarily on **the physical security** of the book itself. That is, the book must be protected from capture by the enemy.

- **Statistical attacks** analogous to those used to break a simple substitution cipher apply to codebooks, although the amount of data required is much larger.
- The reason that a statistical attack on a codebook is more difficult is due to the fact that **the size of the "alphabet" is much larger.**.

Additive Book:

- Ciphers were subject to statistical attack, so codebooks **needed to be periodically replaced with new codebooks**. Since this was an expensive and risky process, techniques were developed to extend the life of a codebook. To accomplish this, a ***additive****book* was used.
- The codewords are all 5- digit numbers. Then the corresponding additive book would consist of a long list of randomly generated 5-digit numbers.
- After a plaintext message had been converted to a series of 5-digit codewords, a starting point in the additive book would be selected and beginning from that point, the sequence of 5- digit additives would be added to the codewords to create the ciphertext.
- To decrypt, the same additive sequence would be **subtracted from the ciphertext** before looking up the codeword in the codebook.
- The starting point in the additive book was selected at random by the sender and sent in the clear at the start of the transmission.
- This additive information was part of the ***message indicator,* or MI**.
- The MI included any non-secret information needed by the intended recipient to decrypt the message.
- If the additive material was only used once, the resulting cipher would be equivalent to a one-time pad and therefore, provably secure.
- If the additive was reused many times and, any messages sent with overlapping additives would have their codewords encrypted with the same key, where the key consists of the codebook and the specific additive sequence.
- Therefore, any messages with overlapping additive sequences could be used to gather the statistical information needed to attack the underlying codebook.
- The additive book dramatically increased the amount of ciphertext required to mount a statistical attack on the codebook, which effects the cryptographers.

2.3.8 Ciphers of the Election of 1876

- The contestants in the election were Republican Rutherford B. Hayes and Democrat Samuel J. Tilden.
- The Electoral College that determines the winner of the presidency.
- In the Electoral College, each state sends a delegation and for almost every state, the entire delegation is supposed to vote for the candidate who received the largest number of votes in that particular state.
- In 1876, the electoral college delegations of four states were in dispute.
- A commission of 15 members was appointed to determine which state delegations were legitimate, and thus determinethe presidency.
- The commission decided that all four states should go to Hayes and he became president of the United States.
- Some months after the election, reporters discovered a large number ofencrypted messages that had been sent from Tilden's supporters to officials inthe disputed states.
- One of the ciphers used was a partial codebook together . with a transposition on the words.
- The codebook was only applied to important words and the transposition was a fixed permutation for all messages of a given length.
- The allowed message lengths were 10, 15, 20, 25, and 30 words, with all messages padded to one of these lengths. A snippet of the codebook appears in Table

Election of 1876 Codebook

Plaintext	Ciphertext
Greenbacks	Copenhagen
Hayes	Greece
votes	Rochester
Tilden	Russia
telegram	Warsaw
⋮	⋮

- The permutation used for a message of 10 words was

  9,3,6,1,10,5,2,7,4,8.

- One actual ciphertext message was

  Warsaw they read all unchanged last are idiots can't situation

which was decrypted by undoing the permutation and substituting telegramfor Warsaw to obtain

Can't read last telegram.

Situation unchanged.

They are all idiots.

- A permutation of a given length was used repeatedly, many messages of particular length were in depth—with respect to the permutation as well as the codebook.
- A cryptanalyst could compare all messages of the same length, making it relatively easy to discover the fixed permutation, even without knowledge of the partial codebook.
- The overuse of a key can be an exploitable flaw.

Modern Crypto History

- Late in the 20th century, cryptography became a critical technology for commercial and business communications as well, and it remains today as well.
- The Zimmermann telegram is one of the first examples that cryptanalysis has had in political and military affairs.

- In the Pacific theatre, the so-called Purple cipher was used for high levelJapanese government communication. This cipher was broken by American cryptanalysts before the attack on Pearl Harbor, but the intelligencegained (code named MAGIC) provided no clear indication of the impending attack.
- The Japanese Imperial Navy used a cipher known as JN-25, which was also broken by the Americans, an inferior American force was able to halt the advance of the Japanese in the Pacific for the first time.
- In Europe, the German Enigma cipher (code named ULTRA) was a major source of intelligence for the Allies during the war.
- The Enigma was initially broken by Polish cryptanalysts. After the fall of Poland, these cryptanalysts escaped to France.
- The Polish cryptanalysts eventually made their way to England, where they provided their knowledge to British cryptanalysts.
- A British team that included the computing pioneer, Alan Turing, developed improved attacks on the Enigma.

An Enigma Cipher

Two fundamental cipher design principles: *confusion* and *diffusion.*

Confusion

- It is defined as obscuring the relationship between the plaintext and ciphertext.
- A simple substitution cipher and a one-time pad employ only confusion
- Confusion is provably secure

Diffusion

- It is the idea of spreading the plaintext statistics through the ciphertext.
- A double transposition is a diffusion-only cipher.
- Diffusion alone is not secure.

The National Bureau of Standards, or NBS, issued a request for cryptographic algorithms. The ultimate result of this process was a cipher known as the Data Encryption Standard, or DES and Public key cryptography was discovered) shortly after the arrival of DES.

A Taxonomy of Cryptography

Three broad categories of ciphers: *symmetric* ciphers, *public key* cryptosystems, and *hash functions.*

***Symmetric* ciphers:**

- Modern symmetric ciphers can be subdivided into *stream ciphers* and *block ciphers*
- Stream ciphers generalize the one-time pad approach, sacrificing provablesecurity for a key that is manageable.

- Block ciphers are the generalization of classic codebooks. In a block cipher, the key determines the codebook, and as long as the key remains fixed, the same codebook is used. Conversely, when the key changes, a different codebook is selected.
- Block ciphers are easier to optimize for software implementations, while stream ciphers are usually most efficient in hardware.

***Public key* cryptosystems**

- In public key crypto, encryption keys can be made public. For each public key, there is a corresponding decryption key that is known as a private key.
- Public key cryptography does not completely eliminate the key distribution problem, since the private key must be in the hands of the appropriate user, and no one else.

Cryptographic hash functions:.

- These functions take an input of any size and produce an output of a fixed size.
- If the input changes in one or more bits, the output should change in about half of its bits.
- It is computationally infeasible to find *any* two inputs that hash to the same output.

A Taxonomy of Cryptanalysis

The goal of cryptanalysis is to recover the plaintext, the key, or both.

- ***Ciphertext only* attack:** If attacker only knows the algorithms and the ciphertext, then he must conduct a *ciphertext only* attack
- ***Known plaintext attack:*** Attacker might know some of the plaintext and observe the corresponding ciphertext. These matched plaintext-ciphertext pairs might provide information about the key.
- ***Chosen plaintext* attack:** Attacker can actually choose the plaintext to be encrypted and see the corresponding ciphertext.

For example, Alice might forget to log out of her computer when she takes her lunch break. Attacker could then encrypt some selected messages before Alice returns. This type of "**lunchtime attack"** takes many forms.

- ***Adaptively Chosenplaintext* attack**: Trudy chooses the plaintext, views the resulting ciphertext, and chooses the next plaintext based on the observed ciphertext.

- ***Related key* attacks:** The idea here is to look for a weakness in the system when the keys are related in some special way.
- ***Forward search (for Public Key Cryptography):*** Suppose attacker intercepts a ciphertext that was encrypted with Alice's public key. If attacker suspects that the plaintext message was either "yes" or "no," then she can encrypt both of these putative plaintexts with Alice's public key. If either matches the ciphertext, then the message has been broken.

Note: The size of the ***keyspace must be large enough*** *to prevent an attacker from trying all possible keys*

Problems:

Question 1

Given that the Caesar's cipher is used find the plaintext from the ciphertext .

Hint :Ceasers cipher is nothing but simple substitution with key=3

VSRQJHEREVTXDUHSDQWU

Solution:

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

VSRQJHEREVTXDUHSDQWU

Ans : spongebobsquarepantr

Question 2

Find the plaintext and the key from the ciphertext

CSYEVIXIVQMREXIH

Given that the cipher is a simple substitution of the shift-by-*n* variety.

Hint :

Exhaustive key search

Solution :

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

Cipher text:CSYEVIXIVQMREXIH

n = 1 brxduhwhuplqdwhg

n = 2 aqwctgvgtokpcugf

n = 3 zpvbsfufsnjobufe

n = 4 youareterminated

Ans : youareterminated

Question 3

If we have a computer that can test 2^{40} keys each second, and if the key space is of size 2^{128}, find the time required to perform exhaustive key search in terms of years.

Hint:

Remark – There are 31,557,600 seconds a year (365.25 days per year)

Solution :

2^{40} keys - 1 sec

2^{128} keys - ?

(2^128 / 2^40) / 31,557,600 =

4,903,494,084,172,197,326.87 years

Question 5

The weak ciphers used during the election of 1876 employed a fixed permutation of the words for a given length sentence. To see that this is weak, find the permutation of *(1, 2, 3, . . . , 10)* that was used to produce the scrambled sentences below, where "San Francisco" is treated as a single word. Note that the same permutation was used for all three sentences.

first try try if you and don't again at succeed
only you you you as believe old are are as
winter was in the I summer ever SanFrancisco coldest spent

- Ans.

4	9	1	5	7	10	2	6	3	8
If	at	first	you	don't	succeed	try	and	try	Again
You	are	only	as	old	as	you	believe	you	Are
The	coldest	winter	I	ever	spent	was	summer	in	San Francisco

Answer : (4 9 1 5 7 10 2 6 3 8)

Question 6

Encrypt the message :

We are all together

Using a double transposition cipher with 4 rows and 4 columns using

the row permutation

(1,2,3,4)-→ (2,4,1,3)

And column permutation

(1,2,3,4)-→(3,1,2,4)

Answer:

row transposition

w	e	a	r	e	a	l	l
e	a	l	l	t	h	e	r
t	o	g	e	w	e	a	r
t	h	e	r	t	o	g	e

col transposition

l	e	a	l
e	t	h	r
a	w	e	r
g	t	o	e

Cipher text : LEALETHRAWERGTOE

Question 7:

Suppose that the following is an excerpt from the decryption codebook for a classic codebook cipher:

code	Message
123	once
199	or
202	maybe
221	twice
233	time
332	upon
451	a

Deccrypt the following cyphertext:

242,554,650,464,532,749,567

Assuming that the following additive sequence was used to encrypt the message:

119,222,199,231,333,547,346

Solution :

Subtract the additive sequence from cyphertext code

242,	554,	650,	464,	532,	749,	567
-119,	222,	199,	231,	333,	547,	346

123 332 451 233 199 202 221

Convert the resultant code to text by looking into codebook cipher

123	332	451	233	199	202	221
Once	upon	a	time	or	maybe	twice

Ans : Once upon a time or maybe twice

Question 8:

Given ciphertext

srlhssthsr

And key

111 101 110 101 111 100 000 101 110 000

Find the plain text.

Letter	e	h	i	k	l	r	s	t
Binary	000	001	010	011	100	101	110	111

Converting these ciphertext bits back into letters, the ciphertext message to be transmitted is

srlhssthsr.

Bob, receives Alice's message, he decrypts it using the same shared key and thereby recovers the plaintext:

	s	r	l	h	s	s	t	h	s	r
ciphertext:	110	101	100	001	110	110	111	001	110	101
key:	111	101	110	101	111	100	000	101	110	000
plaintext:	001	000	010	100	001	010	111	100	000	101
	h	e	i	l	h	i	t	l	e	r

Question 9:

Decrypt the ciphertext

IAUTMOCSMNIMREBOTNELSTRHEREOAEVMWIHTSEEA TMAEOHWHSYCEELTTEO HMUOUFEHTRFT

This message was encrypted with a double transposition using a matrix of 7 rows and 10 columns. Hint: The first word is "there."

I	A	U	T	M	O	C	S	M	N
I	M	R	E	B	O	T	N	E	L
S	T	R	H	E	R	E	O	A	E
V	M	W	I	H	T	S	E	E	A
T	M	A	E	O	H	W	H	S	Y
C	E	E	L	T	T	E	O	H	M
U	O	U	F	E	H	T	R	F	T

First we will try to sort the columns in order to make rows more readable:

2	4	7	6	5	9	3	10	1	8
A	T	C	O	M	M	U	N	I	S
M	E	T	O	B	E	R	L	I	N
T	H	E	R	E	A	R	E	S	O
M	I	S	T	H	E	W	A	V	E
M	E	W	H	O	S	A	Y	T	H
E	L	E	T	T	H	E	M	C	O
O	F	T	H	E	F	U	T	U	R

Then we will try to sort the rows in order to complete the sentence:

T	H	E	R	E	A	R	E	S	O
M	E	W	H	O	S	A	Y	T	H
A	T	C	O	M	M	U	N	I	S
M	I	S	T	H	E	W	A	V	E
O	F	T	H	E	F	U	T	U	R
E	L	E	T	T	H	E	M	C	O
M	E	T	O	B	E	R	L	I	N

Plain text:

There are some who say that communism is the wave of the future, let them come to Berlin.

Question 10:

Letter e h i k l r s t

Binary 000 001 010 011 100 101 110 111

Using the letter encoding in above table the following ciphertext message was encrpted with one time pad

KITLKE

1.If the plaintext is “thrill” what is the key?

2. If the plaintext is “tiller” what is the key?

Plaintext :	t	h	r	i	l	l
	111	001	101	010	100	100
KITLKE	011	010	111	100	011	000
Key	100	011	010	110	111	100

Plaintext :	t	i	l	l	e	r
	111	010	100	100	000	101
KITLKE	011	010	111	100	011	000
Key	100	000	011	000	011	101

HASH FUNCTIONS

Hash Function: It is any functions that takes an input of any size and produce an output of a fixed size. A ***cryptographic hash function h(x)*** must provide all of the following.

• **Compression** — For any size input *x*, the output length of $y = h(x)$ is small. In practice, the output is a fixed size, regardless of the length of the input.

• **Efficiency** — It must be easy to compute $h(x)$ for any input *x*. The computational effort required to compute $h\{x)$ will grow with the length of *x*, but it cannot grow too fast.

• **One-way** — Given any value *y*, it's computationally infeasible to find a value a; such that $h\{x) = y$. Another way to say this is that there is no feasible way to invert the hash.

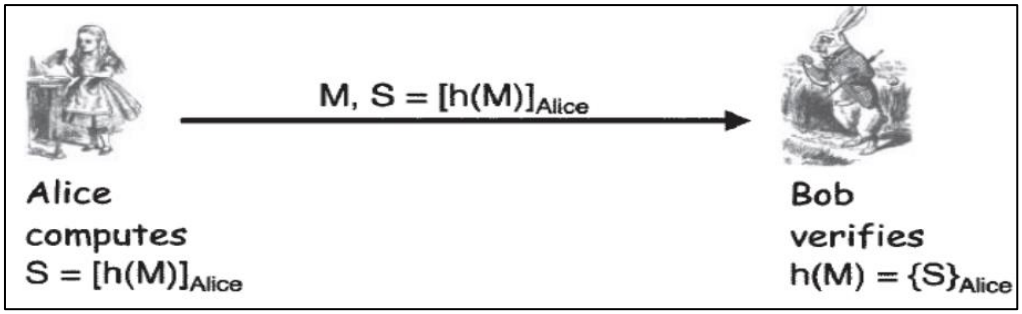
• **Weak collision resistance** — Given *x* and $h(x)$, it's infeasible to find any *y*, with $y \neq x$, such that $h(y) = h(x)$. Another way to state this requirement is that it is not feasible to modify a message without changing its hash value.

• **Strong collision resistance** — It's infeasible to find any *x* and *y*, such that $x \neq y$ and $h(x) = h(y)$. That is, we cannot find any two inputs that hash to the same output.

Hash functions used in the computation of **digital signatures**.

Motivation:

- Alice signs a message *M* by using her private key to "encrypt," that is, she computes $S = [M]_{Alice}$ · If Alice sends *M* and *S* to Bob, then Bob can verify the signature by verifying that $M = \{S\}_{Alice}$, if M is large, $[M]_{Alice}$ is costly to compute.
- If a cryptographic function *h*, Alice will sign *M* by first hashing *M* then signing the hash, that is, Alice computes $S = [h(M)]_{Alice}$. Hashes are efficient (comparable to block cipher algorithms), and only a small number of bits need to be signed.
- Then Alice can send Bob *M* and *S*, as illustrated in Figure. Bob verifies the signature by hashing *M* and comparing the result to the value obtained when Alice's public key is applied to *S*. That is, Bob verifies that $h(M) = \{S\}_{Alice}$·



The Birthday Problem:

- Suppose there are N people in a room.
- How large must N be before the probability someone has same birthday as me is>=1/2?
- Assuming all birth dates are equally likely.
- The probability that person does not have the same probability as you is given as:

1-(1/365)=364/365

- Probability that none of N people have the same birthday as you is

$(364/365)^N$

- The probability that at least one person has the same birthday as you is

$1-(364/365)^N$

- Solving: $1/2 = 1 - (364/365)^N$ for N. Therefore N = 253
- Number of people(N) that must be in a room before probability is ≥ 1/2 that any two (or more) have same birthday is

$1 - (365/365) \cdot (364/365) \cdots (365-N+1)/365 = 1/2$. Therefore N=23

- With N people in a room, the number of comparisons is N^2.
- Since there are 365 different birth dates, a match n be find at the point where $N^2=365$ or $N=\sqrt{365}=19$.
- If h(x) is N bits, then 2^N different hash values are possible.
- So, if you hash about $\text{sqrt}(2^N) = 2^{N/2}$ values then you expect to find a collision
- Secure N-bit hash requires $2^{N/2}$ work to "break" & Secure N-bit symmetric cipher has work factor of 2^{N-1}.

A Birthday Attack

- If M is the message that Alice wants to sign, then she computes $S = [h(M)]_{Alice}$ & sends S and M to Bob.
- Attacker selects an "evil" message E that she wants Alice to sign, but which Alice is unwilling to sign.
- Attacker also creates an innocent message I that she is confident Alice is willing to sign.
- Attacker generates $2^{n/2}$ variants of the innocent message. These innocent messages, which we denote I_i, *all have the same meaning as I, but since the* messages differ, their hash values differ.
- Attacker creates $2^{n/2}$ variants of the evil message, which denoted as E_i. *but their hashes differ.*
- By the birthday problem, attacker can expect to find a collision, $h(E_j) = h(I_k)$.
- Given such a collision, attacker sends I_k to Alice, & asks Alice to sign it.

- Alice signs it and returns I_k *and* $h[I_k]_{Alice}$ *to attacker*.
- *Since* $h(E_j) = h(I_k)$, it follows that $h[E_j]_{Alice} = h[I_k]_{Alice}$. Consequently attacker has obtained Alice's signature on the evil message E_j.
- To prevent this attack, choose a hash function for which *n*, the size of the hash function output, is so large that attacker cannot compute $2^{N/2}$ hashes.

Non-Cryptographic Hashes

- Consider data $X = (X_1,X_2,X_3,\ldots,X_n)$, each X_i is a byte.
- Defining hash function $h(X) = (X_1+X_2+X_3+\ldots+X_n) \bmod 256$. This provides compression, since any size of input is compressed to an 8-bit output. Hash would be easy to break, since the birthday problem tells us that if we hash just $2^4 = 16$ randomly selected inputs, we can expect to find a collision.

  For example: swapping two bytes will always yield a **collision**, such as X = (10101010, 00001111), Hash is h(X) = 10111001. If Y = (00001111, 10101010) then h(X) = h(Y).

  i.e h(10101010,00001111) = h(00001111,10101010) = 10111001.
- Consider data $X = (X_0,X_1,X_2,\ldots,X_{n-1})$
- Suppose hash is defined as $h(X) = (nX_1+(n-1)X_2+(n-2)X_3+\ldots+2\cdot X_{n-1}+X_n) \bmod 256$. It gives different results when the byte order is swapped.

  For example: h(10101010,00001111) ≠ h(00001111,10101010)

- But there exists birthday problem issue and it also happens to be relatively easy to construct collisions.

For example: h(00000001,00001111) = h(00000000,00010001) = 00010001.

- This is not a secure cryptographic hash, it's useful in a particular non-cryptographic application known as **Rsync.**
- Cyclic Redundancy Check is the remainder in a long division calculation, good for detecting burst **errors** and such random errors unlikely to yield a collision.
- CRC has been mistakenly used where crypto integrity check is required (e.g., WEP).

Tiger Hash

- "Fast and strong"
- Designed by Ross Anderson and Eli Biham – leading cryptographers
- Design criteria:

- Secure
- Optimized for 64-bit processors.
- Easy replacement for MD5 or SHA-1.

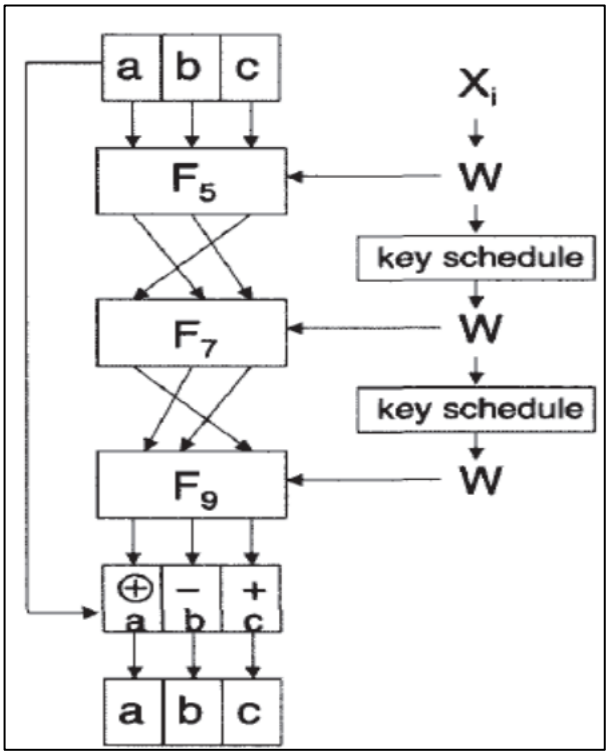
- Input to Hash function is divided into 512 bit blocks (padded).
- Output is **192 bits** (three 64-bit words).
- Intermediate rounds are all 192 bits
- 4 S-boxes(Substitution-box) is used, each maps 8 bits to 64 bits.
- A "key schedule" is used, since there is no key, is applied to the input block.

Tiger Outer Round:

- The input *X* is padded to a multiple of 512 bits and written as

$$X = (X_0, X_1, \ldots, X_{n-1})$$

- Employs one outer round for each X_i
- Initial (a,b,c) constants.
- The final (a, b, c) output from one round is the initial triple for the subsequent round and the final (a, *b*, c) from the final round is the 192-bit hash value.



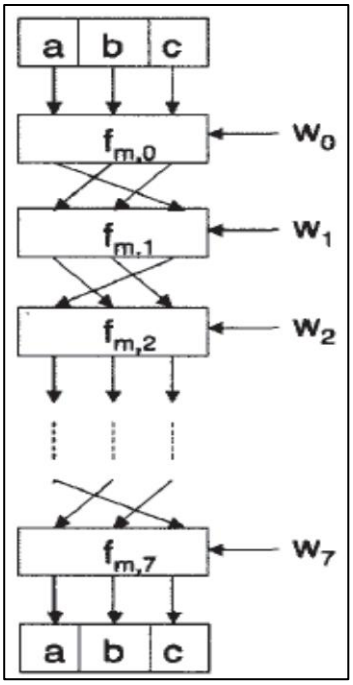
Tiger Outer Round

- In Outer round, input to outer round F5 is (a,b,c).
- The output of F5 as (a,b,c), the input toF7 is (c,a,b), the input to F9 is (b,c,a).
- Each function F_m consists of eight inner rounds.

Tiger Inner Rounds:

- Each F_m consists of precisely 8 inner rounds.
- 512 bit input W to F_m
  - $W=(w_0,w_1,\ldots,w_7)$
  - W is one of the input blocks X_i
- All lines are 64 bits
- The input values for $f_{m,i,}$ for i=0,1,2,…,7 are

$$(a,b,c),(b,c,a),(c,a,b),(a,b,c),(b,c,a),(c,a,b),(a,b,c),(b,c,a)$$



Tiger Inner Round for F_m

Tiger Hash: One Round

- Each $f_{m,i}$ is a function of a,b,c,w_i and m
  - Input values of a,b,c from previous round.
  - And w_i is 64-bit block of 512 bit W.
  - Subscript m is mul tiplier
  - And $c = (c_0,c_1,\ldots,c_7)$
- Output of $f_{m,i}$ is
  - $c = c \oplus w_i$
  - $a = a - (S_0[c_0] \oplus S_1[c_2] \oplus S_2[c_4] \oplus S_3[c_6])$
  - $b = b + (S_3[c_1] \oplus S_2[c_3] \oplus S_1[c_5] \oplus S_0[c_7])$
  - $b = b * m$
- Each S_i is S-box(i.e., lookup table): 8 bits mapped to 64 bits.

Tiger Hash Key Schedule:

- Input is X
  - o $X=(x_0,x_1,\ldots,x_7)$
- Small change in X will produce large change in key schedule output.

$$
\begin{aligned}
w_0 &= w_0 - (w_7 \oplus \texttt{0xA5A5A5A5A5A5A5A5}) \\
w_1 &= w_1 \oplus w_0 \\
w_2 &= x_2 + w_1 \\
w_3 &= w_3 - (w_2 \oplus (\bar{w}_1 \ll 19)) \\
w_4 &= w_4 \oplus w_3 \\
w_5 &= w_5 + w_4 \\
w_6 &= w_6 - (w_5 \oplus (\bar{w}_4 \gg 23)) \\
w_7 &= w_7 \oplus w_6 \\
w_0 &= w_0 + w_7 \\
w_1 &= w_1 - (w_0 \oplus (\bar{w}_7 \ll 19)) \\
w_2 &= w_2 \oplus w_1 \\
w_3 &= w_3 + w_2 \\
w_4 &= w_4 - (w_3 \oplus (\bar{w}_2 \gg 23)) \\
w_5 &= w_5 \oplus w_4 \\
w_6 &= w_6 + w_5 \\
w_7 &= w_7 - (w_6 \oplus \texttt{0x0123456789ABCDEF})
\end{aligned}
$$

Summary:

- Hash and intermediate values are 192 bits.
- 24 (inner) rounds:
  - o **S-boxes:** Claimed that each input bit affects a, b and c after 3 rounds.
  - o **Key schedule:** Small change in message affects many bits of intermediate hash values.
  - o **Multiply:** Designed to ensure that input to S-box in one round mixed into many S-boxes in next.
- S-boxes, key schedule and multiply together designed to ensure strong **avalanche** effect.

Note: A desirable property of any cryptographic hash function is the so-called ***avalanche effect***. The goal is that any small change in the input should cascade and cause a large change in the output.

- At a higher level, Tiger employs
  - o Confusion
  - o Diffusion

HMAC

For message integrity we can compute a message authentication code, or **MAC**, where the MAC is computed using a block cipher in CBC mode. The MAC is the final encrypted block, which is also

known as the ***CBC residue***. Since a hash function effectively gives us a fingerprint of a file, we should also be able to use a hash to verify message integrity.

Motivation:

Consider Alice protect the integrity of M by simply computing *h(M)* and sending both M and *h(M)* to Bob.If *M* changes, Bob will detect the change, provided that *h(M)* has not changed (and vice versa). However, if attacker replaces *M* with *M'* and also replaces *h(M)* with *h(M')*, then Bob will have no way to detect the tampering. But using a hash function to provide integrity protection, involves a key to prevent attacker from changing the hash value.

Approach: Alice encrypt the hash value with a symmetric cipher, *E(h(M),K),* and send this to Bob. A slightly different approach is used to compute a ***hashed MAC,* or HMAC.** Instead of encrypting the hash, directly mix the key into *M* when computing the hash.

Two approaches are to prepend the key to the message, or append the key to the message:

- ***h(K,M)***
- ***h(M,K)***

h(K,M) :

If *h(K,M) is used to* compute an HMAC, then consider cryptographic hashes hash the message in blocks—for MD5, SHA-1, and Tiger, the block size is 512 bits. As a result, if M = *{B1, B2)*, where each *Bi* is 512 bits, then

h(M) = F(F(A, B1),B2) = F(h(B1), B2) .....equation (1)

for some function *F*, where *A* is a fixed initial constant.

For example, in the Tiger hash, the function *F* consists of the outer with each *Bi* corresponding to a 512-bit block of input and *A* corresponding to the 192-bit initial value *{a,b,c).*

If an attacker chooses *M'* so that *M' = (M,X), attacker* might be able to use equation (1) to find *h(K,M')* from *h(K,M)* without knowing *K* since, for *K*, *M*, and *X* of the appropriate size,

$$h(K, M') = h(K, M, X) = F(h(K, M),X)$$

where the function *F* is known.

h(M,K):

If it should happen that there is a known collision for the hash function *h*, that is, if there exists some *M'* with *h(M') = h(M)*, then by equation (1), then

$$h(M, K) = F(h(M), K) = F(h(M'), K) = h(M', K)$$

provided that M and M' are each a multiple of the block size.

Conclusion: If such a collision exists, the hash function is considered insecure.

HMAC:

Approved method to mix the key into the hash for computing an HMAC is as follows.

Let B be the block length of hash, in bytes. For all popular hashes (MD5, SHA-1, Tiger, etc.), $B = 64$. Next, define

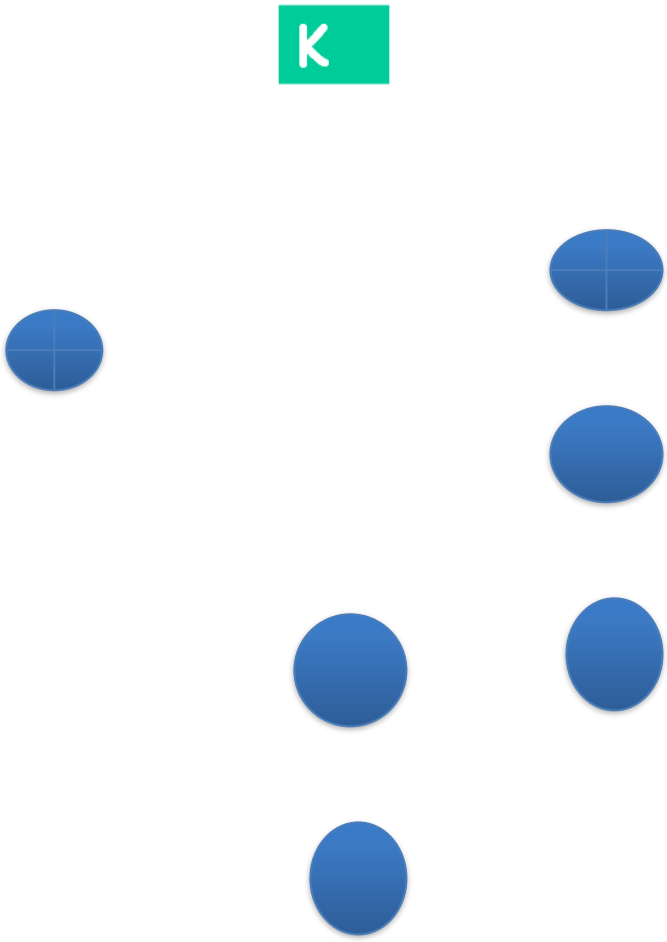
ipad = 0x36 repeated B times

&

opad = 0x5C repeated B times

Then the HMAC of M is defined to be

$$\mathrm{HMAC}(M, K) = H(K \oplus \mathrm{opad}, H(K \oplus \mathrm{ipad}, M))$$



This approach thoroughly mixes the key into the resulting hash. While two hashes are required to compute an HMAC, note that the second hash will be computed on a small number of bits—the output of the first hash with the modified key appended.

5.8 Uses for Hash Functions:

Hash functions include authentication, message integrity (using an HMAC), message fingerprinting, error detection, and digital signature efficiency and can also be used to solve security-related problems.

5.8.1 Online Bids:

- Consider suppose an item is for sale online and Alice, Bob, and Charlie all want to place bids and these are supposed to be sealed bids, that is, each bidder gets one chance to submit a secret bid and only after all bids have been received are the bids revealed.
- The highest bidder wins. Alice, Bob, and Charlie don't necessarily trust each other and they definitely don't trust the online service that accepts the bids.
- Each bidder is concerned that the online service might reveal their bid to the other bidders—either intentionally or accidentally.

For example,

- Suppose Alice places a bid of $10.00 and Bob bids $12.00. If Charlie is able to discover the values of these bids prior to placing his bid (and prior to the deadline for bidding), he could bid $12.01 and win. The point is that nobody wants to be the first (or second) to place their bid, since there might be an advantage to bidding later.

So, the online service proposes the following scheme.

- Each bidder will determine their bids, say, bid *A* for Alice, bid *B* for Bob, and *C* for Charlie, keeping their bids secret.
- Then Alice will submit *h{A),* Bob will submit *h(B),* and Charlie will submit *h(C).*
- Once all three hashed bids have been received, the hash values will be posted online for all to see.
- At this point all three participants will submit their actual bids, that is, *A, B,* and *C.*

Advantage: If the cryptographic hash function is secure, it's one-way, so no disadvantage to submitting a hashed bid prior to a competitor. And since it is infeasible to determine a collision, no bidder can change their bid after submitting their hash value.

i.e, the hash value binds the bidder to his or her original bid, without revealing any information about the bid itself. If there is no disadvantage in being the first to submit a hashed bid, and there is no way to change a bid once a hash value has been submitted, then this scheme prevents the cheating.

Disadvantage: It is subject to a forward search attack.

5.8.2 Spam Reduction

- Spam is defined as unwanted and unsolicited bulk email.
- In this scheme, Alice will refuse to accept an email until she has proof that the sender expended sufficient effort to create the email. Effort will be measured in terms of computing resources, in particular, CPU cycles.
- This scheme would not eliminate spam, but it would limit the amount of such email that any user can send.
- Let *M* be an email message and let *T* be the current time. The message *M* includes the sender's and intended recipient's email addresses, but does not include any additional addresses.
- The sender of message *M* must determine a value *R* such that

$$h(M, R, T) = (\underbrace{00\ldots0}_{N}, X)$$

- That is, the sender must find a value *R* so that the hash in equation (5.5) has zeros in all of its first *N* output bits.
- Once this is done, the sender sends the triple *(M,R,T)*. Before Alice, the recipient, accepts the email, she needs to verify that the time *T* is recent, and that *h(M, R, T)* begins with *N* zeros.
- Again, the sender chooses random values *R* and hashes each until he finds a hash value that begins with TV zeros. Therefore, the sender will need to compute, on average, about 2 hashes.
- The recipient can verify that *h(M, R, T)* begins with *N* zeros by computing a single hash. So the work for the sender (measured in terms of hashes) is about 2^N, while the work for the recipient is always a single hash.
- In this scheme, we would need to choose *N* so that the work level is acceptable for normal email users but unacceptably high for spammers.
- It might also be possible for users to select their own individual value of *N* to match their personal tolerance for spam.

For example, if Alice hates spam, $N = 40$. While this would deter spammers, it might also deter many legitimate email senders. If Bob, doesn't mind receiving some spam and he never wants to deter a legitimate email sender, he might set his value to, $N = 10$. Spammers dislikes this scheme. Legitimate bulk emailers also might not like this scheme, since they would need to spend resources (i.e., money) to compute vast numbers of hashes.

5.9 Miscellaneous Crypto-Related Topics

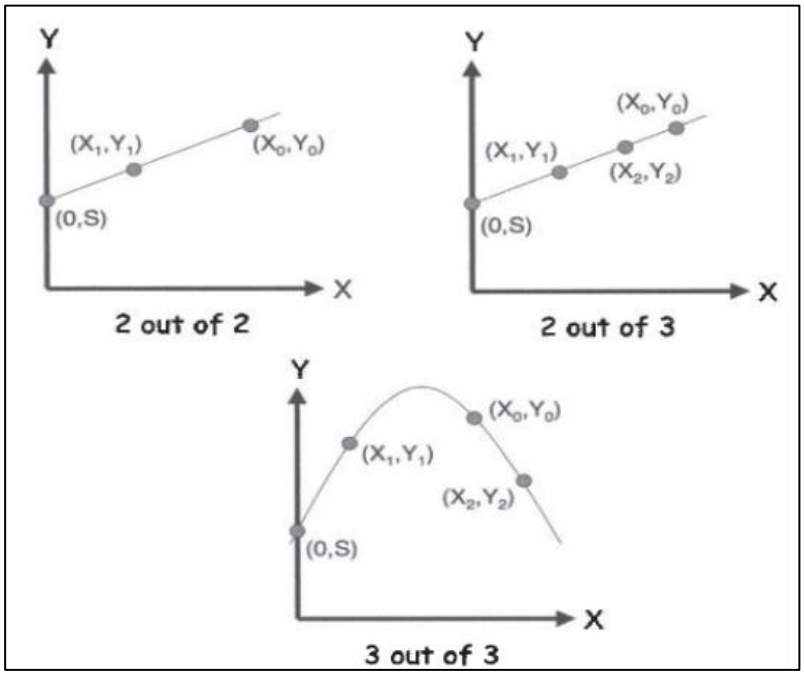
5.9.1 Secret Sharing

Suppose Alice and Bob want to share a secret *S* in the sense that:

• Neither Alice nor Bob alone (nor anyone else) can determine *S* with a probability better than guessing.

• Alice and Bob together can easily determine *S.*

Suppose the secret *S* is a real number. Draw a line *L* in the plane through the point (0, *S)* and give Alice a point $A = (X_0, Y_0)$ on *L* and give Bob another point $B = (X_1, Y_1)$, which also lies on the line *L*. Then neither Alice nor Bob individually has any information about *S*, since an infinite number of lines pass through a single point. But together, the two points *A* and *B* uniquely determine *L*, and therefore the y-intercept, and hence the value *S.* This example is illustrated in the "2 out of 2" scheme which is given below:



Secret Sharing Schemes

It's easy to extend this idea to an "m out of n" secret sharing scheme, for any $m < n$, where *n* is the number of participants, any *m* of which can cooperate to recover the secret. For $m = 2$, a line always works. For example, a "2 out of 3" scheme appears in Figure.

A line, which is a polynomial of degree one, is uniquely determined by two points, whereas a parabola, which is a polynomial of degree two, is uniquely determined by three points. In general, a polynomial of degree m -1 is uniquely determined by m points. This elementary fact is allows us to construct an m out of *n* secret sharing scheme for any $m < n$. For example, a "3 out of 3" scheme is illustrated in Figure.

5.9.1.1 Key Escrow

One particular application where secret sharing would be useful is in the *key escrow* problem . Suppose that we require users to store their keys with an official escrow agency. The government could then get access to keys as an aid to criminal investigations.

- One concern with key escrow is that the escrow agency might not be trustworthy.

- It is possible to ameliorate this concern by having several escrow agencies and allow users to split the key among *n* of these, so that m of the *n* must cooperate to recover the key.
- Alice could select escrow agencies that she considers most trustworthy and have her secret split among these using an m out of *n* secret sharing scheme.

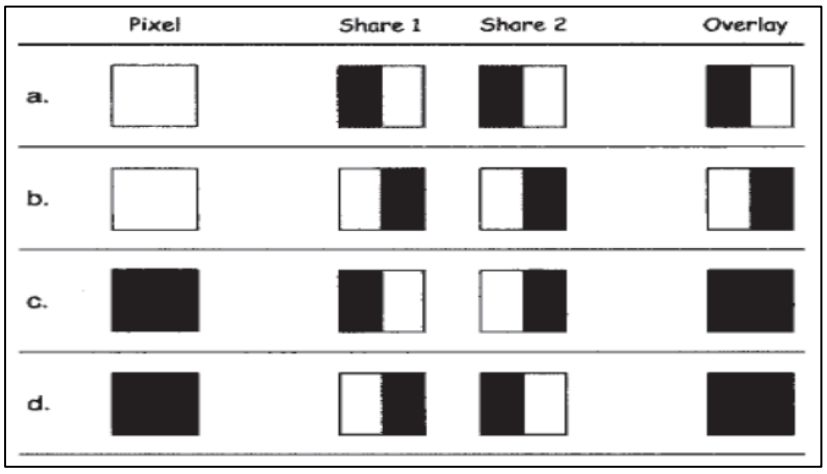
Shamir's secret sharing scheme could be used to implement such a key escrow scheme.

For example, suppose $n = 3$ and m = 2 and Alice's key is *S.* Then the "2 out of 3" scheme illustrated in above Figure could be used.

Alice might choose to have the Department of Justice hold the point = (X_0, Y_0), the Department of Commerce hold = (X_1, Y_1), and Fred's Key Escrow, Inc., hold (X_2,Y_2)· Then at least two of these three escrow agencies would need to cooperate to determine Alice's key *S.*

5.9.1.2 Visual Cryptography

- Visual secret sharing scheme is absolutely secure, as is the polynomial-based secret sharing scheme.
- In visual secret sharing (aka visual cryptography), no computation is required to decrypt the underlying image.
- In the simplest case, we start with a black-and-white image and create two transparencies, one for Alice and one for Bob.
- Each individual transparency appears to be a collection of random black and white subpixels, but if Alice and Bob overlay their transparencies, the original image appears (with some loss of contrast).
- In addition, either transparency alone yields no information about the underlying image.



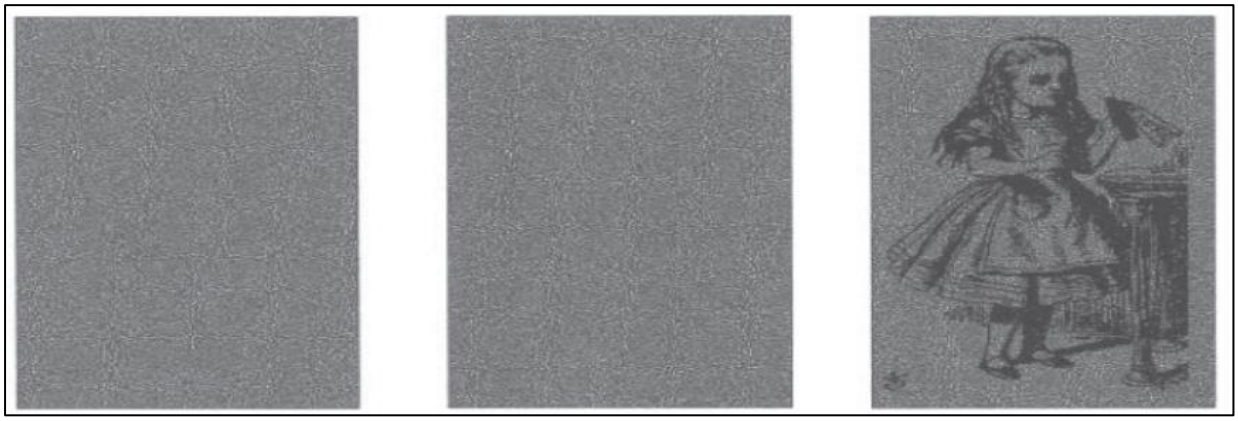
Pixel shares

- Above figure shows various ways that an individual pixel can be split into "shares," where one share goes to Alice's transparency and the corresponding share goes to Bob's.

For example,

- If a specific pixel is white, then we can flip a coin to decide whether to use row "a" or row "b" from above Figure. Then, Alice's transparency gets share 1 from the selected row (either a or b), while Bob's transparency gets share 2.
- The shares are put in Alice's and Bob's transparencies at the same position corresponding to the pixel in the original image. When Alice's and Bob's transparencies are overlaid, the resulting pixel will be half-black/half-white.
- In the case of a black pixel, we flip a coin to select between rows "c" and "d" and we again use the selected row to determine the shares.
- If the original pixel was black, the overlaid shares always yield a black pixel.
- If the original pixel was white, the overlaid shares will yield a half-white/half-black pixel, which will be perceived as gray.
- This results in a loss of contrast (black and gray versus black and white), but the original image is still clearly discernible.

For example, Below Figure illustrates a share for Alice and a share for Bob, along with the resulting overlaying of the two shares.



Alice's Share, Bob's Share, and Overlay Image

5.9.2 Random Numbers

In cryptography, random numbers are needed to generate symmetric keys, RSA key pairs (i.e., randomly selected large primes), and Diffie-Hellman secret exponents as well as in security protocols.

Random numbers are used in many non-security applications such as simulations and various statistical applications. In such cases, the random numbers usually only need to be statistically random.

Cryptographic random numbers must be statistically random and they must be unpredictable.

Consider the following example:

Suppose that a server generates symmetric keys for users. Suppose the following keys are generated for the listed users:

• K_A for Alice

• K_B for Bob

• K_C for Charlie

• K_D for Dave

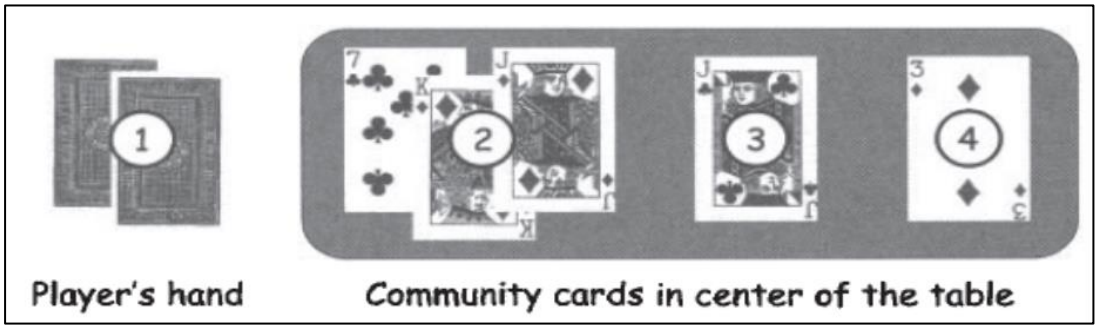
Now, if Alice, Bob, and Charlie don't like Dave, they can pool their information to see if it will help them determine Dave's key. That is, Alice, Bob, and Charlie could use knowledge of their keys, K_A, K_B, and K_C, to see if it helps them determine Dave's key K_D. If K_D can be predicted from knowledge of the keys K_A, K_B, and K_C, then the security of the system is compromised.

Commonly used pseudo-random number generators are predictable. Consequently, pseudo-random number generators are not appropriate for cryptographic applications.

5.9.2.1 Texas Hold 'em Poker

Consider a real-world example that illustrates the wrong way to generate random numbers:

- ASF Software, Inc., developed an online version of the card game known as Texas Hold 'em Poker .
- In this game, each player is first dealt two cards, face down. Then a round of betting takes place, followed by three community cards being dealt face up—all players can see the community cards and use them in their hand.
- After another round of betting, one more community card is revealed, then another round of betting.
- Finally, a final community card is dealt, after which additional betting can occur. Of the players who remain at the end, the winner is the one who can make the best poker hand from his two cards together with the five community cards.
- The game is illustrated in the below Figure.



Texas Hold 'Em Poker

- In this game, random numbers are required to shuffle a virtual deck of cards.

- The AFS poker software had a serious flaw in the way that random numbers were used to shuffle the deck of cards.
- As a result, the program did not produce a truly random shuffle, and it was possible for a player to determining the entire deck in real time.
- A player who could take advantage of this flaw could cheat, since he would know all of the other players' hands, as well as the future community cards before they were revealed.

How was it possible to determine the shuffle?

First, there are $52! > 2^{225}$ distinct shuffles of a 52-card deck.

- The AFS poker program used a "random" 32-bit integer to determine the shuffle. Consequently, the program could generate no more than 2^{32} different shuffles out of the more than 2^{225} possible. This was an inexcusable flaw.
- To generate the "random" shuffle, the program used the pseudo-random number generator, or PRNG, built into the Pascal programming language.
- The PRNG was reseeded with each shuffle, with the seed value being a known function of the number of milliseconds since midnight. Since the number of milliseconds in a day is less than 2^{27} distinct shuffles could actually occur.
- Trudy, the attacker, if she synchronized her clock with the server, Trudy could reduce the number of shuffles that needed to be tested to less than 2^{18}.
- These 2^{18} possible shuffles could all be generated in real time and tested against the community cards to determine the actual shuffle for the hand currently in play.
- After the first set of community cards were revealed, Trudy could determine the shuffle uniquely and she would then know the final hands of all other players—even before any of the other players knew their own final hand!

The AFS Texas Hold 'em Poker program is an extreme example of the ill effects of using predictable random numbers where unpredictable random numbers are required. In this example, the number of possible random shuffles was so small that it was possible to determine the shuffle and thereby break the system.

How can we generate cryptographic random numbers?

Since a secure stream cipher keystream is not predictable, the keystream generated by, say, the RC4 cipher must be a good source of cryptographic random numbers.

5.9.2.2 Generating Random Bits

- True randomness is difficult to achieve. The concept of *entropy*, as developed by Claude Shannon explains Entropy is a measure of the uncertainty or, conversely, the predictability of a sequence of bits.
- Sources of true randomness do exist.

For example,

- Radioactive decay is random.
- Hardware devices are available that can be used to gather random bits based on various physical and thermal properties that are known to be unpredictable.
- Another source of randomness is the lava lamp, which achieves its randomness from its chaotic behaviour.
- Since software is deterministic, true random numbers must be generated external to any code.
- Reasonable sources of randomness include mouse movements, keyboard dynamics, certain network activity, and so on.

5.9.3 Information Hiding

The two methods of information hiding, namely,

- Steganography and
- Digital watermarking.

➢ Steganography, or hidden writing, is the attempt to hide the fact that information is being transmitted.

➢ Watermarks generally involve hidden information, but for a slightly different purpose.

For example,

A copyright holder might hide a digital watermark (containing some identifying information) in digital music is a effort to prevent music piracy.

Digital watermarks: Digital watermarks can be categorized in many different ways.

• *Invisible* — Watermarks that are not supposed to be perceptible in the media.

• *Visible* — Watermarks that are meant to be observed, such as a stamp of TOP SECRET on a document.

Watermarks can also be categorized as follows:

• *Robust* — Watermarks that are designed to remain readable even if they are attacked.

• *Fragile* — Watermarks that are supposed to be destroyed or damaged if any tampering occurs.

For example,

1) Insert a robust invisible mark in digital music in the hope of detecting piracy. Then when pirated music appears on the Internet, we can trace it back to its source. Or we might insert a fragile

invisible mark into an audio file. In this case, if the watermark is unreadable, the recipient knows that tampering has occurred. This latter approach is essential an integrity check.

2) Many modern currencies include (non-digital) watermarks. Several current and recent U.S. bills, including the $20 bill pictured in below figure visible watermarks. In this $20 bill, the image of President Jackson is embedded in the paper itself (in the right-hand section of the bill) and is visible when held up to a light. This visible watermark is designed to make counterfeiting more difficult, since special paper is required to duplicate this easily verified watermark.

Watermarked Currency

Example of a simple approach to steganography:

This particular example is applicable to digital images.

Consider images that employ the well-known 24 bits color scheme— one byte each for red, green, and blue, denoted R, G, and B, respectively.

For example,1) the color represented by (R, G, B) = (0x7E, 0x52,0x90) is much different than (R, G,B) = (OxFE, 0x52,0x90), even though the colors only differ by one bit.

On the other hand, the color (R, G, B) = (OxAB, 0x33, OxFO) is indistinguishable from (R, G,B) = (OxAB, 0x33, OxF1), yet these two colors also differ by only a single bit.

The low-order RGB bits are unimportant, since they represent imperceptible changes in color. Since the low-order bits don't matter, it can use them for any purposes we choose, including information hiding.

2) Consider the two images of Alice in the below Figure. The left-most Alice contains no hidden information, whereas the right-most Alice has the entire *Alice in Wonderland* book (in PDF format) embedded in the low-order RGB bits.

- ➢ To the human eye, the two images appear identical at any resolution. If we compare the bits in these two images, the differences would be obvious.
- ➢ In particular, it's easy for an attacker to write a computer program to extract the low-order RGB bits—or to overwrite the bits with garbage and thereby destroy the hidden information, without doing any damage to the image.
- ➢ It is difficult to apply Kerckhoffs' Principle for this example.

A Tale of Two Alices

3) Consider an HTML file that contains the following text, taken from the well-known poem "To talk of many things

Of shoes and ships and sealing wax

Of cabbages and kings

And why the sea is boiling hot

And whether pigs have wings."

In HTML, the RGB font colors are specified by a tag of the form ... where rr is the value of R in hexadecimal, gg is G in hex, and bb is B in hex.

For example, the color black is represented by #000000, whereas white is #FFFFFF.

Since the low-order bits of R, G, and B won't affect the perceived color, we can hide information in these bits, as shown in the HTML snippet in the Table . Reading the low-order bits of the RGB colors yields the "hidden" information 011 010 100 100 000 101.

Table : Simple Steganography Example-

```
<font color="#010100">"The time has come,"
the Walrus said,</font><br>
<font color="#000100">"To talk of many things :</fontxbr>
<font color="#010100">0f shoes and ships and sealing wax</font><br>
<font color="#000101">0f cabbages and kings</fontxbr>
<font color="#000000">And why the sea is boiling hot</font><br>
<font color="#010001">And whether pigs have wings."</font><br>
```

- Hiding information in the low-order RGB bits of HTML color tags is obviously not as impressive as hiding *Alice in Wonderland* in Alice's image.

- This method is not at all robust—an attacker who knows the scheme can read the hidden information as easily as the recipient.
- Or an attacker could instead destroy the information by replacing the file with another one that is identical, except that the low-order RGB bits have been randomized.

The conclusion here is that for information hiding to be robust, the information must reside in bits that do matter. But this creates a serious challenge, since any changes to bits that do matter must be done very carefully for the information hiding to remain "invisible."

As noted above, if Trudy knows the information hiding scheme, she can recover the hidden information as easily as the intended recipient. Watermarking schemes therefore generally encrypt the hidden information before embedding it in a file. But even so, if Trudy understands how the scheme works, she can almost certainly damage or destroy the information.

Watermarking schemes often use spread spectrum techniques to better hide the information-carrying bits.

Entity Authentication

Entity Authentication is one of the security service. Many cryptographic entity authentication mechanisms rely on randomly generated numbers.

3.1 Random Number Generation:

Many cryptographic primitives cannot function securely without randomness

3.1.1 The need for randomness:

- Most cryptographic primitives take structured input and turn it into something that has no structure.
- Many cryptographic primitives *require* sources of randomness in order to function.

3.1.2 What is randomness?

Randomness is about ideas such as 'unpredictability' and 'uncertainty'. A random number generation process is often assessed by applying a series of statistical tests.

3.1.3 Non-deterministic generators

There are two general approaches to generating randomness.

A *non-deterministic generator* is based on the randomness produced by physical phenomena and therefore provides a source of 'true randomness' in the sense that the source is very hard to control and replicate. Non-deterministic generators can be based on hardware or software.

- **HARDWARE-BASED NON-DETERMINISTIC GENERATORS**

  Hardware-based non-deterministic generators rely on the randomness of physical phenomena. Generators of this type require specialist hardware. Examples include:

➢ measurement of the time intervals involved in radioactive decay of a nuclear atom;

➢ semiconductor thermal (Johnson) noise, which is generated by the thermal motion of electrons;

➢ instability measurements of free running oscillators;

➢ white noise emitted by electrical appliances;

➢ quantum measurements of single photons reflected into a mirror.

Hardware-based generators provide a continuous supply of randomly generated output for as long as the power required to run the generator lasts, or until the process ceases

to produce output. However, because specialist hardware is required, these types of generator are relatively expensive.

- **SOFTWARE-BASED NON-DETERMINISTIC GENERATORS**

Software-based non-deterministic generators rely on the randomness of physical phenomena detectable by the hardware contained in a computing device.

Examples include:

- ➢ capture of keystroke timing;
- ➢ outputs from a system clock;
- ➢ hard-drive seek times;
- ➢ capturing times between interrupts (such as mouse clicks);
- ➢ mouse movements;
- ➢ computations based on network statistics.

These sources of randomness are cheaper, faster and easier to implement than hardware-based techniques.

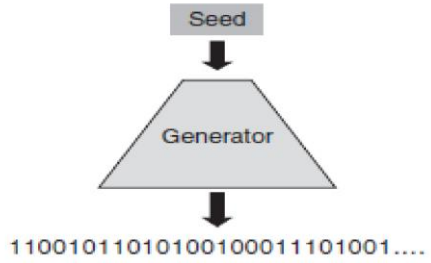
Disadvantage: Two problems with non-deterministic generators:

1. They tend to be expensive to implement.
2. It is, essentially, impossible to produce two identical strings of true randomness in two different places.

3.1.4 Deterministic generators

BASIC MODEL OF A DETERMINISTIC GENERATOR

- A *deterministic generator* is an algorithm that outputs a pseudorandom bit string, in other words a bit string that has no apparent structure.
- The output of a deterministic generator is certainly not randomly generated. If anyone who knows the information that is input to the deterministic generator can completely predict the output.
- If input into the deterministic generator is secret then, with careful design of the generation process, generated output will have no apparent structure.
- It will thus *appear* to have been randomly generated to anyone who does not know the secret input. This is precisely the idea behind a deterministic generator.
- The basic model of a deterministic generator is shown in the below Figure



Basic model of a deterministic generator

The two components of this model are:

A seed. The secret information that is input into the deterministic generator is often referred to as a *seed*. This is essentially a cryptographic key. The seed is the only piece of information that is definitely not known to an attacker. Thus, to preserve the unpredictability of the pseudorandom output sequence it is important both to protect this seed and to change it frequently.

The generator. This is the cryptographic algorithm that produces the pseudorandom output from the seed.

DETERMINISTIC GENERATORS IN PRACTICE

A deterministic generator overcomes the two problems that we identified for non-deterministic generators:

1. They are cheap to implement and fast to run. It is no coincidence that deterministic generators share these advantages with stream ciphers, since the keystream generator for a stream cipher is a deterministic generator whose output is used to encrypt

2. Two identical pseudorandom outputs can be produced in two different locations. All that is needed is the same deterministic generator and the same seed.

The seed is relatively short. It is normally a symmetric key of a standard recommended length, such as 128 bits.

Properties of non-deterministic and deterministic generators:

Non-deterministic generators	Deterministic generators
Close to truly randomly generated output	Pseudorandom output
Randomness from physical source	Randomness from a (short) random seed
Random source hard to replicate	Random source easy to replicate
Security depends on protection of source	Security depends on protection of seed
Relatively expensive	Relatively cheap

Cryptanalysis of the generator. Deterministic generators are cryptographic algorithms and are always vulnerable to potential weaknesses in their design. Use of a well-respected deterministic generator is probably the best way of reducing associated risks.

Seed management. If the same seed is used twice with the same input data then the same pseudorandom output will be generated. Thus seeds need to be regularly updated and managed. The management of seeds brings with it most of the same challenges as managing keys, and presents a likely target for an attacker of a deterministic generator.

3.2 Providing freshness

- *Freshness mechanisms* are techniques that can be used to provide assurance that a given message is 'new', in the sense that it is not a *replay* of a message sent at a previous time.
- The main threat that such mechanisms are deployed against is the capture of a message by an adversary, who then later replays it at some advantageous time.
- Freshness mechanisms are particularly important in the provision of security services that are time-relevant, of which one of the most important is entity authentication.

There are three common types of freshness mechanism.

3.2.1 Clock-based mechanisms

- A *clock-based* freshness mechanism is a process that relies on the generation of some data that identifies the time that the data was created. This is sometimes referred to as a *timestamp*.
- This requires the existence of a clock upon which both the creator of the data and anyone checking the data can rely.

For example, suppose that Alice and Bob both have such a clock and that Alice includes the time on the clock when she sends a message to Bob. When Bob receives the message, he checks the time on his clock and if it 'matches' Alice's timestamp then he accepts the message as fresh.

Clock-based freshness mechanisms seem a natural solution, however, they come with four potential implementation problems:

1. **Existence of clocks**. Alice and Bob must have clocks. For many devices, such as personal computers and mobile phones, this is quite reasonable. It may not be so reasonable for other devices, such as certain types of smart token.

2. **Synchronisation**.

➢ Alice's and Bob's clocks need to be reading the same time, or sufficiently close to the same time.

➢ The clocks on two devices are unlikely to be perfectly synchronised since clocks typically suffer from *clock drift*.

➢ Even if they drift by a fraction of one second each day, this drift steadily accumulates.

Solution:

- One solution might be to only use a highly reliable clock, for example one based on a widely accepted time source such as universal time.
- Another solution might be to regularly run a resynchronisation protocol.
- Next solution is to define a window of time within which a timestamp will be accepted. Deciding on the size of this *window of acceptability* is application dependent and represents a tradeoff parameter between usability and security.

3. **Communication delays**. It is inevitable that there will be some degree of communication delay between Alice sending, and Bob receiving, a message.
4. **Integrity of clock-based data**. Bob will normally require some kind of assurance that the timestamp received from Alice is correct. This can be provided by conventional cryptographic means, for example using a MAC or a digital signature. However, such an assurance can only be provided when Bob has access to the cryptographic key required to verify the timestamp.

3.2.2 Sequence numbers

In applications where clock-based mechanisms are not appropriate, an alternative mechanism is to use ***logical***time. Logical time maintains the order in which messages or sessions occur and is normally instantiated by a ***counter* or *sequence number***.

An example: Suppose Alice and Bob regularly communicate with one another and wish to ensure that messages that they exchange are fresh. Alice can do this by maintaining two sequence numbers for communicating with Bob, which are counters denoted by $N_{AB}$and N_{BA}. Alice uses sequence number N_{AB}as a counter for messages that she sends to Bob, and

sequence number N_{BA}as a counter for messages that she receives from Bob. Both sequence numbers work in the same way.

When Alice sends a message to Bob:

1. Alice looks up her database to find the latest value of the sequence number N_{AB}. Suppose that at this moment in time N_{AB}= *Tnew* .

2. Alice sends her message to Bob along with the latest sequence number value, which is *Tnew* .

3. Alice increments the sequence number $N_{AB}$by one (in other words, she sets N_{AB}= *Tnew* + 1) and stores the updated value on her database. This updated value will be the sequence number that she uses next time that she sends a message to Bob.

When Bob receives the message from Alice:

4. Bob compares the sequence number *Tnew* sent by Alice with the most recent value of the sequence number N_{AB}on his database. Suppose this is N_{AB}= *Told* .

5. If *Tnew* > *Told* then Bob accepts the latest message as fresh and he updates his stored value of $N_{AB}$from *Told* to *Tnew* .

6. If *Tnew* ≤ *Told* then Bob rejects the latest message from Alice as not being fresh.

Sequence numbers address the four concerns that are raised with clock-based mechanisms:

1. **Existence of clocks**. The communicating parties no longer require clocks.
2. **Synchronisation**. In order to stay synchronised, communicating parties need to maintain a database of the latest sequence numbers.
3. **Communication delays**. These only apply if messages are sent so frequently that there is a chance that two messages arrive at the destination in the reverse order to which they were sent. If this is a possibility then there remains a need to maintain the equivalent of a window of acceptability, except that this will be measured in terms of acceptable sequence number differences, rather than time.

For example, Bob might choose to accept the message as fresh not just if *Tnew* > *Told* , but *also* if *Tnew* = *Told* , since there is a chance that the previousmessage from Alice to Bob has not yet arrived. This issue is notrelevant if either:

- delays of this type are not likely (or are impossible);
- Bob is more concerned about the possibility of replays than the implications of rejecting genuine messages.

4. **Integrity of sequence numbers**. Just as for clock-based time, an attacker who can freely manipulate sequence numbers can cause various problems in any protocol that relies on them. Thus sequence numbers should have some level of cryptographic integrity protection when they are sent.

3.2.3 Nonce-based mechanisms

- One problem that is shared by both clock-based mechanisms and sequence numbers is the need for some integrated infrastructure. This problem solved using a mechanism called *Nonce-based* mechanisms.
- Their only requirement is the ability to generate *nonces* (literally, 'numbers used only once'), which are randomly generated numbers for one-off use.
- The general principle is that Alice generates a nonce at some stage in a communication session (protocol).
- If Alice receives a subsequent message that contains this nonce then Alice has assurance that the new message is fresh, whereby 'fresh' we mean that the received message must have been created *after* the nonce was generated.

Suppose that Alice generates a nonce and then sends it in the clear to Bob. Suppose then that Bob sends it straight back. Consider the following three claims about this simple scenario:

Alice cannot deduce anything from such a simple scenario:

- This is not true, although it is true that she cannot deduce very much. She has just received a message consisting of a nonce from someone. It could be from anyone.
- However, it consists of a nonce that she has just generated. This means is that it is virtually certain that whoever sent the nonce back to her (and it might not have been Bob) must have seen the nonce that Alice sent to Bob.
- In other words, this message that Alice has just received was almost certainly sent by someone *after* Alice sent the nonce to Bob.
- In other words, the message that Alice has just received is not authenticated, but it is fresh.

There is a chance that the nonce could have been generated before:

- This is true, there is a 'chance', but if we assume that the nonce has been generated using a secure mechanism and that the nonce is allowed to be sufficiently large then it is a very small chance.

- This is the same issue that arises for any cryptographic primitive. If Alice and Bob share a symmetric key that was randomly generated then there is a 'chance' that an adversary could generate the same key and be able to decrypt ciphertexts that they exchange.
- We can guarantee that by generating the nonce using a secure mechanism, the chance of the nonce having been used before is so small that we might as well forget about it.

Since a nonce was used, Bob is sure that the message from Alice is fresh.

- This is not true, he certainly cannot. As far as Bob is concerned, this nonce is just a number. It could be a copy of a message that was sent a few days before.
- Since Bob was not looking over Alice's shoulder when she generated the nonce, he gains no freshness assurance by seeing it.
- If Bob has freshness requirements of his own then he should also generate a nonce and request that Alice include it in a later message to him.

Nonce-based mechanisms, come with two costs:

1. Any entity that requires freshness needs to have access to a suitable generator, which is not the case for every application.

2. Freshness requires a minimum of two message exchanges, since it is only obtained when one entity receives a message back from another entity to whom they earlier sent a nonce. In contrast, clock-based mechanisms and sequence numbers can be used to provide freshness directly in one message exchange.

3.2.4 Comparison of freshness mechanisms

	Clock-based	Sequence numbers	Nonce-based
Synchronisation needed?	Yes	Yes	No
Communication delays	Window needed	Window needed	Window needed
Integrity required?	Yes	Yes	No
Minimum passes needed	1	1	2
Special requirements	Clock	Sequence database	Random generator

3.3 Fundamentals of entity authentication

- Entity authentication is the assurance that a given entity is involved and currently active in a communication session. This means that entity authentication really involves assurance of both:

Identity. the identity of the entity who is making a claim to be authenticated;

Freshness. that the claimed entity is 'alive' and involved in the current session.

- If we fail to assure ourselves of freshness then we could be exposed to *replay attacks*, where an attacker captures information used during an entity authentication session and replays it a later date in order to falsely pass themselves off as the entity whose information they 'stole'.
- If entity authentication is only used to provide assurance of the identity of one entity to another (and not vice versa) then we refer to it as *unilateral* entity authentication.
- If both communicating entities provide each other with assurance of their identity then we call this *mutual* entity authentication.

3.3.1 A problem with entity authentication

Entity authentication is a security service that is only provided for an 'instant in time'. It establishes the identity of a communicating entity at a specific moment, but just seconds later that entity could be replaced by another entity, and we would be none the wiser.

Consider the following very simple attack scenario:

- Alice walks up to an ATM, inserts her payment card and is asked for her PIN. Alice enters her PIN. This is an example of entity authentication since the card/PIN combination is precisely the information that her bank is using to 'identify' Alice.
- As soon as the PIN is entered, Alice is pushed aside by an attacker who takes over the communication session and proceeds to withdraw some cash.
- The communication session has thus been 'hijacked'. Note that there was no failure of the entity authentication mechanism in this example.
- The only 'failure' is that it is assumed that the communication that takes place just a few seconds after the entity authentication check is still with the entity who successfully presented their identity information to the bank via the ATM.
- This instantaneous aspect of entity authentication might suggest that for important applications we are going to have to conduct almost continuous entity authentication in order to have assurance of the identity of an entity over a longer period of time.
- In the case of the ATM, we would thus have to request Alice to enter her PIN every time she selects an option on the ATM. This will really annoy Alice and does not even protect against the above attack, since the attacker can still push Alice aside at the end of the transaction and steal her money.

The solution is to **combine entity authentication** with the establishment of a **cryptographic key.**

3.3.2 Applications of entity authentication

Entity authentication tends to be employed in two types of situation:

Access control. Entity authentication is often used to directly control access to either physical or virtual resources. An entity, sometimes in this case a human user, must provide assurance of their identity in real time in order to have access. The user can then be provided with access to the resources immediately following the instant in time that they are authenticated.

As part of a more complex cryptographic process. Entity authentication is also a common component of more complex cryptographic processes, typically instantiated by a cryptographic protocol. In this case, entity authentication is normally established at the start of a connection. An entity must provide assurance of their identity in real time in order for the extended protocol to complete satisfactorily.

3.3.3 General categories of identification information

One of the prerequisites for achieving entity authentication is that there is some means of providing information about the identity of a *claimant* (the entity that we are attempting to identify). There are several different general techniques :

- Providing identity information is not normally enough to achieve entity authentication. Entity authentication also requires a notion of freshness.
- Different techniques for providing identity information can be, and often are, combined in real security systems.
- Cryptography has a dual role in helping to provide entity authentication:
    i. Some of these approaches involve identity information that may have little to do with cryptography (such as possession of a token or a password). Cryptography can still be used to support these approaches. For example: cryptography can play a role in the secure storage of passwords.
    ii. Almost all of these approaches require a cryptographic protocol as part of their implementation.

Something The Claimant Has:

For human users, identity information can be based on something physically held by the user. This technique can also be used for providing identity information in the electronic world. Examples of mechanisms of this type include:

Dumb tokens:

Dumb means a physical device with limited memory that can be used to store identity information. Dumb tokens normally require a reader that extracts the identity information from the token and then indicates whether the information authenticates the claimant or not.

One example of a dumb token is a plastic card with a magnetic stripe. The security of the card is based entirely on the difficulty of extracting the identity information from the magnetic stripe, a reader that can extract or copy this information. Hence this type of dumb token is quite insecure.

- ➢ In order to enhance security, it is common to combine the use of a dumb token with another method of providing identification, such as one based on something the user knows.

For example, in the banking community plastic cards with magnetic stripes are usually combined with a PIN, which is a piece of identity information that is required for entity authentication but that is not stored on the magnetic stripe.

Smart cards:

A smart card is a plastic card that contains a chip, which gives the card a limited amount of memory and processing power. The advantage of this over a dumb token is that the smart card can store secret data more securely and can also conduct cryptographic computations. However, like dumb tokens, the interface with a smart card is normally through an external reader.

Smart cards are widely supported by the banking industry, where most payment cards now include a chip as well as the conventional magnetic stripe Smart cards are also widely used for other applications, such as electronic ticketing, physical access control, identity cards etc.

Smart tokens:

Smart tokens have their own user interface. This can be used, for example, to enter data such as a challenge number, for which the smart token can calculate a cryptographic response.

All types of smart token (including smart cards) require an interface to a computer system of some sort. This interface could be a human being or a processor connected to a reader. As

with dumb tokens, smart tokens are often implemented alongside another identification method, typically based on something that the user knows.

SOMETHING THE CLAIMANT IS

The field of *biometrics* is devoted to developing techniques for user identification that are based on physical characteristics of the human body.

A biometric mechanism typically converts a physical characteristic into a digital code that is stored on a database. When the user is physically presented for identification, the physical characteristic is measured by a reader, digitally encoded, and then compared with the template code on the database. Biometric measurements are often classified as either being:

Static, because they measure unchanging features such as fingerprints, hand geometry, face structure, retina and iris patterns.

Dynamic, because they measure features that (slightly) change each time that they are measured, such as voice, writing and keyboard response times.

SOMETHING THE CLAIMANT KNOWS

Basing identity information, at least partially, on something that is known to the claimant is a very familiar technique. Common examples of this type of identity information include PINs, passwords and passphrases. This is the technique most immediately relevant to cryptography since identity information of this type, as soon as it is stored anywhere on a device, shares many of the security issues of a cryptographic key.

In many applications, identity information of this type often *is* a cryptographic key. Strong cryptographic keys are usually far too long for a human user to remember and hence 'know'. This can be some good news and some potentially bad news concerning the use of cryptographic keys as identity information:

1. Most information systems consist of networks of devices and computers. These machines are much better at 'remembering' cryptographic keys than humans!

   Thus, if the claimant is a machine then it is possible that a cryptographic key can be something that is 'known'.
2. Where humans are required to 'know' a cryptographic key, they normally activate the key by presenting identity information that is easier to remember such as a PIN, password or passphrase. This reduces the effective security of that cryptographic key from that of the key itself to that of the shorter information used to activate it.

3.4 Passwords

3.4.1 Problems with passwords

Length.: Since passwords are designed to be memorised by humans, there is a natural limit to the length that they can be. This means that the *password space* (all possible passwords) is limited in size, thus restricting the amount of work required for an exhaustive search of all passwords.

Complexity.

- The full password space is rarely used in applications because humans find randomly generated passwords hard to remember. As a result we often work from highly restricted password spaces, which greatly reduces the security. This makes dictionary attacks possible, where an attacker exhaustively tries all the 'likely' passwords and hopes to eventually find the correct one.
- Clever pneumonic techniques can slightly increase the size of a usable password space.
- Moving from passwords to passphrases can improve this situation by significantly increasing the password space.

Repeatability:

- For the lifetime of a password, each time that it is used it is exactly the same. This means that if an attacker can obtain the password then there is an (often significant) period of time within which the password can be used to fraudulently claim the identity of the original owner.
- One measure that can be taken to restrict this threat is to regularly force password change. However, this again raises a usability issue since regular password change is confusing for humans and can lead to insecure password storage practices.

Vulnerability.:

The consequences of 'stealing' a password can be serious. However, passwords are relatively easy for an attacker to obtain:

- ➢ they are most vulnerable at point of entry, when they can be viewed by an attacker watching the password holder (a technique often referred to as *shoulder surfing*);

- ➢ they can be extracted by attackers during social engineering activities, where a password holder is fooled into revealing a password to an attacker who makes claims.

  for example: to be a system administrator (an attack that is sometimes known as *phishing*);
- ➢ they can be obtained by an attacker observing network traffic or by an attacker who compromises a password database.

3.4.2 Cryptographic password protection

- Consider a large organisation that wishes to authenticate many users onto its internal system using passwords.

One way of implementing this is to use a system that compares offered passwords with those stored on a centralised password database. Since this database potentially contains a complete list of account names and passwords. Even if this database is managed carefully, the administrators of the system potentially have access to this list, which may not be desirable.

- One area where cryptography can be used to help to implement an identification system based on passwords is in securing the password database.

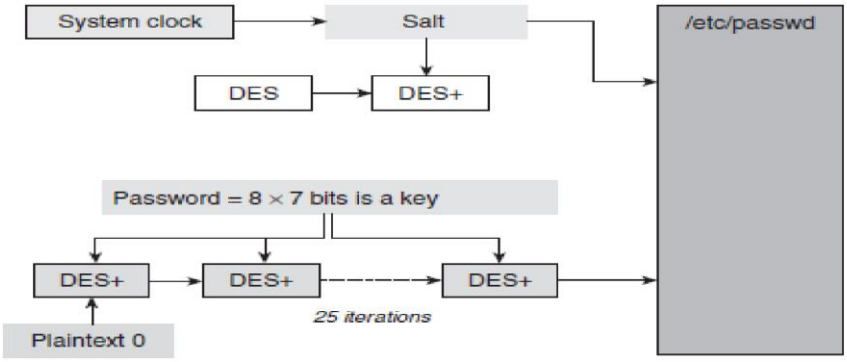
As an example of a cryptographic primitive being used in a different way to create a one-way function, below Figure illustrates the basic idea behind the function that was used in many early UNIX operating systems for password database protection.

- In the password database in the UNIX system, often identified by */etc/passwd*, every user has an entry that consists of two pieces of information:

Salt. This is a 12-bit number randomly generated using the system clock. The salt is used to uniquely modify the DES encryption algorithm nin a subtle way. We denote the result of this unique modification by DES+.

Password image. This is the result that is output after doing the following:

1. Convert the 8 ASCII character password into a 56-bit DES key. This is straightforward, since each ASCII character consists of 7 bits.



One-way function for UNIX password protection

2. Encrypt the plaintext consisting of all zeros (64 zero bits) using the uniquely modified DES+ with the 56-bit key derived from the password.
3. Repeat the last encryption step 25 times (in other words, we encrypt the all zero string 25 times). This last step is designed to slow the operation down in such a way that it is not inconvenient for a user, but much more difficult for an attacker conducting a dictionary attack.

When a user enters their password, the system looks up the salt, generates the modified DES+ encryption algorithm, forms the encryption key from the password, and then conducts the multiple encryption to produce the password image. The password image is then checked against the version stored in */etc/passwd*. If they match then the password is accepted.

3.5 Dynamic password schemes

Two of the main problems with passwords are vulnerability (they are quite easy to steal) and repeatability (once stolen they can be reused).

A ***dynamic password scheme***, also often refered to as a *one-time password scheme*, preserves the concept of a password but greatly improve its security by:

1. limiting the exposure of the password, thus reducing vulnerability;

2. using the password to generate dynamic data that changes on each authentication attempt, thus preventing repeatability.

3.5.1 Idea behind dynamic password schemes

A dynamic password scheme uses a 'password function' rather than a password. If a claimant, is a human user, wants to authenticate to a device, such as an authentication server, then the user inputs some data into the function to compute a value that is sent to the device. There are thus two components that we need to specify:

The password function: Since this function is a cryptographic algorithm, it is usually implemented on a smart token. In the example, we will assume that this is a smart token with an input interface that resembles a small calculator.

The input: We want the user and the device to agree on an input to the password function, the result of which will be used to authenticate the user. Since the input must be fresh, any of the freshness mechanisms could be used. All of these techniques are deployed in different commercial devices, namely:

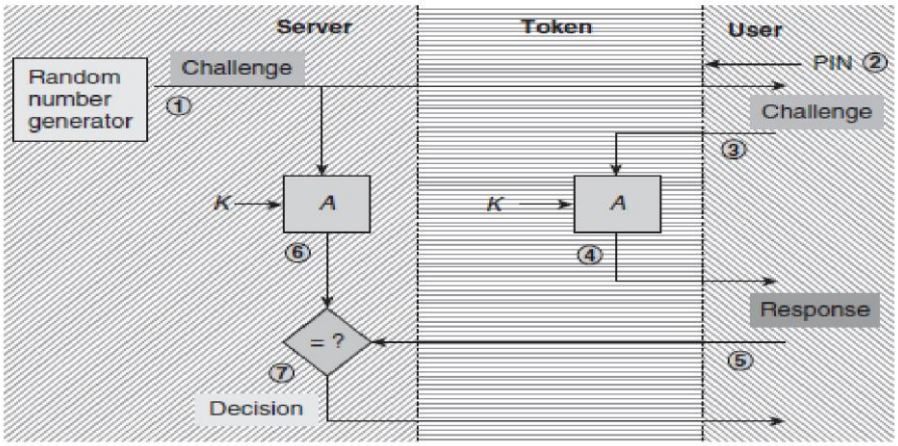
- ***Clock-based***. The user and the device have synchronised clocks and thus the current time can be used to generate an input that both the user and the device will 'understand'.
- ***Sequence numbers*.** The user and the device both maintain synchronised sequence numbers.
- ***Nonce-based***. The device randomly generates a number, known as a *challenge*, and sends it to the user, who computes a cryptographic *response*. Such mechanisms are often referred to as *challenge–response* mechanisms.

3.5.2 Example dynamic password scheme

Figure shows an authentication attempt using this dynamic password scheme:

1. The server randomly generates a challenge and sends it to the user. It is possible that the user first sends a message to the server requesting that the server send them a challenge.
2. The user authenticates themselves to the token using the PIN.
3. If the PIN is correct then the token is activated. The user then uses the token interface by means of a keypad to enter the challenge into the token.
4. The token uses the password function to compute a response to the challenge. If algorithm *A* is an encryption algorithm then the challenge can be regarded as a plaintext and the response is the ciphertext that results from applying encryption algorithm *A* using key *K*. The token displays the result to the user on its screen.
5. The user sends this response back to the server. This step might involve the user reading the response off the screen of the token and then typing it into a computer that is being used to access the authentication server.
6. The server checks that the challenge is still valid. If it is still valid, the server inputs the challenge into the password function and computes the response, based on the same algorithm *A* and key *K*.

7. The server compares the response that it computed itself with the response that was sent by the user. If these are identical then the server authenticates the user, otherwise the server rejects the user.



Example of a dynamic password scheme based on challenge–response

ANALYSIS OF DYNAMIC PASSWORD SCHEME:

There can be several significant improvements:

Local use of PIN:

With regard to security at the user end, the main difference is that the user uses the PIN to authenticate themselves to a small portable device that they have control over. The chances of the PIN being viewed by an attacker while it is being entered are lower than for applications where a user has to enter a PINinto a device not under their control, such as an ATM. Also, the PIN is only transferred from the user's fingertips to the token and does not then get transferred to any remote server.

Two factors. Without access to the token, the PIN is useless. Thus another improvement is that we have moved from *one-factor* authentication (something the claimant knows, namely the password) to *two-factor* authentication (something the claimant knows, namely the PIN, *and* something the claimant has, namely the token).

Dynamic responses. The biggest security improvement is that every time an authentication attempt is made, a different challenge is issued and therefore a different response is needed. Of course, because the challenge is randomly generated there is a *very small* chance that the same challenge is issued on two separate occasions. But assuming that a good source of randomness is used then this chance is so low that we can dismiss it. Hence anyone who

succeeds in observing a challenge and its corresponding response cannot use this to masquerade as the user at a later date.

3.6 Zero-knowledge mechanisms

3.6.1 Motivation for zero-knowledge:

- **Requirement for mutual trust**. Firstly, they are all based on some degree of trust between the entities involved.

For example, passwords often require the user to agree with the server on use of a password, even if the server only stores a hashed version of the password. However, there are situations where entity authentication might be required between two entities who are potential adversaries and do not trust one another enough to share *any* information.

- **Leaking of information**. Secondly, they all give away some potentially useful information on each occasion that they are used. Conventional passwords are catastrophic in this regard since the password is fully exposed when it is entered, and in some cases may even remain exposed when transmitted across a network. Our example dynamic password scheme is much better, but does reveal valid challenge–response pairs each time that it is run.

➢ Entity authentication could be provided in such a way that no shared trust is necessary and *no knowledge at all* is given away during an authentication attempt.

➢ The requirement for a zero-knowledge mechanism is that one entity (the *prover*) must be able to provide assurance of their identity to another entity (the *verifier*) in such a way that it is impossible for the verifier to later impersonate the prover, even after the verifier has observed and verified many different successful authentication attempts.

3.6.2 Zero-knowledge analogy

Consider a popular analogy in which we will play the role of verifier. The setting is a cave shaped in a loop with a split entrance, as depicted in Figure 8.4.

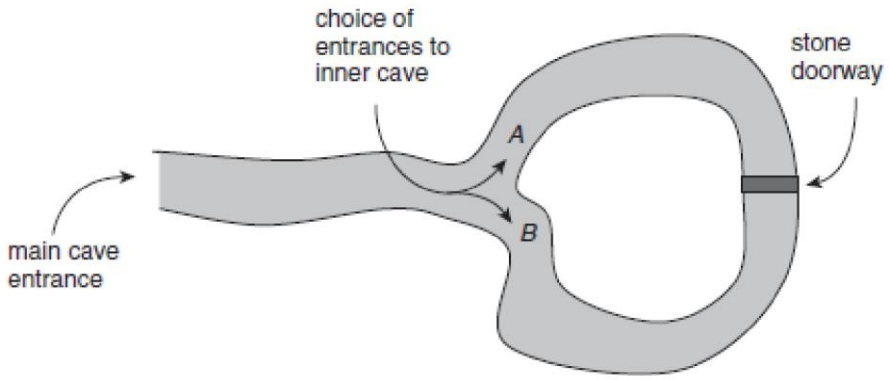


Figure 8.4. Popular analogy of a zero-knowledge mechanism

The back of the cave is blocked by a stone doorway that can only be opened by using a secret key phrase. We wish to hire a guide to make a circular tour of the entire cave system but need to make sure in advance that our guide knows the key phrase, otherwise we will not be able to pass through the doorway. The guide, who will be our prover (the entity authentication claimant), is not willing to tell us the key phrase, otherwise there is a risk that we might go on our own tour without hiring him. Thus we need to devise a test of the guide's knowledge before we agree to hire him.

The guide has a further concern. For all he knows, we are from a rival guiding company and are trying to learn the key phrase. He wants to make sure that no matter how rigorous a test is run, we will not learn *anything* that could help us to try to work out what the key phrase is. i. e he wants to make sure that the test is a zero-knowledge mechanism that verifies his claim to know the key phrase.

So here is what we do:

1. We wait at the main cave entrance and send the guide down the cave to the place where it splits into two tunnels, labelled *A* and *B*. We cannot see this tunnel split from the main entrance, so we send a trusted observer down with him, just to make sure he does not cheat during the test.
2. The guide randomly picks a tunnel entrance and proceeds to the stone door.
3. We toss a coin. If it is heads then we shout down the cave that we want the guide to come out through tunnel *A*. If it is tails then we shout down the cave that we want the guide to come out through tunnel *B*.
4. The observer watches to see which tunnel the guide emerges from.

Suppose we call heads (tunnel *A*). If the guide comes out of tunnel *B* (the wrong entrance) then we decide not to hire him since he does not appear to know the key phrase. However, if he comes out of tunnel *A* (the correct entrance) then one of two things have happened:

- The guide got lucky and chose tunnel *A* in the first place. In that case he just turned back, whether he knows the key phrase or not. In this case we learn nothing.
- The guide chose tunnel *B*. When we called out that he needed to come out of tunnel *A*, he used the key phrase to open the door and crossed into tunnel *A*. In this case the guide has demonstrated knowledge of the key phrase.

So if the guide emerges from tunnel *A* then there is a 50% chance that he has just demonstrated knowledge of the key phrase. The problem is that there is also a chance that he got lucky.

So we run the test again. If he passes a second test then the chances that he got lucky twice are now down to 25%, since he needs to get lucky in both independent tests. Then we run the test again, and again. If we run *n* such independent tests and the guide passes them all, then the probability that the guide does not know the key phrase is:

$$\frac{1}{2} \times \frac{1}{2} \times \cdots \times \frac{1}{2} = \left(\frac{1}{2}\right)^n = \frac{1}{2^n}.$$

Thus we need to insist on *n* tests being run, where $1/2^n$ is sufficiently small that we will be willing to accept that the guide almost certainly has the secret knowledge.

Meanwhile, the guide will have done a great deal of walking around the cave system and using the key phrase, without telling us any information about the key phrase. So the guide will also be satisfied with the outcome of this process.

Cryptographic Protocols

9.1 Protocol basics

9.1.1 Operational motivation for protocols

Many applications:

- **Have complex security requirements**. For example, if we wish to transmit some sensitive information across an insecure network then there should be confidentiality *and* data origin authentication guarantees .
- **Involve different data items with different security requirements**. Most applications involve different pieces of data, each of which may have different security requirements.

  For example, an application processing an online transaction may require the purchase details (product, cost) to be authenticated, but not encrypted, so that this

information is widely available. However, the payment details (card number, expiry date) are likely to be required to be kept confidential. It is also possible that different requirements of this type arise for efficiency reasons, since all cryptographic computations (particularly public-key computations) have an associated efficiency cost. It can thus be desirable to apply cryptographic primitives only to those data items that strictly require a particular type of protection.

- **Involve information flowing between more than one entity**. It is rare for a cryptographic application to involve just one entity, such as when a user encrypts a file for storage on their local machine. Most applications involve at least two entities exchanging data.

  For example, a card payment scheme may involve a client, a merchant, the client's bank and the merchant's bank (and possibly other entities).

- **Consist of a sequence of logical (conditional) events**. Real applications normally involve multiple operations that need to be conducted in a specific order, each of which may have its own security requirements.

  For example, it does not make any sense to provide confidentiality protection for the deduction of a sum from a client's account and issue some money from a cash machine until entity authentication of the client has been conducted.

9.1.2 Components of a cryptographic protocol

A cryptographic protocol needs to specify:

The protocol assumptions – any prerequisite assumptions concerning the environment in which the protocol will be run. This involves assumptions about the entire environment (including, for example, security of devices used in the protocol). *What needs to have happened before the protocol is run*?

The protocol flow – the sequence of communications that need to take place between the entities involved in the protocol. Each message is often referred to as being a *step* or *pass* of the protocol. *Who sends a message to whom, and in which order*?

The protocol messages – the content of each message that is exchanged between two entities in the protocol. *What information is exchanged at each step*?

The protocol actions – the details of any actions (operations) that an entity needs to perform after receiving, or before sending, a protocol message. *What needs to be done between steps*?

9.2 From objectives to a protocol

9.2.1 Stages of protocol design

There are three main stages to the process of designing a cryptographic protocol:

- **Defining the objectives**. This is the problem statement, which identifies what the problem is that the protocol is intended to solve. Particularly performance-related objectives.
- **Determining the protocol goals**. This stage translates the objectives into a set of clear cryptographic requirements. The protocol goals are typically statements of the form *at the end of the protocol, entity X will be assured of security service Y*. We will see some examples shortly.

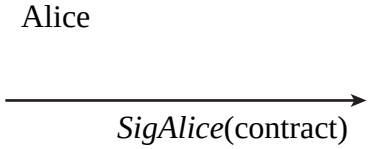


Figure 9.1. A simple cryptographic protocol providing non-repudiation

- **Specifying the protocol**. This takes the protocol goals as input and involves determining some cryptographic primitives, message flow and actions that achieve these goals.

A very simple example of these stages would be the following:

Defining the objectives. Merchant Bob wants to make sure that a contract that he will receive from Alice cannot later be denied.

Determining the protocol goals. At the end of the protocol Bob requires non- repudiation of the contract received from Alice.

Specifying the protocol. A protocol to achieve this simple goal is given in Figure 9.1. In this protocol there is only one message, which is sent from Alice to Bob. This message consists of the contract, digitally signed by Alice. The notation Sig_{Alice}represents a generic digital signature algorithm. We assume that if a digital signature scheme with appendix is used then part of Sig_{Alice}(contract) is a plaintext version of the contract.

9.3 Analysing a simple protocol

9.3.1 A simple application

THE OBJECTIVES

In this scenario we suppose that Alice and Bob have access to a common network. Periodically, at any time of his choosing, Bob wants to check that Alice is still ‘alive’ and

connected to the network. This is main security objective, which will be referred to as a check of liveness.

Lets assume that Alice and Bob are just two entities in a network consisting of many such entities, all of whom regularly check the liveness of one another, perhaps every few seconds. Thus set a secondary security objective that whenever Bob receives any confirmation of liveness from Alice, he should be able to determine precisely which liveness query she is responding to.

THE PROTOCOL GOALS

Whenever Bob wants to check that Alice is alive he will need to send a *request* to Alice, which she will need to respond to with a *reply*.

At the end of any run of a suitable cryptographic protocol, the following three goals should have been met:

1. **Data origin authentication of Alice's reply**. If this is not provided then Alice may not be alive since the reply message might have been created by an attacker.
2. **Freshness of Alice's reply.** If this is not provided then, even if there is data origin authentication of the reply, this could be a replay of a previous reply.

   In other words, an attacker could observe a reply that Alice makes when she *is* alive and then send a copy of it to Bob at some stage after Alice has expired. This would be a genuine reply created by Alice. But she would not be alive and hence the protocol will have failed to meet its objectives.
3. **Assurance that Alice's reply corresponds to Bob's request**. If this is not provided then it is possible that Bob receives a reply that corresponds to a different request (either one of his own, or of another entity in the network).

Table 9.1: Notation used during protocol descriptions

r_B	A nonce generated by Bob
$\|$	Concatenation
Bob	An *identifier* for Bob (perhaps his name)
$MAC_K(data)$	A MAC computed on *data* using key *K*
$E_K(data)$	Symmetric encryption of *data* using key *K*
$Sig_A(data)$	A digital signature on *data* computed by Alice
T_A	A timestamp generated by Alice
T_B	A timestamp generated by Bob
ID_S	A session identifier

CANDIDATE PROTOCOLS

Figure 9.2 shows the protocol flow and messages of our first candidate protocol.

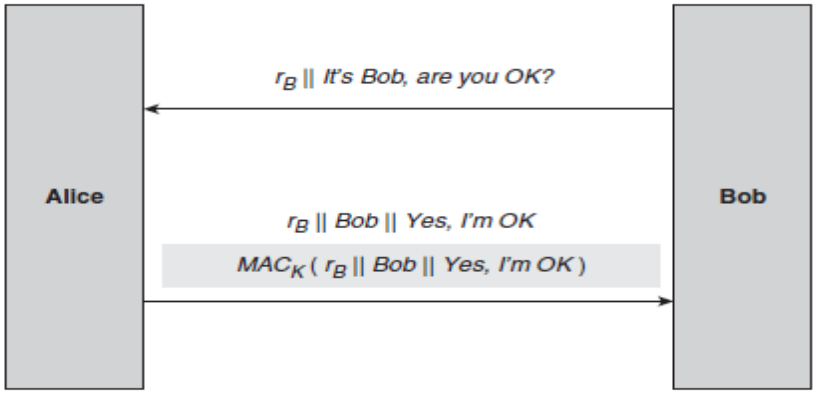


Figure 9.2. Protocol 1

PROTOCOL ASSUMPTIONS

There are three assumptions that we make before running this protocol:

1. **Bob has access to a source of randomness**. This is necessary because the protocol requires Bob to be able to generate a nonce and also assume that this generator is 'secure' in order to guarantee unpredictability of the output.
2. **Alice and Bob already share a symmetric key *K* that is known only to them**. This is necessary because the protocol requires Alice to be able to generate a MAC that Bob can verify.
3. **Alice and Bob agree on the use of a strong MAC algorithm**. This is necessary because if the MAC algorithm is flawed then data origin authentication is not necessarily provided by it.

If Alice and Bob do not already share a symmetric key then they will need to first run a different protocol in order to establish a common symmetric key *K* . If Alice and Bob have not already agreed on the use of a strong MAC algorithm to compute the MAC then Alice could indicate the choice of MAC algorithm that she is using in her *reply*.

PROTOCOL DESCRIPTION

Protocol 1 consists of the following steps:

1. Bob conducts the following steps to form the request:
   a) Bob generates a nonce *rB* (this is an implicit action that is not described in Figure 9.2.
   b) Bob concatenates *rB* to the text *It's Bob, are you OK?*. This combined data string is the request.
   c) Bob sends the request to Alice.
2. Assuming that she is alive and able to respond, Alice conducts the following steps to form the reply:
   a) Alice concatenates the nonce *rB* to identifier *Bob* and the text *Yes, I'm OK*. We will refer to this combined data string as the *reply text*.
   b) Alice computes a MAC on the reply text using key *K* (this is an implicit action). The reply text is then concatenated to the MAC to form the reply.
   c) Alice sends the reply to Bob.
3. On receipt of the reply, Bob makes the following checks.:
   a) Bob checks that the received reply text consists of a valid *rB* (which he can recognise because he generated it and has stored it on a local database) concatenated to his identifier *Bob* and a meaningful response to his query (in this case, *Yes, I'm OK*).
   b) Bob computes a MAC on the received reply text with key *K* (which he shares with Alice) and checks to see if it matches the received MAC.

c) If both of these checks are satisfactory then Bob accepts the reply and ends the protocol. We say that the protocol successfully *completes* if this is the case.

PROTOCOL ANALYSIS

Protocol 1 meets the required goals:

a) **Data origin authentication of Alice's reply**. Under second assumption, the only entity other than Bob who can compute the correct MAC on the reply text is Alice. Thus, given that the received MAC is correct, the received MAC must have been computed by Alice. Thus Bob indeed has assurance that the reply (and by implication the reply text) was generated by Alice.

b) **Freshness of Alice's reply**. The reply text includes the nonce r_B, which Bob generated at the start of the protocol. Thus, by the principles, the reply is fresh.

c) **Assurance that Alice's reply corresponds to Bob's request**.

There are two pieces of evidence in the reply that provide this:

- Firstly, and most importantly, the reply contains the nonce *rB*, which Bob generated for this run of the protocol. By our first protocol assumption, this nonce is very unlikely to ever be used for another protocol run, thus the appearance of *rB* in the reply makes it almost certain that the reply corresponds to his request.
- The reply contains the identifier *Bob*.

Four of the components of a cryptographic protocol:

- **The protocol assumptions**. If the protocol assumptions do not hold then, even when the protocol successfully completes, the security goals are not met.

  For example, if a third entity Charlie also knows the MAC key *K* then Bob cannot be sure that the reply comes from Alice, since it could have come from Charlie.
- **The protocol flow**. Clearly the two messages in this protocol must occur in the specified order, since the reply cannot be formed until the request is received.
- **The protocol messages**. The protocol goals are not necessarily met if the content of the two messages is changed in any way.
- **The protocol actions**. The protocol goals are not met if any of the actions are not undertaken.

  For example, if Bob fails to check that the MAC on the reply text matches the received MAC then he has no guarantee of the origin of the reply.

9.3.3 Protocol 2

Figure 9.3 shows the protocol flow and messages of our second candidate protocol.

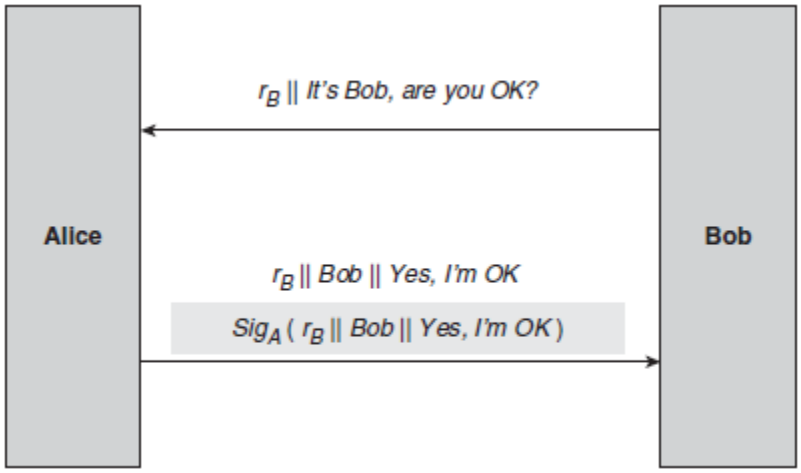


Figure 9.3. Protocol 2

PROTOCOL ASSUMPTIONS

As can be seen from Figure 9.3, Protocol 2 is very similar to Protocol 1. In fact, it is in the protocol assumptions that the main differences lie:

Bob has access to a source of randomness. As for Protocol 1.

Alice has been issued with a signature key and Bob has access to a verification key corresponding to Alice's signature key. This is the digital signature scheme equivalent of the second assumption for Protocol 1.

Alice and Bob agree on the use of a strong digital signature scheme.

PROTOCOL DESCRIPTION

The description of Protocol 2 is exactly as for Protocol 1, except that:

Instead of computing a MAC on the reply text, Alice digitally signs the reply text using her signature key.

Instead of computing and comparing the received MAC on the reply text, Bob verifies Alice's digital signature on the reply text using her verification key.

PROTOCOL ANALYSIS

The analysis of Protocol 2 is exactly as for Protocol 1, except for:

Data origin authentication of Alice's reply. Under second assumption, the only entity who can compute the correct digital signature on the reply text is Alice. Thus, given that her digital signature is verified, the received digital signature must have been computed by Alice.

Thus Bob indeed has assurance that the reply (and by implication the reply text) was generated by Alice.

Therefore deduce that Protocol 2 also meets the three security goals.

REMARKS

Protocol 2 can be thought of as a public-key analogue of Protocol 1. So which one is better?

- It could be argued that, especially in resource-constrained environments, Protocol 1 has an advantage in that it is more computationally efficient, since computing MACs generally involves less computation than signing and verifying digital signatures.
- However, it could also be argued that Protocol 2 has the advantage that it could be run between an Alice and Bob who have not pre-shared a key, so long as Bob has access to Alice's verification key.

9.3.4 Protocol 3

PROTOCOL ASSUMPTIONS

These are identical to Protocol 1.

PROTOCOL DESCRIPTION

This is identical to Protocol 1, except that in Protocol 3 the identifier *Bob* is omitted from the reply text.

PROTOCOL ANALYSIS

This is identical to Protocol 1, except for:

Assurance that Alice's reply corresponds to Bob's request.

As argued for Protocol 1, the inclusion of the nonce rB in the reply appears, superficially, to provide this assurance since rB is in some sense a unique identifier of Bob's request. However, there is an attack that can be launched against Protocol 3 in certain environments which shows that this is not always true. Since the attacker plays the role of a 'mirror', and this is a reflection *attack* against Protocol 3. The attack is depicted in Figure 9.5.

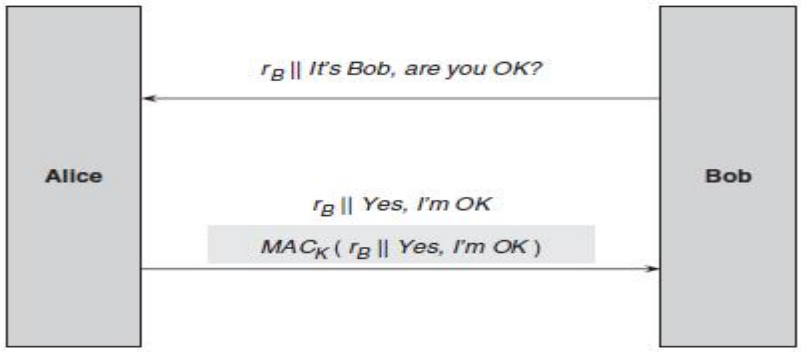


Figure 9.4. Protocol 3

The reflection attack working:

Lets assume that an attacker is able to intercept and block all communication between Alice and Bob and also assume that Bob normally recognises that incoming traffic may be from Alice through the use of the channel, rather than an explicit identifier. This is perhaps an unreasonable assumption, but we are trying to keep it simple. Thus, even if Alice is no longer alive, the attacker can pretend to be Alice by sending messages on this channel. The reflection attack works as follows:

1. Bob initiates a run of Protocol 3 by issuing a request message.
2. The attacker intercepts the request message and sends it straight back to Bob, except that the text It's Bob is replaced by the text It's Alice.
3. At this point it is to suggest that Bob will regard the receipt of a message containing his nonce rB as rather strange and will surely reject it. However, resist the temptation to anthropomorphise the analysis of a cryptographic protocol and recall that in most applications of this type of protocol both Alice and Bob will be computing devices following programmed instructions. In this case Bob will simply see a request message that appears to come from Alice and, since he is alive, will compute a corresponding reply message. He then sends this reply to Alice.
4. The attacker intercepts this reply message and sends it back to Bob.
5. Bob, who is expecting a reply from Alice, checks that it contains the expected fields and that the MAC is correct. Of course it is, because he computed it himself!

The reflection attack described in Figure 9.5 as two nested runs of Protocol 3:

• The first run is initiated by Bob, who asks if Alice is alive. He thinks that he is running it with Alice, but instead he is running it with the attacker.

• The second run is initiated by the attacker, who asks if Bob is alive. Bob thinks that this request is from Alice, but it is from the attacker. Note that this run of Protocol 3 begins after the first run of the protocol has begun, but completes before the first run ends.

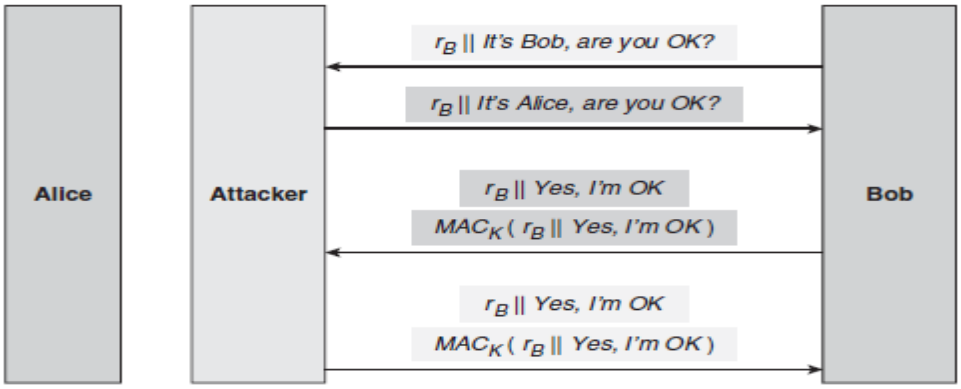


Figure 9.5. Reflection attack against Protocol 3

A better response would be to repair Protocol 3. There are two options:

1. **Include an action to check for this attack:**

This would involve Bob keeping a note of all Protocol 3 sessions that he currently has open. He should then check whether any request messages that he receives match any of his own open requests.

2. **Include an identifier.**

Include some sort of identifier in the reply that prevents the reflection attack from working. There is no point in doing so in the request since it is unprotected and an attacker could change it without detection.

9.3.5 Protocol 4

Figure 9.6 shows the protocol flow and messages of our fourth candidate protocol.

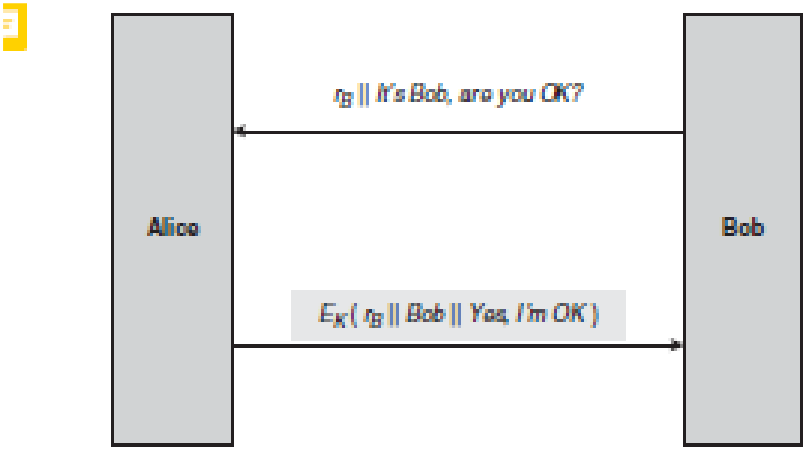


Figure 9.6. Protocol 4

PROTOCOL ASSUMPTIONS

These are identical to Protocol 1, expect that we assume that Alice and Bob have agreed on the use of a strong symmetric encryption algorithm E (rather than a MAC).

PROTOCOL DESCRIPTION

The description of Protocol 4 is exactly as for Protocol 1, except that:

- Instead of computing a MAC on the reply text, Alice uses E to encrypt the reply text using key K.
- Alice does not send the reply text to Bob.
- Instead of computing and comparing the received MAC on the reply text, Bob simply decrypts the received encrypted reply text.

PROTOCOL ANALYSIS

The analysis of Protocol 4 is exactly as for Protocol 1, except for the issue of data origin authentication of Alice's reply. Consider whether encryption can be used in this context to provide data origin authentication.

There are two arguments:

The case against:

This is perhaps the purist's viewpoint. Protocol 4 does not provide data origin authentication because encryption does not, in general, provide data origin authentication.

The case for:

Problems that may arise if encryption is used to provide data origin authentication. These mainly arose when the plaintext was long and unformatted. In this case the reply text is short and has a specific format. Thus, if a block cipher such as AES is used then it is possible that the reply text is less than one block long, hence no 'block manipulation' will be possible. Even if the reply text is two blocks long and ECB mode is used to encrypt these two blocks, the format of the reply text is specific and any manipulation is likely to be noticed by Bob (assuming of course that he checks for it).

9.3.6 Protocol 5

Protocol 5, depicted in Figure 9.7, is very similar to Protocol 1, except that the nonce generated by Bob is replaced by a timestamp generated by Bob.

PROTOCOL ASSUMPTIONS

These are the same as the assumptions for Protocol 1, except that the need for Bob to have a source of randomness is replaced by:

Bob can generate and verify integrity-protected timestamps:

This requires Bob to have a system clock. Requiring T_B to be integrity-protected means that it cannot be manipulated by an attacker without subsequent detection of this by Bob.

PROTOCOL DESCRIPTION : The description of Protocol 5 is exactly as for Protocol 1, except that:

- Instead of generating a nonce rB, Bob generates an integrity-protected timestamp T_B. This is then included in both the request (by Bob) and the reply (by Alice).
- As part of his checks on the reply, Bob checks that the reply text includes T_B.

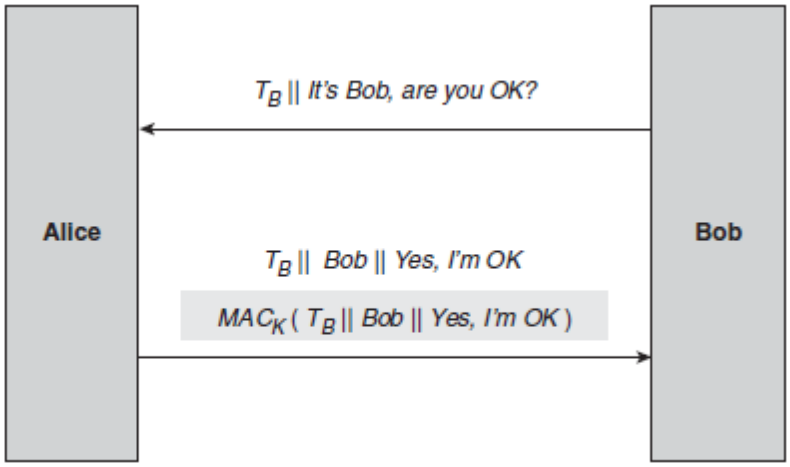


Figure 9.7. Protocol 5

PROTOCOL ANALYSIS

The analysis of Protocol 5 is similar to Protocol 1.

Data origin authentication of Alice's reply. As for Protocol 1.

Freshness of Alice's reply. The reply text includes the timestamp TB, which Bob generated at the start of the protocol. The reply is fresh.

Assurance that Alice's reply corresponds to Bob's request. There are two pieces of evidence in the reply that provide this:

1. The reply contains the timestamp T_B, which Bob generated for this run of the protocol. Assuming that the timestamp is of sufficient granularity that it is not possible for Bob to have issued the same timestamp for different protocol runs (or that it includes a unique session identifier), the presence of T_B indicates that the reply matches the request.
2. The reply contains the identifier Bob, preventing reflection attacks. Thus Protocol 5 meets the three security goals.

Thus Protocol 5 meets the three security goals.

REMARKS

Protocol 5 can be thought of as the 'clock-based' analogue of Protocol 1.

No need for Alice to share a synchronised clock with Bob for Protocol 5 to work. This is because only Bob requires freshness, hence it suffices that Alice includes Bob's timestamp without Alice necessarily being able to 'make sense' of, let alone verify, it.

One consequence of this is that it is important that T_B is integrity-protected. To see this, suppose that T_B just consists of the time on Bob's clock, represented as an unprotected timestamp (perhaps just a text stating the time). In this case the following attack is possible:

1. At 15.00, the attacker sends Alice a request that appears to come from Bob but has T_B set to the time 17.00, which is a time in the future that the attacker anticipates that Bob will contact Alice.
2. Alice forms a valid reply based on T_B being 17.00 and sends it to Bob.
3. The attacker intercepts and blocks the reply from reaching Bob, then stores it.
4. The attacker hits Alice over the head with a blunt instrument. (Less violent versions of this attack are possible!)
5. At 17.00, Bob sends a genuine request to Alice (recently deceased).
6. The attacker intercepts the request and sends back the previously intercepted reply from Alice.
7. Bob accepts the reply as genuine (which it is) and assumes that Alice is OK (which she most definitely is not). This attack is only possible because, in this example, we allowed the attacker to 'manipulate' T_B. By assuming that T_B is a timestamp that cannot be manipulated in such a way, this attack is impossible.

9.3.7 Protocol 6

Protocol 6 is shown in Figure 9.8.

PROTOCOL ASSUMPTIONS These are the same as the assumptions for Protocol 1, except that the need for Bob to have a random generator is replaced by:

Alice can generate timestamps that Bob can verify. As part of this assumption we further require that Alice and Bob have synchronised clocks

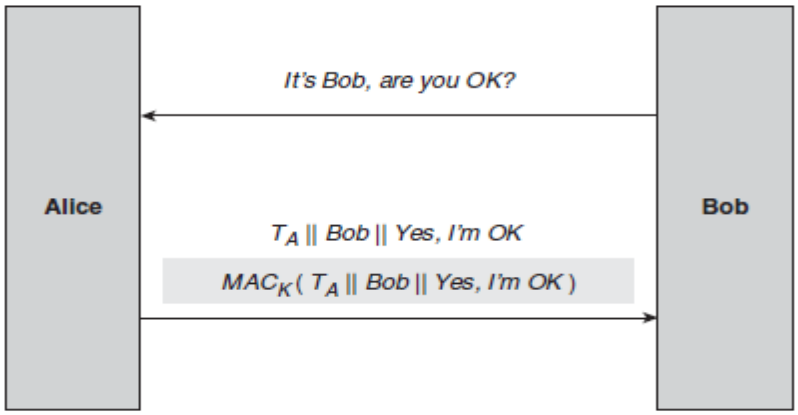


Figure 9.8. Protocol 6

PROTOCOL DESCRIPTION

The description of Protocol 6 is slightly different from Protocol 1, so we will explain it in more detail than the last few protocols.

1. Bob conducts the following steps to form the request:
    a) Bob forms a simplified request message that just consists of the text It's Bob, are you OK?.
    b) Bob sends the request to Alice.
2. Assuming that she is alive and able to respond, Alice conducts the following steps to form the reply:
    (a) Alice generates a timestamp T_A and concatenates it to identifier Bob and the text Yes, I'm OK, to form the reply text.
    (b) Alice computes a MAC on the reply text using key K. The reply text is then concatenated to the MAC to form the reply.
    (c) Alice sends the reply to Bob.
3. On receipt of the reply, Bob makes the following checks:
    a) Bob checks that the received reply text consists of a timestamp TA concatenated to his identifier Bob and a meaningful response to his query (in this case, Yes, I'm OK).
    b) Bob verifies TA and uses his clock to check that it consists of a fresh time.
    c) Bob computes a MAC on the received reply text with key K and checks to see if it matches the received MAC.
    d) If both these checks are satisfactory then Bob accepts the reply and ends the protocol.

PROTOCOL ANALYSIS

The analysis of Protocol 6 is similar to Protocol 1.

Data origin authentication of Alice's reply. As for Protocol 1.

Freshness of Alice's reply. The reply text includes the timestamp T_A. The reply is fresh.

Assurance that Alice's reply corresponds to Bob's request. Unfortunately this is not provided, since the request does not contain any information that can be used to uniquely identify it. Thus Protocol 6 does not meet all three security goals.

REMARKS

Protocol 6 has only failed on a technicality. It could easily be 'repaired' by including a unique session identifier in the request message, which could then be included in the reply text.

9.3.8 Protocol 7

Seventh protocol variant is closely related to Protocol 6, and is depicted in Figure 9.9.

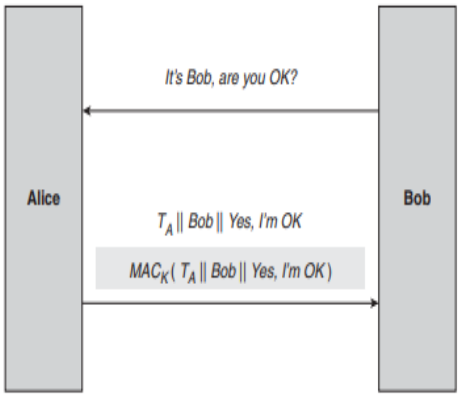


Figure 9.8. Protocol 6

PROTOCOL ASSUMPTIONS

These are the same as the assumptions for Protocol 6.

PROTOCOL DESCRIPTION

The description of Protocol 7 is almost the same as Protocol 6. The only differences are:

- Bob includes a unique session identifier IDS in the request, which Alice includes in the reply text. This identifier is not necessarily randomly generated (unlike the nonces that were used in some of the previous variants).
- The reply text that is sent in the clear by Alice differs from the reply text on which Alice computes the MAC. The difference is that T_A is included in the latter, but not the former.

PROTOCOL ANALYSIS

The analysis of Protocol 7 is similar to Protocol 6. The inclusion of the session identifier IDS is intended to remove the concerns about linking the reply to the request. The omission of T_A from the reply text that is sent in the clear at first just looks like a saving in bandwidth, since:

- Alice and Bob have synchronised clocks, by our assumptions,
- it is not strictly necessary that the data on which the MAC is computed matches the reply text, so long as Bob receives all the critical data that he needs to check the MAC.

Problem:

Bob does not know T_A. Even if they have perfectly synchronised clocks, the time that Alice issues T_A will not be the same time that Bob receives the message due to communication delays. Thus Bob does not know all the reply text on which the MAC is computed, and hence cannot verify the MAC to obtain data origin authentication. The only option is for Bob to check all the possible timestamps T_A within a reasonable window and hope that he finds one that matches. While this is inefficient, it is worth noting that this technique is sometimes used in real applications to cope with time delays and clock drift.

REMARKS

Protocol 7 is easily fixed by including T_A in both versions of the reply text, as is done in Protocol 6. Nonetheless, this protocol flaw demonstrates how sensitive cryptographic protocols are to even the slightest 'error' in their formulation.

9.4 Authentication and key establishment protocols

AKE protocols (authentication and key establishment):

The two main security objectives of an AKE protocol are always:

Mutual entity authentication:

Occasionally just unilateral entity authentication.

Establishment of a common symmetric key:

Regardless of whether symmetric or public-key techniques are used to do this. It should not come as a surprise that these two objectives are required together in one protocol.

Need to authenticate key holders:

Key establishment makes little sense without entity authentication. It is hard to imagine any applications where we would want to establish a common symmetric key between two parties without at least one party being sure of the other's identity. Indeed, in many applications mutual entity authentication is required. The only argument for not featuring entity

authentication in a key establishment protocol is for applications where the authentication has already been conducted prior to running the key establishment protocol.

Prolonging authentication:

The result of entity authentication can be prolonged by simultaneously establishing a symmetric key. A problem with entity authentication is that it is achieved only for an instant in time.

9.4.1 Typical AKE protocol goals

Mutual entity authentication:

Alice and Bob are able to verify each other's identity to make sure that they know with whom they are establishing a key.

Mutual data origin authentication: Alice and Bob are able to be sure that information being exchanged originates with the other party and not an attacker.

Mutual key establishment: Alice and Bob establish a common symmetric key.

Key confidentiality: The established key should at no time be accessible to any party other than Alice and Bob.

Key freshness: Alice and Bob should be happy that (with high probability) the established key is not one that has been used before.

Mutual key confirmation: Alice and Bob should have some evidence that they both end up with the same key.

Unbiased key control: Alice and Bob should be satisfied that neither party can unduly influence the generation of the established key.

9.4.2 Diffie–Hellman key agreement protocol

IDEA BEHIND THE DIFFIE–HELLMAN PROTOCOL

The Diffie–Hellman protocol requires the existence of:

- A public-key cryptosystem with a special property, We denote the public and private keys of Alice and Bob in this cryptosystem by (P_A, S_A) and (P_B, S_B), respectively. These may be temporary key pairs that have been generated specifically for this protocol run, or could be long-term key pairs that are used for more than one protocol run.
- A combination function F with a special property. By a 'combination' function, means a mathematical process that takes two numbers x

and y as input, and outputs a third number which we denote F(x, y). Addition is an example of a combination function, with F(x, y) = x + y.

The Diffie–Hellman protocol is designed for environments where secure channels do not yet exist, it is often used to establish a symmetric key, which can then be used to secure such a channel.

The basic idea behind the Diffie–Hellman protocol is that:

1. Alice sends her public key P_A to Bob.
2. Bob sends his public key P_B to Alice.
3. Alice computes $F(S_A, P_B)$. Note that only Alice can conduct this computation, since it involves her private key S_A.
4. Bob computes $F(S_B, P_A)$. Note that only Bob can conduct this computation, since it involves his private key S_B. The special property for the public-key cryptosystem and the combination function F is that $F(S_A, P_B) = F(S_B, P_A)$.

At the end of the protocol Alice and Bob will thus share this value, which we denote Z_{AB}, this shared value Z_{AB} can then easily be converted into a key of the required length. Since the private keys of Alice and Bob are both required to compute Z_{AB}, it should only be computable by Alice and Bob, and not anyone else (an attacker) who observed the protocol messages. Note that this is true despite the fact that the attacker will have seen P_A and P_B.

INSTANTIATION OF THE DIFFIE–HELLMAN PROTOCOL

For ElGamal, choose two public system wide parameters:

• a large prime p, typically 1024 bits in length;

• a special number g (a primitive element).

The Diffie–Hellman protocol is shown in Figure 9.10 and proceeds as follows.

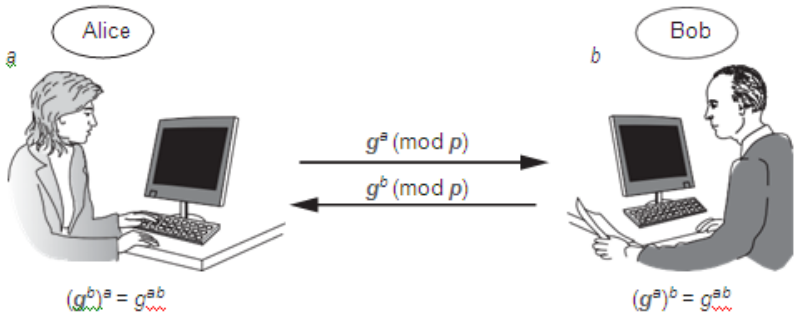


Figure 9.10. Diffie-Hellman protocol

Note that all calculations are performed modulo p and thus we omit mod p in each computation for convenience.

a) Alice randomly generates a positive integer a and calculates g^a. This is, effectively, a temporary ElGamal key pair. Alice sends her public key g^a to Bob.

b) Bob randomly generates a positive integer b and calculates g^b. Bob sends his public key g^b to Alice.

c) Alice uses g^b and her private key a to compute $(g^b)^a$.

d) Bob uses g^a and his private key b to compute$(g^a)^b$

e) The special combination function property that is needed that raising a number to the power a and then raising the result to the power b is the same as raising the number to the power b and then raising the result to the power a, which means that: $(g^a)^b = (g^b)^a$.

So Alice and Bob have ended up with the same value at the end of this protocol.

There are several important issues to note:

1. It is widely believed that the shared value $Z_{AB} = g^{ab}$ cannot be computed by anyone who does not know either a or b. An attacker who is monitoring the communication channel only sees g^a and g^b.
2. The main purpose of the Diffie–Hellman protocol is to establish a common cryptographic key *KAB*. There are two reasons why the shared value $ZAB = gab$ is unlikely to itself form the key in a real application:
    - Z_{AB} is not likely to be the correct length for a cryptographic key. If we conduct the Diffie–Hellman protocol with p having 1024 bits, then the shared value will also be a value of 1024 bits, which is much longer than a typical symmetric key.
    - Having gone through the effort of conducting a run of the Diffie–Hellman protocol to compute Z_{AB}, Alice and Bob may want to use it to establish several

different keys. Hence they may not want to use Z_{AB} as a key, but rather as a seed from which to derive several different keys. The rationale behind this is that Z_{AB} is relatively expensive to generate, both in terms of computation and communication, whereas derived keys K_{AB} are relatively cheap to generate from Z_{AB}.

3. Any public-key cryptosystem that has the right special property and for which a suitable combination function F can be found, could be used to produce a version of the Diffie–Hellman protocol. In this case:
    - very informally, the special property of ElGamal is that public keys of different users can be numbers over the same modulus p, which means that they can be combined in different ways;
    - the combination function F , which is $F(x, g^y) = (g^y)^x$, has the special property that it does not matter in which order the two exponentiations are conducted, since:

$F(x, g^y) = (g^y)^x = (g^x)^y = F(y, g^x)$.

It is not possible to use keys pairs from any public-key cryptosystem to instantiate the Diffie–Hellman protocol. In particular, RSA key pairs cannot be used because in RSA each user has their own modulus n, making RSA key pairs difficult to combine in the above manner.

ANALYSIS OF THE DIFFIE–HELLMAN PROTOCOL

Mutual entity authentication: There is nothing in the Diffie–Hellman protocol that gives either party any assurance of who they are communicating with. The values *a* and *b* (and hence *ga* and *gb*) have been generated for this protocol run and cannot be linked with either Alice or Bob. Neither is there any assurance that these values are fresh.

Mutual data origin authentication: This is not provided, by the same argument as above.

Mutual key establishment: Alice and Bob do establish a common symmetric key at the end of the Diffie–Hellman protocol, so this goal is achieved.

Key confidentiality. The shared value $Z_{AB}=gab$ is not computable by anyone other than Alice or Bob. Neither is any key K_{AB} derived from Z_{AB}. Thus this goal is achieved.

Key freshness. Assuming that Alice and Bob choose fresh private keys *a* and *b* then Z_{AB} should also be fresh. Indeed, it suffices that just one of Alice and Bob choose a fresh private key.

Mutual key confirmation. This is not provided, since neither party obtains any explicit evidence that the other has constructed the same shared value Z_{AB}.

Unbiased key control. Both Alice and Bob certainly contribute to the generation of Z_{AB}. Technically, if Alice sends *ga* to Bob before Bob generates *b*, then Bob could 'play around' with a few candidate choices for *b* until he finds a *b* that results in a $Z_{AB}=gab$ that he particularly 'likes'.

MAN-IN-THE-MIDDLE ATTACK ON THE DIFFIE–HELLMAN PROTOCOL

The man-in-the-middle attack is applicable to any situation where an attacker (Fred, in Figure 9.11) can intercept and alter messages sent on the communication channel between Alice and Bob.

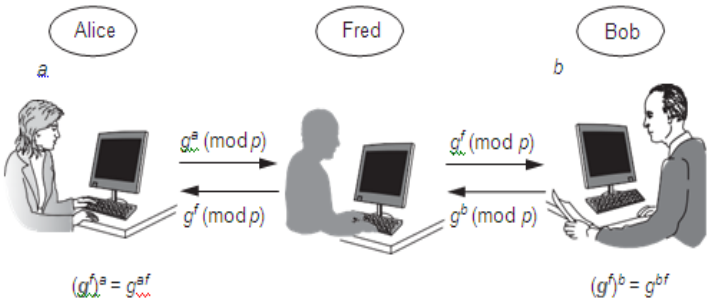


Figure 9.11. Man-in-the-middle attack against the Diffie-Hellman protocol

The man-in-the middle attack works as follows (where all calculations are modulo p):

1. Alice begins a normal run of the Diffie–Hellman protocol depicted in Figure 9.10. She randomly generates a positive integer a and calculates g^a. Alice sends g^a to Bob.
2. Fred intercepts this message before it reaches Bob, generates his own positive integer f , and calculates g^f . Fred then claims to be Alice and sends g^f to Bob instead of g^a.
3. Bob continues the Diffie–Hellman protocol as if nothing untoward has happened. Bob randomly generates a positive integer b and calculates g^b. Bob sends g^b to Alice.
4. Fred intercepts this message before it reaches Alice. Fred then claims to be Bob and sends g^f to Alice instead of g^b.
5. Alice now believes that the Diffie–Hellman protocol has successfully completed. She uses gf and her private integer a to compute $g^{af}= (g^f)^a$.
6. Bob also believes that it has successfully completed. He uses g^f& b to compute $g^{bf}= (g^f)^b$.
7. Fred computes $g^{af}=(g^a)^f$ and $g^{bf}= (g^b)^f$. He now has two different shared values, g^{af} , which he shares with Alice, and g^{bf} , which he shares with Bob.

At the end of this man-in-the-middle attack, all three entities hold different beliefs:

- Alice believes that she has established a shared value with Bob. But she is wrong, because she has established a shared value with Fred.
- Bob believes that he has established a shared value with Alice. But he is wrong, because he has established a shared value with Fred.
- Fred correctly believes that he has established two different shared values, one with Alice and the other with Bob.

At the end of this man-in-the-middle attack, Fred cannot determine the shared value *gab* that Alice and Bob would have established had he not interfered, since both *a* and *b* remain secret to him, protected by the difficulty of the discrete logarithm problem. Nonetheless, Fred is now in a powerful position:

- If Fred's objective was simply to disrupt the key establishment process between Alice and Bob then he has already succeeded. If Alice derives a key *KAF* from *gaf* and then encrypts a message to Bob using this key, Bob will not be able to decrypt it successfully because the key *KBF* that he derives from his shared value *gbf* will be different from *KAF* .
- Much more serious is the situation that arises if Fred remains on the communication channel. In this case, if Alice encrypts a plaintext to Bob using key *KAF* , Fred (who is the only person who can derive both *KAF* and *KBF*) can decrypt the ciphertext using *KAF* to learn the plaintext. He can then re-encrypt the plaintext using *KBF* and send this to Bob. In this way, Fred can 'monitor' the *encrypted* communication between Alice and Bob without them being aware that this is even happening.
- This man-in-the middle attack was only able to succeed because neither Alice nor Bob could determine from whom they were receiving messages during the Diffie–Hellman protocol run.

AKE PROTOCOLS BASED ON DIFFIE–HELLMAN

Way of building in authentication: The *station-to-station* (STS) protocol makes an additional assumption that Alice and Bob have each established a long-term signature/verification key pair and have had their verification keys certified. The STS protocol is shown in Figure 9.12 and proceeds as follows (where all calculations are modulo *p*):

1. Alice randomly generates a positive integer *a* and calculates *ga*. Alice sends *ga* to Bob, along with the certificate *CertA* for her verification key.
2. Bob verifies *CertA*. If he is satisfied with the result then Bob randomly generates a positive integer *b* and calculates *gb*. Next, Bob signs a message that consists of Alice's

name, *ga* and *gb*. Bob then sends *gb* to Alice, along with the certificate *CertB* for his verification key and the signed message.

3. Alice verifies *CertB*. If she is satisfied with the result then she uses Bob's verification key to verify the signed message. If she is satisfied with this, she signs a message that consists of Bob's name, *ga* and *gb*, which she then sends back to Bob. Finally, Alice uses *gb* and her private key *a* to compute (*gb*)*a*.
4. Bob uses Alice's verification key to verify the signed message that he has just received. If he is satisfied with the result then Bob uses *ga* and his private key *b* to compute (*ga*)*b*.

With the exception of the first two, the goals are met for the STS protocol are just as for the basic Diffie–Hellman protocol (in other words, they are all met except for key confirmation). It remains to check whether the first two authentication goals are now met:

Mutual entity authentication. Since *a* and *b* are randomly chosen private keys, *ga* and *gb* are thus also effectively randomly generated values. Hence we can consider *ga* and *gb* as being nonces. At the end of the second STS protocol message, Alice receives a digital signature from Bob on a message that includes her 'nonce' *ga*. Similarly, at the end of the third STS protocol message, Bob receives a digital signature from Alice on a message that includes his 'nonce' *gb*. Hence, by the principles , mutual entity authentication is provided, since both Alice and Bob each perform a cryptographic computation using a key only known to them on a nonce generated by the other party.

Mutual data origin authentication. This is provided, since the important data that is exchanged in the main messages is digitally signed.

9.4.3 An AKE protocol based on key distribution

The STS protocol is an AKE protocol based on key agreement and the use of public-key cryptography. We will now look at an AKE protocol based on key distribution and the use of symmetric cryptography. This protocol is a simplified version of one from ISO 9798-2. This protocol involves the use of a trusted third party (denoted TTP).

PROTOCOL DESCRIPTION

The idea behind this AKE protocol is that Alice and Bob both trust the TTP. When Alice and Bob wish to establish a shared key K_{AB}, they will ask the TTP to generate one for them, which will then be securely distributed to them. The protocol involves the following assumptions:

- Alice has already established a long-term shared symmetric key *KAT* with the TTP.
- Bob has already established a long-term shared symmetric key *KBT* with the TTP.

- Alice and Bob are both capable of randomly generating nonces.

There is a further assumption made on the type of encryption mechanism used, but we will discuss that when we consider data origin authentication.

The protocol is shown in Figure 9.13 and proceeds as follows:

1. Bob starts the protocol by randomly generating a nonce *rB* and sending it to Alice.
2. Alice randomly generates a nonce *rA* and then sends a request for a symmetric key to the TTP. This request includes both Alice's and Bob's names, as well as the two nonces *rA* and *rB*.
3. The TTP generates a symmetric key *KAB* and then encrypts it twice. The first ciphertext is intended for Alice and encrypted using *KAT* . The plaintext consists of *rA*, *KAB* and Bob's name. The second ciphertext is intended for Bob and encrypted using *KBT* . The plaintext consists of *rB*, *KAB* and Alice's name. The two ciphertexts are sent to Alice.
4. Alice decrypts the first ciphertext using *KAT* and checks that it contains *rA* and Bob's name. She extracts *KAB*. She then generates a new nonce *rA*j . Next, she generates a new ciphertext by encrypting *rA*j and *rB* using *KAB*. Finally, she forwards the second ciphertext that she received from the TTP, and the new ciphertext that she has just created, to Bob.
5. Bob decrypts the first ciphertext that he receives (which is the second ciphertext that Alice received from the TTP) using *KBT* and checks that it contains *rB* and Alice's name. He extracts *KAB*. He then decrypts the second ciphertext using *KAB* and checks to see if it contains *rB*. He extracts *rA*j . Finally, he encrypts *rB*, *rA*j and Alice's name using *KAB* and sends this ciphertext to Alice. Alice decrypts the ciphertext using *KAB* and checks that the plaintext consists of *rB*, *rA*j and Alice's name. If it does then the protocol concludes successfully.

PROTOCOL ANALYSIS

Mutual entity authentication: We will divide this into two separate cases.

First we look at Bob's perspective. At the end of the fourth protocol message, the second ciphertext that Bob receives is an encrypted version of his nonce *rB*. Thus this ciphertext is fresh. But who encrypted it? Whoever encrypted it must have known the key *KAB*. Bob received this key by successfully using *KBT* to decrypt a ciphertext, which resulted in a correctly formatted plaintext message consisting of *rB, KAB* and Alice's name. Thus Bob can be sure that this ciphertext originated with the TTP, since the TTP is the only entity other than Bob who knows *KBT* . The format of this plaintext is essentially an 'assertion' by the

TTP that the key *KAB* has been freshly issued (because *rB* is included) for use in communication between Bob (because it is encrypted using *KBT*) and Alice (because her name is included). Thus the entity that encrypted the second ciphertext in the fourth protocol message must have been Alice because the TTP has asserted that only Alice and Bob have access to *KAB*. Hence Bob can be sure that he has just been talking to Alice.

1. Alice's perspective is similar. At the end of the last protocol message, Alice receives an encrypted version of her nonce *rA*j . This ciphertext, which was encrypted using *KAB*, is thus fresh. In the third protocol message Alice receives an assertion from the TTP that *KAB* has been freshly (because *rA* is included) issued for use for communication between Alice (because it is encrypted using *KAT*) and Bob (because his name is included). Thus the entity that encrypted the last protocol message must have been Bob, again because the TTP has asserted that only Alice and Bob have access to *KAB*. Thus Alice can be sure that she has just been talking to Bob.

Mutual data origin authentication: We use symmetric encryption throughout this protocol and do not apparently employ any mechanism to explicitly provide data origin authentication, such as a MAC. While symmetric encryption does not normally provide data origin authentication. Throughout current protocol, the plaintexts are strictly formatted and fairly short, hence it might be reasonable to claim that encryption alone is providing data origin authentication, so long as a strong block cipher such as AES is used. However, the standard ISO 9798-2 goes further, by specifying that the 'encryption' used in this protocol must be such that data origin authentication is also essentially provided. One method would be to use an authenticated-encryption primitive. This goal is thus also met.

Mutual key establishment: At the end of the protocol Alice and Bob have established K_{AB}, so this goal is met.

Key confidentiality: The key K_{AB}can only be accessed by an entity who has knowledge of either K_{AT}, K_{BT}or K_{AB}. This means either the TTP (who is trusted), Alice or Bob. So this goal is met.

Key freshness: This goal is met so long as the TTP generates a fresh key K_{AB}. Again, we are *trusting* that the TTP will do this.

Mutual key confirmation: Both Alice and Bob demonstrate that they know $K_{AB}$by using it to encrypt plaintexts (Alice in the fourth protocol message; Bob in the last protocol message). Thus both confirm knowledge of the shared key.

Unbiased key control: This is provided because $K_{AB}$is generated by the TTP.

Thus we conclude that all the goals are provided. A similar AKE protocol is used by the widely deployed *Kerberos* protocol.

KEY MANAGEMENT

10.1 Key management fundamentals

10.1.1 What is key management?

- Key management is the *secure administration of cryptographic keys*.
- Cryptographic keys are special pieces of *data*.
- Key management thus involves most of the diverse processes associated with information security. These include:
  - ➢ **Technical controls.** These can be used in various aspects of key management. For example, special hardware devices may be required for storing cryptographic keys, and special cryptographic protocols are necessary in order to establish keys.
  - ➢ **Process controls.** Policies, practices and procedures play a crucial role in key management. For example, business continuity processes may be required in order to cope with the potential loss of important cryptographic keys.
  - ➢ **Environmental controls.** Key management must be tailored to the environment in which it will be practiced. For example, the physical location of cryptographic keys plays a big role in determining the key management techniques that are used to administer them.
  - ➢ **Human factors.** Key management often involves people doing things. Many key management systems rely on manual processes.

10.1.2 The key lifecycle

The main phases of the key lifecycle are depicted in the Figure.

Key generation concerns the creation of keys.

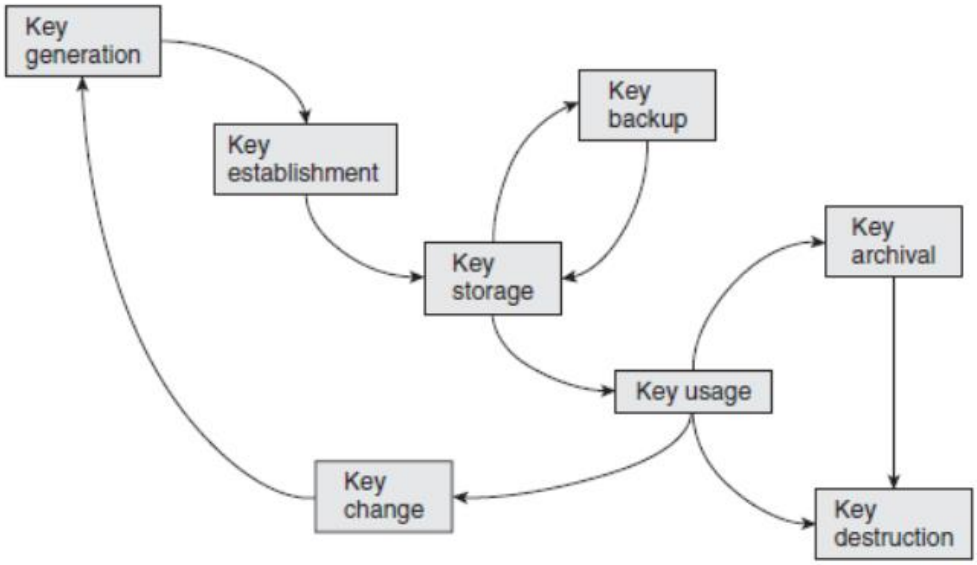
Key establishment is the process of making sure that keys reach the end points where they will be used.

Key storage deals with the safekeeping of keys. It may also be important to conduct

key backup so that keys can be recovered in the event of loss of a key and, ***key archival***.

Key usage is about how keys are used.

key change. How a key's life ends in ***key destruction***.



The key lifecycle

10.1.3 Fundamental key management requirements

There are two fundamental key management requirements that apply throughout the various phases of the key lifecycle:

Secrecy of keys. Throughout the key lifecycle, secret keys (in other words, symmetric keys and private keys) must remain secret from all parties except those that are authorised to know them.

Example:

- if a weak key generation mechanism is used then it might be possible to determine information about a secret key more easily than intended;
- secret keys are vulnerable when they are 'moved around', thus secure key distribution mechanisms must be used;
- secret keys are perhaps most vulnerable when they are 'sitting around', thus key storage mechanisms must be strong enough to resist an attacker who has access to a device on which they reside;
- if secret keys are not destroyed properly then they can potentially be recovered after the supposed time of destruction.

Assurance of purpose of keys. Throughout the key lifecycle, those parties relying on a key must have *assurance of purpose* of the key.

This 'purpose', may include :

• information concerning which entities are associated with the key

• the cryptographic algorithm that the key is intended to be used for.

• key usage restrictions, for example, that a symmetric key can only be used for creating and verifying a MAC, or that a signature key can only be used for digitally signing transactions of less than a certain value.

10.1.4 Key management systems

A key management system may depend on:

Network topology. Key management is much simpler if it is only needed to support two parties who wish to communicate securely, rather than a multinational organisation that wishes to establish the capability for secure communication between any two employees.

Cryptographic mechanisms. Some of the key management system requirements of symmetric and public-key cryptography differ.

Compliance restrictions. For example, depending on the application, there may be legal requirements for key recovery mechanisms or key archival .

Legacy issues. Large organisations whose security partly depends on that of other related organisations may find that their choice of key management system is restricted by requirements to be compatible with business partners, some of whom might be using older technology.

10.2 Key lengths and lifetimes

- ➢ Longer keys are better from a security perspective.
- ➢ A cryptographic computation normally takes more time if the key is longer. In addition, longer keys involve greater storage and distribution overheads.

10.2.1 Key lifetimes

- ➢ The key can only be used for a specified period of time, during which it is regarded as being *live*.
- ➢ Once that lifetime has been exceeded, the key is regarded as *expired* and should no longer be used. At this point it may need to be *archived* or perhaps *destroyed.*

There are many reasons why cryptographic keys have finite lifetimes. These include:

- **Mitigation against key compromise**. Having a finite lifetime prevents keys being used beyond a time within which they might reasonably be expected to be compromised,
  For example: by an exhaustive key search or compromise of the storage medium.

- **Mitigation against key management failures**. Finite key lifetimes help to mitigate against failures in key management.

  For example: forcing an annual key change will guarantee that personnel who leave an organisation during the year, but for some reason retain keys, do not have access to valid keys the following year.
- **Mitigation against future attacks**. Finite key lifetimes help to mitigate against future advances in the attack environment. For this reason, keys are normally set to expire well before current knowledge suggests that they need to.
- **Enforcement of management cycles**. Finite lifetimes enforce a key change process, which might be convenient for management cycles.

  For example: if keys provide access to electronic resources that are paid for on an annual subscription basis, then having a one-year key lifetime allows access to keys to be directly linked to subscriptions to the service.
- **Flexibility**. Finite key lifetimes introduce an additional 'variable' which can be adjusted to suit application requirements.

  For example, a relatively short key (which is relatively inexpensive to generate, distribute and store) could be adopted under the pretext that the key lifetime is also suitably short.
- **Limitation of key exposure**. Consider some information relating to a key is 'leaked' to an attacker every time the attacker sees a cryptographic value computed using that key. This is because the result of every cryptographic computation provides the attacker with information that they did not have before they saw the ciphertext. We refer to this as ***key exposure*.**

10.3 Key generation

Key generation processes for symmetric and public-key cryptography are fundamentally different.

10.3.1 Direct key generation

- ➢ Symmetric keys are just randomly generated numbers (normally bit strings).
- ➢ Method for generating a cryptographic key is to randomly generate a number, or more commonly a pseudorandom number.
- ➢ The choice of technique will depend on the application.

For example, use of a hardware-based non-deterministic generator might be appropriate for a master key, whereas a software-based non-deterministic generator based on mouse movements might suffice for generating a local key to be used to store personal files on a home PC.

10.3.2 Key derivation

The term *key derivation* is the generation of cryptographic keys from other cryptographic keys or secret values.

Advantages of deriving keys from other keys:

1. **Efficiency**. Generating and establishing one key (sometimes called a *base key*), and then using it to derive many keys.

For example:

- ➢ Many applications require both confidentiality and data origin authentication.
- ➢ If separate cryptographic mechanisms are to be used to provide these two security services then they require an encryption key and a MAC key
- ➢ It is good practice to make sure that the keys used for each of these mechanisms are different. Rather than generating and establishing two symmetric keys for this purpose, a cost efficient solution is to generate and establish one key K and then derive two keys $K1$ and $K2$ from it.

For example, a very simple key derivation process might involve computing:

$$K_1 = h(K||0) \text{ and } K_2 = h(K||1),$$

where h is a hash function.

2. **Longevity**:

- In some applications, long-term symmetric keys are preloaded onto devices before deployment.
- Using these long-term keys directly to encrypt data exposes them to cryptanalysis.
- Randomly generating a new key requires a key establishment mechanism to be used, which may not always be possible or practical.
- A good solution is to derive keys for use from the long-term key.

There are standards for key derivation:

For example: PKCS#5 defines how a key can be derived from a password or a PIN, which can be regarded as a relatively insecure type of cryptographic key, but one which is often long term (such as the PIN associated with a payment card).

Key derivation in this case is defined as a function ***f* (*P, S, C, L*),** where:

- *f* is a key derivation function that explains how to combine the various inputs in order to derive a key;
- *P* is the password or PIN;
- *S* is a string of (not necessarily all secret) pseudorandom bits, used to enable *P* to be used to derive many different keys;
- *C* is an iteration counter that specifies the number of 'rounds' to compute.
- *L* is the length of the derived key.

10.3.3 Key generation from components

For extremely important secret keys it may not be desirable to trust one entity with key generation in such cases we need to distribute the key generation process amongst a group of entities in such a way that no members of the group individually have control over the process, but collectively they do. One technique for facilitating this is to generate a key in *component form*.

Considering a simple scenario involving three entities: Alice, Bob and Charlie. Assume that we wish to generate a 128-bit key:

1. Alice, Bob and Charlie each randomly generate a *component* of 128 bits. This component is itself a sort of key, so any direct key generation mechanism could be used to generate it. The generation of components should be performed as securely as possible. Let us denote the resulting components by K_A, K_B and K_C , respectively.
2. Alice, Bob and Charlie securely transfer their components to a secure *combiner*. In most applications this combiner will be represented by a hardware security module. The input of the components to the secure combiner is normally conducted according to a strict protocol that takes the form of a *key ceremony.*
3. The secure combiner derives a key *K* from the separate components.

In this example, the best derivation function is XOR i.e :

$$\boldsymbol{K = K_A \oplus K_B \oplus K_C}\ .$$

- Thus Alice, Bob and Charlie are able to jointly generate a key in such a way that all three of their components are necessary for the process to complete.
- If only two of the components are present then no information about the key can be derived, even if the components are combined.

10.3.4 Public-key pair generation

Key generation for public-key cryptography is algorithm-specific.

Not every number in the 'range' of the keyspace of a public-key cryptosystem is a valid key. **For example**: for RSA the keys *d* and *e* are required to have specific mathematical properties. If we choose an RSA modulus of 1024 bits then there are, in theory, 2^{1024} candidates for *e* or *d*. However, only some of these 2^{1024} numbers *can* be an *e* or a *d*, the other choices are ruled out.

- Some keys in public-key cryptosystems are chosen to have a specific format.
  For example, RSA public keys are sometimes chosen to have a specific format that results in them being 'faster than the average case' when they are used to compute exponentiations,
- The generation of a key pair can be slow and complex. Some devices, such as smart cards, may not have the computational resources to generate key pairs.

In such cases it may be necessary to generate key pairs off the card and import them.

10.4 Key establishment

Key establishment is the process of getting cryptographic keys to the locations where they will be used.

The key does not need to be shared. This applies to any keys that can be locally generated and do not need to be transferred anywhere, such as symmetric keys for encrypting data on a local machine..

The key does not need to be secret. This applies mainly to public keys.

The key can be established in a controlled environment. In some cryptographic applications it is possible to establish all the required keys within a controlled environment before the devices containing the keys are deployed. This is termed *key predistribution*.

10.4.1 Key hierarchies

- *key hierarchy* is a technique used for managing symmetric keys.
- This consists of a ranking of keys, with high-level keys being more 'important' than low-level keys.
- Keys at one level are used to encrypt keys at the level beneath.

PHILOSOPHY BEHIND KEY HIERARCHIES

There are two advantages of deploying keys in a hierarchy:

Secure distribution and storage:

By using keys at one level to encrypt keys at the level beneath, most keys in the system can be protected by the keys above them. This allows keys to be securely distributed and stored in encrypted form.

Facilitating scalable key change.

- The risk of a key being compromised, keys that are directly used to perform cryptographic computations, such as encryption of transmitted data.
- Use of a key hierarchy makes it easy to change these low-level keys without the need to replace the high-level keys, which are expensive to establish.

A SIMPLE KEY HIERARCHY

The idea of a key hierarchy is illustrated considering a simple example which is shown in the Figure. The three levels of this hierarchy consist of:

***Master keys*.** These are the top-level keys that require careful management. They are only used to encrypt key encrypting keys. Since the key management of master keys is expensive, they will have relatively long lifetimes.

Key encrypting keys. These are distributed and stored in encrypted form using master keys. They are only used to encrypt data keys. Key encrypting keys will have shorter lifetimes than master keys, since they have greater exposure and are easier to change.

Data keys. These are distributed and stored in encrypted form using key encrypting keys. These are the working keys that will be used to perform cryptographic computations. They have high exposure and short lifetimes. This corresponds to the lifetime a single session, hence data keys are often referred to as *session keys*.

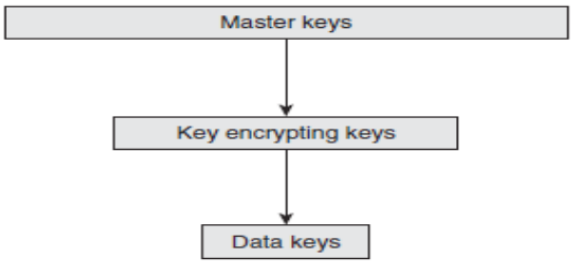


Figure 10.2. A three-level key hierarchy

MANAGING THE TOP-LEVEL KEYS

Top-level (master) keys need to be securely managed, or the whole key hierarchy is compromised. Most key management systems using key hierarchies will employ hardware security modules (HSMs) to store master keys. These top-level keys will never leave the HSMs in unprotected form.

- Master keys are commonly generated, established and backed up in component form.
- If a master key needs to be shared between two different HSMs then one option is to generate the same master key from components separately on each HSM.

- An alternative is to run a key agreement protocol between the two HSMs in order to establish a shared master key.

SCALABLE KEY HIERARCHIES

A key hierarchy works for a relatively simple network, but quickly becomes unmanageable for large networks.

Problem: Consider a simple two-level hierarchy consisting of only master and data keys. If there is a network of n users, then the number of possible pairs of users is 12 $n(n - 1)$. This means that, if there are 100 users then there are $12 \times 100 \times 99 = 4950$ possible pairs of users. Hence, in the worst case, we might have to establish 4950 separate master keys amongst the 100 HSMs in the network, which is not practical.

Alternatively, install the same master key in all HSMs. Data keys for communication between Alice and Bob could then be derived from the common master key and Alice and Bob's identities.

However, compromise of Alice's HSM would now not only compromise data keys for use between Alice and Bob, but data keys between *any* pair of users in the network. This is not normally acceptable.

Solution:

- It is common to deploy the services of a trusted third party, whom all the users trust, refer to as a *key centre* (KC).
- The idea is that each user in the network shares a key with the KC, which acts as a 'go between' any time any pairs of users require a shared key.
- In this way we reduce the need for 4950 master keys in a network of 100 users to just 100 master keys, each one shared between a specific user and the KC.

There are **two key distribution approaches** to acquiring shared keys from a KC.

Consider a simple scenario.

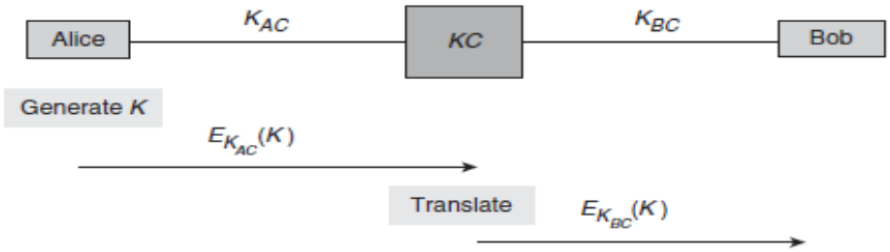
In each case let us assume that Alice wishes to establish a shared data key K with Bob, also assume that both Alice and Bob have respectively established master keys K_{AC} and K_{BC} with the KC, and that a simple two-level key hierarchy is being employed.

The two approaches are:

- ***Key translation*.**
- ***Key despatch*.**

***Key translation*.** In this approach the KC simply translates an encrypted key from encryption using one key to encryption using another. In this case the KC is acting as a type of *switch*. This process is depicted in the below figure:

1. Alice generates a data key *K*, encrypts it using K_{AC} and sends this to KC.
2. KC decrypts the encrypted *K* using K_{AC} , re-encrypts it using *KBC* and then sends this to Bob.
3. Bob decrypts the encrypted *K* using K_{BC} .



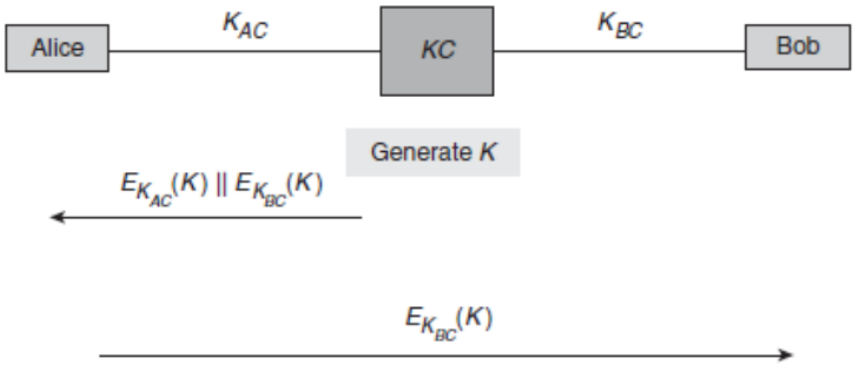
Key translation

***Key despatch*.** In this approach the KC generates the data key and produces two encrypted copies of it, one for each user. This process is depicted in the below figure and runs as follows:

1. KC generates a data key *K*, encrypts one copy of it using K_{AC} and another copy of it using K_{BC} , and then sends both encrypted copies to Alice.

2. Alice decrypts the first copy using K_{AC} and sends the other copy to Bob.

3. Bob decrypts the second copy using K_{BC} .



Key despatch

10.4.2 Unique key per transaction schemes

Unique key per transaction **(UKPT)** schemes is a different way of establishing a cryptographic key and they establish a new key each time that they are used.

MOTIVATION FOR UKPT SCHEMES

Previous key establishment mechanisms involves one, or both, of the following:

• Use of long-term (top-level) secret keys, for example, the use of master keys or key encrypting keys in key hierarchies.

• A special transfer of data explicitly for the purposes of key establishment. This applies to every technique except key predistribution.

- ➢ The first requires devices that can securely store and use long-term keys, and the second introduces a communication overhead.
- ➢ One of the reasons that most of the previous schemes require above mentioned features is that the new key that is being established has been generated *independently*, in the sense that it has no relationship with any existing data.
- ➢ An alternative methodology is to generate new keys by deriving them from information already shared by Alice and Bob.
- ➢ If key derivation is used to generate new keys then the processes of key generation and key establishment essentially 'merge'. This brings several advantages:
    1. Alice and Bob do not need to store a long-term key;
    2. Alice and Bob are not required to engage in any special communication solely for the purpose of key establishment;
    3. Key generation and establishment can be 'automated'.

APPLICATION OF UKPT SCHEMES

UKPT schemes adopts the methodology where it updates the keys using a key derivation process after each use.

A good example of an application of UKPT schemes is retail point-of-sale terminals, which are used by merchants to verify PINs and approve payment card transactions.

The advantages of a UKPT scheme all apply to this scenario:

1) Terminals have limited security controls, since they must be cheap enough to deploy widely.
    - In addition, they are typically located in insecure public environments such as stores and restaurants.
    - They are also portable, so that they can easily be moved around, hence easily stolen.
2) Transactions should be processed speedily to avoid delays, hence efficiency is important.
3) Terminals may be managed and operated by unskilled staff, hence full automation of the key establishment process is a necessity.

EXAMPLE UKPT SCHEMES :

- Consider a UKPT scheme operating between a merchant *terminal* and a *host* (a bank or card payment server). The terminal maintains a *key register*, which is essentially the running 'key' that will be updated after every transaction.
- A generic UKPT scheme in terms of the protocol that is run between the terminal and the host during a transaction.

Note:

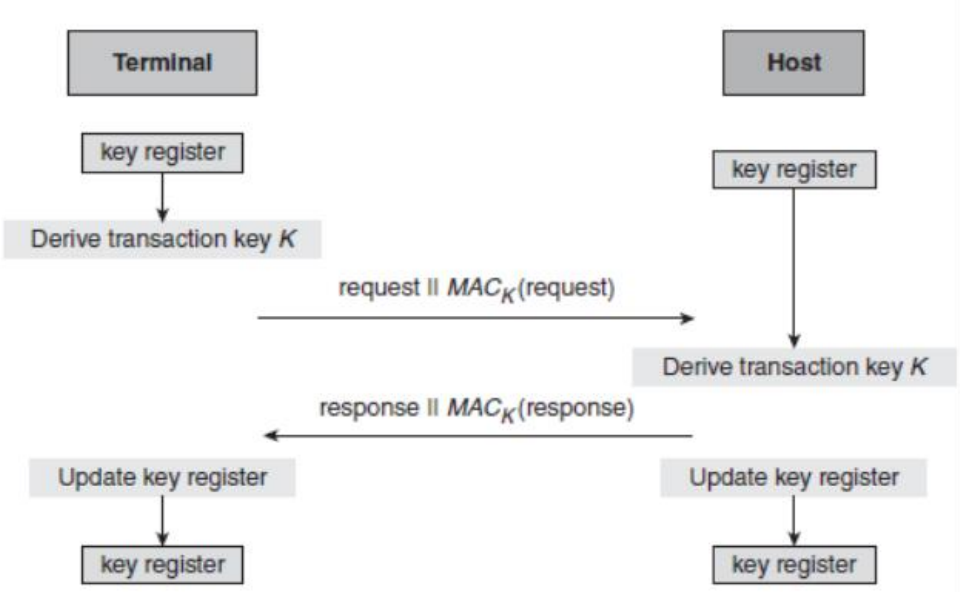
• Let us assume at the start of the protocol that the terminal and the host share an initial value that is stored in the terminal key register. This may or may not be a secret.

• a simple protocol that uses a single *transaction key* to compute MACs on the exchanged messages.

for example, an encryption key might also be needed to encrypt the PIN of the card.

Below figure illustrates generic UKPT scheme:

1. The terminal derives the transaction key using the contents of the key register and shared information that will be available to the host.
2. The terminal sends a *request* message to the host. The transaction key is used to compute a MAC on the request message.
3. The host derives the transaction key.
4. The host validates the MAC on the request message.
5. The host sends a *response* message to the terminal. The transaction key is used to compute a MAC on the response message.
6. The terminal validates the MAC on the response message.
7. The terminal updates the contents of the key register.



Generic UKPT scheme

Two examples of real UKPT schemes are:

Racal UKPT scheme:

1. The initial value is a secret seed, which is agreed between the terminal and the host.
2. The host maintains an identical key register to the terminal. The transaction key is derived from the key register and the card data (more precisely, the primary account number on the card), both of which are known by the terminal and the host.
3. At the end of the protocol, the new key register value is computed as a function of the old key register value, the card data (primary account number) and the transaction data (more precisely, the two *MAC residues* of the MACs on the request and response messages, both of which can be computed by the host and the terminal but neither of which are transmitted during the protocol,

Both the terminal and the host conduct the same computation to update their key registers.

Derived UKPT scheme. This scheme is supported by Visa:

1. The initial value is a unique initial key that is installed in the terminal.
2. The transaction key is derived by the terminal from the contents of the terminal key register, a transaction counter, and the terminal's unique identifier. The host has a special base (master) key. The host does not need to maintain a key register, but can calculate this same transaction key from the base key, the transaction counter and the terminal identifier.
3. At the end of the protocol the new terminal key register value is derived from the old key register value and the transaction counter. The host does not need to store this value because it can compute transaction keys directly.

10.4.3 Quantum key establishment

MOTIVATION FOR QUANTUM KEY ESTABLISHMENT

- One motivation for *quantum key establishment*, is an attempt to make the establishment of long, shared, randomly generated symmetric keys 'easy'.
- Quantum key establishment is a technique for establishing a conventional symmetric key, which can then be used in any symmetric cryptosystem, including a one-time pad.

Problem: In a conventional communication channel, one simple way of establishing a symmetric key is for Alice to generate a key and then send it to Bob. The problem with this approach is that an attacker could be listening in on the communication and thus learn

the key. Even worse, neither Alice nor Bob would be aware that this has happened.

THE BASIC IDEA

- Quantum key establishment takes place over a *quantum channel*.
- This is typically instantiated by an optical fibre network or free space.
- Alice and Bob must have devices capable of sending and receiving information that is encoded as quantum states, often termed *qubits*, which are the quantum equivalent of bits on a conventional communication channel. These qubits are represented by *photons*..
- The basic idea behind quantum key establishment is to take advantage of the fact that in a quantum channel such an attacker cannot 'listen in' without changing the information in the channel. This is a very useful property, which Alice and Bob can exploit to test whether an attacker has been listening to their communication.

The most well known quantum key establishment protocol is the *BB84 protocol*. The BB84 protocol involves the following steps:

- Alice randomly generates a stream of qubits, and sends these as a stream of polarised photons to Bob.
- Bob measures them using a *polarisation detector*, which will return either a 0 or a 1 for each photon.
- Bob contacts Alice over a conventional authenticated channel (perhaps a secure email, a telephone call, or a cryptographically authenticated channel), and Alice then provides him with information that probably results in Bob discarding approximately 50% of the measurements that he has just taken. This is because there are two different types of polarisation detector that Bob can use to measure each photon, and if he chooses the wrong one then the resulting measurement has only a 50% chance of being correct. Alice advises him over the authenticated channel which polarisation detector she used to encode each qubit, and Bob throws away the returns of all the wrongly measured photons.
- Alice and Bob now conduct a check over the authenticated channel on the stream of bits that they think they have just agreed upon. They do this by randomly choosing some positions and then check to see if they both agree on the bits in these positions. If they find no discrepancies then they throw away the bits that were used to conduct the check, and form a key from the bits that they have not yet checked.

QUANTUM KEY ESTABLISHMENT IN PRACTICE

There are a number of substantial limitations of quantum key establishment. These include:

Distance limitations. Implementations of quantum key establishment has still only been demonstrated to work over limited distances. For example, in 1988 it was shown to work over a 30 cm distance. This had improved by 2010 to around 150 km in an optical fibre network, and several demonstration networks had been built that used quantum key establishment. In contrast, there are no technical limits on the distance over which most conventional key establishment techniques can be used.

Data rates. There are limits to the rate at which key material can be exchanged over the quantum channel. This is also related to the distance over which the key establishment is being conducted.

Cost. Use of quantum key establishment requires expensive hardware devices and suitable quantum channels. Although the associated costs will doubtless reduce over time, most conventional key establishment techniques do not require such special technology.

The need for conventional authentication. Quantum key establishment requires a conventional means of authentication to be used. For example, in the BB84 protocol it is important that Alice and Bob establish an authenticated channel. One way is to use symmetric cryptography. So how do they establish the key used for authentication? If a conventional key establishment technique is used then the security of the quantum key establishment relies on the security of conventional key establishment.

10.5 Key storage

Secret keys need to be protected from exposure to parties other than the intended ‘owners’. It is thus very important that they are stored securely.

10.5.1 Avoiding key storage

- The best solution of all would be not to store cryptographic keys anywhere and just generate them on the fly whenever they are required.
- Since the same key must be generated on the fly every time we need to use it, we require a deterministic key generator to generate the key.
- Deterministic generators require a seed, so we will require this seed to be consistently used each time we generate the key so the seed is stored inside the human brain in the form of a passphrase or strong password.
- The user generates a passphrase, which they are required to remember.
- The passphrase is then used as a seed for a deterministic generator that generates the key encrypting key on the fly.

- The key encrypting key is then used to decrypt the encrypted private key.

Drawback: is that the security of the stored key is now dependent on the security of the seed (passphrase) that is used to generate the key encrypting key.

But it is not always possible to avoid storing a key. For example:

- Suppose that a symmetric key is being used to secure communication between Alice and Bob, who are in different locations. In some applications Alice and Bob may be able to locally generate the key precisely when they require it. However, in many other applications the key will need be stored somewhere, at least for a short while (for example, if Alice and Bob are both issued with the key in advance by a mutually trusted third party).
- Many uses of cryptography require long-term access to certain keys. For example, keys used for secure data storage may themselves need to be stored for a long time in order to facilitate future access to the protected data.
- Public-key pairs are expensive to generate. Generating them precisely when they are needed is inefficient. In many cases this is impossible, since the devices on which the private keys reside (for example, a smartcard) may have no user interface. Thus private keys almost always need to be securely stored.

10.5.2 Key storage in software

- One option for storing a cryptographic key is to embed the key into software.
- One common approach is to try to 'hide' the keys somewhere in the software.
- There are two fundamental problems with hiding cryptographic keys in software:
    1. The developer who designs the software will know where the keys are, so there is at least one potential attacker out there who knows where to look for the keys.
    2. Assuming that the hidden keys are specific to different versions (users) of the software, an attacker who obtains two versions of the software could compare them. Any locations where differences are noted are potential locations of material relating to a key.

STORING KEYS USING CRYPTOGRAPHY

There are really only four options:

- **Encrypt it with yet another key**. So where do we store that key?
- **Generate it on the fly**.

- **Store it in hardware**. This is the most common approach but, requires access to a suitable hardware device. The key encrypting key remains on the hardware device, which is also where all encryption and decryption using this key is performed.
- **Store it in component form**. By using components we make the task of obtaining a key harder since, in order to recover the key, all of the necessary components need to be obtained. As components are essentially keys themselves, hence not easily memorised, the most common way to store components is on hardware (such as a smart card). Thus component form is really a strengthening of a hardware- based solution, not an alternative.

10.5.2 Key storage in hardware

HARDWARE SECURITY MODULES

- The securest hardware storage media for cryptographic keys are *hardware security modules* (HSMs).
- These dedicated hardware devices that provide key management functionality are known as *tamper-resistant devices*.
- Many HSMs can also perform bulk cryptographic operations, often at high speed. An HSM can either be peripheral or can be incorporated into a more general purpose device such as a point-of-sale terminal.
- HSMs are mechanisms for the secure storage of cryptographic keys, HSMs are often used to enforce other phases of the key lifecycle.
- Keys stored on HSMs are physically protected by the hardware. If anyone attempts to penetrate an HSM, for example, to extract a key from the device, tamper-resistant circuitry is triggered and the key is normally deleted from the HSM's memory.
- There are various techniques that can be used to provide tamper resistance. These include:
  - ➢ **Micro-switches**. A simple mechanism that releases a switch if an HSM is opened. This is not particularly effective, since a clever attacker can always drill a hole and use glue to force the switch off.
  - ➢ **Electronic mesh**. A fine-gauge electronic mesh that can be attached to the inside of an HSM case. This mesh surrounds the sensitive components. If broken, it activates the tamper-detection circuitry. This mechanism is designed to protect against penetrative attacks, such as drilling.

- ➢ **Resin**. A hard substance, such as epoxy resin, that can be used to encase sensitive components. Sometimes electronic mesh is also embedded in resin. Any attempt to drill through the resin, or dissolve the resin using chemicals, will generally damage the components and trigger the tamper-detection circuitry.
- ➢ **Temperature detectors**. Sensors that are designed to detect variations in temperature outside the normal operating range. Abnormal temperatures may be an indication of an attack. For example, one type of attack involves, literally, freezing the device memory.
- ➢ **Light-sensitive diodes**. Sensors that can be used to detect penetration or opening of an HSM casing.
- ➢ **Movement or tilt detectors**. Sensors that can detect if somebody is trying to physically remove an HSM. One approach is to use mercury tilt switches, which interrupt the flow of electrical current if the physical alignment of an HSM changes.
- **Voltage or current detectors**. Sensors that can detect variations in voltage or current outside the normal operating range. Such anomalies may be indication of an attack.
- **Security chips**. Special secure microprocessors that can be used for cryptographic processing within an HSM. Even if an attacker has penetrated all the other defences of an HSM, the keys may still remain protected inside the security chip.

KEY STORAGE ON AN HSM

There is at least one key, often referred to as a ***local master key*** **(LMK),** that resides inside the HSM at all times. Some HSMs may store many LMKs, each having its own specific use. Any other keys that need to be stored can either be:

1. stored inside the HSM;
2. stored outside the HSM, encrypted using an LMK.

- When a key stored outside the HSM needs to be used, it is first submitted to the HSM, where it is recovered using the LMK and then used.
- This approach places a great deal of reliance on the LMK. It is thus extremely important to back up the LMK in order to mitigate against loss of the LMK. Such loss can occur if the HSM fails, or if it is attacked, since the tamper-resistance controls are likely to delete the HSM memory.

- Indeed this applies to any keys that are only stored inside the HSM. Thus the issue of whether to store a key inside or outside the HSM involves a tradeoff between:

Efficiency – storing keys inside the HSM is more efficient in terms of processing speed since they do not need to be imported and then recovered before use.

Need for backups – since every key only stored inside the HSM needs to be securely backed up, perhaps in component form.

10.5.4 Key storage risk factors

The risks to key storage media depend not only on the devices on which keys are stored, but also on the environments within which the devices reside. This relationship is indicated in Figure 10.6, which identifies four zones based on different environmental and device controls.

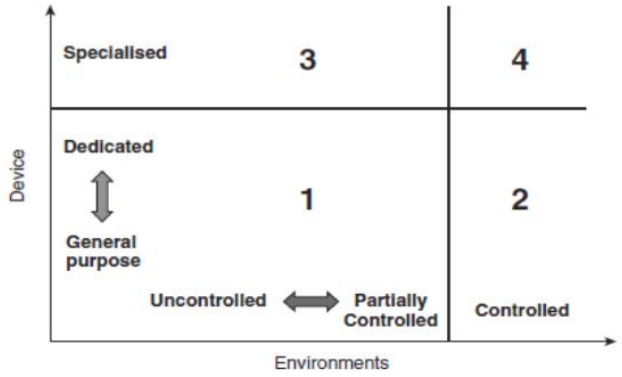


Figure 10.6. Key storage risk zones

The two dimensions depicted in Figure 10.6 represent:

Environments, which range from

Uncontrolled: public environments such as shops and restaurants, where it is not possible to implement strict access control mechanisms;

Partially controlled: environments such as general offices and homes, where it is possible to implement basic access control mechanisms (for example, a physical door key);

Controlled: environments such as high-security offices and military installa- tions, where it is possible to implement strong access control mechanisms (for example, biometric swipe cards).

Devices, which range from

General purpose: general devices running conventional operating systems with their default in-built security controls (for example, a laptop);

Dedicated: dedicated devices that offer some specialist security controls, such as limited tamper resistance (for example, a point-of-sale terminal or a mobile phone);

Specialised: specialised devices whose main functionality is to provide security (for example, an HSM).

The four zones identified in Figure 10.6 are mainly conceptual, but illustrate the importance of both dimensions.

Zone 1. This is the lowest security zone and thus offers the highest risk. However, for many applications this may provide sufficient security. For example, a key stored in encrypted form on the hard disk of a home PC may well be good enough for protection of the user's personal files. Likewise, any keys stored under the limited protection of a portable point-of-sale terminal are probably still secure from anything other than attacks from experts.

Zone 2. The security offered by Zone 1 devices is increased substantially when they are moved into a controlled environment. In the extreme, a key stored in the clear in software on a general PC provides excellent security if the PC is not networked and is kept in a physically secure room with an armed guard at the door! More realistically, encrypted keys stored on PCs that are located in an office with strong physical security (such as smart card access control to the rooms) and good network security controls should have better protection than those on a PC located in a public library or an internet cafe.

Zone 3. Specialised devices sometimes have to be located in insecure environ- ments because of the nature of their application. A good example is provided by Automated Teller Machines (ATMs), which need to be 'customer facing'. Such devices are thus exposed to a range of potentially serious attacks that are made possible by their environment, such as an attacker attempting to physically remove them with the intention of extracting keys back in a laboratory.

Zone 4. The highest-security zone is provided when a specialist device is kept in a controlled environment. This is not just the most secure, but the most expensive zone within which to provide solutions. This level of security is nonetheless appropriate for important keys relating to high-security applications such as data processing centres, financial institutions, and certification authorities.

10.5.5 Key backup, archival and recovery

For example:

- Data stored in encrypted form will itself be lost if the corresponding decryption key is lost, since nobody can recover the data from the ciphertext;
- A digital signature on a message becomes ineffective if the corresponding verification key is lost, since nobody has the ability to verify it.
    - ➢ The first scenario illustrates the potential need for key backup of critical secret keys.

➢ The second scenario more broadly illustrates the potential need for key archival, which is the long-term storage of keys beyond the time of their expiry.

KEY BACKUP

- Important keys are often stored on HSMs.
- An obvious attack against a Zone 3 (see Figure 10.6) HSM would be to physically attack the HSM to the point that one of its tamper-resistant triggers is activated and the device wipes its memory. The attacker does not learn the keys stored on the device but, without a backup, the potential impact on the organisation relying on the HSM is high.
- Zone 4 HSMs are subject to risks such as a careless cleaner bumping into a device and accidentally wiping its memory.
- the security of a key backup process must be as strong as the security of the key itself.
- The backed-up key will need to be stored on media that is subject to at least the same level of device and environmental security controls as the key itself.
- Indeed for the highest levels of key, the use of component form might be the only appropriate method for key backup.

KEY ARCHIVAL

- Key archival is essentially a special type of backup, which is necessary in situations where cryptographic keys may still be required in the period between their expiry and their destruction. Such keys will no longer be 'live' and so cannot be used for any new cryptographic computations, but they may still be required.

For example:

- There may be a legal requirement to keep data for a certain period of time. If that data is stored in encrypted form then there will be a legal requirement to keep the keys so that the data can be recovered.
- Managing the storage of archived keys is just as critical as for key backups.
- Once a key no longer needs to be archived, it should be destroyed.

KEY RECOVERY

- Key recovery is the key management process where a key is recovered from a backup or an archive.
- The term 'key recovery' is also associated with initiatives to force a 'mandatory'

backup, also referred to as key escrow.

- The idea behind key escrow is that if any data is encrypted then a copy of the decryption key is stored (escrowed) by a trusted third party in such a way that, should it be necessary and the appropriate legal authority obtained, the decryption key can be obtained and used to recover the data. Such a situation might arise if the encrypted data is uncovered in the course of a criminal investigation.
- Many suggested key escrow mechanisms employed component form storage of escrowed keys in an attempt to reassure potential users of their security.

10.6 Key usage

10.6.1 Key separation

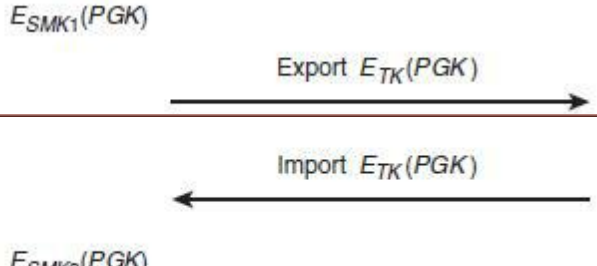
The principle of key separation is that cryptographic keys must only be used for their intended purpose.

THE NEED FOR KEY SEPARATION

- In many applications the need for key separation may be quite obvious. For example, it may be the case that encryption and entity authentication are conducted by distinct processes.
- In other applications it may be tempting to use a key that has already been established for one purpose and then use it for some other purpose.
- Disadvantages of doing this are:

Example 1. PINs are often stored in encrypted form using a symmetric PIN encrypting key. This key should only ever be used to encrypt PINs. It should never be used to decrypt an encrypted PIN. In contrast a normal symmetric data key is used both for encryption and decryption. If these two keys are somehow interchanged within an HSM then it may become possible to decrypt and reveal a PIN or it may not be possible to recover any normal data encrypted with the PIN encrypting key.

- One method of enforcing key separation in an HSM is to store keys in the HSM encrypted under a master key that is specific to one usage purpose.
- Many HSMs have export and import functions that allow keys to be transferred between different HSMs.
- Keys are encrypted using a transport key during export and import. Figure 10.7 shows the apparent usage purpose of a key.



1. A PIN generation key PGK is stored on the HSM, encrypted by a storage master key SMK1, which is the local key on the HSM that is used to store PIN generation keys.
2. The HSM is instructed to export PGK. It decrypts the encrypted PGK using SMK1, then re-encrypts PGK using the transport key TK. This is then exported.
3. The HSM is then instructed by the attacker to import a new MAC key. The attacker submits PGK, encrypted under TK.
4. The HSM decrypts the encrypted PGK using TK, then re-encrypts it using storage master key SMK2, which is the HSM key used to store MAC keys. The HSM thus regards PGK as a MAC key.

This attack will not be possible if different variants of transport key are used for separate export and import functions.

ENFORCING KEY SEPARATION

Keys are often unstructured bit strings.There are two main techniques that can be used to enable the purpose of a key to be distinguished:

1. **Encrypting a key using a specified variant key**. This is a hardware-enforced method which involves using specific higher level keys to encrypt keys for particular purposes. This technique can be applied to keys being distributed, as well as keys being stored. This method can be used to enforce any type of key separation.
2. **Embedding the key in a larger data block**. This involves embedding the key into a larger data object that also includes a statement on the key usage.

Examples:

1. **Employing redundancy.** DES key has an effective length of 56 bits, but is usually a 64-bit value. Thus, there are 8 ‘spare’ bits that can be used for other purposes. The original DES standard recommends that the spare bits be used to provide error detection in the event that a DES key becomes corrupted. Key tagging allows the eight spare bits to define the key usage. When a key is presented in a command to an HSM, the tagging bits are checked by the HSM to ensure that the key is a valid key for the command that

it is being used for.

2. **Key blocks.** This is a formatted data string that allows a key to be represented along with other data relating to the key. One example is the ANSI TR-31 key block, which is shown in Figure 10.8 and has the following fields:

Figure 10.8. ANSI TR-31 key block

- Header includes information that clarifies the purpose of the key.
- Optional header includes optional data such as the expiry date of the key.
- Key is encrypted using a suitable key encrypting key.
- Authenticator is a MAC on the rest of the key block, which provides data origin authentication (data integrity) of the key block data.

iii. **Public-key certificates.** These are types of key block used to provide assurance of purpose for public keys. A public-key certificate often includes a field that defines the key usage.

10.6.2 Key change

THE NEED FOR KEY CHANGE

1. **Planned key changes**. These will most likely occur at regular intervals. One reason for a planned key change might be the end of the key lifetime.
2. **Unplanned key changes**. These may occur for a variety of reasons.

- A key is compromised
- A security vulnerability such as operating system vulnerability, a breakthrough in cryptanalysis, or a failure of a tamper-resistance mechanism in an HSM.
- An employee unexpectedly leaves an organization.

IMPACT OF KEY CHANGE

- Key change can be a very expensive process.
- An unplanned key change is particularly problematic.
- The minimum impact of a key change is that a new key needs to be generated and established.
- The impact can be severe, especially in the case of high-level key compromise. For example, if a master key is compromised in a financial system then the resulting costs

might include costs of investigation into the compromise, costs related to any transactions conducted using compromised keys, damage to reputation and loss of customer confidence.

- Generation and establishment of a new key.
- Withdrawing the old key (destroying or archiving it).

Planned key changes should happen automatically and require very little intervention.

More intervention may be required in the case of unplanned key changes.

There are two reasons why changing public-key pairs is normally more challenging:

1. **Knowledge of public keys**. The 'public' nature of a public key means that a key management system may have little control over which entities have knowledge of a public key. In open environments such as the Internet, a public key could be known by anyone.
2. Open application environments. Public-key cryptography tends to be used in open environments where this may be more challenging. Since private and public keys are interdependent, any requirement to change one of them requires the other also to be changed. Changing a private key is simpler than changing a symmetric key.

10.6.3 Key activation

In many applications key activation requires human interaction. As an example, consider a signature key stored on a computer for digitally signing emails. If RSA is being used then this signature key might be up to 2048 bits long, which is a value that the human owner of the keywill not be capable of memorizing. When the user decides to digitally sign an email, the user needs to instruct the email client to activate their signature key. Several scenarios depending on how the key is stored on the computer. These include:

1. **Key stored on the computer in the clear**. the user might activate the key simply by entering an instruction, selecting the key from a list of potential keys stored on the computer. Key activation is possible for anyone with access to the computer. In this case the effective security of the keys is simply linked to the security required to access the computer itself, which requires a valid username and password.
2. **Key stored on the computer in encrypted form**. The user might activate the key in this case by providing some secret identity information, such as a passphrase. This passphrase would then be used to generate the key that can be used to recover the signature key. In this case the effective security is linked to the security of the

passphrase.

3. **Key generated on the fly**. In this case the key is not stored on the computer, but is generated on the fly. The activation of the key is linked to the generation of the key. One way of implementing this is to request some identity information such as a passphrase from the user. Thus the effective security of the key is again determined by the security of this passphrase.
4. **Key stored off the computer**. Key is stored on a peripheral device. The key activation takes place when the user connects the device to the computer. In this case the effective security is linked to the security of the peripheral device.

10.6.4 Key destruction

When a key is no longer required for any purpose then it must be destroyed in a secure manner. The point at which key destruction is required may either be:

1. When the key expires (the natural end of the key's lifetime).
2. When the key is withdrawn (before its expiry, in the event of unplanned events).
3. At the end of a required period of key archival.

- Since keys are sensitive data, secure mechanisms must be used. Suitable techniques are sometimes referred to as data erasure or data sanitization mechanisms.
- Simply deleting a key from a device is not sufficient if the key is to be truly destroyed.
- Operating systems may well have other (temporary) copies of the key in different locations.
- Even if the key was stored on the device in encrypted form, this may be useful to a determined attacker. Any other media storing information about keys, such as paper, should also be destroyed.

10.7 Governing key management:

- Key management is the main interface between the technology of cryptography and the users and systems that rely on it.
- Key management is a much more complex process for an organisation, due to the diversity of processes that affect key management,

10.7. Key management policies, practices and procedures

Within an organisation, the most common way to govern key management is through the specification of:

- **Key management policies.** These define the overall requirements and strategy for providing key management. For example, a policy might be that all cryptographic keys are stored only in hardware.
- **Key management practices.** These define the tactics that will be used in order to achieve the key management policy goals. For example, that all devices using cryptography will have an in-built HSM.
- **Key management procedures**. These document the step-by-step tasks necessary in order to implement the key management practices. For example, the specification of a key establishment protocol that will be used between two devices.

Important outcome of key management governance is:

- **By design**: in other words, that the entire key management lifecycle has been planned from the outset, and not made up in response to events as they occur.
- **Coherent**: the various phases of the key lifecycle are considered as linked component parts of a larger unified process and designed with 'big picture' in mind.
- **Integrated**: the phases of the key management lifecycle are integrated with the wider requirements and priorities of the organisation.

10.7.2 Example procedure: key generation ceremony

- **Key ceremony**: It is an important type of key management procedure that might be required by a large organization and used to implement key generation from components.
-

In key generation ceremony the participants are:

1. **Operation manager:** responsible for the physical aspects, including the venue, hardware, software and any media on which components are stored or transported.
2. **Key manager:** responsible for making sure that the key ceremony is performed in accordance with the relevant key management policies, practices and procedures.
3. **Key custodians**: the parties physically in possession of the key components, responsible for handling them appropriately and following the key ceremony as instructed.
4. **Witnesses:** responsible for observing the key ceremony and providing independent assurance that all other parties perform their roles in line with the appropriate policies,

practices and procedures.

Key ceremony involves several phases:

1. **Initialization**. The operation manager installs and configures the required hardware and software, including the HSM, within a controlled environment. This process might need to be recorded by witnesses.
2. **Component retrieval**. The components required for the key ceremony are transported to the key ceremony location.
3. **Key generation/establishment**. The key is installed onto the HSM under the guidance of the key manager. This process will involve the various key custodians taking part in the key ceremony, but not necessarily simultaneously. Throughout the key ceremony, the witnesses record the events and any deviations from the defined procedure are noted. At the end, an official record is presented to the key manager.
4. **Validation**. The official record can be scrutinized to validate that the correct procedure was followed.

Public-Key Management

11.1 Certification of public keys

The main challenge for the management of public keys is providing assurance of purpose of public keys. The most popular mechanism for providing this assurance of purpose, the ***public-key certificate*.**

11.1.1 Motivation for public-key certificates

A SCENARIO

Suppose that Bob receives a digitally signed message that claims to have been signed by Alice and that Bob wants to verify the digital signature. This requires Bob to have access to Alice's verification key. Suppose that Bob is presented with a key that is alleged to be Alice's verification key.

Bob uses this key to 'verify' the digital signature and it appears to be correct. What guarantees does Bob have that this is a valid digital signature by Alice on the message? And many more questions:

- *Does the verification key actually belong to Alice?*
- *Could Alice deny that this is her verification key?*
- *Is the verification key valid?*
- *Is the verification key being used appropriately?*

PROVIDING ASSURANCE OF PURPOSE

The above scenario requires:

1. provide a 'strong association' between a public key and the *owner* of that key (the entity whose identity is linked to the public key);
2. provide a 'strong association' between a public key and any other relevant data (such as expiry dates and usage restrictions).

PROVIDING A POINT OF TRUST

The problem in designing any public-key management system is that we need to find a source for the provision of the 'strong association' between a publickey value and its related data. In public-key management systems this is provided by introducing points of trust in the form of trusted third parties who 'vouch' for this association.

USING A TRUSTED DIRECTORY

An approach to providing assurance of purpose for public keys is to use a trusted 'directory', which lists all public keys next to their related data (including the name of the owner). Anyone requiring assurance of purpose of a public key, simply looks it up in the trusted directory.

This approach has several significant problems:

- **Universality**. The directory has to be trusted by all users of the public-key management system.
- **Availability**. The directory has to be online and available at all times to users of the public-key management system.
- **Accuracy**. The directory needs to be maintained accurately and protected from unauthorised modification.

11.1.2 Public-key certificates

A *public-key certificate* is data that binds a public key to data relating to the assurance of purpose of that public key.

CONTENTS OF A PUBLIC-KEY CERTIFICATE

A public-key certificate contains four essential pieces of information:

- **Name of owner**:

The name of the owner of the public key. This owner could be a person, a device, or even a role within an organisation. The format of this name will depend upon the application, but it should be a unique identity that identifies the owner within the environment in which the public key will be employed.

- **Public-key value**:

The public key itself. This is often accompanied by an identifier of the cryptographic algorithm with which the public key is intended for use.

- **Validity time period**. This identifies the date and time from which the public key is valid and, more importantly, the date and time of its expiry.
- **Signature**. The creator of the public-key certificate digitally signs all the data that forms the public-key certificate, including the name of owner, public-key value and validity time period. This digital signature not only binds all this data together, but is also the guarantee that the creator of the certificate believes that all the data is correct. This provides the 'strong association'.

Example:

Table 11.1: Fields of an X.509 Version 3 public-key certificate

Field	Description
Version	Specifies the X.509 version being used (in this case V3)
Serial Number	Unique identifier for the certificate
Signature	Digital signature algorithm used to sign the certificate
Issuer	Name of the creator of the certificate
Validity	Dates and times between which the certificate is valid
Subject	Name of the owner of the certificate
Public-Key Info.	Public-key value; Identifier of public-key algorithm
Issuer ID	Optional identifier for certificate creator
Subject ID	Optional identifier for certificate owner
Extensions	A range of optional fields that include:
	Key identifier (in case owner owns several public keys);
	Key usage (specifies usage restrictions);
	Location of revocation information;
	Identifier of policy relating to certificate;
	Alternative names for owner.

INTERPRETING A PUBLIC-KEY CERTIFICATE

- A public-key certificate cannot be used to encrypt messages or verify digital signatures.
- A public-key certificate is not a proof of identity.

PUBLIC-KEY CERTIFICATE CREATORS

A creator of a public-key certificate is referred to as a certificate authority (CA). The certificate authority normally plays three important roles:

1. **Certificate creation:** The CA takes responsibility for ensuring that the information in a public-key certificate is correct before creating and signing the public-key certificate, and then issuing it to the owner.
2. **Certificate revocation**: The CA is responsible for revoking the certificate in the event that it becomes invalid.
3. **Certificate trust anchor**. The CA acts as the point of trust for any party relying on the correctness of the information contained in the public-key certificate. To fulfill this role, the CA will need to actively maintain its profile as a trusted organization. It may also need to enter into relationships with other organizations in order to facilitate wider recognition of this trust.

RELYING ON A PUBLIC-KEY CERTIFICATE

In order to obtain assurance of purpose of the public key follow three steps:

1. Trust the CA: The relying party needs to be able to trust the CA to have performed its job correctly when creating the certificate.

2. Verify the signature on the certificate: The relying party needs to have access to the verification key of the CA in order to verify the CA's digital signature on the public-key certificate. If the relying party does not verify this signature then they have no guarantee that the contents of the public-key certificate are correct.

3. **Check the fields**: The relying party needs to check all the fields in the public-key certificate. In particular, they must check the name of the owner and that the public-key certificate is valid. If the relying party does not check these fields then they have no guaranteed that the public key in the certificate is valid for the application for which they intend to use it.

DIGITAL CERTIFICATES

Public-key certificates represent a special class of digital certificates. An example of another type of digital certificate is an attribute certificate, which can be used to provide a strong association between a specific attribute and an identity, such as:

• The identity specified is a member of the access control group administrators.

• The identity specified is over the age of 18.

Attribute certificates might contain several fields that are similar to a public-key certificate (for example, owner name, creator name, validity period) but would not contain a public-key value.

11.2 The certificate lifecycle

11.2.1 Differences in the certificate lifecycle

1. Key generation.

The generation of a public-key pair is an algorithm-specific, and often technically complex, operation. Creation of a public-key certificate is even harder from a process perspective, since it involves determining the validity of information relating to the public key.

2. **Key establishment**.

Private Key establishment is potentially easier than symmetric key establishment since the private key only needs to be established by one entity. This entity could even generate the private key themselves. If another entity generates the private key then private key establishment may involve the private key being distributed to the owner using a secure channel of some sort, such as physical distribution of a smart card on which the private key is installed. Public-key certificate establishment is not a sensitive operation, since the public-key certificate does not need to be kept secret. Most techniques can either be described as:

Pushing a public-key certificate: The owner of the public-key certificate provides the certificate whenever it is required by a relying party.

Pulling a public-key certificate: The relying parties must retrieve public-key certificates from some sort of repository when they first need them. One potential advantage of pulling public-key certificates is that they could be pulled from a trusted database that only contains valid public-key certificates.

3. Key storage, backup, archival:

They are all less-sensitive operations when applied to public key certificates.

4.Key usage:

Many public-key certificate formats, such as the X.509 Version 3 certificate format include fields for specifying key usage.

5.Key destruction:

Destruction of public-key certificates is less sensitive, and may not even be required.

11.2.2 Certificate creation

LOCATION OF KEY PAIR AND CERTIFICATE CREATION

- Key pair generation and Certificate creation two separate processes here:
- Key pair generation can be performed either by the owner of the public-key pair or a trusted third party (who may or may not be the CA).

The choice of location for this operation results in different certificate creation scenarios:

➢ **Trusted third party generation**. In this scenario, a trusted third party (which could be the CA) generates the public-key pair. If this trusted third party is not the CA then they must contact the CA to arrange for certificate creation.

Advantages:

- The trusted third party may be better placed than the owner to conduct the relatively complex operations involved in generation of the public-key pair.
- The key pair generation process does not require the owner to do anything.

Disadvantages:

- The owner needs to trust the third party to securely distribute the private key to the owner;
- The owner needs to trust the third party to destroy the private key after it has been distributed to the owner

➢ **Combined generation**. In this scenario, the owner of the key pair generates the public-key pair. The owner then submits the public key to a CA for generation of the public-key certificate.

Advantages:

- **T**he owner is in full control of the key pair generation process;
- The private key can be locally generated and stored, without any need for it to be distributed.

Disadvantages:

- The owner is required to have the ability to generate key pairs;
- The owner may need to demonstrate to the CA that the owner knows the private key that corresponds to the public key submitted to the CA for certification

➢ **Self-certification**. In this scenario, the owner of the key pair generates the key pair and certifies the public key themselves. Since a public-key certificate generated by a CA provides 'independent' assurance of purpose of a public key, whereas self-certification requires relying parties to trust in the assurance of purpose provided by

the owner of the public key. However, if relying parties trust the owner then this scenario may be justifiable.

Examples:

- The owner is a CA; it is not uncommon for CAs to self-certify their own public-keys.
- All relying parties have an established relationship with the owner and hence trust the owner's certification;

for example, a small organisation using a self-certified public key to encrypt content on an internal website.

REGISTRATION OF PUBLIC KEYS

- If either **trusted third-party generation or combined generation of a publickey pair** is undertaken then the owner of the public-key pair must engage in a ***registration* process** with the CA before a public-key certificate can be issued.
- This is when the owner presents their credentials to the CA for checking.
- These credentials not only provide a means of authenticating the owner, but also provide information that will be included in some of the fields of the public-key certificate.
- Registration process that varies greatly between different applications.

In many application environments a separate entity known as a *Registration Authority* (RA) performs this operation.

The roles of RA and CA can be separated for several reasons:

- Registration involves a distinct set of procedures that generally require an amount of human intervention, whereas certificate creation and issuance can be automated.
- Checking the credentials of a public-key certificate applicant is the most complex part of the certificate creation process.
    - Centralised checking of credentials represents a major bottleneck in the process, particularly for large organisations.
    - Distributing the registration activities across a number of local RAs, which perform the checking and then report the results centrally.

What credentials should be presented to the RA during registration?

Some examples of credentials:

- A very low level of public-key certificate might simply require a valid email address to be presented at registration. The registration process might include checking that the applicant can receive email at that address.
- Registration for public-key certificates for use in a closed environment, such as an organisation's internal business environment, might involve presentation of an employee number and a valid internal email address.
- Commercial public-key certificates for businesses trading over the Internet might require a check of the validity of a domain name and the confirmation that the applicant business is legally registered as a limited company.
- Public-key certificates for incorporation into a national identity card scheme require a registration process that unambiguously identifies a citizen. Credentials might include birth certificates, passports, domestic utility statements, etc.

PROOF OF POSSESSION

If a public key and its certificate are created using combined generation then, strictly speaking, it is possible for an attacker to attempt to register a public key for which they do not know the corresponding private key. Such an **'attack'** on a verification key for a digital signature scheme might work as follows:

1. The attacker obtains a copy of Alice's verification key. This is a public piece of information, so the attacker can easily obtain this.

2. The attacker presents Alice's verification key to an RA, along with the attacker's legitimate credentials.

3. The RA verifies the credentials and instructs the associated CA to issue a publickey certificate in the name of the attacker for the presented verification key.

4. The CA issues the public-key certificate for the verification key to the attacker.

Problem:

- The attacker now has a public-key certificate issued in their name for a verification key for which they do not know the corresponding signature key.
- A problem arises if Alice now digitally signs a message with her signature key, since the attacker will be able to persuade relying parties that this is actually the attacker's digital signature on the message.
- This is because the attacker's name is on a public-key certificate containing a verification key that successfully verifies the digital signature on the message.
- This attack can be prevented if the CA conducts a simple check that the publickey certificate applicant knows the corresponding private key.

- This type of check is referred to as *proof of possession* (of the corresponding private key).

If the public key is an encryption key then one possible proof of possession is as follows:

- The RA encrypts a test message using the public key and sends it to the certificate applicant, along with a request for the applicant to decrypt the resulting ciphertext.
- If the applicant is genuine, they decrypt the ciphertext using the private key and return the plaintext test message to the RA. An applicant who does not know the corresponding private key will not be able to perform the decryption to obtain the test message.

GENERATING CA PUBLIC-KEY PAIRS

Public-key certificates involve a CA digitally signing the owner's public key together with related data. This in turn requires the CA to possess a public-key pair. This raises the question of how assurance of purpose of the CA's verification key will be provided.

Solution: is to create a public-key certificate for the CA's public key.

But who will sign the public-key certificate of the CA?

The two most common methods of certifying the CA's verification key are:

Use a higher-level CA. If the CA is part of a *chain* of CAs then the CA may choose to have their public key certified by another CA.

Self-certification. A top-level CA has no choice other than self certification.

As an example, CAs who certify public keys that are used in web-based commercial applications need to have their public-key certificates incorporated into leading web browsers, or have them certified by a higher-level CA who has done this.

11.2.3 Key pair change

REVOCATION OF PUBLIC-KEY CERTIFICATES

Withdrawing an existing public key is very difficult. This process is referred to as *revoking* the public key. Revoking a public key essentially means revoking the public-key certificate.

REVOCATION TECHNIQUES

Three ways of Revocation of public-key certificates are:

Blacklisting.

- This involves maintaining a database that contains serial numbers of public-key certificates that have been revoked.

- This type of database is referred to as a *certificate revocation list* (or CRL).
- These CRLs need to be maintained carefully, normally by the CA who is responsible for issuing the certificates, with clear indications of how often they are updated.
- The CRLs need to be digitally signed by the CA and made available to relying parties.

Whitelisting.

- This involves maintaining a database that contains serial numbers of public-key certificates that are valid.
- This database can then be queried by a relying party to find out if a public-key certificate is valid.

An example is the *Online Certificate Status Protocol* (OCSP).

Rapid expiration. This removes the need for revocation by allocating very short lifetimes to public-key certificates. This, of course, comes at the cost of requiring certificates to be reissued on a regular basis.

Blacklisting is a common technique when real-time revocation information is not required. There are many different ways of implementing the blacklisting concept, often involving networks of distributed CRLs rather than one central CRL.

The main problem with blacklisting is one of synchronisation. i.e:

- Reporting delays between the time that a public-key certificate should be revoked (for example, the time of a private key compromise) and the CA being informed;
- CRL issuing delays between the time that the CA is informed of the revocation of a public-key certificate and the time that the next version of the CRL is signed and made publicly available.

Thus a relying party could rely on a public-key certificate in the gap period between the time the public-key certificate should have been revoked and the publication time of the updated CRL. This issue must be 'managed' through suitable processes and procedures.

For example:

• The CA should inform all relying parties of the update frequency of CRLs.

• The CA should clarify who is responsible for any damage incurred from misuse of a public key in such a gap period.

Address these issues by:

– the CA accepting limited liability during gap periods;

– relying parties accepting full liability if they fail to check the latest CRL before relying on a public-key certificate.

11.3 Public-key management models

11.3.1 Choosing a CA

Choosing an organization to play the role of a CA in an open environment is less straightforward. Currently, most CAs serving open environments are commercial organizations who have made it their business to be 'trusted' to play the role of a CA. While CAs serving open environments can be regulated to an extent by commercial pressure the importance of their role may demand tighter regulation of their practices. Options for this include:

Licensing. This approach requires CAs to obtain a government license before they can operate. Government, thus, ultimately provides the assurance that a CA conforms to minimum standards.

Self-regulation. This approach requires CAs to form an industry group and set their own minimum operational standards through the establishment of best practices.

11.3.2 Public-key certificate management models

1. CA-FREE CERTIFICATION MODEL

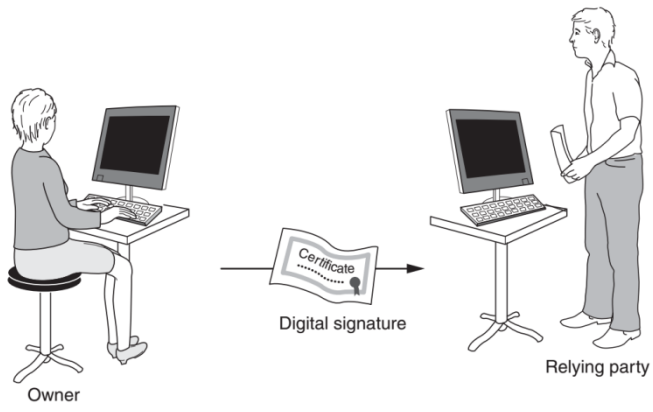


Figure 11.1. CA-free certification model

- The CA-free certification model is depicted in Figure 11.1 and applies when there is no CA involved.
- In the CA-free certification model, the owner generates a key pair and then either self-certifies the public key or does not use a public-key certificate.
- Any relying party obtains the (self-certified) public key directly from the owner.
- For example, the owner could include their public key in an email signature or write it on a business card. The relying party then has to make an independent decision as to

whether they trust the owner or not. The relying party thus carries all the risk in this model.

2. REPUTATION-BASED CERTIFICATION MODEL

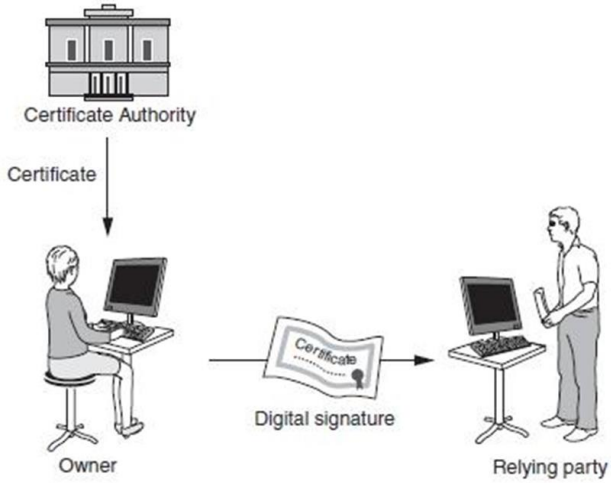


Figure 11.2. Reputation-based certification model

➢ The reputation-based certification model is depicted in Figure 11.2 and applies when the owner has obtained a public-key certificate from a CA, but the relying party has no relationship with this CA.

➢ Even if the relying party obtains the verification key of the CA, which enables them to verify the public-key certificate, because they do not have any relationship with the CA itself they do not by default gain assurance of purpose from verification of the certificate.

➢ They are left to choose whether to trust that the CA has done its job correctly and hence that the information in the public-key certificate is correct. In the worst case, they may not trust the CA at all, in which case they have no reason to trust any information affirmed by the CA.

➢ The only assurance that they might gain is through the reputation of the CA that signed the public-key certificate. If the relying party has some trust in the reputation of the CA, for example, it is a well-known organization or trust service provider, and then the relying party might be willing to accept the information in the public-key certificate.

3.CLOSED CERTIFICATION MODEL

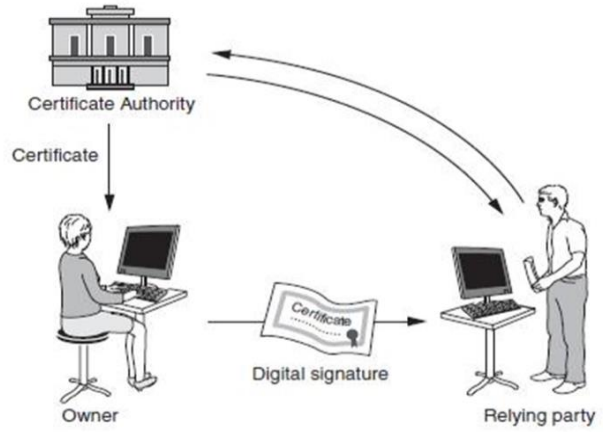


Figure 11.3. Closed certification model

- The closed certification model is depicted in Figure 11.3 and applies when the relying party has a relationship with the owner's CA.
- The closed certification model is the most 'natural' certification model, but is only really applicable to closed environments where a single CA oversees the management of all public-key certificates.
- In the closed certification model, the more onerous issues concerning public-key management, such as those relating to liability and revocation, are more straightforward to solve than for the other models.
  - This is because public key certificate owners and relying parties are all governed by the same certificate management policies and practices.

4.CONNECTED CERTIFICATION MODEL

- The connected certification model is depicted in Figure 11.4 and applies when the relying party has a relationship with a trusted third party, which in turn has a relationship with the owner's CA.
- The trusted third party that the relying party has a relationship with could be another CA.
- The role of validation authority is to assist the relying party to validate the information in the owner's public-key certificate.
  - This validation authority may not necessarily be a CA. the CA and validation authority could both be members of a federation of organizations who have agreed to cooperate in the validation of public-

key certificates and have signed up to common certificate management policies and practices.

✓ The connected certification model is a pragmatic 'stretching' of the closed certification model, in order to allow public-key certificates to be managed in environments that are either:

Open: Owners and relying parties are not governed by any single management entity. **Distributed:** In the case of a closed environment that is distributed. For example a large organization with different branches or regional offices.

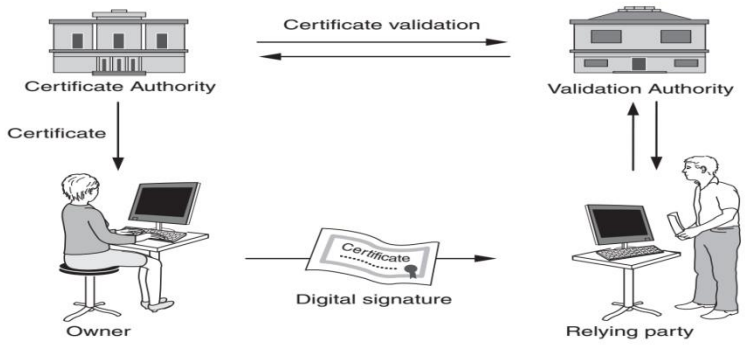


Figure 11.4. Connected certification model

11.3.3 Joining CA domains

There are three approaches:

1. **CROSS-CERTIFICATION**

Figure 11.5. Cross-certification

The first technique for joining two CA domains is to use cross-certification; each CA certifies the other CA's public key. This idea is depicted in Figure 11.5. Cross-certification implements a transitive trust relationship. By cross-certifying, relying party Bob of CA2, who wishes to trust a public-key certificate issued to Alice by CA1, can do so by means of the following argument:

1. Bob trust CA2 (because bob have a business relationship with CA2).
2. CA2 trusts CA1 (because they have agreed to cross-certify one another).
3. CA1 has vouched for the information in Alice's public-key certificate (because CA1 generated and signed it)

4. Bob trust the information in Alice's public-key certificate.

2.CERTIFICATION HIERARCHIES

The second technique is certification hierarchy consisting of different levels of CA. A higher-level CA, which both CA1 and CA2 trust, can then be used to 'transfer' trust from one CA domain to the other. The simple certification hierarchy in Figure 11.6 uses a higher-level CA (termed the root CA) to issue public-key certificates for both CA1 and CA2. Relying party Bob of CA2, who wishes to trust a public-key certificate issued to Alice by CA1, can do so by means of the following argument:

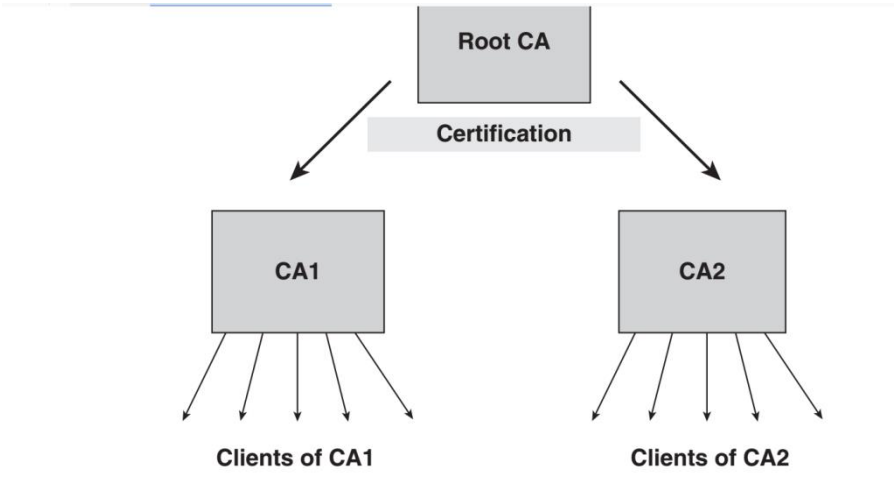


Figure 11.6. Certification hierarchy

1. Bob trust CA2 (because Bob have a business relationship with CA2).
2. CA2 trusts root CA (because CA2 has a business relationship with root CA).
3. Root CA has vouched for the information in CA1's public-key certificate (because root CA generated and signed it).
4. CA1 has vouched for the information in Alice's public-key certificate (because CA1 generated and signed it).
5. Bob trust the information in Alice's public-key certificate.

3.CERTIFICATE CHAINS

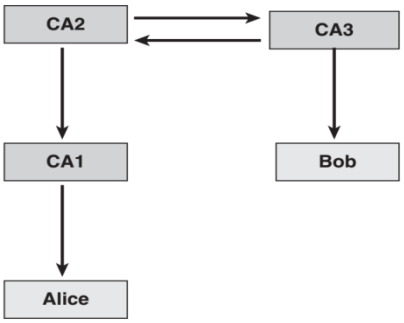


Figure 11.7. A simple CA topology

The joining of CA domains makes verification of public certificates a potentially complex process. It results in the creation of certificate chains, which consist of a series of public-key certificates that must all be verified in order to have trust in the end public-key certificate. Example: consider the apparently simple CA topology in Figure 11.7. In this topology Alice has a relationship with CA1, which is a low-level CA, whose root CA is CA2. Bob has a relationship with CA3, which has cross-certified with CA2. Now suppose that Bob wishes to verify Alice's public-key certificate. He will need to verify a certificate chain that consists of the three public- key certificates shown in Table 11.2. To properly verify the above certificate chain, for each of these public key certificates Bob should:

1. Verify the signature on the public-key certificate.
2. Check all the fields in the public-key certificate.
3. Check whether the public-key certificate has been revoked.

Table 11.2: Example of a certificate chain

Certificate	Containing public key of	Certified by
1	Alice	CA1
2	CA1	CA2
3	CA2	CA3

11.4 Alternative approaches

11.4.1 Webs of trust

✓ A stronger assurance can be provided if a web of trust is implemented. Suppose that Alice wishes to directly provide relying parties with her public key.

✓ The idea of a web of trust involves other public-key certificate owner's acting as 'lightweight CAs' by digitally signing Alice's public key.

✓ Alice gradually develops a key ring, which consists of her public key plus a series of digital signatures by other owners attesting to the fact that the public-key value is indeed Alice's.

✓ These other owners are not acting as formal CAs, and the relying party may have no more of a relationship with any of these other owners than with Alice herself.

✓ Alice builds up her key ring there are two potentially positive impacts for relying parties:

1 A relying party sees that a number of other owner's have been willing to sign Alice's public key. This is at least some evidence that the public key may indeed belong to Alice.

2 There is an increasing chance (as the key ring size increases) that one of the other owners is someone that the relying party knows and trusts. If this is the case then the relying party might use a transitive trust argument to gain some assurance about Alice's public key.

Webs of trust have limitations.

- They represent a lightweight and scalable means of providing some assurance of purpose of public keys in open environments.
- Webs of trust make a real impact is unclear since, for the types of open applications in which they make most sense, relying parties are often likely to choose to simply trust the owner.

11.4.2 Identity-based public-key cryptography

THE IDEA BEHIND IDPKC

✓ A significant difference between IDPKC and certificate-based approaches to management of conventional public-key cryptography is that IDPKC requires a trusted third party to be involved in private-key generation.

✓ This trusted third party is referred as a trusted key centre (TKC), since its main role is the generation and distribution of private keys.

- A public-key owner's 'identity' is their public key. There is a publicly known rule that converts the owner's 'identity' into a string of bits, and then some publicly known rule that

converts that string of bits into a public key.

• The public-key owner's private key can be calculated from their public key only by the TKC, who has some additional secret information.

A MODEL FOR IDPKC ENCRYPTION

•

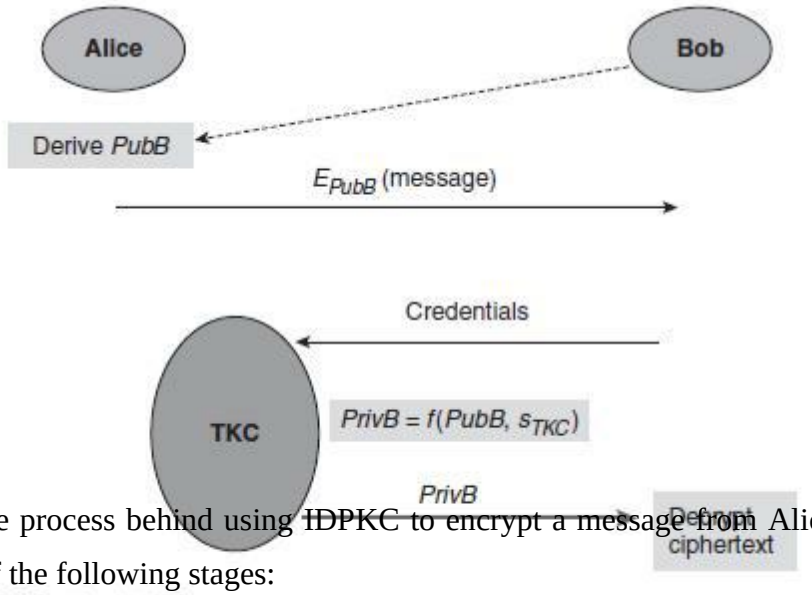


Figure 11.8. The IDPKC process

Figure 11.8 shows the process behind using IDPKC to encrypt a message from Alice to Bob. The model consists of the following stages:

1. **Encryption**. Alice derives Bob's public key PubB from Bob's identity using the publicly known rules. Alice then encrypts her message using PubB and sends the resulting ciphertext to Bob.
2. **Identification**. Bob identifies himself to the TKC by presenting appropriate credentials and requests the private key PrivB that corresponds to PubB.
3. **Private key derivation**. If the TKC accepts Bob's credentials then the TKC derives PrivB from PubB and a system secret value STKC, known only by the TKC.
4. **Private-key distribution**. The TKC sends PrivB to Bob using a secure channel.
5. **Decryption**. Bob decrypts the ciphertext using PrivB.

One of the most interesting aspects of this IDPKC encryption model is that encryption can occur before private-key derivation and distribution. It is possible to send an encrypted message to someone who has not yet established a private decryption key. Once a user has obtained their private key, there is no need for them to repeat the middle three stages until either PubB or the system secret STKC change.

IDPKC ENCRYPTION ALGORITHMS

The most important issue regarding algorithms for IDPKC is that conventional public-key cryptosystems cannot be used for IDPKC. There are two principal reasons for this:

1 In conventional public-key algorithms, such as RSA, it is not possible for any value to be

a public key. Rather, a public key is a value that satisfies certain specific mathematical properties.

2 Conventional public-key algorithms do not feature a system secret STKC that can be used to 'unlock' each private key from the corresponding public key.

PRACTICAL ISSUES WITH IDPKC

1. The need for an online, centrally trusted TKC.
    - The TKC should be online, since it could be called upon at any time to establish private keys.
    - Only the TKC can derive the private keys in the system, hence it provides a source of key escrow, which can either be seen as desirable or undesirable.
    - The TKC requires secure channels with all private key owners.

2. **Revocation issues.**

- One of the problems with tying a public-key value to an identity is the impact of revocation of a public key.

- An obvious solution to this is to introduce a temporal element into the owner's 'identity', perhaps featuring a time period for which the public key is valid.

- For example, Bob's public key might become PubB 3rdApril for any encrypted messages intended for Bob on 3rd April, but change to PubB 4thApril the next day. The cost of this is a requirement for Bob to obtain a new private key for each new day.

3. Multiple Applications

- In conventional public-key cryptography, one owner can possess different public keys for different applications.

- By tying an identity to a public-key value, this is not immediately possible. One solution is to introduce variety into the encryption key using PubB Bank as Bob's public key for his online banking application.

- IDPKC offers a potential advantage in this area since it is possible for the same public key PubB to be used across multiple applications.

MORE GENERAL NOTIONS OF IDPKC

- The idea behind IDPKC is both compelling and intriguing, since it represents a quite different approach to implementing public-key cryptography.

- They could take the form of almost any string of data. One of the most promising extensions of the IDPKC idea is to associate public keys in an IDPKC system with decryption policies. The idea involves only a slight modification of the process described in Figure 11.8:

Encryption. Alice derives a public key PubPolicy based on a specific decryption policy using publicly known rules. For example, this policy could be qualified radiographer working in a UK hospital. Alice then encrypts her message using PubPolicy and stores it on a medical database, along with an explanation of the decryption policy.

Identification. Qualified UK radiographer Bob, who wishes to access the health record, identifies himself to the TKC by presenting appropriate medical credentials and requests the private key PrivPolicy that corresponds to PubPolicy.

Private-key derivation. If the TKC accepts Bob's credentials then the TKC derives PrivPolicy from PubPolicy and a system secret value sTKC, known only by the TKC. **Private-key distribution**. The TKC sends PrivPolicy to Bob using a secure channel.

Decryption. Bob decrypts the ciphertext using PrivPolicy.

Cryptographic Applications

Cryptographic Applications are:

1. Cryptography on the Internet.
2. Cryptography for wireless local area networks.
3. Cryptography for mobile telecommunications.
4. Cryptography for secure payment card transactions.
5. Cryptography for video broadcasting.

12.1 Cryptography on the Internet

- Cryptography on the internet uses SSL protocol.
- SSL is most important cryptographic protocols for establishing a secure network channel.
- The Internet is often modeled as a four-layer Internet Protocol Suite. SSL operates at the Transport Layer of the Internet Protocol Suite, secure channels can also be established at the higher Application Layer using the Secure Shell (SSH) protocol and at the lower Internet Layer using the Internet Protocol Security (IPsec) suite.
- SSL was developed by Netscape in the mid-1990s for use with their Navigator browser. It subsequently became the responsibility of the Internet Engineering Task Force (IETF).

12.1.2 SSL security requirements

SSL is designed to establish a 'secure channel' between two entities. The main security requirements are:

1. **Confidentiality**. Data transferred over the secure channel should only be accessible tothe entities at either end of the channel, and not by any attacker who monitors the channel.
2. **Data origin authentication**. Data transferred over the secure channel should be integrity-protected against an attacker who can conduct active attacks on the channel.
3. **Entity authentication**. In order to set up the secure channel, it should be possible to establish the identity of each communicating entity.

12.1.3 Cryptography used in SSL

SSL uses a wide range of cryptographic primitives:

1. Public-key cryptography is used to enable symmetric key establishment.
2. Digital signatures are used to sign certificates and facilitate entity authentication.
3. Symmetric encryption is used to provide confidentiality.
4. MACs are used to provide data origin authentication and facilitate entity authentication.
5. Hash functions are used as components of MACs and digital signatures, and for key derivation.

SSL supports a range of different algorithms, which include:

• Many well-known block ciphers, such as AES, normally in CBC mode.

• HMAC, implemented using a choice of well-known hash functions such as SHA-256.

• Digital signature algorithms such as RSA and DSA.

12.1.4 SSL protocols

SSL consists of two cryptographic protocols:

Handshake Protocol. This protocol performs all the tasks that require agreement between the two entities before they set up the secure SSL channel. This protocol can be used to:

- Agree on the cryptographic algorithms to be used to establish the secure Channel.
- Establish entity authentication.
- Establish the keys that will be needed to secure the channel.

Record Protocol. This protocol implements the secure channel. This includes:

- Formatting the data.
- Computing MACs on the data.
- Encrypting the data.

SIMPLE SSL HANDSHAKE PROTOCOL DESCRIPTION

The message flow of the simplified SSL Handshake Protocol is indicated in Figure 12.1.

i. **Client Request:** This message from the client initiates the communication session and requests the establishment of an SSL-protected channel. The client sends some data, including:

   - **A session ID**, which acts as a unique identifier for the session.
   - **a pseudorandom number r_C** , which will be used for the provision of freshness.
   - a list of **cipher suites** that the client supports.

ii. **Server Response:** The server responds by sending some initialization data, including:

- the session ID.
- a pseudorandom number r_S, which is the server's freshness contribution to the protocol.
- the particular cipher suite that the server has decided to use.
- a copy of the server's public-key certificate, including details of any certificate chain required to verify this certificate.

iii. **Pre-master Secret Transfer:**

The client now generates another pseudorandom number K_P, which encrypts using the server's public key and sends to the server.

A) r_C and r_S are nonces used to provide freshness, hence they are not encrypted when they are exchanged.

B) K_P will be used to derive the keys that are used to secure the session. This value K_P is referred to as the pre-master secret and is a value known only by the client and the server.

iv. **Client Finished**: The client computes a MAC on the hash of all the messages sent thus far. This MAC is then encrypted and sent to the server.

v. **Server Finished:** The server checks the MAC received from the client. The server then computes a MAC on the hash of all the messages that have been sent. This MAC is then encrypted and sent to the client. The client checks the MAC received from the server.

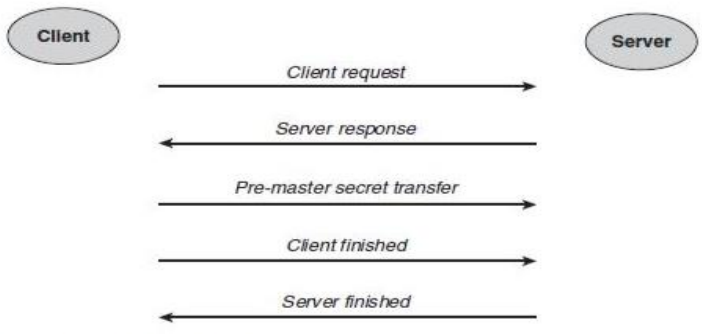


Figure 12.1. Simple SSL Handshake Protocol message flow

ANALYSIS OF THE SIMPLE SSL HANDSHAKE PROTOCOL

The simple SSL Handshake Protocol achieves its three main goals:

Agreement of cryptographic algorithms. This is achieved at the end of the second protocol message, when the server informs the client which cipher suite has been selected from the list provided by the client.

Entity authentication of the server. This relies on the following argument, assuming that the protocol run has been successful and that all checks (including certificate validity checks) have been correctly made:

- The entity who sent the Server Finished message must know the master secret KM, since the final check was correct and relied on knowledge of KM.
- Any entity other than the client who knows KM must also know the pre-master secret KP , since KM is derived from KP .
- Any entity other than the client who knows KP must know the private decryption key corresponding to the public-key certificate sent in the message Server Response, since this public key was used to encrypt KP in the message Pre- master Secret Transfer.
- The only entity with the ability to use the private decryption key is the genuine server, since the public-key certificate provided by the server in the message Server Response was checked and found to be valid.
- The server is currently 'alive' because KM is derived from fresh pseudorandom values (KP and rC) generated by the client and thus cannot be an old value.

Key establishment. SSL establishes several keys, as we will shortly discuss. These are all derived from the master secret K_M, which is a value that is established during the SSL Handshake Protocol. The master secret is derived from the pre-master secret K_P, which is a value that only the client and the server know.

SSL HANDSHAKE PROTOCOL WITH CLIENT AUTHENTICATION

The simple SSL Handshake Protocol does not provide mutual entity authentication, only entity authentication of the server. This is reasonable because many applications do not require client authentication at the network layer where SSL is deployed.

For example, when a user purchases goods from an online store, the merchant may not care about who they are communicating with, so long as they get paid at the end of the transaction. In

this scenario, client authentication is more likely to be performed at the application layer, perhaps by using a password-based mechanism.

The simple SSL Handshake Protocol can be modified by adding an extra message from the client to the server after the message Pre-master Secret Transfer, as follows:

Client Authentication Data:

The client sends a copy of its public-key certificate to the server. The public key in this certificate is used as a verification key. The certificate includes any details of the certificate chain required for verification. In addition, the client hashes all the protocol messages so far and digitally signs the hash using the client's signature key.

The server should now check that the client's public-key certificate (chain) is valid. The server should also verify the client's digital signature. If these checks are successful then the server has entity authentication assurance of the client by the following argument:

1. The entity who sent the Client Authentication Data message must know the signature key corresponding to the public key in the client's certificate, since the digital signature verified correctly.

2. The only entity who knows the signature key is the genuine client, since the public-key certificate provided by the client was checked and found to be valid.

3. The client is currently 'alive' because the digital signature was computed on a hash of some data that included the fresh pseudorandom value rS generated by the server, and thus cannot be a replay.

SSL RECORD PROTOCOL

The SSL Record Protocol is the protocol used to instantiate the secure channel after the SSL Handshake Protocol has successfully completed. Before running the SSL Record Protocol, both the client and the server derive the cryptographic data that they will need to secure the session. This includes symmetric session keys for encryption, symmetric MAC keys and any required IVs. These are all generated using a key derivation function to compute a key block. This key derivation function uses K_M as a key and takes as input, amongst other data, r_C and r_S. The key block is then 'chopped up' to provide the necessary cryptographic data. In particular, the following four symmetric keys are extracted from the key block:

• K_{ECS} for symmetric encryption from the client to the server;

• K_{ESC} for symmetric encryption from the server to the client;

• K_{MCS} for MACs from the client to the server;

• K_{MSC} for MACs from the server to the client.

The SSL Record Protocol specifies the process for using these keys to protect traffic exchanged between the client and the server. For example, for data sent from the client to the server, the process is:

1. compute a MAC on the data (and various other inputs) using key K_{MCS};

2. append the MAC to the data and then pad, if necessary, to a multiple of the block length;

3. encrypt the resulting message using key K_{ECS}.

Upon receipt of the protected message, the server decrypts it using K_{ECS} and then verifies the recovered MAC using K_{MCS}.

12.1.5 SSL key management

KEY MANAGEMENT SYSTEM

SSL essentially relies on two 'separate' key management systems:

Public-key management system. Since SSL is designed for use in open environments, it relies on an external key management system that governs the public-key pairs that are required by SSL users. This key management system is beyond the scope of an SSL specification and is relied on to establish and maintain publickey certificates and information concerning their validity. If this system fails then the security provided by SSL is undermined.

Symmetric key management system. Within SSL is a self-contained symmetric key management system. SSL is used to generate symmetric sessions keys, which are designed to have limited lifetimes.

KEY GENERATION

There are two types of keys deployed in SSL:

Asymmetric keys. These are generated using the public-key management system, which is not governed by the specification of SSL.

Symmetric keys. These are all generated within SSL. The session keys are all derived from the master secret that is established following the SSL Handshake Protocol. Key derivation is a suitable technique for key generation because:

- it is a lightweight key generation technique, which does not impose significant overheads;
- it allows several different session keys to be established from just one shared secret;
- as the SSL Handshake Protocol is relatively expensive to run (it requires the use of public-key cryptography), the shared master secret can be used to establish several batches of session keys, should this be desirable.

KEY ESTABLISHMENT

The most important key establishment process in SSL is the establishment of the pre-master secret during the SSL Handshake Protocol. Probably the most common technique for conducting this is to use RSA public-key encryption during the protocol message Pre-master Secret Transfer. However, a variant based on Diffie–Hellman is also supported by SSL.

KEY STORAGE

Key storage is beyond the scope of SSL, but it relies on both the client and the server securely storing relevant secret keys. The most sensitive keys to store are the private keys, since they are relied upon across multiple SSL sessions. In contrast, the symmetric keys negotiated during the SSL Handshake Protocol are only used for a relatively short period of time. Nonetheless, if they are compromised then so are any sessions that they are used to protect.

KEY USAGE

Separate encryption and MAC keys are derived from the master secret, which are then used to establish the secure channel. However, SSL takes this principle a step further by deploying separate keys for each communication direction, which provides security against reflection attacks. The cost of this is low because these separate keys are derived from the common master secret.

12.1.6 SSL security issues

1. **Process failures**. The most common 'failure' of SSL arises when a client does not perform the necessary checks to validate the server's public-key certificate.

    - A web user who is presented with a dialogue box warning them of their browser's inability to verify a public-key certificate is quite likely to disregard it and proceed with establishing an SSL session.

    - A particularly common manifestation of this problem on the Internet is when a rogue webserver, holding a legitimate public-key certificate in it so wnname,tries to pass itself off as another webserver.

    - Even if the client web browser successfully verifies the rogue webserver's certificate chain, if the client does not notice that the public-key certificate is not in the name of the expected webserver then the rogue webserver will succeed in establishing an SSL protected channel with the client.

    - This is an entity authentication failure because the client has succeeded in setting up an SSL session, but it is not with the server that they think it is with.

    - This failure is often exploited during phishing attacks.

    - It is a failure in the surrounding processes that support the protocol. In this case the client has failed to conduct a protocol action (validating the server's certificate chain) with a sufficient degree of rigour.

2 **Implementation failures**.

Because it is an open protocol that can be adopted for many different applications, on different platforms, by anyone, SSL is particularly vulnerable to implementation failures. Even if the protocol specification is followed correctly, it could fail if a supporting component is weak.

For example, if the client uses a weak deterministic generator to generate the pre-master secret K_P then the protocol can be compromised because the session keys become too predictable.

3 **Key management failures**. If either the client or the server mismanages their cryptographic keys then the protocol can be compromised.

For example, if an attacker obtains the server's private key then the attacker can recover the pre-master secret. The attacker can then compute all the resulting session keys and hence undermine any secure channel that these session keys are used to establish.

4 **Usage failures**. SSL has such a high profile that it runs the risk of being used inappropriately. Alternatively it may be appropriately deployed, but its security properties overestimated under the misapprehension that use of SSL 'guarantees' security.

12.1.7 SSL design issues

Support for a range of publicly known cryptographic algorithms.

Since SSL(in this case were ally mean TLS) is an open standard targeted at wide-scale public use, it is fundamental that it supports not just publicly known algorithms, but a *range of* publicly known algorithms. This supports cross-platform use and has helped to foster confidence in SSL as a protocol.

Flexibility. SSL is not only flexible in terms of the components that can be used to implement it, but it is also flexible in the ways in which it can be used (for example, to provide unilateral or mutual entity authentication). This, again, is because SSL has been targeted at a wide range of application environments.

Minimal use of public-key operations. The use of hybrid encryption restricts the number of public-key operations to the minimum necessary to establish a secure channel. Although we have not discussed this in any detail, SSL is also designed so that the relatively expensive SSL Handshake Protocol may not need to be rerun if a client requires another session with the same server within a specified time period.

Unbalanced computational requirements: In the SSL Handshake Protocol it is the client who is required to generate the pre-master secret and send it encrypted to the server. This means that the client performs one public- key *encryption* operation and the server performs one public-key *decryption* operation. Onereason for this is that some public-key cryptosystems, and RSA is a good example, have certain public keys that are considerably more computationally efficient to use than others.

12.2 Cryptography for wireless local area networks

12.2.1 WLAN background

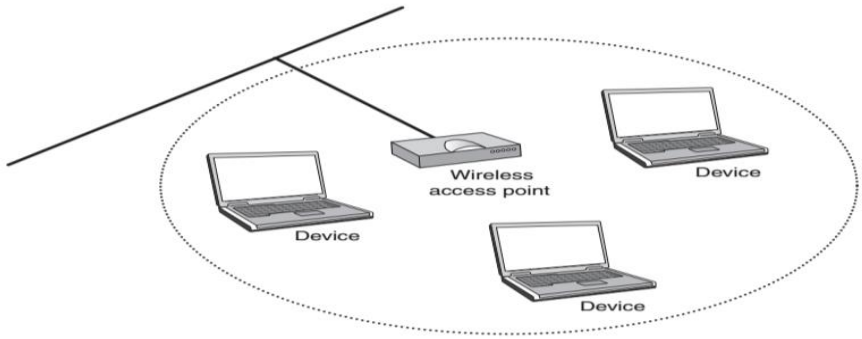


Figure 12.2. Simple WLAN architecture

A simple WLAN architecture is shown in Figure 12.2.

- A wireless access point is a piece of hardware that acts as a bridge between the wireless network and a wired network (for example, the wired network that delivers a connection to the Internet from a home).
- The access point consists of a radio, an interface with the wired network and bridging software.
- A device is any computer (for example, a desktop PC, laptop or PDA) which has a wireless network interface card that allows it to communicate over a wireless network.
- A WLAN may consist of many devices all communicating with the one access point, or indeed may involve several different access points.
- The original 802.11 standard defined the Wired Equivalent Privacy (WEP) mechanism to protect WLAN communication. WEP was designed to provide security at the data link layer, which means that it operates at a virtual networking layer that is close to being the equivalent of physical wires in a wired network.
- An improved security mechanism known as Wi-Fi Protected Access (WPA) was designed as a solution.

12.2.2 WLAN security requirements

The security requirements for a WLAN are:

Confidentiality. Data transferred over the WLAN should be kept confidential..

Mutual entity authentication. Communicating entities can identify one another when setting up a WLAN connection. This is motivated by the fact that a degree of inherent (very weak) ‘entity

authentication' is provided by physical wires, but there are no such guarantees once we are in a wireless environment.

Data origin authentication. The source of all data transferred over the WLAN should be assured. This is because an attacker could easily modify data transmitted during a WLAN session after the initial entity authentication has been conducted. The original WLAN security standard WEP only provides a weak level of data integrity, which is not good enough.

12.2.3 WEP

There are three cryptographic design decisions that are common to all of the WLAN security mechanisms

• Since WLANs may be comprised of many different types of device, from different manufacturers, it is important that the cryptography used in a WLAN is widely available. Hence it would not be wise to deploy proprietary cryptographic algorithms.

• Since these mechanisms are dedicated to WLAN security and do not require the full flexibility of the likes of SSL, it makes sense to decide which cryptographic algorithms to use in advance and then deploy them universally, rather than require an expensive equivalent of the SSL Handshake Protocol to negotiate them.

• Since speed and efficiency are important, and WLANs are usually linked to some sort of fixed infrastructure, symmetric cryptography is a natural choice.

CONFIDENTIALITY AND INTEGRITY MECHANISMS IN WEP

The first WEP design decision was to use a shared, fixed symmetric key in each WLAN. This same key is used by all devices, for several different purposes, when communicating using a WEP-secured WLAN. This almost eliminates any issues regarding key establishment, however, it introduces considerable risks. In particular, if one of the devices is compromised then this key may become known to an attacker, and hence the entire network will be compromised. The original version of WEP only used a 40-bit key, but later adaptations allow much longer keys.

One problem with deploying a stream cipher such as RC4 is the need for synchronisation, especially in a potentially noisy channel such as a wireless one. Thus WEP requires each packet of data to be encrypted separately, so that loss of a packet does not affect the rest of the data being sent. This introduces a new problem. the negative consequences of re-using keystream for

more than one plaintext. It follows that WEP requires a mechanism for making sure that the same keystream is not reused for subsequent packets.

The solution to this problem in WEP was to introduce an initialisation vector (IV), which just like the IVs used in several of the modes of operation of a block cipher, varies each time the WEP key is used to encrypt a packet. However, RC4 does not easily allow an IV to be incorporated into the encryption process, hence the WEP IV is directly appended to the key. In this way WEP defines a 'per-packet' key, which consists of a 24-bit IV appended to the WEP key. If Alice wants to set up a secure WLAN connection with Bob, based on the shared, fixed WEP key K, the encryption process for each packet of data to be sent is depicted in Figure 12.3 and is as follows.

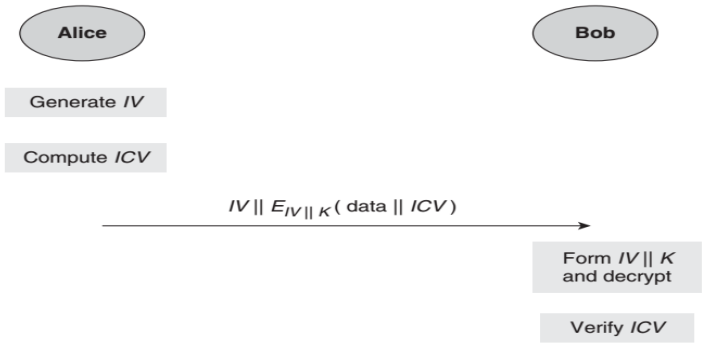


Figure 12.3. WEP encryption process

Alice:

1. generates a 24-bit pseudorandom initialisation vector IV and appends the WEP key K to IV to form the key:

$$K' = IV \,||\, K;$$

2. computes a 32-bit CRC checksum ICV of the data, and appends this to the data;
3. encrypts the data plus ICV using key K';
4. sends IV (in the clear) and the resulting ciphertext to Bob.

Bob then:

1. appends the WEP key K to the received IV to form the key K';
2. decrypts the ciphertext using K' and extracts the checksum ICV;
3. verifies that the checksum ICV is correct.

If the verification of ICV is successful then Bob accepts the data packet.

ENTITY AUTHENTICATION IN WEP

The WEP entity authentication technique is very simple. It is based on the challenge–response principle If Alice (a device) wants to identify herself to Bob (a wireless access point):

1. Alice sends a request to authenticate to Bob;

2. Bob sends a nonce r_B to Alice;

3. Alice uses WEP encryption to encrypt r_B (importantly for later, note from our above explanation of the WEP encryption process that this also involves Alice generating an IV that is used to 'extend' the WEP key).

4. Alice sends the IV and the resulting ciphertext to Bob;

5. Bob decrypts the ciphertext and checks that it decrypts to r_B; if it does, he authenticates Alice.

12.2.4 Attacks on WEP

WEP KEY MANAGEMENT WEAKNESSES

There are several serious problems with WEP key management:

- **Use of a shared fixed key**: The WEP key K acts as an overall 'master key' for the WLAN and, as such, is a single point of failure. If the WEP key can be compromised (and it suffices that this compromise arises on just one of the entities forming the WLAN) and an attacker learns the WEP key then the entire WLAN security is compromised.

- **Exposure of the WEP key**. In its role as a master key, the WEP key is unnecessarily 'exposed' through direct use as a component of an encryption key. It is also exposed in this way each time an authentication attempt is made.

- **No key separation**.WEP abuses the principle of key separation by using the WEP key for multiple purposes.

- **Key length**. While WEP does allow the WEP key length to vary, the smallest RC4 key length is 40 bits, which is far too short to be secure against contemporary exhaustive key searches. Many WEP implementations allow WEP keys to be generated from passwords which, if not long enough, reduce the effective keyspace that an attacker needs to search.

WEP ENTITY AUTHENTICATION WEAKNESSES

Attacks concerning the entity authentication mechanism.

Rogue wireless access point. WEP only provides unilateral entity authentication from a device (Alice) to a wireless access point (Bob). This means that an attacker could set up a rogue access point and allow Alice to authenticate to it, without Alice realising that she was not dealing with the genuine access point.

Lack of session key. WEP does not establish a session key during entity authentication that is later used to protect the communication session. As a result, WEP entity authentication is only valid for the 'instant in time' at which it is conducted. WEP thus suffers from the potential for a 'hijack' of the communication session.

Keystream replay attack. Another serious problem is that there is no protection against replays of the WEP authentication process. An attacker who observes Alice authenticating to Bob is able to capture a plaintext (the challenge r_B and its CRC checksum) and the resulting ciphertext (the encrypted response). Since WEP uses the stream cipher RC4, the keystream can be recovered by XORing the plaintext to the ciphertext. We will denote this keystream by KS(IV || K), since it is the keystream produced by RC4 using the encryption key IV || K. Good stream ciphers are designed to offer protection against an attacker who knows corresponding plaintext/ciphertext pairs, and hence can recover keystream from this knowledge. However, this relies on the same keystream not being reused in a predictable manner . This is where WEP fails, since the attacker can now falsely authenticate to Bob as follows (and depicted in Figure 12.4):

1. The attacker requests to authenticate to Bob;
2. Bob sends a nonce r'_B to the attacker (assuming that Bob is properly generating his nonces, it is very unlikely that $r'_B = r_B$);
3. The attacker computes the CRC checksum ICV on r'_B. The attacker then encrypts $r'_B \,||\, ICV$ by XORing it with the keystream $KS(IV \,||\, K)$; note:
   - the attacker does not know the WEP key K, but does know this portion of keystream;
   - in line with WEP encryption, the attacker also first sends the IV that was observed during Alice's authentication session to Bob;
4. Bob decrypts the ciphertext, which should result in recovery of r'_B, in which case Bob accepts the attacker.

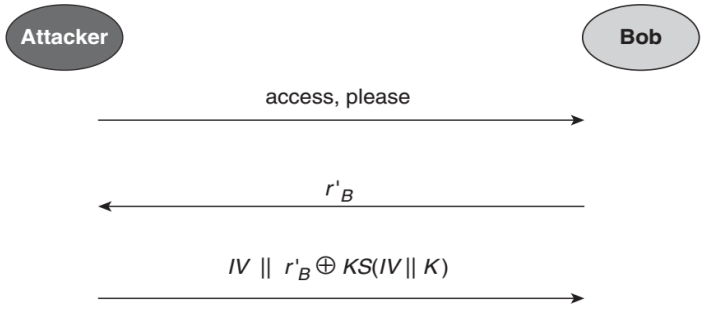


Figure 12.4. Keystream replay attack on WEP authentication

This attack works because WEP allows the attacker to 'force' Bob to use the same IV that Alice used in the genuine authentication session, and hence use the same encryption key IV || K, which in turn validates the use of the previous keystream. Of course, having authenticated to the access point, the attacker cannot do much more since the attacker still does not know the WEP key K and hence cannot perform valid encryptions and decryptions. Nonetheless, the authentication process has been successfully attacked.

12.2.5 WPA and WPA2

MUTUAL ENTITY AUTHENTICATION AND KEY ESTABLISHMENT

In order to avoid all the problems relating to use of a shared, fixed WEP key, a key hierarchy is employed. The top key in this key hierarchy is known as the pairwise master key PMK, which is a key that is shared between a device and a wireless access point. There are two ways in which this key PMK can be established:

1. During an AKE protocol that is run between a device and a central authentication server. Both WPA and WPA2 support the use of a central authentication server to provide authentication in a way that is scalable and can be tailored to fit the needs of the specific application environment. A wide range of authentication techniques are supported by the Extensible Authentication Protocol (EAP), which is a suite of entity authentication mechanisms that includes methods that deploy SSL to secure a connection to an authentication server.

2. As a pre-shared key that is programmed directly into the device and the wireless access point. This is most suitable for small networks. The most common method for generating PMK is by deriving it from a password. Any users requiring access to the WLAN must be made aware of this password. A home user who purchases a wireless router may be provided with a (weak) default password from the manufacturer or service provider. It is important that this is changed on first installation to something less predictable.

The master key PMK is also used to derive session keys using the following AKE protocol that runs between Alice (a device) and Bob (a wireless access point) and is shown in Figure 12.5:

1. Alice generates a nonce r_A and sends r_A to Bob.

2. Bob generates a nonce r_B. Bob then uses r_A, r_B and PMK to derive the following four 128-bit session keys:

• an encryption key EK;

• a MAC key MK;

• a data encryption key DEK;

• a data MAC key DMK

3. Bob then sends rB to Alice, along with a MAC computed on rB using MAC key MK.

4. Alice uses r_A, r_B and PMK to derive the four session keys. She then checks the MAC that she has just received from Bob.

5. Alice sends a message to Bob stating that she is ready to start using encryption. She computes a MAC on this message using MAC key MK.

6. Bob verifies the MAC and sends an acknowledgement to Alice.

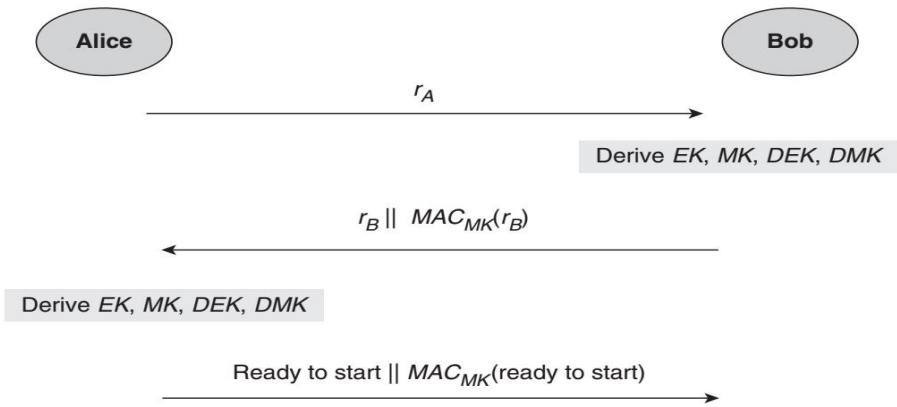


Figure 12.5. WPA authentication and key establishment protocol

12.2.7 WLAN design issues

The main cryptographic design issues concerning WLAN security are as follows:

- **Use of symmetric cryptography**. This is a sensible decision because WLANs transfer bulk traffic between networked devices, hence speed of encryption is important. For small networks, such as a home network, key establishment is straightforward. Larger enterprise WLANs may optionally choose to use public-key mechanisms as part of the initial authentication between a device and a central authentication server, but the core WPA2 security protocol CCMP uses only symmetric cryptography.

- **Use of recognised cryptographic mechanisms.** This was not adhered to in WEP, where the cryptographic design was rather ad hoc. WEP thus provides a useful lesson regarding

the potential folly of adopting unconventional mechanisms. In contrast, WPA2 adopts more widely accepted cryptographic mechanisms.

- **Flexibility, but only when appropriate.** While WLANs may be deployed in quite different environments, they do not require the same cryptographic flexibility as open applications such as SSL. Thus it makes sense to 'lock down' the cryptographic mechanisms, where appropriate. WPA2 does this for the confidentiality and data origin authentication services. However, WPA2 allows for flexibility in choosing the initial entity authentication mechanism (between the device and a centralised authentication server), recognising that different environments may well have different approaches to identifying network users.

- **The potential need to cater for migration.** When the flaws in WEP became apparent, it was clear that due to the difficulty of upgrading a widely deployed technology, any complete redesign of the WLAN security mechanisms could not be rolled out quickly. It was thus necessary to design a 'fix' that was based on the existing cryptographic mechanisms, which would provide 'good enough' security. The 'fix' is WPA, which is based on RC4. The 'complete redesign' is WPA2, which is based on AES.

12.3 Cryptography for mobile telecommunications

12.3.1 GSM and UMTS background

- The shift from analogue to digital communications brought with it the opportunity to use cryptographic techniques to provide security. In doing so, the development of the Global System for Mobile Communication (GSM) standard by the European Telecommunications Standards Institute (ETSI) brought security to mobile telecommunications.

- Third generation, or 3G, mobile phones are characterised by higher data transmission rates and a much richer range of services. The enhanced security of GSM's successor for 3G phones, the Universal Mobile Telecommunications System (UMTS).

- The basic architecture of a mobile telecommunications network is shown in Figure 12.6. The network is divided into a large number of geographic cells, each of which is controlled by a base station. A mobile phone first connects with its nearest base station, which directs communications either to the home network of the mobile phone user or to other networks in order to transfer call data.

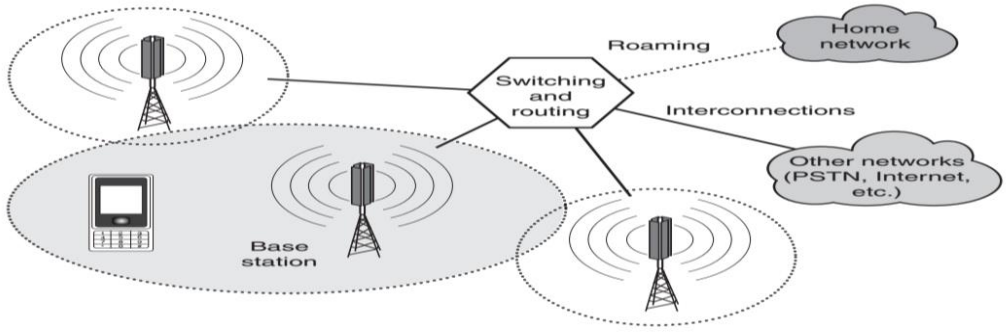


Figure 12.6. Basic architecture of mobile telecommunications network

12.3.2 GSM security requirements

The following are the specific security requirements:

- **Entity authentication of the user**. Mobile operators need to have strong assurance of the identity of users connecting with their services in order to reduce fraud. This issue is much simpler to deal with in traditional telephone networks, since a user needs to have physical access to the end of a telephone wire in order to use the services.

- **Confidentiality on the radio path.** In simple terms, a mobile connection passes 'over the air' (the radio path) between the handset and a base station, after which it is passed through a switching centre and enters the traditional PSTN (see Figure 12.6). Thus in order to provide 'PSTN-equivalent security', the main link for which GSM needs to provide additional security is the radio path. Since this path is easily intercepted by anyone with a suitable receiver it is necessary to provide confidentiality on this radio path.

- **Anonymity on the radio path**. GSM provides a degree of anonymity (confidentiality of the identity of users) on the radio path in order to prevent an attacker from linking the source of several intercepted calls. This is handled by using temporary user identities for each call, rather than permanent ones.

12.3.3 Cryptography used in GSM

The main cryptographic design decisions for GSM were:

- **A fully symmetric cryptographic architecture**.GSM is an entirely closed system. All key material can be loaded onto the necessary equipment prior to it being issued to users, so there is no need to use public-key cryptography for this purpose.

- **Stream ciphers for data encryption**. The requirement for fast real-time encryption over a potentially noisy communication channel means that, a stream cipher is the most appropriate primitive.

- **Fixing the encryption algorithms**. It is necessary that the mobile operators agree on which encryption algorithms to use, so that the devices on which they operate can be made compatible with one another. However, other cryptographic algorithms, such as those used in GSM authentication, donot have to be fixed. In the case of authentication, an individual mobile operator is free to choose the cryptographic algorithm that it deploys to authenticate its own users(since users of another mobile operator are not directly impacted by this decision).

- **Proprietary cryptographic algorithms**. The designers of GSM chose to develop some proprietary cryptographic algorithms, rather than use open standards. While the use of proprietary algorithms is not wise in many application environments, in the case of GSM there were three factors that favoured at least considering this option:

- GSM is a closed system, hence deploying proprietary algorithms is feasible.
- ETSI have a degree of cryptographic expertise, and maintain links with the open research community.
- The need for fast real-time encryption means that an algorithm designed explicitly to run on the hardware of a mobile phone will probably perform better than an 'off-the-shelf' algorithm.

SIM: The fundamental component involved in GSM security is the *Subscriber Identification Module* (SIM) *card*, which is a smart card that is inserted into the mobile phone of the user. This SIM card contains all the information that distinguishes one user account from another. As a result, a user can potentially change phone equipment simply by removing the SIM and inserting it into a new phone. The SIM contains two particularly important pieces of information:

1. The *International Mobile Subscriber Identity* (IMSI), which is a unique number that maps a user to a particular phone number;
2. A unique 128-bit cryptographic key K_i, which is randomly generated by the mobile operator.

These two pieces of data are inserted onto the SIM card by the mobile operator before the SIM card is issued to the user. The key K_i forms the basis for all the cryptographic services relating to the user. The SIM card also contains implementations of some of the cryptographic algorithms required to deliver these services.

GSM AUTHENTICATION

- Entity authentication of the user in GSM is provided using a challenge–response protocol, This is implemented as part of an AKE protocol, which also generates a key K_c for subsequent data encryption.
- GSM does not dictate which cryptographic algorithms should be used as part of this AKE protocol, but it does suggest one candidate algorithm and defines the way in which algorithms should be used.
- As indicated in Figure12.7, an algorithm A3 is used in the challenge–response protocol and an algorithm A8 is used to generate the encryption key K_c.
- Both of these algorithms can be individually selected by the mobile operator and are implemented on the SIM and in the operator's network.
- Both A3 and A8 can be loosely considered as types of key derivation function, since their main purpose is to use K_i to generate pseudorandom values.

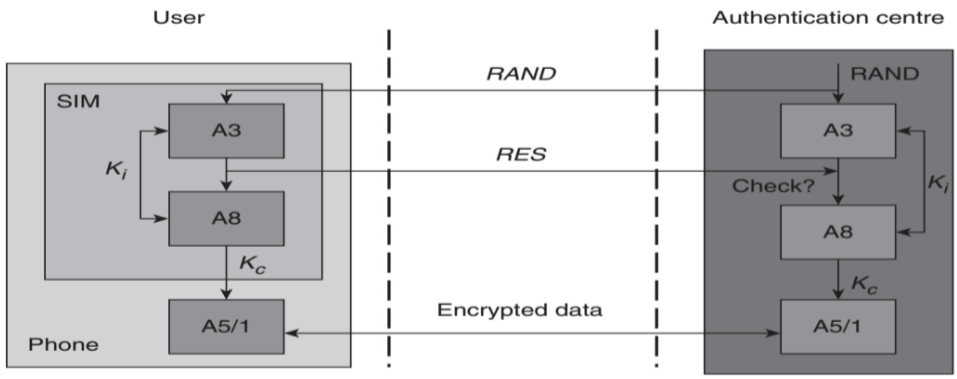


Figure 12.7. GSM authentication and encryption

The notation $A3_K$ (*data*) to denote the result of computing algorithm A3 on the input *data* using key K (the notation $A8_K$ (*data*) should be similarly interpreted). If Alice (a mobile) is able to directly authenticate to Bob (the authentication centre of a mobile operator) then the GSM AKE protocol is as follows:

1. Alice sends an authentication request to Bob.
2. Bob generates a 128-bit randomly generated challenge number *RAND* and sends it Alice.

3. Alice's SIM card uses K_i and *RAND* to compute a response *RES* using algorithmA3:

$$RES = A3_{K_i}(RAND).$$

The response *RES* is sent back to Bob.

4. Bob, who maintains a database of all the user keys, selects the appropriate key Ki for Alice and then computes the expected response in the same way. If the result matches the received *RES* then Alice is authenticated.

5. Alice and Bob both use K_i and *RAND* to compute an encryption key K_C using algorithm A8:

$$K_C = A8_{K_i}(RAND).$$

This simple protocol relies on the belief that only the mobile user and the mobile operator authentication centre can possibly know the key K_i that has been installed on the user's SIM card.

GSM ENCRYPTION

- While authentication is a service that is 'private' to a mobile user and their mobile operator, encryption must be provided using a mechanism that is common to all mobile operators, in order to facilitate cross-network calls.
- Thus the encryption algorithm A5/1 is fixed by the GSM standard (in fact GSM offers three different versions of A5, but A5/1 is the most commonly deployed). As indicated in Figure 12.7, it is implemented on the mobile phone itself, not the SIM card, since the phone has more computation power than the SIM.
- The A5/1 algorithm is a stream cipher with a 64-bit key. It was designed to be implemented very efficiently in the hardware of a mobile phone.
- In GSM, A5/1 is used to encrypt all radio path communication (both signalling information and the message data) using the key K_c. Potentially, this key maybe freshly generated each time a user makes a mobile call.
- Encryption is also used to protect the transfer of temporary identification numbers, which are used instead of the IMSI to provide user anonymity.
-

FACILITATING GSM ROAMING

Scenario : when a mobile user is traveling outside the area serviced by their mobile operator, for example, overseas (this is referred to as roaming). Although different mobile operators are in some sense part of a wider 'closed' GSM network, they are still individual businesses with their own private user relationships. It would thus be unacceptable for one operator to share its security critical data (particularly key Ki) with another for the purpose of facilitating roaming. On the other hand, it is equally unacceptable from a practical perspective for every authentication request from a roaming user to be referred back to the user's mobile operator, since this might result in extensive delays.

Solution: GSM has a solution to this problem, through the use of authentication triplets. When a roaming mobile user Alice first connects with Charlie, a local mobile operator with whom she has no direct business relationship, the following procedure is followed:

1. Charlie contacts Bob (Alice's mobile operator) and requests a batch of GSM authentication triplets.
2. Bob generates a fresh batch of randomly generated challenge numbers $RAND(1), RAND(2), \ldots, RAND(n)$ and computes the matching values for RES and K_c using Alice's key K_i. These form the batch of triplets:

$$TRIP(1) = (RAND(1), RES(1), K_c(1))$$
$$TRIP(2) = (RAND(2), RES(2), K_c(2))$$
$$\vdots$$
$$TRIP(n) = (RAND(n), RES(n), K_c(n)),$$

   where $RES(j) = A3_{K_i}(RAND(j))$ and $K_c(j) = A8_{K_i}(RAND(j))$. Bob sends this batch of triplets to Charlie.
3. Charlie sends the challenge $RAND(1)$ to Alice.
4. Alice computes the response $RES(1)$ using $RAND(1)$ and key K_i and sends $RES(1)$ to Charlie.
5. Charlie checks that the received $RES(1)$ matches the value in the first triplet that he received from Bob. If it does then Charlie authenticates Alice. Note that Charlie has done this without needing to know the key K_i. Alice and Charlie can now safely assume that they share the encryption key $K_c(1)$.
6. The next time Alice contacts Charlie to request a new authentication, Charlie uses the second triplet received from Bob and sends the challenge $RAND(2)$. Thus although Bob has to be involved in the first authentication attempt, there is no need to contact Bob again until the current batch of triplets have all been used up.

12.3.4 UMTS

The main cryptographic improvements over GSM are as follows:

- **Mutual entity authentication**. GSM offers entity authentication only of the mobile user. Since the development of GSM, so-called false base station attacks have become much more feasible due to reductions in the costs of suitable equipment. In one example of such an attack, a mobile user connects to the false base station, which immediately suggests to the user that encryption is turned off. By additionally requiring the user to authenticate to the mobile base station, such attacks are prevented.

- **Prevention of triplet reuse.** A GSM triplet can be reused many times for the particular mobile that it was generated for. In UMTS this is prevented by upgrading authentication triplets to quintets, which additionally include a sequence number that prevents successful replay and a MAC key.

- **Use of publicly known algorithms**. UMTS adopts cryptographic algorithms based on well-established and well-studied techniques. While it does not quite use 'off-the-shelf' algorithms, due to the desire to tailor algorithms to the underlying hardware, the algorithms deployed are very closely based on standard algorithms and the modifications have been publicly evaluated.

- **Longer key lengths**. Following the relaxation of export restrictions that were in place at the time of GSM development, the key lengths of the underlying cryptographic algorithms were increased to 128 bits.

- **Integrity of signalling data.** UMTS provides additional integrity protection to the critical signalling data. This is provided using a MAC, whose key is established during the UMTS authentication (AKE) protocol.

12.3.5 GSM and UMTS key management

KEY MANAGEMENT SYSTEM

GSM and UMTS have an entirely symmetric key management system, facilitated by the fact that a mobile operator is completely in control of all keying material relating to their users. Underlying key management system as a very simple key hierarchy with the user keys K_i acting as individual user 'master keys' and the encryption keys K_c acting as data (session) keys.

KEY GENERATION

The user keys K_i are randomly generated, normally by the SIM manufacturer (on behalf of the mobile operator) using a technique of their choice. The encryption keys K_c are derived from the user keys K_i, using the mobile operator's chosen cryptographic algorithm.

KEY ESTABLISHMENT

The establishment of user key K_i is under the control of the SIM manufacturer (on behalf of the mobile operator) who installs K_i on the user's SIM card before it is issued to the user. The significant key management advantage that is being exploited here is that a mobile service has no utility until a customer obtains a physical object from the mobile operator (in this case a SIM card), hence key establishment can be tied to this process. The keys K_c are established during the AKE protocol used for entity authentication. It is clearly very important that the SIM manufacturer transfers all the keys Ki to the mobile operator using highly secure means, perhaps in the form of an encrypted database.

KEY STORAGE

The critical user keys K_i are stored in the hardware of the user's SIM card, which offers a reasonable degree of tamper-resistance. Only the encryption key K_c, and in UMTS a MAC key derived from K_i, leave the SIM card. These are session keys that are discarded after use.

KEY USAGE

Both GSM and UMTS enforce a degree of key separation by making sure that the long-term user key K_i is only ever indirectly 'exposed' to an attacker through its use to compute the short responses to the mobile operator's challenges. The key Kc that is used for bulk data encryption, and is thus most 'exposed' to an attacker, is a derived key that is not used more than once. In UMTS, separate keys for encryption and MACs are derived from K_i. The use of a SIM also makes key change relatively straightforward.

12.3.7 GSM and UMTS design issues

The main design issues emerging from our study of GSM and UMTS are the following:

Use of symmetric cryptography. The closed nature of the application environment lends itself to adoption of a fully symmetric solution. The properties of stream ciphers are highly suited to mobile telecommunications.

Adaptation to evolving constraints. GSM was designed under several constraints, including cryptographic export restrictions and the apparent lack of a need for mobile operator authentication. As the environment determining these constraints evolved, the redesigned security mechanisms of UMTS took these into account.

Shift from proprietary to publicly known algorithms. Mobile telecommunications provide a plausible environment for the adoption of proprietary cryptographic algorithms. However, subsequent weaknesses in some of the original GSM algorithms may well have influenced the use of publicly known algorithms in UMTS.

Flexibility, but only when appropriate. GSM and UMTS only prescribe particular cryptographic algorithms when this is essential, leaving a degree of flexibility to mobile operators. That said, in UMTS mobile operators are strongly encouraged to follow central recommendations.

12.4 Cryptography for secure payment card transactions

- Financial sector organisations are the most established commercial users of cryptography.
- They oversee global networks that use cryptographic services to provide security for financial transactions.

12.4.1 Background to payment card services

A *payment card organisation* (PCO), such as Visa and MasterCard, essentially operates as a 'club' of member banks who cooperate in order to facilitate transactions. Figure 12.8 indicates the key players in this cooperative organisation. *Issuing banks* issue payment cards to customers. *Acquiring banks* have relationships with merchants of goods. PCOs run networks that connects these banks and facilitate payments from issuing bank customers to acquiring bank merchants. The two main uses of a payment card network are to:

1. Authorize payments;
2. Arrange clearing and settlement of payments.

PCOs oversee the use of both credit and debit cards. The main difference between the two is the process by which the issuing bank decides to bill the customer. From a cryptographic perspective we will not distinguish between these two types of payment card.

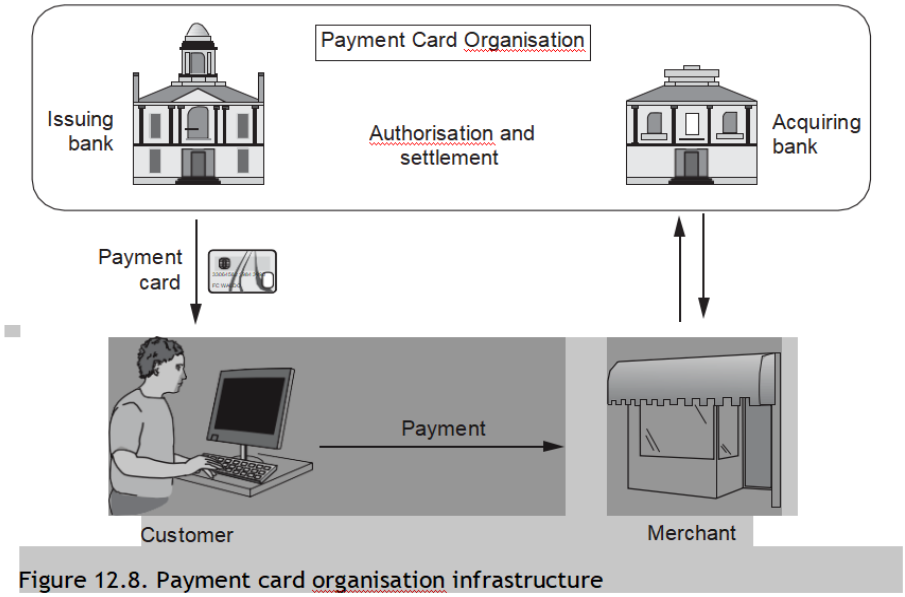


Figure 12.8. Payment card organisation infrastructure

12.4.2 Magnetic stripe cards

Most payment cards have magnetic stripes. Even payment cards with chips often retain the magnetic stripe and may resort to using it when they are deployed in environments that do not support EMV. The following description of cryptography used by magnetic stripe cards is based on the practices of Visa and MasterCard.

PIN PROTECTION

Consider example of cryptography being used by payment cards concerns online authentication of a user who inserts their magnetic stripe payment card into an ATM. Before releasing any funds, the ATM needs to know whether the user is genuine and whether they are entitled to make the requested withdrawal.

The process begins when the user is asked to enter their PIN into the ATM. The ATM clearly cannot verify this PIN on its own, so it needs to refer the PIN to the user's issuing bank. Since PINs are sensitive, this information should be encrypted. It is impractical for every ATM to share an encryption key with every issuing bank, so a process of key translation is used:

- The ATM encrypts the PIN and the authentication request message using a key shared by the ATM and the acquiring bank responsible for that ATM (each ATM should have a unique key of this type).
- The acquiring bank decrypts the cipher text and then re-encrypts it under a key known as the acquirer working key, which is a key shared by the acquiring bank and the PCO.

- The PCO decrypts the cipher text and re-encrypts it using an issuer working key, which is a key that the PCO shares with the issuing bank.
- The issuing bank decrypts the cipher text and makes the necessary checks of the PIN and the authentication request message. The response is then relayed back to the ATM.

It would be dangerous simply to encrypt the PIN directly during this process, since the limited number of PINs will result in a limited number of possible cipher texts representing encrypted PINs. If the same key were to be used to encrypt several PINs then an attacker could conduct a dictionary attack that matched a cipher rtext representing an unknown PIN against a 'dictionary' of cipher rtexts corresponding to known PINs. This threat is prevented by two important mechanisms:

Use of a PIN block. The PIN is never encrypted directly. Instead a *PIN block* is formed, one example of which consists of a 64-bit string containing the PIN being XORed to a 64-bit string containing the *Personal Account Number* (PAN) corresponding to the card. This means that two cards with the same PIN will not be encrypted to the same ciphertext under the same encryption key.

Session key encryption. Further security is provided by ensuring that ATMs use session keys, which are generated for a single PIN encryption event and then destroyed.

CARD VERIFICATION VALUES

- One major problem with magnetic stripe cards is that they are relatively easy to clone.
- Early payment cards only included routine information such as the PAN and expiry date on the magnetic stripe. Since this information is easily obtained by a potential attacker (most of it is even displayed on the card itself, or can be obtained from receipts), it was very easy for an attacker to forge such a card.
- The problem was alleviated by the inclusion of a cryptographic value known as the *Card Verification Value* (CVV) on the magnetic stripe .
- The CVV consists of three digits that are extracted from a hex ciphertext, which is computed by encrypting the routine card information using a key known only to the issuer. The CVV is not displayed on the card and can only be created and verified by the card issuer.

- The CVV can be obtained by an attacker who has read off all the information contained on the magnetic stripe, for example, a rogue merchant.
- Payment cards thus include a second CVV value, CVV2, which is a cryptographic value computed in a similar (but slightly different) way to the CVV.
- The CVV2 is displayed on the reverse of the payment card, but is not included in the magnetic stripe. The CVV2 is primarily used as a simple check of the physical presence of a card, particularly in transactions made over the telephone or online.

PIN VERIFICATION VALUE

- In order to improve availability, PCOs also provide a service which allows PINs to be verified when the card issuer is unable to process PIN verification requests.
- This is conducted using a *PIN Verification Value* (PVV), which is computed in a similar way to the CVVs, except that the PIN itself forms part of the plaintext that is encrypted in order to generate the PVV.
- The issuing bank needs to share the key that it uses to compute this PVV with the PCO.
- The PVV is four digits long, so that its security is 'equivalent' to that of the PIN itself.
- Like the CVV, the PVV is normally stored on the magnetic stripe but not displayed on the card. During a PIN verification request, the PCO recomputes the PVV using the PIN that has been offered by the customer and checks whether this value matches the PVV on the magnetic stripe .If It does then the PIN verification is accepted.

PAYMENT CARD AUTHORISATION

- When a payment card is inserted into a terminal, the main goal of the terminal is normally to determine the validity of the card and decide whether the transaction that is being requested is likely to go through.
- Prior to magnetic stripe cards, this process required a merchant to make a telephone call to the issuer.
- The ability for a terminal to extract data from the magnetic stripe and automatically contact the issuer in order to authorise a transaction certainly makes this process easier.

- However, it is important to note that with magnetic stripe cards this process still requires direct (online) communication with the card issuer.
- This requirement has restricted the adoption of payment cards of this type in countries with poor communication infrastructures.

12.4.3 EMV cards

EMV cards were introduced for two main reasons.

- The first reason was in order to improve the security of payment card transactions.
- The other reason was to lower telecommunication costs by introducing a secure means of authorising a transaction offline, hence reducing the number of times that a merchant might have to contact a card issuer.

PIN VERIFICATION

PIN verification becomes much more straightforward for EMV than for magnetic stripe cards, since the PIN can be stored on the chip itself. This allows a terminal to easily verify the PIN without having to contact the card issuer, or use a service based on a PVV.

OFFLINE DATA AUTHENTICATION

- In order to authorise an EMV card transaction, a terminal must first decide whether to do an offline check, or whether to conduct a stronger online check that involves communicating with the card issuer.
- The decision as to which check to conduct depends on the transaction amount and the number of transactions conducted since the last online check.
- Offline data authentication does not involve the card issuer. In its most basic form, it provides a means of gaining assurance that the information stored on an EMV card has not been changed since the payment card was created by the card issuer.
- In other words, it provides data origin authentication of the fundamental card data. The stronger mechanisms also provide entity authentication of the card.
- Offline data authentication of a payment card can be conducted directly by a terminal that the card has been inserted into.

- It is impractical to provide this offline service using symmetric cryptography, since each terminal would need to share a symmetric key with every possible issuer.
- The use of key translation, for magnetic stripe PIN verification, requires the issuer to be online.
- Thus public-key cryptography, in the form of a digital signature scheme, is used to provide offline data authentication.
- For space efficiency reasons, EMV cards use a type of RSA digital signature scheme with message recovery to provide this assurance.
- EMV provides three offline data authentication mechanisms:

  - **Static Data Authentication (SDA)** is the simplest technique. All that is checked is the digital signature on the card data that is stored on the card. Verification of this digital signature requires access to the issuer's verification key. Clearly it is not reasonable to expect every terminal to have direct access to every issuer's verification key. Thus EMV employs a simple certificate hierarchy. In this case the card stores a public-key certificate containing the verification key of the issuer. This certificate is signed by the PCO, and the PCO's verification key is installed in every terminal supporting EMV.

  - **Dynamic Data Authentication (DDA**) goes one step further and provides this assurance in a dynamic way that differs for each transaction, hence providing another layer of security against card counterfeiting. During DDA, a challenge–response protocol is run that provides entity authentication of the card. In this case each card has its own RSA key pair and includes a public-key certificate for the card's verification key, signed by the issuer, as well as the issuer's public-key certificate, signed by the PCO. We thus have a three-level public-key certificate chain. The card computes a digital signature on the card data as well as some information unique to the current authentication session. The terminal uses the certificates offered by the card to verify this digital signature.

  - **Combined Data Authentication (CDA)** is similar to DDA, except that the card also signs the transaction data, thus providing assurance that the card and terminal

have the same view of the transaction. This protects against man-in-the-middle attacks that seek to modify the transaction data communicated between card and terminal. In contrast, DDA can take place before the transaction details have been established.

ONLINE AUTHENTICATION

Online authentication is the stronger check, which requires communication with the card issuer. As with DDA, the objective of online card authentication is for a terminal to gain entity authentication assurance of a payment card that is involved in a transaction. This is provided by means of a simple challenge– response protocol, based on a symmetric key that is shared by the card issuer and the payment card, which stores it on the chip. The only complication is that the terminal does not share this key, hence the issuer must be contacted online in order to verify the response. More specifically:

1. The terminal generates transaction data (which includes the payment card details) and a randomly generated challenge, which it then sends to the card.

2. The card computes a MAC on this data with the key that it shares with the issuer. This MAC is called the authorisation request cryptogram, and is passed on to the issuer.

3. The issuer computes its own version of the authorisation request cryptogram and compares it with the value received from the card. The issuer is also able to conduct a check that there are sufficient funds in the account to proceed with the transaction.

TRANSACTION CERTIFICATES

At the end of each transaction a transaction certificate (TC) is generated. This is a MAC computed on the details and outcome of the transaction and is passed back to the card issuer. The TC is computed using the key shared by the card and the card issuer. The TC is normally only required as evidence in the event of a subsequent dispute about certain aspects of the transaction.

SECURITY OF MANAGEMENT FUNCTIONS

A number of important management functions concerning security features of the payment card can be remotely managed by sending instructions to the card.

These include PIN changes, PIN unblocking instructions and changes to card data items (such as credit limits). These instructions are sent by the card issuer to the card (via a terminal). They

are authorised by computing and verifying a MAC on the instruction, which is generated using a symmetric key that is shared by the card issuer and the card. Since this is a very different use of symmetric cryptography, in line with the principle of key separation this key is different from the one used in online authentication.

12.4.4 Using EMV cards online

- An increasing number of transactions are conducted when the card is remote from the merchant, most commonly when a customer makes an online transaction. These are referred to as card-not-present (CNP) transactions.
- The potential for fraud in such transactions is high, since the most common information used to authenticate CNP transactions is simple card data (PAN, expiry date, CCV2), which is relatively easily acquired by a determined attacker.
- From the card holder perspective, the counter to this fraud threat has been the ability to challenge fraudulent transactions. However, this brings significant costs to the merchants, as well as being an inconvenience to the PCOs and cardholders when new cards have to be reissued to customers who have been fraud victims.
- Secure Electronic Transactions (SET) was a standard that proposed a heavy architecture and set of procedures for securing CNP transactions. It relied on an overarching public-key management system and required all merchants to acquire special supporting equipment. Its complexity prevented it from being successful and so Visa and Mastercard developed a more lightweight approach known as 3DSecure.
- The two main goals of 3DSecure are:
    1. The card issuer is able to authenticate its payment card holders during a CNP transaction.
    2. A merchant gains assurance that it will not later be financially punished because of a fraudulent transaction.

3DSecure relies on the following process:

1. A merchant that is 3DSecure-enabled puts in a request for authorisation of the card.
2. The card issuer contacts the card holder and requests authentication information. While EMV-CAP provides a natural way to enable this, a common instantiation is for the card issuer and card holder to have

preagreed a password, which the card holder must enter into a form presented to them in an embedded frame on their browser.

3. 3. If authentication is successful, the card issuer computes a MAC on the critical transaction data, using a symmetric key known only to them. This MAC is known as a Cardholder Authentication Verification Value (CAVV) and acts as sort of 'signature', vouching for the authentication of the card holder and the transaction data. The CAVV will be used to resolve any subsequent disputes about the transaction.

12.4.5 Using EMV cards for authentication

Since EMV cards have cryptographic capability, and EMV-supporting bank customers have such a card by default, it is natural to consider using the EMV card as part of an entity authentication mechanism. This is precisely the thinking behind the Chip Authentication Program (CAP), which specifies a range of entity authentication options (EMV-CAP explicitly refers to MasterCard technology, while Visa have a similar scheme known as Dynamic Passcode Authentication). These are supported by a CAP reader, which is a handheld device with a display and keypad. The customer authenticates directly to the CAP reader by means of a PIN. The CAP reader can then support several different entity authentication mechanisms:

- **Identify.** This option displays a number on the CAP reader that is computed from a symmetric key on the EMV card and an EMV customer transaction counter, which is also stored and updated on the card. This mechanism is a type of sequence-number-based dynamic password scheme. The cryptographic computation essentially involves computing a CBC-MAC on the input.

- **Response.** In this case the bank provides the customer with a randomly generated challenge. The customer types the challenge into the CAP reader, which computes a response using the symmetric key on the EMV card (again, based on CBC-MAC). Finally, the customer provides the bank with the displayed response.

- **Sign.** This is stronger version of the response mechanism, which involves the CBC-MAC being computed on basic transaction data (amount and recipient account) as well as the challenge value. This can be used to provide a type of 'digital signature' on the transaction. This is an example of the 'asymmetric trust relationship' use of MACs to provide non-repudiation.

12.4.6 Payment card key management

KEY MANAGEMENT SYSTEM

While the cryptography used by magnetic stripe cards is entirely symmetric, EMV uses a hybrid of symmetric and public-key cryptography. While PCOs allow issuing and acquiring banks to manage the keys of their own customers, the PCOs provide overarching key management services that link up these banks and facilitate secure transactions. The model depicted in Figure 12.8 that underlies payment card transactions is essentially the same as the connected certification model.. It is thus a good model to adopt given the distributed nature of a PCO's network of banks.

KEY GENERATION

A PCO generates its own master public-key pair. PCOs maintain master RSA key pairs of different lengths in order to cope with potential improvements in factorisation techniques. Individual banks are responsible for the genealso maintained on the card and is communicated to relying parties during a transaction.

KEY ESTABLISHMENT

The advantage of a closed system of this type is that the keys stored on a card can be pre-installed during the manufacturing (or personalisation) process. This is slightly more complex for RSA key pairs, since they cannot be mass generated as efficiently as symmetric keys. The session keys used in individual transactions are established on the fly during the transaction, as just discussed. A PCO's verification key is installed into terminals during their manufacture. PCOs also oversee an important symmetric key hierarchy. At the top level are zone control master keys, which are manually established using component form. These are used to establish the acquirer working keys and issuer working keys.

KEY STORAGE

All the long-term secret or private keys used in EMV payment card systems are protected in tamper-resistant hardware, either in the form of an issuer's hardware security module or the chip on the payment card.

KEY USAGE

In general, key separation is enforced in EMV. The two main security functions that involve encryption using keys stored on a card are conducted using separate symmetric keys.

12.4.8 Payment card cryptographic design issues

The main cryptographic design issues concerning payment card cryptographic security mechanisms are:

Use of well-respected cryptographic algorithms. Payment cards use 2TDES and RSA, which are well-established algorithms.

Targeted use of public-key cryptography. Payment cards uses public-key cryptography precisely when it delivers substantial benefits, namely in simplified key management for the support of offline data authentication.

Balance of control and flexibility. PCOs strictly control the part of the key management infrastructure that they need to, but otherwise devolve control to participating banks. This provides scalable key management and allows banks to develop their own relationships with their customers.

Efficient use of related data. Payment cards use data in a number of imaginative ways. For example, PANs are used to derive keys, and items of transaction data are used as challenges in authentication protocols. This is both efficient and clever.

12.5 Cryptography for video broadcasting

12.5.1 Video broadcasting background

Commercial television broadcasters have traditionally financed the provision of their services either through government subsidy or advertising revenue. This is primarily because most analogue broadcast content can be received by anyone with access to a suitable device, such as a television set. This makes alternative business models, such as those based on annual subscription, hard to enforce.

An alternative option is to ‘encrypt’ analogue content using special techniques that are developed for particular broadcast technologies. This process is often referred to asscrambling. This requires a consumer of content to acquire dedicated hardware in order to use decryption to recover the content. This requirement thus presents an opportunity for revenue collection. Digital video broadcast networks process digital content, thus making it possible to use the full range of modern cryptographic mechanisms to protect content. This, in turn, enables a wide variety of different business models. Most of these require consumers to obtain specific hardware (or occasionally software) in order to recover content. Common models include full subscription services that allow consumers to access all broadcast content for a specific period

of time, package subscription services that allow consumers to access ‘bundles’ of predefined broadcast content, and pay-per-view services that allow the purchase of specific broadcast content (for example, a live broadcast of a sports event). The compression of digital video broadcasts also allows more content to be broadcast than that of analogue over a similar bandwidth. It thus creates the opportunity for a much more diverse provision environment.

- Figure 12.9 shows a simple example of a possible infrastructure for a digital video broadcast network.
- The broadcast source transmits the broadcast content, and is under the control of the broadcast provider. The broadcast content is transmitted to the consumer of the content, who requires access to a suitable broadcast receiver in order to receive the signal.
- In the example in Figure 12.9, the communication channel is over the air via a satellite link, hence the broadcast receiver takes the form of a satellite dish.
- However, a digital video broadcast could just as well be transmitted by other media, such as a fibre optic cable, in which case the broadcast receiver is any hardware device capable of receiving the content.
- As well as receiving the data transmitted by the broadcast source, the consumer requires a content access device, which has the capability of decrypting to recover the broadcast content. While this can be implemented in software, most content access devices are hardware devices that contain a smart card.
- The critical data that is required to control access to the broadcast content, such as cryptographic keys, will normally be stored on the smart card, thus allowing the potential for a content access device to be used to obtain content from different broadcast providers.
- In such cases choose to regard the ‘content access device’ as the hardware and the smart card, unless otherwise specified.

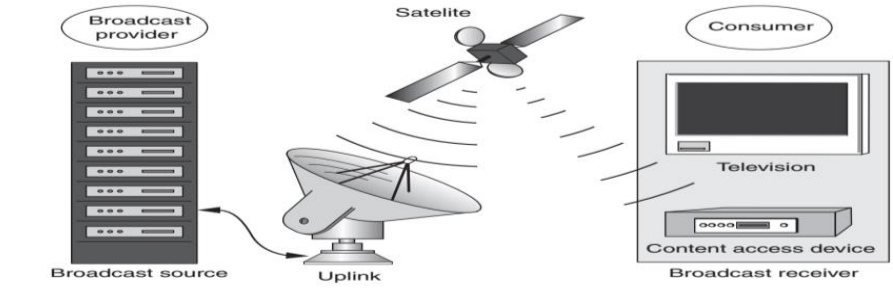


Figure 12.9. Digital video broadcast network

12.5.2 Video broadcasting security requirements

Two important constraints on the broadcast network environment:

One-way channel. The broadcast communication channel only operates in one direction: from broadcast source to broadcast receiver. There is no means by which a consumer can send information back to the broadcast source on this communication channel.

Uncontrolled access. Just as for analogue broadcasts, digital video broadcast content can be received by anyone with the right broadcast receiver technology (a satellite dish in our example in Figure 12.9).

The security requirement for digital video broadcast is thus, simply:

Confidentiality of the broadcast content. In order to control the revenue stream the broadcast provider must make the broadcast content essentially 'worthless' to anyone who has not purchased the necessary content access device. In other words, confidentiality is required on the broadcast channel, with only authorised consumers having access to the necessary decryption keys.

Entity authentication. Most of our previous applications required some level of entity authentication, which would be one way of controlling which consumers get access to broadcast video content. However, this requires the consumer to be able to communicate with the broadcast source, which in this case is not possible. Entity authentication of the broadcast source is possible, but unnecessary, since the threat of an attacker posing as a broadcast source and sending false video broadcasts is not particularly relevant to most commercial broadcast environments.

Data integrity. A video broadcast channel is potentially prone to errors in the transmission channel. However, the threat against data integrity is more likely to be accidental errors rather

than deliberate ones introduced by a malicious attacker. Hence the solutions lie in the area of error-correcting codes and not cryptographic mechanisms.

12.5.3 Cryptography used in video broadcasting

The cryptographic design decisions for GSM encryption namely:

A fully symmetric cryptographic architecture. Video broadcast networks are closed systems.

Stream ciphers for data encryption. Video broadcasts involve streaming data in real time over potentially noisy communication channels.

Fixing the encryption algorithm. Agreeing on use of a fixed encryption algorithm allows this algorithm to be implemented in all broadcast receivers, aiding interoperability.

Proprietary encryption algorithm. Choosing to design a proprietary encryption algorithm was justifiable for the same reasons as for GSM. In this case the expertise lay with members of the Digital Video Group (DVB), which is a consortium of broadcasters, manufacturers, network operators, software developers, and regulatory bodies with interests in digital video broadcasting. As for GSM, one of the influences behind the design was to make decryption as efficient as possible, since content access devices are less powerful than broadcast sources.

The proprietary encryption algorithm that was designed was CSA. While CSA was standardised by ETSI. The CSA was implemented in a software application and subsequently reverse-engineered. The CSA is essentially a double stream cipher encryption. The first encryption is based on a proprietary block cipher deployed in CBC mode, which means that it operates as a stream cipher . The second layer of encryption uses a dedicated stream cipher to encrypt the ciphertext produced during the first encryption (this is a slight simplification). The key length is 64 (only 48 of the bits are actually used for encryption) and the same encryption key is used for both encryption processes.

12.5.4 Key management for video broadcasting

The primary key management task for digital video broadcasting is simple to state: the keys required to recover broadcast content should be available only to those consumers who are authorised to view the broadcast content. However, there are several complications:

The number of potential consumers. A digital video broadcast network is likely to have a large number of consumers (in some cases this could be several million), hence the key management system design must be sufficiently scalable that it works in practice.

Dynamic groups of authorised consumers. The groups of consumers who are authorised to view digital broadcast content is extremely dynamic. Payper-view services provide the extreme example of this, where the group of authorised consumers is likely to be different for every content broadcast.

Constant service provision. In many applications a broadcast source will be constantly streaming digital video content that needs to be protected. There are no break periods in which key management operations could be conducted. Most key management must therefore be conducted on the fly.

Precision of synchronisation. Stream ciphers require the keys at each end of the communication channel to be synchronised. In digital video broadcasting this synchronisation has to happen between the broadcast source and all (and as we have just pointed out, this could be 'millions of ') authorised consumers. This synchronisation must be close to being perfect, otherwise some consumers may incur a temporary loss of service.

Instant access. Consumers normally want instant access to broadcast content and will not tolerate delays imposed by key management tasks.

VIDEO BROADCAST KEY MANAGEMENT SYSTEM DESIGN

- All video broadcast content must be encrypted during transmission. A symmetric key, which will be refer to as the content encryption key (CEK). Since the broadcast source only transmits one version of an item of broadcast content, the content encryption key used to encrypt a specific item of content must be the same for all consumers.
- Since consumers have different access rights to digital content, the CEK for two different items of broadcast content must be different.
- The challenge is thus to make sure that only consumers who are authorised to access content can obtain the appropriate CEK.

The following key management design decisions:

- **Encrypted CEK is transmitted in the broadcast signal.** The CEK is transmitted along with the content itself and is made 'instantly available' by being continuously repeated, perhaps every 100 milliseconds or so. Clearly the CEK cannot be transmitted in the clear, otherwise anyone receiving the broadcast signal could obtain it and hence recover the content. Thus the CEK is transmitted in encrypted form. We will refer to the key used to encrypt the CEK as the key encrypting key (KEK).

- **CEK is frequently changed.** Once someone has access to the CEK, they can use it to recover all broadcast content that is encrypted using it. Thus it is important to frequently change the CEK. In most video broadcast systems the CEK typically changes every 30 seconds, but this can happen as often as every five seconds.

- **CEK is transmitted in advance.** In order to aid synchronisation and instant access, the CEK is issued in advance of the transmission of any content broadcast using it. Clearly this cannot be too far in advance because of the dynamic nature of the authorised consumer base. The compromise is to constantly transmit two (encrypted) CEKs, which consist of:

  1. the current CEK that is being used to encrypt the current broadcast content;

  2. the 'next' CEK that will be used to encrypt the next broadcast content. Hence the content access device has time to recover the next CEK and have it instantly available as soon as the CEK is changed.

- **Use of symmetric key hierarchies**. Video broadcast schemes uses KEKs to encrypt the CEKs. This of course just 'transfers' the access problem to making sure that only authorised consumers have access to the required KEKs.

VIDEO BROADCAST KEY ESTABLISHMENT

- A video broadcast scheme establishes the KEKs that are necessary for authorised consumers to obtain the CEKs that they are entitled to.

- A video broadcast schemes use symmetric key hierarchies. At the 'top' of each these hierarchies are keys that are shared only by the broadcast provider and a particular consumer, which we refer to as consumer keys (CKs).

- In a simple system, with relatively few consumers, these CKs could be used to encrypt the KEKs. However, there are two reasons why this is not very practical:

  1. Most video broadcast systems have so many consumers that sending an encrypted KEK in this way would require too much bandwidth, since a unique ciphertext would have to be sent for each consumer.

  2. Each KEK itself must be frequently changed, for similar reasons to the CEKs. This might happen, say, on a daily basis. Thus the bandwidth problems are further exacerbated by the need to frequently update the KEKs.

  - The compromise is to deploy zone keys (ZKs), which are keys shared by groups of consumers. Zone keys have longer lifetimes than KEKs, but shorter lifetimes than CKs.

  - A relevant ZK is initially sent to a consumer encrypted using their CK. The consumer then uses the ZK to recover KEKs, which are used to recover CEKs.

  - When a ZK needs to be changed, the new ZK does need to be sent to every consumer who requires it, but this event occurs much less frequently than for KEKs (which in turn occurs much less frequently than for CEKs).

  - The consumer keys, which sit at the top of these key hierarchies, are stored on the smart cards of the content access devices. They are thus established prior to the issuing of the smart cards to the consumers.

  - A simple example set of key hierarchies is shown in Figure 12.10. In this example there are are five consumers, divided into two zones. In practice, multiple layers of zone keys can be deployed in order to enhance scalability.

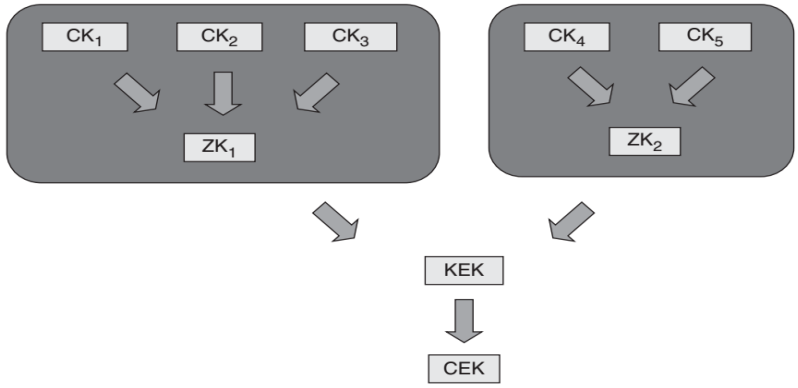


Figure 12.10. Digital video broadcast scheme key hierarchy

12.5.6 Video broadcast design issues

Use of symmetric cryptography. The closed nature of a video broadcast scheme facilitates the use of a fully symmetric cryptosystem.

Use of a symmetric key hierarchy. Video broadcast schemes provide a good example of the benefits of deploying a key hierarchy to support symmetric key management.

The influence of operational constraints. While the security requirements for video broadcast networks are fairly straightforward, the operational constraints require some innovative key management controls.

Partially standardised infrastructure. Video broadcast schemes follow some common standards, for example, for content encryption, while leaving other aspects such as higher-level key establishment open to custom design by individual broadcast providers. While this provides the opportunity for a diverse market of interoperable schemes, it also presents a potential source of vulnerability in specific systems.

12.6 Cryptography for identity cards

12.6.1 eID background

Within a specific context, such as a workplace, most people accept cards that contain and/or display data relating to the identity of the holder. However, the attitude towards national identity card schemes is surprisingly diverse and, to an extent, cultural. In some countries, such as the UK, there is a great deal of hostility to such schemes. This is largely due to concerns over privacy issues, costs of deployment, data management and doubts about the utility of such a

scheme. In many other countries, such as Belgium, national identity card schemes have been rolled out and are integrated into daily life.

The main application of national identity cards is to present independently issued evidence of the identity of the card holder. Such cards typically display a photograph of the card holder and some personal details, which may include a handwritten signature. However, the progress in smart card technology and the development of cryptographic applications has presented the opportunity for national identity cards to provide additional functionality and thus, perhaps, become more useful.

- The eID card scheme was motivated by the establishment of the 1999 European Directive on Electronic Signatures, which created a framework that enabled electronic signatures to become legally binding.

- The first eID cards were issued to Belgian citizens in 2003 and from 2005 all newly issued identity cards were eID cards.

- The eID card has four core functions:

  Visual identification. This allows the card holder to be visually identified by displaying a photograph on the card alongside a handwritten signature and basic information such as date of birth (see Figure 12.11). This functionality is also provided by previous Belgian identity cards.

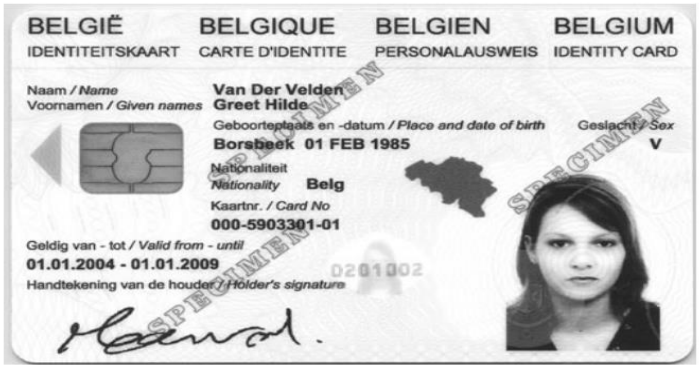


Figure 12.11. eID card

  Digital data presentation. This allows the data on the eID card to be presented in electronic form to a verifying party. The card data has a specific format and includes:

  • a digital photograph of the card holder;

• an identity file which consists of:

- personal data such as name, national identity number, date of birth, and special status (for example, whether the card holder has a disability);
- a hash of the digital photograph of the card holder;
- card-specific data such as chip number, card number and validity period;

• an address file which consists of the card holder's registered address.

Applications of digital data presentation include access control to facilities such as libraries, hotel rooms and sports halls.

Digital card holder authentication. This allows a card holder to use the eID card to 'prove' their identity in real time to a verifying party. In other words, it facilitates entity authentication of the card holder. The many listed applications of digital card holder authentication include remote access to various internet services, including official document requests (for example, birth certificates), access to an online tax declaration application, and access to patient record information.

Digital signature creation. This allows the card holder to use the eID card to digitally sign some data. Applications of digital signature creation include signing of electronic contracts and social security declarations. Digital signatures created using an eID card are legally recognised.

12.6.2 eID security requirements

Data origin authentication of the card data. In order to provide digital data presentation, assurance that the card data has not been changed since the card was issued must be provided.

Ability to provide a data origin authentication service. In order to support digital card holder authentication, it is necessary for an eID card to be used as part of an entity authentication service. The eID card's role in this is to provide a data origin authentication service, which can then be used to support an entity authentication protocol between the card holder and a verifying party.

Ability to provide a non-repudiation service. In order to support digital signature creation, an eID card must be able to provide non-repudiation.

12.6.3 Cryptography used in eID cards

The cryptography in the eID card is relatively straightforward. The following design issues are important in determining the eID card's cryptographic capability:

Use of public-key cryptography. The open nature of the potential application space for eID cards dictates that public-key cryptography must be supported. It is impractical for an eID card to contain pre-loaded symmetric keys that will be 'meaningful' to all unknown future applications.

A digital signature scheme suffices. All three of the security requirements for eID cards can be met by using a digital signature scheme. The first requirement does not even require the digital signature scheme to be implemented on the eID card, however, the second and third requirements do need this. Note that the eID card is not required to have the capability to encrypt or decrypt data.

Use of a publicly known digital signature scheme. In order to encourage use of the eID card and aid interoperability, it is imperative that the digital signature scheme that is deployed is widely respected and supported.

12.6.4 Provision of the eID card core functions

The eID card scheme is governed by an entity called the National Register (NR). The NR can be considered as a trusted third party that facilitates the scheme. The NR is responsible for issuing eID cards and hence also takes 'ownership' of the personal data contained on them.

Each eID card contains two signature key pairs and one additional signature key:

Authentication key pair. This key pair is used to support digital card holder authentication.

Non-repudiation key pair. This key pair is used to support digital signature creation.

Card signature key. This signature key can be used to authenticate the card, rather than the card holder. Only the NR knows the verification key that corresponds to a particular eID card. This signature key is only used for administrative operations between the card and the NR.

DIGITAL DATA PRESENTATION

This involves a verifying party reading the card data and then gaining assurance that the data on the card is correct. To gain this assurance, the verifying party needs to verify two digital signatures that are created by the NR and stored on the eID card:

Signed identity file. This is a digital signature generated by the NR on the identity file.

Signed identity and address file. This is a digital signature generated by the NR on a concatenation of the signed identity file and the address file. In other words, this takes the form:

$$sig_{NR}(sig_{NR}(\text{identity file}) \,||\, \text{address file}).$$

- A verifying party can then verify the card data by first using the verification key of the NR to verify the signed identity file. If this check is fine then they can proceed to verify the signed identity and address file.
- The reason that the NR does not simply sign all the card data is that address changes are much more frequent than changes to the content of the identity file. Thus the NR can update an address on the card without having to reissue a new eID card.

DIGITAL CARD HOLDER AUTHENTICATION

Each eID card holder can activate the signature keys on the eID card through the use of a PIN. The card holder also requires access to an eID card reader, which may include a PIN pad. This provides an interface between the eID card and the card holder's computer. A typical card holder authentication process is illustrated in Figure 12.12. In this example, a visited web server is requesting authentication of the card holder:

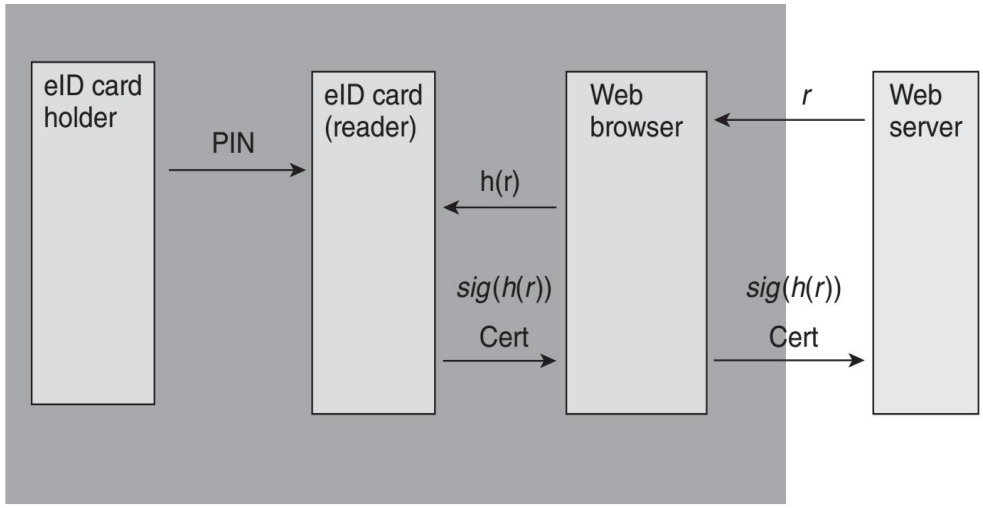


Figure 12.12. eID card holder authentication

1. The web server randomly generates a challenge r. This is sent to the card holder's browser, which displays a request to login.
2. The card holder enters their PIN into the eID card reader which, if correct, authorises the eID card to proceed with the authentication.
3. The card holder's browser computes a hash $h(r)$ of the challenge r, using a suitable hash function (see Section 6.2) and sends this to the eID card via the card reader.
4. The eID card digitally signs $h(r)$ using the authentication signature key and sends this to the web server via the card holder's browser, along with the card holder's authentication verification key certificate.
5. The web server verifies the received certificate and, if this is successful, verifies the signature and checks that it corresponds to the challenge r. If everything is in order, the card holder is successfully authenticated.

12.6.5 eID key management

eID CERTIFICATES The eID card scheme key management is based on the closed certification model. It uses a certification hierarchy, in order to provide a scalable approach to certificate issuing. This certification hierarchy is indicated in Figure 12.13. The main CAs involved are:

Belgium Root CA. This CA is the root CA that oversees all the eID scheme certification. It possesses a 2048-bit RSA verification key certificate that is both self-signed and signed by a commercial CA.

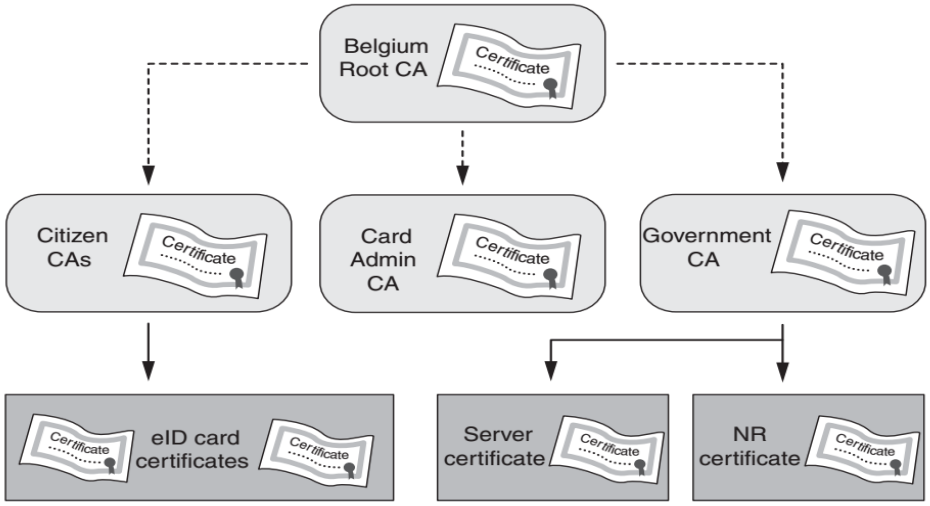


Figure 12.13. eID certification hierarchy

Citizen CAs. These CAs issue certificates to card holders and are responsible for signing the eID card authentication and non-repudiation verification key certificates. Citizen CAs have a 2048-bit RSA verification key signed by the Belgium Root CA.

Card Admin CA. This CA issues certificates to organisations carrying out administrative operation of the eID card scheme, such as those managing address changes and key pair generation. The Card Admin CA has a 2048-bit RSA verification key signed by the Belgium Root CA.

Government CA. This CA issues certificates to government organisations and web servers, including the NR. The Government CA has a 2048-bit RSA verification key signed by the Belgium Root CA.

Each eID card stores five certificates:

1. the Belgium Root CA certificate;
2. the Citizen CA certificate for the Citizen CA that issued the eID card's certificates;
3. the eID card authentication verification key certificate;
4. the eID card non-repudiation verification key certificate;
5. the NR certificate.

eID CARD ISSUING PROCESS

The process of issuing an eID card is quite complex and involves several different organisations. It serves as a good illustration of the intricacies of generating public- key certificates,. The process is indicated in Figure12.14 and consists of the following steps:

1. Either after requesting, or being invited to apply for, an eID card, the eID applicant attends a local government office. This office essentially acts as the RA. The applicant presents a photograph to the RA, which then verifies the personal details of the applicant and formally signs an eID card request.
2. The eID card request is sent from the local government office to the *card* personaliser(CP), and the NR is notified. The CP checks the eID card request. For simplicity we will assume the existence of a single CP, who is responsible for creating the physical aspects of the card and for inputting the relevant data onto the chip on the card.
3. The CP creates a new eID card and generates the required key pairs on the card itself. The CP then sends are quest for certificates to the relevant Citizen CA via the NR, who issues a certificate serial number for each certificate.
4. The Citizen CA generates certificates and sends them to the CP, who stores the month ecard. The CA then immediately suspends these certificates.
5. The CP writes all the remaining card data onto the card and then deactivates the card.
6. The CP sends:
   - The first part of an activation code *AC*1 to the NR;
   - These cond part of the activation code *AC*2 and a PIN to the applicant;
   - The inactive eID card to the RA.
7. The applicant revisits the RA and presents *AC*2. This is then combined with *AC*1, which the RA requests from the database of the NR.
8. The CA activates the suspended card certificates and the active eID card is issued to the applicant.

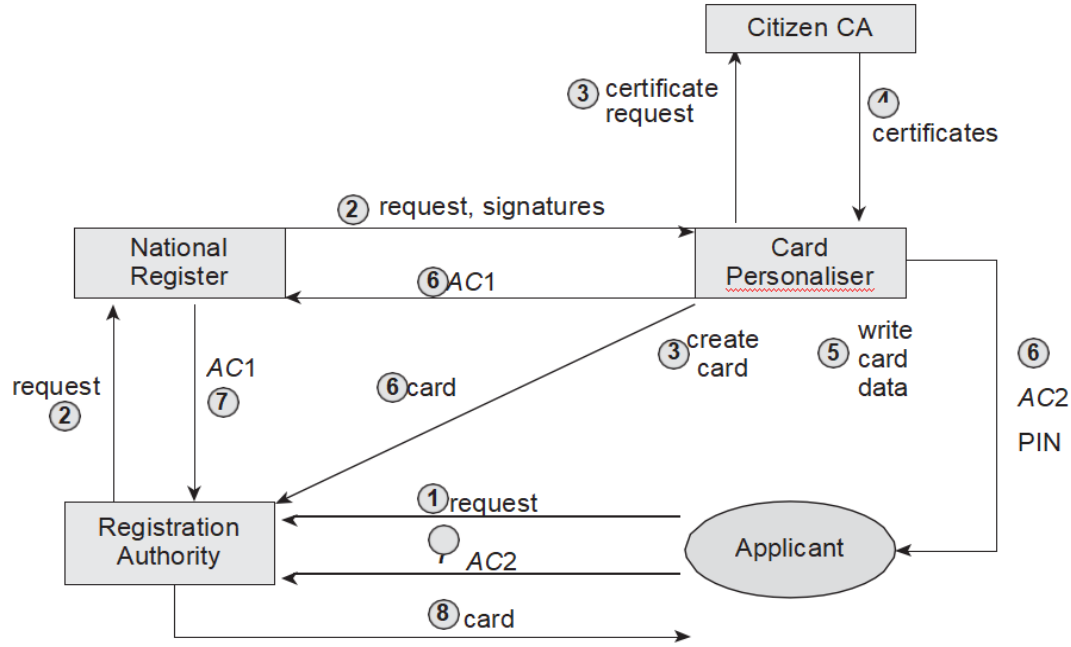


Figure 12.14. eID card issuing process

eID CERTIFICATE REVOCATION

There are two special situations in which an eID card certificate has the status of being revoked:

1. the eID card non-repudiation verification key certificate is revoked for juveniles under the age of 18;

2. the eID card authentication verification key certificate is revoked for children under the age of 6.

- The main technique used to manage certificate revocation in the eID card scheme is CRLs .
- A significant problem of the eID card scheme is that the potential size of CRLs is considerable.
- The eID card scheme Citizen CAs issue new base CRLs every three hours. During the period between updates of the base CRL, much smaller delta CRLs are issued, which identify changes to the last base CRL.
- In this way anyone who wishes to maintain their own local copy of the complete CRLs for the eID card scheme does not have to regularly download the full database.
- All CRLs are digitally signed by the issuing Citizen CA using 2048-bit RSA.

eID SIGNATURE VALIDITY

the potential validity of digital signatures during two specific periods of time:

Digital signatures created after an incident but before revocation. A potential problem arises if a relying party verifies an eID card signature in the period between occurrence of a security incident (of a type that invalidates the eID card non-repudiation verification key certificate) and the revocation of that certificate. If the time of the incident can be precisely verified then, technically speaking, a digital signature created during this period is unlikely to be valid. Applications need to be aware of this potential problem and have procedures for coping with it. The Citizen CAs assist this process by frequently issuing base and delta CRLs.

Validity of digital signatures after expiry or revocation of the eID card (nonrepudiation verification key certificate). So long as a digital signature is verified before expiry or revocation of the eID card (or its non-repudiation verification key certificate) then it should still be regarded as valid (and, indeed, may be legally binding) after the expiry or revocation date. One method for making this more explicit is for the signer who signs some data to obtain a digital signature from a trusted third party that attests to the validity of that signature at a specific point in time. Namely, the signer Alice presents her digital signature sigA(data) to the TTP, who verifies this signature at time t and then generates the digital signature:

$$sig_{TTP}(sig_A(\text{data}) \,||\, t).$$

The TTP thus acts as an archiving service. After the expiry or revocation of her eID card, Alice can still present the archived signature as evidence of its validity. Note that any future relying party does not need to verify Alice's original signature, but does have to trust the TTP. This process assumes that:

• The TTP's verification key has a longer lifetime than Alice's. On expiry of the TTP's verification key, Alice can always ask the TTP to resign the archived signature with its new signature key.

• No flaws are subsequently found in any of the processes or algorithms used to generate or validate Alice's digital signature.

12.6.7 Design issues

The main design issues concerning the eID card scheme are as follows:

Use of public-key cryptography. While eID cards are issued within a closed environment, they are intended for use in open environments. Thus the use of public-key cryptography is appropriate.

Use of publicly known algorithms. To increase confidence and support interoperability, the eID card scheme uses the well-respected RSA digital signature scheme.

Use of certification hierarchies. The eID card scheme's national reach lends itself very naturally to a certification hierarchy, with central CAs supporting regional registration authorities.

Specific data handling. The eID card design demonstrates that in real applications different data items may require different management. This is reflected in the way that card data is digitally signed, which recognises that address data normally changes much more frequently than other types of personal data.

Flexibility. The eID card scheme is primarily an enabler for cryptographic applications. It therefore leaves specific applications a degree of flexibility on how they manage security of applications interacting with eID cards. In particular, applications must manage their own certificate revocation processing.

12.7 Cryptography for home users

12.7.1 File protection

There are two main reasons for a home user wanting to use cryptography to protect a file:

Additional storage protection. Most computer systems, including desktops, laptops, PDAs and smart phones, have basic security controls that provide some protection against unauthorised parties from accessing the files that are stored on them. Most home users rely on basic user access control mechanisms for this protection. The commonest such control is to provide entity authentication to the computer itself through the use of a passwordbased mechanism. However, such controls do not normally provide strong protection, since it is relatively easy to overcome them. In addition, different types of portable media exist for storing files, such as DVDs, memory cards and USB tokens, many of which have no default file storage protection mechanisms.

File transfer security. A user may wish to transfer a file from one computer system to another. While the end computer systems may be protected, the communication channel is potentially insecure.

FULL DISK ENCRYPTION

- One option for a home user who is concerned about the security of files stored on their desktop or laptop is to deploy full disk encryption, which encrypts every bit of data contained on the computer system.
- Full disk encryption mechanisms are available both in hardware and software, with hardware mechanisms typically offering greater security and performance.
- Full disk encryption is particularly attractive for laptops, which are at risk of becoming lost or stolen.
- The 'classical' physical attack on a stolen computer is for an attacker to remove the disk and reinstall it on a computer for which the attacker has administrator access.
- There are two constraints which motivate the type of encryption deployed in full disk encryption mechanisms:
  - **Performance**. Encryption and decryption operations need to take place as fast as possible, ideally without any apparent delay. Thus most full disk encryption mechanisms encrypt each disk sector, which typically consist of around 512 bytes, independently.
  - **Avoidance of storage overhead.** In order to use disk space efficiently, the encryption operation should not result in significantly more data being stored than would otherwise have been stored without full disk encryption.

VIRTUAL DISK ENCRYPTION

- ✓ An alternative to encrypting an entire disk is to use virtual disk encryption mechanisms, which can be used to encrypt chunks of data, usually referred to as containers.
- ✓ Virtual disk encryption can be deployed on devices such as USB tokens, as well as on desktops and laptops.
- ✓ In most solutions the user is required to authenticate to the device, usually by means of a password, in order to access the encrypted files within the container.
- ✓ There are several advantages of virtual disk encryption over full disk encryption:
  - Virtual disk encryption can be used to encrypt selected data on a disk, rather than the full disk.

- An encrypted container is normally portable, in the sense that it can be copied onto media such as a DVD. Thus virtual disk encryption can provide security for data transfer, as well as storage, in cases where the data can be physically transferred using portable media. Just as for full disk encryption, care needs to be taken to make sure that the mechanisms and processes used to support user (entity) authentication to the device and key management are adequately addressed.

FILE ENCRYPTION

- The greatest granularity of control over data encryption is to deploy file encryption, which encrypts individual files (or folders).
- One of the other main advantages of file encryption is that it can protect a file on a running computer system that an attacker has gained access to.
- Contrast this situation with, for example, a full disk encryption mechanism running on a computer that the user has authenticated to and then (foolishly) walked off and left unattended.
- Unlike full and virtual disk encryption, however, file (and folder) encryption do not normally prevent an attacker from learning data associated with the file, such as file size, file type and the folder name in which the file resides.
- Some operating systems provide in-built file encryption, such as the Encrypting File System (EFS) deployed in many Microsoft operating systems.
- EFS uses hybrid encryption to protect a file by first encrypting it with a unique symmetric key, which is then itself encrypted using the user's public key. The user's private key is then required in order to decrypt.
- One issue with in-built file encryption of this type is that the protection is not always maintained when the encrypted file is transferred to another storage medium.
- However, there are many third-party software applications providing general file encryption capability, some of which support transfer of encrypted data.
- File encryption is also appropriate for a user who only occasionally needs to encrypt a file, usually for transfer purposes.

- An example of encryption software for casual encryption of this type is GNU Privacy Guard (GPG). This uses hybrid encryption to encrypt files, as well as supporting digital signatures.

- A range of patent-free symmetric and public-key algorithms are supported. Users generate their own key pairs locally, using a passphrase to generate, and later activate, a key encrypting key that is used to protect the decryption key.

- Public-key management is lightweight and left at the user's discretion. Users could, for example, exchange public keys directly with known contacts or use a web of trust .

- Finally, some application software supports encryption for specific data formats. For example, Adobe software allows users to encrypt pdf files. Adobe originally used RC4 but now also supports AES. The key is activated using a password, which can be sent to the recipient of an encrypted file in order to allow them to decrypt and view it.

12.7.2 Email security

EMAIL SECURITY REQUIREMENTS

There are two potential concerns about the security of email:

Confidentiality. By default, email messages are unprotected during their transfer from the email sender's device to the email receiver's device. During that transfer the email message resides on several email servers and internet routers, as well as passing through various potentially unprotected networks. There are many points at which, at least in theory, the contents of an email message could be viewed by someone other than the intended recipient. In addition, users sometimes mistakenly send email to the wrong recipient, for example, by replying to all the recipients of an email rather than just the original sender. Thus there is certainly a case that could be made for requiring confidentiality of some types of email message.

Data origin authentication. Email messages are structured using a simple protocol that facilitates their transfer. This protocol includes fields for specifying the sender, recipient and subject, as well as the message itself. An informed attacker can fairly easily generate forged emails. In addition, at most of the points at which an attacker can read a genuine email, the attacker could intercept and make changes to the email message before forwarding it on to the recipient. This also makes a case for requiring data origin authentication of email messages.

Indeed, for some email messages we might even want to go further and require non-repudiation, but data origin authentication probably suffices for most traffic.

EMAIL SECURITY APPLICATIONS

- There are two well-known standards for protection of email, each of which are implemented by a wide range of email security applications.
- Both Open Pretty Good Privacy (OpenPGP) and Secure/Multipurpose Internet Mail Extensions (S/MIME) broadly work in the same way, although precise implementations may have minor differences.
- They both provide confidentiality and data origin authentication (non-repudiation) through support for encryption and digital signatures.
- They are either supported by default in certain email clients or can be installed through plug-ins.
- There are three ways in which email messages can be protected using these applications:
    - **Confidentiality only**. This is provided by hybrid encryption. The symmetric encryption key is either generated using a deterministic generator or a software-based non-deterministic generator. The body of the email message is then encrypted using this symmetric key, and the symmetric key is encrypted using the public key of the recipient. Data origin authentication only. This is provided by a digital signature scheme with appendix .The email message is first hashed and then signed using the signature key of the sender. The receiver will need to obtain the corresponding verification key in order to verify the resulting digital signature.
    - **Confidentiality and data origin authentication.** This is typically provided by following the MAC-then-encrypt construction. In other words, a symmetric encryption key is generated and the email message is digitally signed, as described above. The email message and the resulting signature are then both encrypted using the symmetric encryption key. Finally the symmetric encryption key is itself encrypted using the public encryption key of the recipient.

The main differences between OpenPGP and S/MIME are with respect to:

Cryptographic algorithms supported. OpenPGP implementations support a range of cryptographic algorithms. On the other hand, S/MIME is more restrictive and specifies the use of AES or Triple DES for symmetric encryption and RSA for digital signatures and public-key encryption (the original S/MIME proposal came from RSA Data Security Inc.).

Public-key management. Again, OpenPGP is more flexible and can be supported by almost any form of public-key management system. The default public key management model for OpenPGP is to use a web of trust, although more formal public-key management can also be supported. On the other hand, S/MIME is based on the use of X.509 Version 3 certificates supported by a structured public-key management system relying on Certificate Authorities.

AN ALTERNATIVE APPROACH TO EMAIL SECURITY

- Since the approaches to email security rely on the use of public-key cryptography, the problem of assurance of purpose of public keys needs to be addressed by whichever public-key management system is used to support an email security application.

- The IDPKC concept requires unique identifiers that can be associated with users of the system.

- In email security applications such a potential unique identifier exists in the form of the email address of the recipient. Thus, using IDPKC, an email sender is potentially able to send an encrypted email to any recipient simply by encrypting the email using the recipient's email address.

- The advantages offered by this concept have resulted in the commercial development of email security applications based on IDPKC.

- one of the potential drawbacks with IDPKC is the need for an online centrally-trusted key centre (TKC). Thus IDPKC is most suited to large organisations where such a TKC can easily be provided, rather than home users.

- However, a home user could well receive encrypted email from such an organisation without needing any formal relationship with the sender. In this case:

1. The sender (from the organisation supporting IDPKC) sends an encrypted email to the recipient (the home user), using the recipient's email address as the encryption key.
2. The recipient receives an email message informing them that they have received an encrypted email message and inviting them to visit a secure website in order to view the contents.
3. The recipient clicks on the provided web link and is directed via an SSL-protected channel to the organisation's TKC web server. This generates the necessary private decryption key and recovers the email, which is then displayed to the recipient.