Computer Security: Secret Key Crypto

B. Jacobs and J. Daemen
Institute for Computing and Information Sciences – Digital Security
Radboud University Nijmegen
Version: fall 2016

Outline

Crypto intro

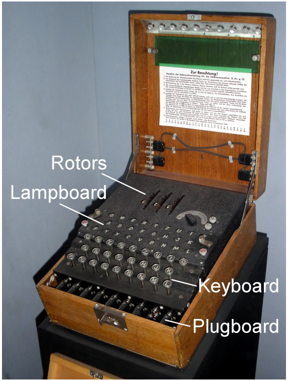
Symmetric crypto

Achieving security goals with symmetric crypto
- Confidentiality
- Integrity
- Authentication

Modes for encryption and authentication

e-Passport example

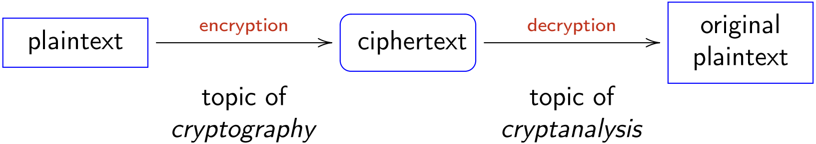
Old cryptographic systems



Scytala from Sparta **German Enigma from WWII**

Check out http://cryptomuseum.com/ for a large collection of (Dutch) devices

Situation & terminology



Officially,

$$\textit{cryptology} \quad = \quad \text{cryptography} + \text{cryptanalysis}$$

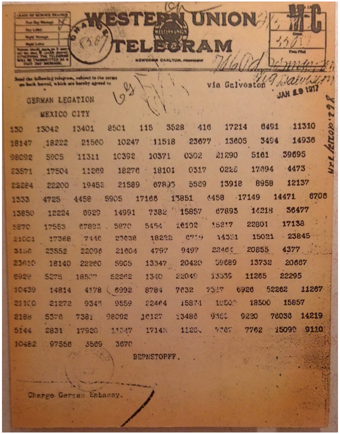
This is the official, somewhat outdated terminology. But often "crypto" or "cryptography" is used for "cryptology".

Cryptanalysis that changed the course of history

- The Zimmermann telegram in WWI, sent by Germany to incite war between Mexico & US, intercepted by the British and passed on the US; it brought the US into the war.
- The breaking of the German Enigma in WWII by the British, shortening the war by probably at least a year.
- The breaking of the Japanese JN25 code in WWII by the US
  - it provided crucial intelligence in the Midway battle (1942)
  - and for ambushing the plane of Marshal Yamamoto (1943)

(In the 1960s and 1970s cryptography in NL was probably third best in the world, with great work at MID and Philips Usfa.)

Zimmermann telegram, ciphertext and cleartext



WESTERN UNION TELEGRAM

NEWCOMB CARLTON, PRESIDENT

Send the following telegram, subject to the terms on back hereof, which are hereby agreed to

via Galveston

JAN 19 1917

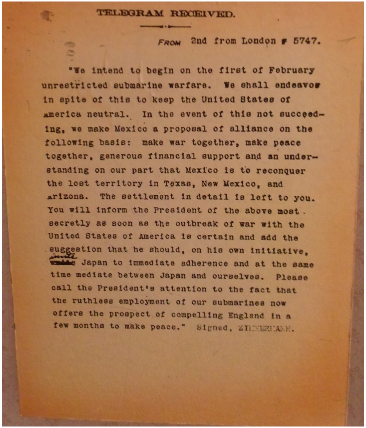
GERMAN LEGATION

MEXICO CITY

130 13042 13401 8501 115 3528 416 17214 6491 11310
18147 18222 21560 10247 11518 23677 13605 3494 14936
98092 5905 11311 10392 10371 0302 21290 5161 39695
23571 17504 11269 18276 18101 0317 0228 17694 4473
23284 22200 19452 21589 67893 5569 13918 8958 12137
1333 4725 4458 5905 17166 13851 4458 17149 14471 6706
13850 12224 6929 14991 7382 15857 67893 14218 36477
5870 17553 67893 5870 5454 16102 15217 22801 17138
21001 17388 7446 23638 18222 6719 14331 15021 23845
3156 23552 22096 21604 4797 9497 22464 20855 4377
23610 18140 22260 5905 13347 20420 39689 13732 20667
6929 5275 18507 52262 1340 22049 13339 11265 22295
10439 14814 4178 6992 8784 7632 7357 6926 52262 11267
21100 21272 9346 9559 22464 15874 18502 18500 15857
2188 5376 7381 98092 16127 13486 9350 9220 76036 14219
5144 2831 17920 11347 17142 11264 7667 7762 15099 9110
10482 97556 3569 3670

BERNSTORFF.

Charge German Embassy.



TELEGRAM RECEIVED.

FROM 2nd from London # 5747.

"We intend to begin on the first of February unrestricted submarine warfare. We shall endeavor in spite of this to keep the United States of America neutral. In the event of this not succeeding, we make Mexico a proposal of alliance on the following basis: make war together, make peace together, generous financial support and an understanding on our part that Mexico is to reconquer the lost territory in Texas, New Mexico, and Arizona. The settlement in detail is left to you. You will inform the President of the above most secretly as soon as the outbreak of war with the United States of America is certain and add the suggestion that he should, on his own initiative, invite Japan to immediate adherence and at the same time mediate between Japan and ourselves. Please call the President's attention to the fact that the ruthless employment of our submarines now offers the prospect of compelling England in a few months to make peace." Signed, ZIMMERMANN.

(pictures from National Cryptologic Museum)

Example encryption

Example

The message:

Dit wil ik versleutelen!

becomes (with PGP-encrypt, in hexadecimals):

30a4 efde f665 d409 4946 c8b0 d82b 7620
312c bf1b 7f3a 8781 086d 069b b6e0 60a2
94c2 9b27 440c affd 5343 ca47 d0b4 afce 5719

Modern, software-based crypto systems are virtually unbreakable, when:

- well-designed and openly evaluated
- properly used, esp. when keys are kept secret

Crypto system

The en/de-cryption is done with:

$$\begin{array}{c}\text{crypto system}\\ \text{(or encryption scheme, or cipher)}\end{array} \quad = \quad \left\{\begin{array}{l}\text{algorithm}\\ \quad + \\ \text{key (parameter of the algorithm)}\end{array}\right.$$

Kerckhoffs principle

The strength of the crypto system must rely solely on the secrecy of the key; the algorithm must be (assumed to be) public.

Modern interpretation of this principle:

- Algorithm must arise from public scrutiny, eg. via competition (organised by NIST for AES & Keccak/Sha3)
- Non-public algorithms must be distrusted (think of DVD-encryption, GSM, Mifare, . . . , all broken)

Ordering crypto primitives via numbers of keys

number of keys	name	key names	notation
0	hash functions	—	$h(m)$
1	symmetric crypto	shared, secret	$K\{m\}$
2	asymmetric crypto (or public key crypto)	public & private keypair	$\{m\}_K$

We start with symmetric key crypto.

First a few words on . . . words

- Crypto systems transform plaintext to cipher text
- They transform words to words
- Words (aka. strings) are sequences of letters, taken from an alphabet; in practice words are bitstrings

Alphabets

In principle, an alphabet is an arbitrary set A. In this context, the elements $a \in A$ are called letters.

In practice, an alphabet is a finite set $A = \{a_1, \ldots, a_n\}$ of letters. Examples:

- $A = \{0, 1\}$, the alphabet of bits
- $A = \{a, b, c, \ldots, z\}$, the alphabet of lowercase Latin characters;
- $A = \{00, 01, \ldots, 7F\}$ the ASCII alphabet, as hexadecimals; (Recall: $7F = 127 = 2^7 - 1$.)
- The extended ASCII alphabet of 256 characters
- UTF alphabets involve even more characters (depending on version, like UTF-16, UTF-32)

Words

A word over an alphabet A is a finite sequence $w = a_1 a_2 \cdots a_n$ of letters $a_i \in A$. The length of this w is n, obviously.

One writes $A^\star$ for the set of words over A (aka. the Kleene star)

For instance, $\{0,1\}^\star$ is the set of binary words.

We write $|$, or sometimes just a comma, for concatenation of words. Hence:

$$a_1 a_2 \cdots a_n \mid b_1 b_2 \cdots b_m \quad = \quad a_1 a_2 \cdots a_n b_1 b_2 \cdots b_m.$$

On binary words with the same length we write $\oplus$ for bitwise XOR:

$$\begin{aligned} & (a_1 a_2 \cdots a_n) \oplus (b_1 b_2 \cdots b_n) \\ & = (a_1 \textbf{ XOR } b_1)(a_2 \textbf{ XOR } b_2) \cdots (a_n \textbf{ XOR } b_n). \end{aligned}$$

Encryption/decryption are functions from words to words (usually binary).

Symmetric crypto: three basic techniques

Suppose we have a message/word m and wish to (symmetrically) encrypt it to $K\{m\}$, using key K. There are three basic techniques:

(1) Substitution: exchange characters from the alphabet, like in Caesar's cipher.

The key K is: the character substitution/exchange function

(2) Transposition: exchange positions of characters, block-by-block.

The key K is: the position exchange function

(3) One-time-pad: take bitwise XOR with keystream, for binary messages only.

The key K is: the keystream, which must have at least the same length as the message

Ciphers like DES and AES involve repeated combinations of substitution and transposition, depending on a secret key

iCIS | Digital Security
Radboud University

Substitution: exchange of characters

The key is a function $K\colon A \longrightarrow A$, which is bijective: it has an inverse $K^{-1}\colon A \longrightarrow A$, satisfying

$$K^{-1} \circ K = \text{identity} = K \circ K^{-1}.$$

This reversibility is needed for decryption.

This substition function K is extended to words via:

$$m = a_1 a_2 \cdots a_n \quad \text{becomes} \quad K\{m\} = K(a_1)K(a_2)\cdots K(a_n).$$

Substitution: Example

- Caesar's cipher is determined by the substitution function/key

$$C\colon \{a, b, \ldots, z\} \longrightarrow \{a, b, \ldots, z\},$$

given by:

$$C(a) = d, \quad C(b) = e, \quad \ldots \quad C(z) = c.$$

- **Example**:

$$\begin{aligned} C\{\text{ikbengek}\} &= C(\text{i})C(\text{k})C(\text{b})C(\text{e})C(\text{n})C(\text{g})C(\text{e})C(\text{k}) \\ &= \text{lnehqjhn}. \end{aligned}$$

- What is the inverse function $C^{-1}\colon \{a, \ldots, z\} \longrightarrow \{a, \ldots, z\}$? Use it to describe decryption!
- rot13 is a 13-step-shift, which is its own inverse.

Substitution: weakness

The main attack on substitution ciphers is frequency analysis.

- In English, e is the most common letter, followed by t, o, a, n, i, etc. There are frequency tables on the web.
- The most frequently occurring letter in a (substitution) ciphertext corresponds thus most probably to e. You will see this most clearly by doing an exercise.

Transposition: exchange of positions

Transposition via blocks and keys

- For a transposition cipher one first chooses a blocksize N, like $N = 64$, or $N = 128$, or $N = 256$.
- The key K is an exchange of positions within such a block, via a bijective function $K\colon \{1, 2, \ldots, N\} \longrightarrow \{1, 2, \ldots, N\}$.

Encryption of words/messages

- A word m is first chopped-up into blocks of length N, as in:

$$m = \underbrace{a_1 a_2 \cdots a_N}\, \underbrace{b_1 b_2 \cdots b_N}\, \cdots$$

- At the end arbitrary letters (like x) are added to fill the remaining block: such padding must be chosen carefully
- For encryption of m the transposition K is applied per block:

$$K\{m\} = \underbrace{a_{K(1)} a_{K(2)} \cdots a_{K(N)}}\, \underbrace{b_{K(1)} b_{K(2)} \cdots b_{K(N)}}\, \cdots$$

Transposition: Example

Transposition of *ikbengek*

- Choose blocksize, say $N = 3$
- Choose key $K\colon \{1,2,3\} \longrightarrow \{1,2,3\}$ by:

$$K(1) = 3, \quad K(2) = 1, \quad K(3) = 2.$$

- Now encrypt a message block-by-block:

$$\begin{aligned} K\{\text{ikbengek}\} &= K\{\underbrace{\text{ikb}}\ \underbrace{\text{eng}}\ \underbrace{\text{ekx}}\} \\ &= \underbrace{\text{bik}}\ \underbrace{\text{gen}}\ \underbrace{\text{xek}} \\ &= \text{bikgenxek}. \end{aligned}$$

The letter 'x' is add for padding: filling up empty spaces

Columnar transposition example

- The key is an ordinary word, say bart
- The plain text is written under the key, as in:

b	a	r	t
i	k	b	e
n	k	n	e
t	t	e	r
g	e	k	x

- Now read off the cipher text as columns, using the alphabetical order of the key:

kkteintgbnekeerx

- See e.g. http://practicalcryptography.com/ciphers/columnar-transposition-cipher/
- Or many software tools, like GCipher under linux

Transposition: weakness

- First, a transposition does not change the letter frequencies. This is often an indication of transposition
- Next, via a lot of fiddling, frequent 2-letter combinations can be exploited to see the structure of transpositions.

Combining substitution and transposition

Example: Vigenère cipher (from 16th century)

- It applies different (shift) substitution ciphers, depending on the letters of a keyword
- This is called a polyalphabetic cipher
- Broken in 19th century by Babbage, and also by Kasiski

DES and AES

Combine substitution and transposition in several rounds

Vigenère in practice: en/de-cryption by hand

Confederate cipher disc, from the American Civil War (1861-1865)

Such discs are the precursors of rotors, in mechanical crypto devices like the Enigma.

One-time pad (OTP), or Vernam cipher

- Assume a binary message $m = b_1 b_2 \cdots b_n \in \{0,1\}^\star$, so that $b_i \in \{0,1\}$.
- Assume a key of (at least) the same length $K = k_1 k_2 \cdots k_n \in \{0,1\}^\star$.
- For encryption, perform bitwise XOR, as in:

$$K\{m\} \;=\; m \oplus K \;=\; (b_1 \textbf{ XOR } k_1)(b_2 \textbf{ XOR } k_2)\cdots(b_n \textbf{ XOR } k_n).$$

- Decryption is the same as encryption, using basic properties of XOR:

$$\begin{aligned}(b \textbf{ XOR } k) \textbf{ XOR } k &= b \textbf{ XOR } (k \textbf{ XOR } k)\\ &= b \textbf{ XOR } 0\\ &= b.\end{aligned}$$

One-time pad in practice

- OTPs are very secure, in principle, when the key material is truly random
- . . . but OTPs require a lot of key material
  (one can use, say a DVD as shared secret key)
- Running out of key material is a problem, because keys may never be re-used, XOR-ing ciphertexts reveals information:

$$(b \textbf{ XOR } k) \textbf{ XOR } (c \textbf{ XOR } k) \quad = \quad b \textbf{ XOR } c.$$

Key-reuse blunders do happen in practice!

- In the Mifare CLASSIC cipher, part of the key stream is re-used (for parity bits), leaking some information. Also, the "abort" command is sent encrypted, leaking further keystream.

- By Russian spies in the 1940s, who ran out of keys.

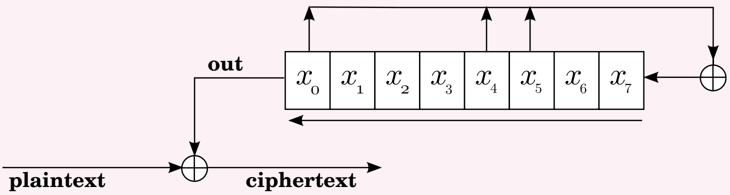
  The US-UK *Venona* project recoverd a lot of traffic, and revealed famous atom spies like Klaus Fuchs

  Even today there are US intelligence officials working on Venona material

Public Venona files in National Cryptologic Museum

One-time pad key stream generator via LFSR

Example LFSR = Linear Feedback Shift Register



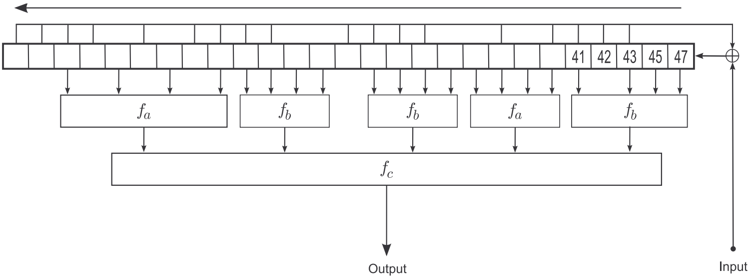
- With every clock cycle the register shifts to the left, and a new value $x_7 = x_0$ **XOR** x_4 **XOR** x_5 is shifted in on the right.
- Illustration: if the current state is 11001010, then the next state is: 10010100
- (“good” LFSRs, with well-chosen feedback, contain all 2^n words)

LFSR usage

- LFSRs are frequently used, since they are fast and easy to implement in cheap hardware
- They can be analysed using basic linear algebra
  (eg. are all possible states actually reached?)
- The Mifare CLASSIC chipcard (from early 1990s) has 2 LFSRs:
  - a 16-bit register for generating (very weak!) "randoms"
  - a 48-bit register (plus "filter" function) for its Crypto1 stream cipher

  This system is completely broken (too few bits, design errors)
- The A5/1 encryption cipher used in GSM works with three different LFSRs. It is also broken.

Mifare CLASSIC LFSR



- The Mifare producer (NXP) tried to prevent publication of this LFSR via a court case (*kort geding*) in July 2008.
- Probably all of you have this LFSR in your pocket!

Symmetric crypto, in practice I

Common implementations (see Wikipedia for details)

- DES from 1977, with 64 bit blocks and 56 bits keys.
  DES is now obsolete, only surviving as triple-DES, in:

$$3\mathrm{DES} = \left(\cdot \xrightarrow[\text{encrypt}]{K_1} \cdot \xrightarrow[\text{decrypt}]{K_2} \cdot \xrightarrow[\text{encrypt}]{K_3} \cdot\right)$$

  Backwards compatibility with (single) DES is achieved via $K_3 = K_2$.
  DES is fast in hardware, relatively slow in software.
- AES from 1997 (elected standard since 2001).
  Standard block length is 128 bit, key lengths are 128 and 256. AES is also fast in software, via software-oriented design.

Different modes of use will be discussed later.

Symmetric crypto, in practice II

In this course

We often use $K\{m\}$ as a *black box* for symmetric encryption, without being very specific about which kind of cipher is used;
in practice we assume the cipher is unbreakable.

Main disadvantages of symmetric crypto

- Large number of keys: if N people wish to communicate pairwise securely, one needs: $\binom{N}{2} = \frac{N(N-1)}{2}$ different secret keys.
  - By using a Trusted Third Party (TTP) it can be reduced to N.
- If Alice and Bob share a key K, and Bob is sloppy and looses K, this affects Alice.

Security protocols are notoriously difficult

Roger Needham:

> *Security protocols are three-line programs that people still manage to get wrong*

Famous example: The Needham-Schroeder mutual authentication protocol (see later) which contained an error that remained undetected for some 20 years

- An attack was found in 1996 by Gavin Lowe, using a model checker
- The attack involved two different interleaved runs of the protocol

What is a security protocol, really?

- A security protocol is a list of communications of the form

$$A \longrightarrow B : m$$

  which is read as: Alices sends message m to Bob.
- The sequence of such messages is intended to achieve a security goal, like confidentiality, integrity, one-way/mutual authentication, non-repudiation, etc.
- At each step of the protocol the beliefs of the participants change: eg. after receiving such return message, Alice knows that Bob has seen ...
- if something goes wrong, the protocol is aborted.

Attacker model

- Implicitly there is an attacker ("Eve") who tries to undermine the goal of the protocol
  - "Dolev-Yao" attacker capabilities are assumed: the attacker can read, delete, copy, rebuild messages
  - but the attacker cannot break encryptions (with unknown keys) or hashes
- Security protocols are important part of the field (and of this course)
  - You must known basic protocol primitives by heart

Protocol basics for confidentiality

Assume Alice and Bob share a secret key K_{AB}, and can do symmetric encryption.

(The index 'AB' in K_{AB} has no mathematical meaning; it suggests notationally that it it is a shared key between A and B.)

Confidential exchange of a message m proceeds via:

$$A \longrightarrow B : K_{AB}\{m\}$$

Is confidentiality achieved? Can Eve read the plaintext m? What are the assumptions involved?

Sequence numbers

- We study abstract security protocols — not actual implementations
- But in such implementations, all messages should be numbered. Hence we should really send:

$$A \longrightarrow B\colon K_{AB}\{i, m\}$$

where $i \in \mathbb{N}$ is a so-called sequence number. It should be incremented with every message (overflow must be handled)
- Sequence numbers are used primarily against loss and replay of messages
  - an additional advantage is that identical message yield different ciphertexts.
- We do not mention sequence numbers explicitly, and assume they are already included implicitly (when needed)

Also integrity?

Question: does $A \longrightarrow B\colon K\{m\}$ also guarantee integrity?

NO! For example,

- Assume the encryption is done via a one-time pad
- An attacker can easily change one bit in the ciphertext
- Possibly the result still makes sense — but has a different meaning

Hence: there is no automatic (cryptographic) test that B can perform in order to verify that the message he receives is the one that was sent by A.

Security in the future

- Recall that the attacker can read (and store) all messages; he can do this over a long time period.
- Hence the strength of the encryption (e.g. keylength) must be chosen appropriately.
  - Tables available online, e.g. `keylength.com`
- Remember *Venona*: if a key ever gets (partially) compromised, old messages may become readable.
  - Some protocols protect against such compromise, and are called forward secure
  - Such forward security is important e.g. in e-voting

Protocol basics for integrity

Suppose Alice and Bob wish to be really sure that what Bob receives is what has been sent by Alice.

They use:

$$A \longrightarrow B \colon m, K_{AB}\{m\}$$

$$\big(\text{or, shorter} \quad A \longrightarrow B \colon m, K_{AB}\{h(m)\}\big)$$

where h is a hash function (see later).

- Is the integrity goal achieved? How? What will Bob detect when Eve replaces the plaintext m by m'?
- What are the assumptions?
- Is confidentiality also achieved?
- Better explanation comes later, in terms of MACs (message authentication codes).

Both confidentiality and integrity

Obvious combinations:

$$A \longrightarrow B\colon K\{m\}, K\{K\{m\}\} \qquad \text{or} \qquad A \longrightarrow B\colon K\{m, K\{m\}\}$$

- This is not wise for one-time pads, since the message is revealed by two successive encryptions.
- One should use two different keys, one for confidentiality, and one for integrity.
- One can then still argue where to put the emphasis of the protection
  - confidentiality first $K_1\{m\}, K_2\{K_1\{m\}\}$
  - integrity first $K_1\{m, K_2\{m\}\}$.

  In general integrity is more important than confidentiality, so it needs to be protected better, like in the second option.

Authentication via shared secret

It is quite common to use a shared secret for authentication

- if I first share a secret with you, then I will henceforth conclude that anyone who can produce this secret is you.
- Example of authentication by "something you know"
- Problem: in every authentication session, the secret is used in the clear.

Something you know examples

- Passwords used by (military) guards to allow access.

  (The use of the secret word *Scheveningen* for this purpose in May 1940 also involved authenticaton "by skill")

- PINs in ATM/payment transactions: one-way authentication between a customer (C) and the bank (B).

  $C \longrightarrow B$: number of card of C (e.g. via magnetic stripe)
  $B \longrightarrow C$: "prove that you are C"
  $C \longrightarrow B$: PIN of C

  This is very weak and has led to widespread skimming

Authentication by challenge-response

It is much better to achieve authentication without using the shared secret *in the clear*.

- **Idea:** send a riddle that can only be solved (efficiently) with the secret key
- It is important that the riddle is fresh upon every use.

  (Which attacker capabilities are used to exploit a non-fresh riddle?)
- Typically this freshness is achieved via a nonce: a number used once.
  - Range of numbers is relevant (say 2^{128})
  - Also randomness / unpredictability

Challenge-response authentication examples

$$\begin{aligned} A &\longrightarrow B\colon A, N_A \\ B &\longrightarrow A\colon K_{AB}\{N_A\} \end{aligned}$$

(N_A is a fresh nonce)

At this stage A knows she is talking to B, because only B, so she assumes, posseses the shared key K_{AB} and can compute $K_{AB}\{N_A\}$.

There are several inessential variations:

$$\begin{aligned} A &\longrightarrow B\colon A, K_{AB}\{N_A\} \\ B &\longrightarrow A\colon N_A \end{aligned}$$

Or:

$$\begin{aligned} A &\longrightarrow B\colon A, K_{AB}\{N_A\} \\ B &\longrightarrow A\colon K_{AB}\{N_A + 1\} \end{aligned}$$

NOTE: authentication key must be different from encryption key!

Two-way authentication options

Naive two-way, combined version:

$$\begin{aligned} A &\longrightarrow B\colon\ A, N_A \\ B &\longrightarrow A\colon\ K_{AB}\{N_A\}, N_B \\ A &\longrightarrow B\colon\ K_{AB}\{N_B\} \end{aligned}$$

Or:

$$\begin{aligned} A &\longrightarrow B\colon\ K_{AB}\{N_A, \text{timestamp}\} \\ B &\longrightarrow A\colon\ N_A \end{aligned}$$

Nonces, timestamps, sequence numbers

All of these alternatives for freshness have pros and cons:

- Random nonces require a secure random number generator
  - if there is one thing that computers are *not* good at, it is generating random numbers
- Timestamps require reliable/secure/synchronised clocks
- sequence numbers are predictable (so should be used more carefully) and can wrap around. They are a special form of nonce.

Reflection attack (*Koekje van eigen deeg*)

A reflection attack is possible for the "naive" two-way protocol by mixing two sessions (written as 'a' and 'b'):

Protocol

$$A \longrightarrow B\colon A, N_A$$
$$B \longrightarrow A\colon K_{AB}\{N_A\}, N_B$$
$$A \longrightarrow B\colon K_{AB}\{N_B\}$$

Attack

$$(a)\quad E \longrightarrow B\colon A, N_A$$
$$(a)\quad B \longrightarrow E\colon K_{AB}\{N_A\}, N_B$$
$$(b)\quad E \longrightarrow B\colon A, N_B$$
$$(b)\quad B \longrightarrow E\colon K_{AB}\{N_B\}, N$$
$$(a)\quad E \longrightarrow B\colon K_{AB}\{N_B\}$$

In the end B thinks that he is talking to A, but in reality he is talking to the intruder Eve (E). Note that Eve can take the initiative for this attack.

Attack prevention

A solution is to this attack is to use different keys for the two directions, as in:

$$
\begin{aligned}
A &\longrightarrow B\colon A, N_A \\
B &\longrightarrow A\colon K_{AB}\{N_A\}, N_B \\
A &\longrightarrow B\colon K_{BA}\{N_B\}
\end{aligned}
$$

A more economical solution is to use *domain separation*:

- code challenge *from* Alice differently than challenge *for* Alice
- e.g., challenge for Alice starts with bit 1, from Alice with bit 0

This gives

$$
\begin{aligned}
A &\longrightarrow B\colon A, N_A \\
B &\longrightarrow A\colon K_{AB}\{1\|N_A\}, N_B \\
A &\longrightarrow B\colon K_{AB}\{0\|N_B\}
\end{aligned}
$$

Initiator must authenticate first

Another solution is to let the initiator authenticate itself first, as in:

$A \longrightarrow B$: "Hi, I'm A; let's talk"
$B \longrightarrow A$: "Sure, but first increment $K_{AB}\{N_B\}$"
$A \longrightarrow B$: $K_{AB}\{N_B + 1\}, K_{AB}\{N_A\}$
$B \longrightarrow A$: "Wow, you're really A; this shows I'm B: $K_{AB}\{N_A - 1\}$"
$A \longrightarrow B$: "Great; we now also have a session key K"
(namely $K = N_A \oplus N_B$)

- This protocol has additional benefit that it sets up a session key determined from both sides.

Man-in-the-middle attack

All presented protocols are vulnerable to a man in the middle attack:

Protocol	**Attack**
$A \longrightarrow B: A, N_A$	$A \longrightarrow E: A, N_A$
$B \longrightarrow A: K_{AB}\{N_A\}, N_B$	$E \longrightarrow B: A, N_A$
$A \longrightarrow B: K_{AB}\{N_B\}$	$B \longrightarrow E: K_{AB}\{N_A\}, N_B$
	$E \longrightarrow A: K_{AB}\{N_A\}, N_B$
	$A \longrightarrow E: K_{AB}\{N_B\}$
	$E \longrightarrow B: K_{AB}\{N_B\}$

- As a result, A thinks that E is B, and B thinks that E is A.
- Note that Eve does not take the initiative, but waits until she can intercept an initiative of A.

  (any router performs a relay attack, in a strict sense)

More on man-in-the-middle (MITM) attacks

- Car key relay attack
- Serious attack scenario in internet banking
  - Often occurring as "man-in-the-browser" attack
  - Attacker manipulates what is shown in the browser, and sends false date to the bank (via usual encrypted connection)
- Forged certificates obtained in DigiNotar (2011) attack were probably used by Iran to do a man-in-the-middle attack on local, Iranian Gmail users
  - by setting up a false intermediate Gmail site
  - the NSA is now accused of similar attacks, against Petrobas
- Nice story, but not historically correct: Mig-in-the-middle see Ross Anderson's book *Security Engineering* (freely available, google it)

Diversified keys, AKA Key Derivation

Recall the key management problem of secret key crypto:

- n interacting users require $\frac{n(n-1)}{2}$ keys
- In payment/transport smart cards: n cards and m terminals: nm keys

Solution: Diversified keys: secret key K_C of card C from its identity, using some (super secret) masterkey K_M: $K_C = K_M\{\mathsf{Id}_C\}$.

The card can then authenticate itself to a terminal T via:

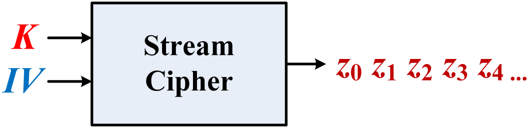
$$
\begin{array}{ll}
C \longrightarrow T : \mathsf{Id}_C & (T \text{ checks } \mathsf{Id}_C \text{ is in range, and computes } K_C) \\
T \longrightarrow C : N & \\
C \longrightarrow T : K_C\{N\}. &
\end{array}
$$

- Used in OV-chip, PIN transactions etc.
- Offline: K_M in all terminals; online K_M only in central system
- Multi-level: session keys K_S derived from K_C and card transaction counter: $K_S = K_C\{\mathsf{Nr}_C\}$

Active attack overview

- Replay attack
  - eavesdropped e.g., login name + password, is sent again
  - countermeasure: include nonce, checked by verifier
- Reflection attack
  - typical attack on challenge-response protocols
  - data from one session is re-used in another session
  - countermeasure: include ID info; key or domain separation
- Man-in-the-middle (MITM) attack
  - *passive* MITM version, without modification: relay attack
  - *active* MITM version involves re-encryption
  - countermeasure: protecting keys, out-of-band methods
- Lunch-break attack
  - typical for physical access: car keys, access badge
  - attacker gets *responses* from prover and uses them later
  - countermeasure: unpredictable challenge from verifier (freshness)

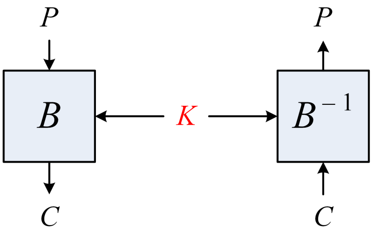
Stream ciphers



$$K\{IV\} = Z = z_0 z_1 z_2 \cdots$$

- Generates keystream bits z_t from
  - K: secret, typically 128 or 256 bits
  - IV: initial value, for generating multiple keystreams per key
- z_t can be a bit or a sequences of bits, e.g. a 32-bit word
- Z typically used for one-time pad encryption: $C = M \oplus Z$
- Sometimes also for authentication: response $= K\{N\}$

Block ciphers

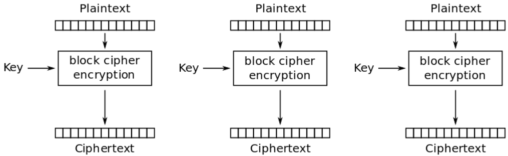


- Function B mapping b-bit string P to b-bit string C
  - depends on a key $C = K\{P\}$
  - must be invertible ("*permutation*"): $P = K^{-1}\{C\}$
- Dimensions: block length b and key length
- Examples:
  - DES: 64-bit block, 56-bit key
  - AES: 128-bit block, keys of 128, 192 or 256 bits

Block cipher modes for encryption

- DES can encipher 8-byte messages, AES of 16-byte messages
  - what about longer and shorter messages?
  - what about real-time datastreams: audio or video?
  - two approaches: block encryption and stream encryption
- Block encryption modes
  - split the message in blocks
  - after padding last *incomplete* block if needed
  - apply block cipher to blocks *in some way*
- Stream encryption modes
  - build a stream cipher with a block cipher as building block

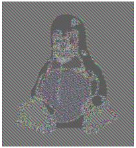
Electronic CodeBook Mode (ECB)



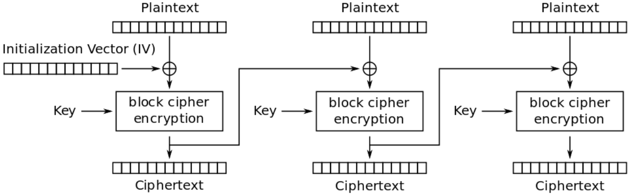
Electronic Codebook (ECB) mode encryption

- Simplest possible way to encrypt with a block cipher
- Advantage: parallelizable
- Limitation: equal plaintext blocks $\rightarrow$ equal ciphertext blocks:

under ECB gives



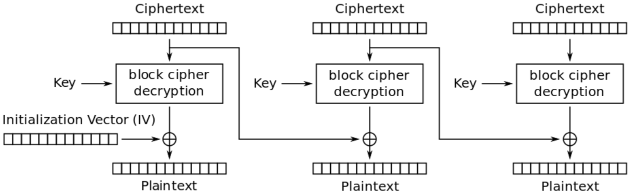
Cipher Block Chaining mode (CBC)



Cipher Block Chaining (CBC) mode encryption

- *ECB with plaintext block randomized by previous ciphertext block*
- First plaintext block randomized with Initial Value (*IV*)
- Solves information leakage in ECB (partially):
  - equal plaintext blocks do not lead to equal ciphertext blocks
  - requires randomly generating and transferring *IV*
  - this is in practice often neglected, e.g. *IV* fixed to 0

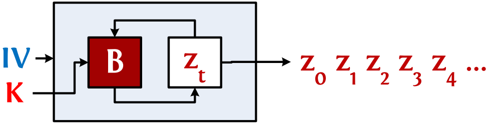
Cipher Block Chaining mode (cont'd)



Cipher Block Chaining (CBC) mode decryption

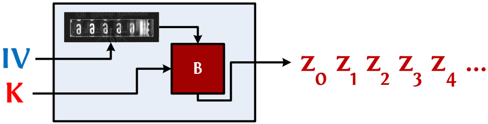
- Properties of CBC
  - encryption strictly serial, decryption can be parallel
  - *IV* must be managed and transferred

Stream encryption: Output FeedBack mode (OFB)



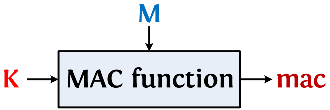
- Stream cipher:
  - $z_0 = K\{IV\}$
  - $z_1 = K\{z_0\}$
  - $z_i = K\{z_{i-1}\}$
  - key stream: $K\{IV\}, K\{K\{IV\}\}, K\{K\{K\{IV\}\}\}, \cdots$
- Properties
  - strictly serial
  - no need for K^{-1} (valid for all stream encryption)

Stream encryption: Counter mode



- Stream cipher:
  - $z_0 = K\{IV\}$
  - $z_1 = K\{IV + 1\}$
  - $z_i = K\{IV + i\}$
  - key stream: $K\{IV\}, K\{IV + 1\}, K\{IV + 2\}, \cdots$
- Properties
  - fully parallelizable
  - most used block cipher mode
  - good *IV* management is critical for security

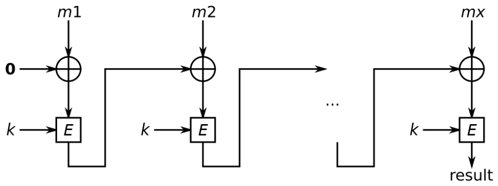
Message authentication code (MAC) functions



$$\mathrm{mac} = K_{\mathrm{mac}}\{M\}$$

- MAC: cryptographic checksum
  - input: key K_{mac} and arbitrary-length message M
  - output: ℓ-bit mac or tag T with ℓ some length
- Applications:
  - message integrity: append mac to message
  - authentication protocols: compute mac over challenge

Cipher Block Chaining MAC mode (CBC-MAC)



- Observation: in CBC encryption Ci depends on $m1$ to mi
- Idea:
  - Apply CBC encryption to (padded) message
  - take mac equal to last ciphertext block
  - throw away other blocks (essential for security)
- This is the basis for most block-cipher based mac functions

Protection of sensitive data on e-passports

- Since 2006 NL passport contain contactless chip with name, date-of-birth, BSN etc. plus a digital photograph
- Since 2009 also fingerprints
- Main purpose: combat look-alike fraud, i.e., using someone else's passport
- Access to data on chip is delicate matter:
  - should be impossible for "someone next to you in the bus"
  - should require consent of passport holder
  - sensitive data (fingerprints) only for countries that are "friends" (currently, none)
- Chosen approach: accessibility of
  - picture, name etc. after user consent, via *Basic Access Control*
  - fingerprints only after terminal authentication: *Extended Access Control*

Protection of e-passport data: consent

- Passports contain a (thick) plastic page, with embedded:
  - photo of cardholder + authenticity marks
  - chip + antenna
  - at bottom: 2-line Machine Readable Zone (MRZ) containing, date-of-issuance, BSN, document nr. etc.
- Essence of Basic Access Control (BAC):
  - cryptographic key for chip communication, derived from MRZ
  - standardized by International Civil Aviation Organization (ICAO)
  - currently rolled out on airports worldwide at border control
- Idea of consent: when you hand over your e-passport, the receiver can read the MRZ and communicate with the chip

BAC keys for e-passports

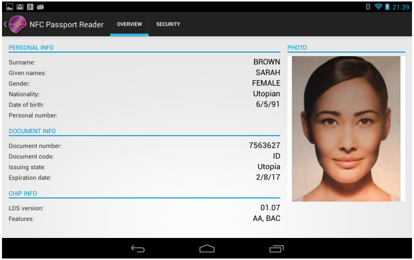
- Two 3DES keys are derived from MRZ:
  - K_{enc}, for confidentiality
  - K_{mac}, for integrity

  These keys are fixed, but are used to obtain session keys to protect the communication between card and reader

- Relevant MRZ-input for these 2 keys
  - passport nr.
  - birth date
  - expiry date

- In early approaches the MRZ was too predictable, e.g. because document numbers were sequential

Read your own passport, on Android with NFC



- Requires MRZ as input (via optical character recognition (OCR)), for BAC keys
- This one is developed by former group member Martijn Oostdijk, now at InnoValor; there are many others.

BAC protocol for e-passports

Assume a card reader Rdr has derived the keys K_{enc} and K_{mac} of a passport PsP

$$PsP \xrightarrow[\text{(8 byte nonce)}]{N_P} Rdr$$

$$PsP \xleftarrow[\text{where } m = (N_P, N_R, K_R)]{K_{\text{enc}}\{m\},\ K_{\text{mac}}\{m\}} Rdr$$

$$PsP \xrightarrow[\text{where } n = (N_P, N_R, K_P)]{K_{\text{enc}}\{n\},\ K_{\text{mac}}\{n\}} Rdr$$

K_P and K_R are contributions from both sides to a session key, as in: $K = K_P \oplus K_R$.

Two passport vulnerabilities

- These are "level below" attacks, using implementation details
- They exploit differences in how different smart cards react to different events—without knowing secret keys
  - Not all countries have the same card producers, so low level (hardware) differences are likely
  - The international standards (from ICAO) do not precisely specify how to react to each possible failure
- Sources are research papers (on the web):

(1) [RMP'08] H. Richter, W. Mostowski, and E. Poll, *Fingerprinting Passports*, NLUUG, 2008.

(2) [CS'10] T. Chothia and V. Smirnov, *A Traceability Attack Against e-Passports*, Financial Crypto, 2010.

Fingerprinting e-passports [RMP'08]

Idea: send deliberately wrong (out-of-protocol) messages and inspect the resulting byte-sequences for different countries:

	Commands						
	44	**82**	**84**	**88**	**A4**	**B0**	**B1**
	Rehab. CHV	Ext. Auth.	Get Chall.	Int. Auth.	Select File	Read Binary	Read Binary
Australian	6982	6985	6700	6700	9000	6700	6700
Belgian	—	6E00	—	6700	6A86	6986	6700
Dutch	—	6700	6700	6982	6A86	6982	6982
French	6982	6F00	6F00	6F00	6F00	6F00	6F00
German	—	6700	6700	—	6700	6700	—
Greek	6982	63C0	6700	6982	9000	6986	6700
Italian	—	6700	—	—	—	—	—
Polish	6982	6700	6700	6700	9000	6700	—
Swedish	6982	6700	6700	—	9000	6700	—
Spanish	—	6700	6700	—	6700	6700	—

Hence, passports from different countries can be distinguished externally, via their reactions. Is this a problem?

Excursion on timing attacks

- Suppose you write a software module for checking a PIN
- A stupid way is to check the digits one-by-one, after the whole PIN has been entered, giving an error message as soon as a digit is wrong.
- This approach is vulnerable to a timing attack:
  - accurately measure the time that it takes to get an error message
  - you will see timing differences between an error in the n-th digit and in the $n+1$-th digit.
  - hence you can try to find the PIN digit-by-digit.
- Such timing attacks occur in practice, and can be quite subtle
  - For e-passports they were found in [CS'10].
  - They exist(ed) in many implementations
  - Including the open source version (from Nijmegen), now fixed, see: `http://jmrtd.org`

Timing attack on the e-passport [CS'10]

- Recall the second message from the BAC protocol:

$$PsP \xleftarrow[\text{where } m = (N_P, N_R, K_R)]{K_{\text{enc}}\{m\},\ K_{\text{mac}}\{m\}} Rdr$$

- Many implementations do the following consecutively:
  (1) integrity/MAC check: decrypt, recompute mac and check
  (2) nonce check: compare incoming N_P to last-generated nonce
- An error in the first integrity-check will thus appear sooner than an error in the second nonce-check
- (Some implementations, like the French one, even give different error messages)

How to exploit the e-passport timing attack

- Suppose I can eavesdrop an entire session for the e-passport of, say, Wilders
  - this means that I have a pair $K_{enc}\{m\}$, $K_{mac}\{m\}$
  - with secret keys K_{enc} and K_{mac} from his e-passport
- Now I can check for an arbitrary passport if it is the one from Wilders or not!
  - ask a passport for a nonce
  - replay the above message pair, and time the response
  - the nonce-check will always fail, but:
    - if the MAC-check succeeds, the passport is from Wilders!
    - if the MAC-check fails, it is not
- In order to exploit this in a physical attack, you need to get pretty close to Wilders
  - in that case you also have other attack options
  - but note: the timing attack can be fully automated

Intermediate conclusion

Security in practice is subtle and bloody difficult!