Computer Security: Secret Key Crypto

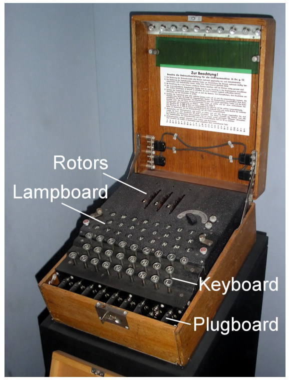
B. Jacobs and J. Daemen
Institute for Computing and Information Sciences – Digital Security
Radboud University Nijmegen
Version: fall 2017

Outline

- Crypto intro
- Symmetric crypto
- Modern symmetric crypto: block ciphers and stream ciphers
  - Block ciphers
  - Stream ciphers
- Achieving security goals with symmetric crypto
  - Confidentiality
  - Integrity
  - Authentication, of entities
  - Some interesting attacks
- Block cipher modes of use
  - Block cipher modes for encryption
  - Block cipher MAC modes
- Management of symmetric keys
- e-Passport example

Old cryptographic systems

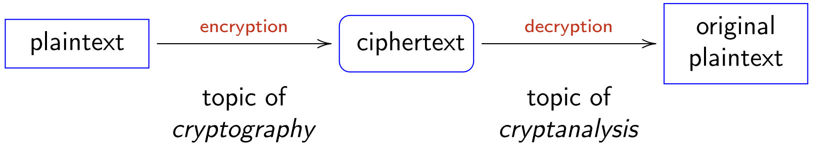
Scytala from Sparta



German Enigma from WWII

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

Situation & terminology



Officially,

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

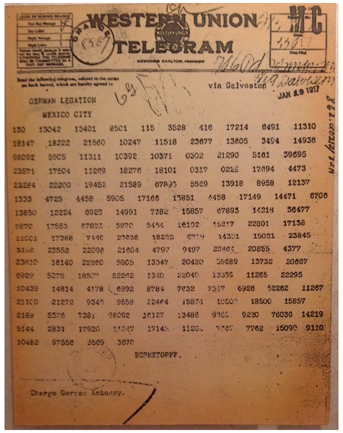
This is the official, somewhat outdated terminology. But often “crypto” or “cryptography” is used for “cryptology”.

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

via Galveston

JAN 19 1917

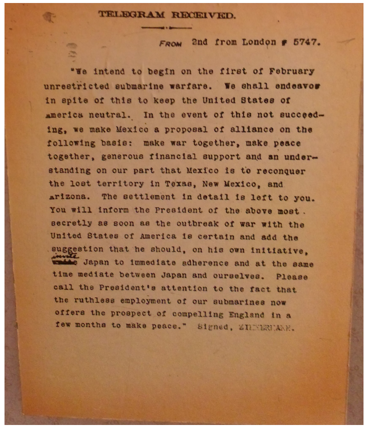
GERMAN LEGATION

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
22284  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, ~~[illegible]~~ 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:

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

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/SHA-3)
- 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

Substitution: exchange of characters from an alphabet A

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.

Polyalphabetic cipher: Vigenère (16th century)

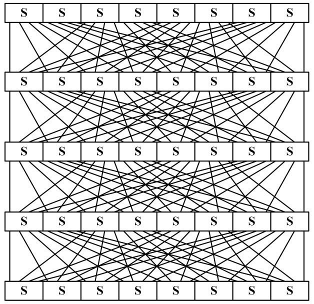
- Different substitutions (shifts), depending on letters of a keyword
- Broken in 19th century by Babbage, and also by Kasiski

Used in confederate cipher disc, from American Civil War (1861-1865):

It's a precursor of rotors in mechanical crypto devices like Enigma (WW II)

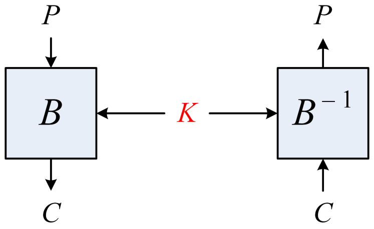
Product ciphers [Claude Shannon, 1949]

SP-network or SPN is used as shorthand name for alternating substitutions and transpositions (permutations), as in:



This lead to modern cryptography: block ciphers

Block ciphers: mini encryption schemes



- Enciphers plaintexts and deciphers ciphertexts under secret key K
  - looks like random permutation if key is unknown
  - fixed-length: plaintexts and ciphertexts must be N bits long
- Relevant dimensions: block length N and key length
- Built as SPN or more sophisticated variants
- There are many block ciphers, but two important ones: DES and AES
- In this course we ignore the internals and take a more abstract perspective

Data encryption standard (DES)

- Block cipher with block length $N = 64$ bits and 56-bit key
- US Federal Information Processing Standard (FIPS) 46 (1977)
  - standard for US government designed by IBM, and NSA
  - NSA-involvement was/is controversial, but backdoors were never found
  - massively adopted by banks and industry worldwide
  - dominated symmetric crypto for more than 20 years
  - targeted at hardware, relatively slow in software
- Key length too short to resist exhaustive key search by end of 80's
- The patch: triple-DES, in:

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

Backwards compatibility with (single) DES is achieved via $K_1 = K_2 = K_3$.

Advanced Encryption Standard (AES)

- Block cipher with block length $N = 128$ and variable key lengths: $128, 192$ and 256
- US Federal Information Processing Standard (FIPS) 197 (2001)
  - result of an open competition (1997-2000)
  - designed by Joan Daemen and Vincent Rijmen: aka. "Rijndael"
- AES has replaced (Triple-)DES in most applications
  - designed with software performance in mind
  - dedicated instructions on many CPUs, e.g. Intel
  - massively deployed worldwide

Using block ciphers

- A block cipher can only process blocks of fixed N-bit length
- Encrypting shorter messages, e.g. PINs, requires padding
  - this means: appending bits to obtain a length of N bits
  - must be invertible; works for fixed length messages
- Encrypting longer message m
  - naive way: split m in N bit blocks and encrypt separately
  - if last block is shorter than N bits: first apply padding
- There are more sophisticated ways to do this: modes of use
  - see later in this course

One-time pad (OTP) encryption aka. Vernam cipher

Bob encrypts message M with key K via the bitwise addition $\oplus =$ XOR:

$$\begin{array}{rccccccccccccccccl}
M = & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & \\
K = & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & \bigoplus \\
\hline
C = & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 &
\end{array}$$

Alice decrypts via XOR with the same key!

$$\begin{array}{rccccccccccccccccl}
C = & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & \\
K = & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & \bigoplus \\
\hline
M = & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 &
\end{array}$$

Shannon proved that this one-time pad encryption offers "perfect" secrecy

Why does One-time pad (OTP) work?

- Let b be a single message bit, and k a key bit
- OTP-encryption of b gives $b \oplus k$
- OTP-decryption of this ciphertext is $(b \oplus k) \oplus k$.
- This yields the original bit b because of basic properties of XOR:

$$0 \oplus 0 = 0 \quad \text{and} \quad 1 \oplus 1 = 0 \qquad \text{so} \qquad k \oplus k = 0$$

And:

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

One-time pad: limitations

- Shannon's proof requires that:
  - The key K is completely random for the adversary
  - Every bit of K is used to encipher a single message bit
- Consequences of key re-use are catastrophic:
  - Say we encipher messages M_1 and M_2 with the same key K. So

    $$C_1 = M_1 \oplus K \text{ and } C_2 = M_2 \oplus K$$

  - An adversary can compute bitwise difference between M_1 and M_2 from ciphertexts:

    $$C_1 \oplus C_2 = M_1 \oplus K \oplus M_2 \oplus K = M_1 \oplus M_2$$

  - (partial) knowledge of M_1 gives (partial) knowledge of M_2
- Impractical aspect: encryption of n bits requires n bits of key
  - Alice and Bob must pre-share a key as long as their expected communication
  - e.g. disk encryption requires an equally-sized disk of key material

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.

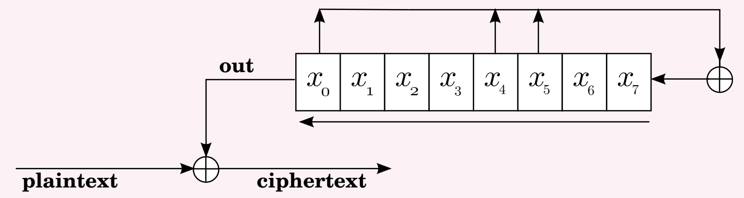
  The US-UK *Venona* project recoverd a lot of traffic, and revealed famous atom spies like Klaus Fuchs

  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



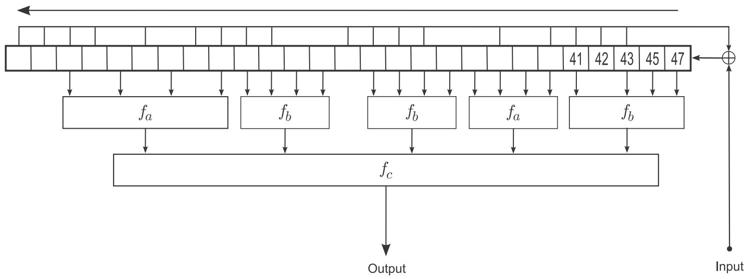
- With every clock cycle the register shifts to the left, and a new value $x_7 = x_0$ **XOR** x_4 **XOR** x_5 is shifted in on the right.
- Illustration: if the current state is 11001010, then the next state is: 10010100
- ("good" LFSRs, with well-chosen feedback, contain all 2^n words)

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!

LFSR's generalised: stream encryption I

LFSR's combine two constructions:

- generation of a keystream — of arbitrary length
- XOR of this keystream and plaintext yields ciphertext

This combination is called stream encryption or a stream cipher.

The keystream Z is described abstractly as $Z = F(K, D)$

- F is called a *stream cipher*. Its inputs are:
  - K: a short secret key
  - D: a short diversifier
- D allows generating multiple different keystreams per key K

The output stream $Z = F(K, D)$ can be of arbitrary length.

In stream encryption we instantiate encryption $K\{M\}$ as $M \oplus F(K, D)$.

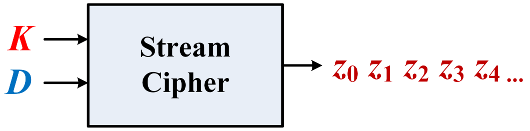
(We often leave the diversifier D implicit, unless it plays an explicit role)

LFSR's generalised: stream encryption II

Example (LFSR, in this abstract format)

- the key K is the initial content of the register
- there is no diversifier — alternatively, $D = 1$
- the function F captures the register cycles

Stream cipher schematic and security


K

D

Stream Cipher

$z_0\ z_1\ z_2\ z_3\ z_4 \ldots$

Informal definition of stream cipher security

A stream cipher $Z = F(K, D)$ is secure if its output Z looks random for someone that does not know the secret key K — whereas the diversifier D is public.

The mapping $F(K, \cdot)$ behaves like a "pseudorandom function" (PRF).

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 sequence of data exchanges such as:

  $$A \longrightarrow B : m$$

  this means: Alice sends message m to Bob.
- In between the data exchanges, Alice or Bob perform operations
- The goal is to achieve a security goal, like confidentiality and/or integrity of the communication, entity authentication, non-repudiation, etc.
- At each step of the protocol the beliefs of the participants may change: eg. after receiving such return message, Alice knows that Bob has seen . . .
- in some cases, Alice or Bob may decide to abort the protocol due to failure of a cryptographic check

Attacker model

- Implicitly there is an attacker ("Eve") who tries to undermine the goal of the protocol
- We assume a very powerful attacker model: "Dolev-Yao"
  - attacker can read, delete, copy, rebuild messages
  - attacker cannot break the crypto such as deciphering cryptograms or finding hash collisions or pre-images
- Security protocols are important part of the field (and of this course)
  - You must known basic protocol primitives by heart

In our protocol descriptions

We often use $K\{m\}$ as a *black box* for (assumed secure) symmetric encryption, without being specific about the cipher. The result is called a cryptogram

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?

Confidentiality of multiple messages

Assume now Alice sends multiple messages $m_0, m_1, \ldots$ to Bob and they use a stream cipher $F(K_{AB}, \cdot)$

The keystream must be different for each message.

This can be achieved with a sequence number. For the i-th message:

$$A \longrightarrow B \colon c_i = m_i \oplus F(K_{AB}, i), i$$

Here $i \in \mathbb{N}$ is the sequence number:

- Alice shall increment it for each message
- If messages can be lost or arrive out of order, i sent along can be used for recovery
- If encoded on n bits, only 2^n messages can be encrypted. Sequence number may never overflow.

Also integrity?

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

NO!

For example,

- Assume a stream cipher is used: $c = m \oplus F(K_{AB}, i)$
- Attacker can flip plaintext bit $m[i]$ by flipping $c[i]$
- Possibly the result still makes sense — but has different meaning

Integrity from encryption? Counterexample

Example: PINPAS amount at coffee shop cash register

- Alice: PINPAS payment terminal in coffee shop
- Bob: Bank
- Eve: coffee shop owner

- Message: transaction request to Bank, containing amount
- Message starts with 32-bit transaction amount in # Eurocents
- Say amount is: 34.50 Euro: gives (in Hex) `00 00 0D 7A`
- Eve changes ciphertext by flipping 16th bit
- Bank decrypts ciphertext and obtains plaintext: `00 01 0D 75`
- This translates to amount $3450 + 2^{16}$ Eurocents ≈ 690 Euro

Lesson: Encryption does not automatically protect integrity

Secrecy in the long term

- Remember: Dolev-Yao attacker can store all messages over any time period
- Sometimes we care about long-term secrecy: e-voting, state secrets, …
- Strength of the encryption must be sufficient
  - for key lengths, guidance can be found at, e.g., `keylength.com`
- Consequences of key compromise must be limited
  - 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
  - Principle: compromise of keys from device does not allow to decipher old messages

Protocol basics for integrity

- Goal: Alice and Bob wish to be certain that what Bob receives is what Alice sent

- Recall the stream cipher $Z = F(K, D)$
- Trick: we will use the message m as diversifier D

Alice sends a message to Bob consisting of two parts:

$$A \longrightarrow B:\ m, T \qquad \text{where} \qquad T = F(K_{AB}, m)$$

- Is integrity achieved? What will Bob detect when Eve replaces the plaintext m by m'?
- What are the assumptions?
- Is confidentiality also achieved?
- This T is called a message authentication code (MAC), or also tag.

MAC functions

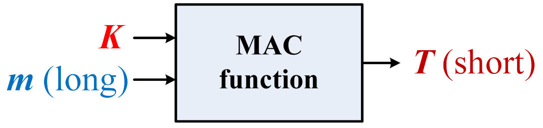
- Problem with using stream cipher
  - in real-world stream ciphers D has fixed length, e.g. 96 bits
  - message m can be much longer
  - real-world stream cipher generate long outputs Z
  - for MAC T we only use first 128 (or so) bits

- Solution: we build functions similar to stream ciphers but
  - with input m of arbitrary length
  - with short output T:

$$T = F(K, m)$$

  - This is called a MAC function

MAC function schematic and security



Informal definition of MAC function security

A MAC function $F(K, m) = T$ is secure if its output T looks random for someone that does not know the secret key K — whereas the message m is public.
The mapping $F(K, \cdot)$ behaves like a pseudorandom function (PRF).

Back to protocol for integrity: replay

- Possible attack: Eve copies valid message and later sends it again
  - the message does come from Alice
  - but only the first time
- Example: back to our coffee shop example
  - Each replay of payment message brings 34.50 Euro extra income
- We call this a replay attack
- Protection: include sequence counter in MAC computation ($i \| m$ means concatenation of i and m)

$$A \longrightarrow B\colon\ i, m, T \qquad \text{where} \qquad T = F(K_{AB}, i\|m)$$

- Receiver Bob must check that sequence number does not repeat

Recall: for confidentiality sender Alice must ensure i does not repeat

Both confidentiality and integrity

Good practice in computer security: key separation

Use different cryptographic keys for different goals

Assume we have two keys: one for confidentiality and one for integrity:

$$K_{\text{conf}} \qquad \text{and} \qquad K_{\text{int}}$$

Then we achieve both goals in a message transfer from Alice to Bob via:

$$A \longrightarrow B\colon\ i, c, T \qquad \text{with} \qquad c_i = m_i \oplus F(K_{\text{conf}}, i) \qquad \text{and} \qquad T = F(K_{\text{int}}, c_i\|i)$$

Here i is the sequence counter for diversifying keystreams and anti-replay

Upon receipt, Bob checks the MAC T and only proceeds to decrypt c_i if it is correct.

Authentication via shared secret: password/PIN

It is quite common to use a shared secret for authentication

- if I first share a special secret exclusively with you, then I will henceforth conclude that anyone who can produce this secret is you
- Example of authentication by "something you know"
- Problems:
  - you present your secret often to (some) other party that may be impersonated, called phishing
  - other party may be hacked: password file compromise (possibly protected by hashing, see later)

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 involved authentication "by skill" and not by secret)

- PINs in ATM/payment transactions with magnetic stripe cards: the bank B authenticates the customer C.

$$\begin{aligned} &C \longrightarrow B\colon \text{number of card of } C && \text{(via magnetic stripe)} \\ &B \longrightarrow C\colon \text{“prove that you are } C\text{”} \\ &C \longrightarrow B\colon \text{PIN of } C \end{aligned}$$

  This is very weak and has led to widespread skimming

- Passwords authentication in about every WWW site

Authentication with crypto: challenge-response

It is much better to achieve authentication without sending the shared secret but rather *using* it

- **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 an unpredictable number:
  - Range of numbers is relevant (say 2^{128})

Challenge-response authentication examples

The riddle can be: encipher the challenge with K_{AB}. This gives:

$$\begin{array}{l} A \longrightarrow B\colon A, N_A \\ B \longrightarrow A\colon K_{AB}\{N_A\} \end{array} \qquad (N_A \text{ is an unpredictable nonce})$$

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 $K_{AB}\{N_A\}$.

Some inessential variations:

$$\begin{array}{l} A \longrightarrow B\colon A, K_{AB}\{N_A\} \\ B \longrightarrow A\colon N_A \end{array}$$

Or:

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

Beware: these examples require that $K_{AB}\{\cdot\}$ is a block cipher

Challenge-response with a MAC function

Riddle: the generation of MAC over the challenge.

This gives:

$$\begin{array}{l} A \longrightarrow B\colon A, N_A \\ B \longrightarrow A\colon F(K_{AB}, N_A) \end{array}$$

What does Alice do upon receiving the reply?

Two-way authentication options

Naive two-way, combined version:

$$\begin{array}{l} A \longrightarrow B\colon A, N_A \\ B \longrightarrow A\colon F(K_{AB}, N_A), N_B \\ A \longrightarrow B\colon F(K_{AB}, N_B) \end{array}$$

Another philosophy, making use of timestamps:

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

Nonces, random numbers and timestamps

There are several alternatives for freshness and protection against replay.

Definitions:

Sequence number number that increments with each message, step

Nonce *number used once*

Random number number that is (believed to be) unpredictable

Pseudo-random number number unpredictable by someone not knowing some secret

Timestamp an encoding of date and time (e.g. up to the ms.)

Nonces, random numbers and timestamps, II

Some relations:

- Sequence number is special case of nonce, at least if it does not roll over
- Timestamp is special case of nonce, at least if the clock is reliable
- Random number is special case of a nonce, at least if
  - it is long enough for number of random numbers used per key
  - it has a sufficiently uniform distribution
- A pseudorandom number can be a nonce (see later)

Essentially two orthogonal properties:

nonce one-time use, requires keeping state

random unpredictability, requires some magic

Use a nonce or a random number?

- Stream encryption of multiple messages: nonce
- Protection against replay attack: nonce
- Ensuring freshness of messages: random
- Ensuring freshness of messages without using randomness:
  - timestamps, assuming reliable clocks synchronised between sender and receiver
  - very hard to get right
- Generation of cryptographic keys: random

On (pseudo-)random generation

- Random generation is not easy to do
- This is where many systems fail

There are two very different types of generators:

- True random number generators (TRNG)
- Pseudo-random number generators (PRNG)

Sometimes they are combined in a hybrid RNG

True random number generators

- Based on some physically hard to model process
- Examples:
  - throwing dice, coin flip, ...
  - user input: mouse or keyboard activity
  - thermal noise
  - radioactive decay
- physics instead of mathematics!
- hard to build
- even harder to test

Pseudo-random number generator

- There are cryptographic ones and non-cryptographic ones
  - non-crypto example: middle-square method (see wikipedia)
  - only cryptographic ones are useful in our protocols
- Very similar to a stream cipher
  - output should be unpredictable if some secret is unknown
  - keeps state (like LFSR)
  - generates pseudorandom output while updating the state
- Some differences:
  - The secret is not called key, but *seed*
  - New seed material can be injected at any time
- seed compromise is fatal
- Often forward secrecy is built in:
  - knowledge of full state does not allow reconstructing previously generated random output
  - relevant if used for generating cryptographic keys

Pseudo-random number generators (cont'd)

Unlike TRNG, PRNG is cryptographic primitive

- If badly designed, it may be weak
- It may be broken, or worse ...
- ...random generator in Android weakened by RSA
- ...NIST standard and RSA Labs default DUAL_EC_DRBG backdoored by NSA

Hybrid RNG: PRNG seeded with output of TRNG

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

$$\begin{array}{l} A \longrightarrow B\colon A, N_A \\ B \longrightarrow A\colon F(K_{AB}, N_A), N_B \\ A \longrightarrow B\colon F(K_{AB}, N_B) \end{array}$$

Attack

$$\begin{array}{ll} (a) & E \longrightarrow B\colon A, N_A \\ (a) & B \longrightarrow E\colon F(K_{AB}, N_A), N_B \\ (b) & E \longrightarrow B\colon A, N_B \\ (b) & B \longrightarrow E\colon F(K_{AB}, N_B), N \\ (a) & E \longrightarrow B\colon F(K_{AB}, N_B) \end{array}$$

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.

Fixing the protocol

Observation: Alice authenticates Bob and Bob authenticates Alice with the same key K_{AB}.

Good practice of key separation: use different keys for different purposes.

In this case:

- K_{AB} for Alice to authenticate Bob
- K_{BA} for Bob to authenticate Alice

This gives:

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

Often one computes both keys from a single one, e.g., $K_{BA} = K_{AB} + 1$.
Bad idea: opens attack routes making the cryptosystem more vulnerable.

Another fix: initiator must authenticate first

Observation: Eve can launch an attack as outsider

Countermeasure: the initiator should be authenticated first.

This gives:

$$\begin{aligned} A &\longrightarrow B\colon \text{“Hi, I'm } A\text{; let's talk”} \\ B &\longrightarrow A\colon \text{“Sure, here is my challenge } N_B\text{”} \\ A &\longrightarrow B\colon F(K_{AB}, N_B), N_A \\ B &\longrightarrow A\colon \text{“you're } A \text{ alright; this shows I'm } B\text{:”} F(K_{AB}, N_A) \end{aligned}$$

- Solves the issue at the expense of one additional message exchange
- Still a very good protocol design rule in general

A fix that requires only a single key

Observation: in her attack Eve is replaying a message

Solution: MAC computation should include a nonce!

- Idea: make sure challenges for Alice are always different than challenges for Bob
- e.g., challenge for Alice ends with bit 1, from Alice with bit 0
- This is called domain separation

This gives

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

- More economical and as secure as key separation
- Can be combined with initiator authenticates first

Man-in-the-middle attack

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

Protocol

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

Attack

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

- As a result, A thinks that E is B, and B thinks that E is A.
- Note: Eve cannot take the initiative, but must wait 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)

Active attack overview I

Replay attacks

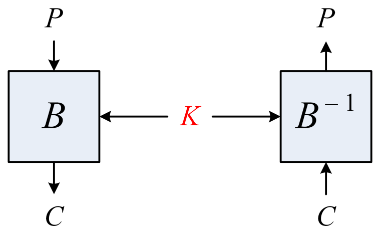
- eavesdropped e.g., financial transaction, is sent again
  - countermeasure: include nonce, checked by verifier
- Special case: reflection attack
  - typical attack on two-way challenge-response protocols
  - data from one session is re-used in another session
  - countermeasures: key separation or domain separation

Active attack overview II

Man-in-the-middle (MITM) attacks

- *passive* version, without modification: relay attack
  - not really a cryptographic attack
  - countermeasures: distance bounding, user interface measures
- Special case: delayed passive MITM: Lunch-break attack
  - typical for physical access: car keys, access badge
  - attacker gets *responses* from prover and uses them later
  - countermeasure: random challenge from verifier (freshness)
- *active* version involves breaking crypto
  - actually out of scope here
  - countermeasures: protect keys and use decent crypto

Remember block ciphers



- Function B mapping N-bit string P to N-bit string C
  - depends on a key $C = K\{P\}$
  - must be invertible (“*permutation*”): $P = K^{-1}\{C\}$
- Dimensions: block length N 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

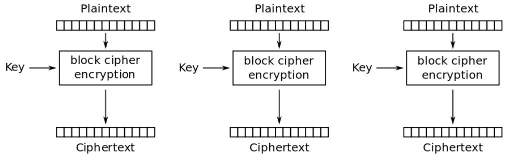
Block encryption modes

- 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

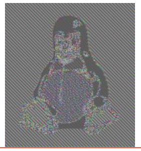
Block encryption: Electronic CodeBook Mode (ECB)



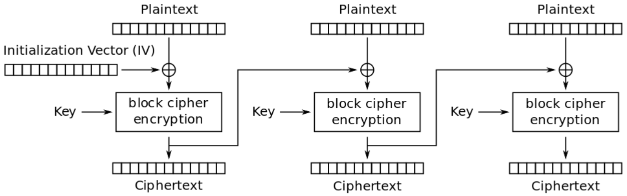
Electronic Codebook (ECB) mode encryption

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

under ECB gives



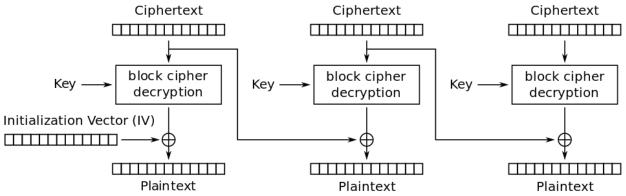
Block encryption: Cipher Block Chaining (CBC)



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 (cont'd)

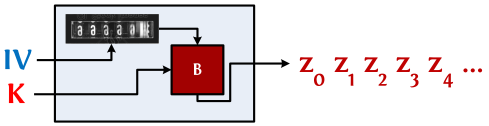


Cipher Block Chaining (CBC) mode decryption

Properties of CBC:

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

Stream encryption: Counter mode



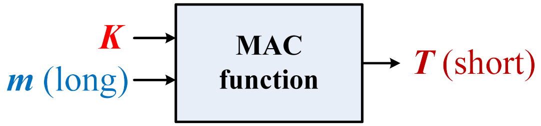
Stream cipher:

- keystream: $z_0 = K\{IV\}, z_1 = K\{IV+1\}, \ldots z_i = K\{IV+i\}$

Properties

- fully parallelizable
- most frequently used block cipher mode
- good IV management is critical for security
- $IV = D \| i$ with
  - D the diversifier
  - i the sequence number of the block in the keystream

Remember MAC functions



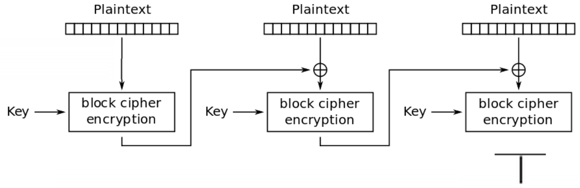
$$T = F(K, m)$$

with:

- m: message of arbitrary length
- T: MAC of fixed length

Can we use block ciphers for building a MAC function?

Cipher Block Chaining MAC mode (CBC-MAC)



- Observation: in CBC encryption $c[i]$ depends on $m[0]$ to $m[i]$
- Idea:
  - get rid of IV
  - 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

The symmetric key management problem

- Say we have a community of N people
- We want a confidential channel between any pair of users
- This requires one key K_{AB} per pair

Symmetric key distribution problem

If N people wish to communicate pairwise securely, one needs: $\binom{N}{2} = \frac{N(N-1)}{2} \approx N^2/2$ different secret keys.

Examples:

- A 10-person company: 45 keys
- Radboud universiteit has around 20.000 students: 4×10^8 keys
- Facebook has 2×10^9 people: 2×10^{18} keys

Key distribution by Trusted Third Party (TTP)

The principle:

- Every user shares one key with a TTP: say Alice has K_A
- Bob can ask the TTP for key to share with Alice with a message:

$$\begin{array}{lll} B & \longrightarrow & \mathrm{TTP}: \text{ key request for } B \text{ to } A \\ \mathrm{TTP} & \longrightarrow & B: K_B\{K_{AB}\} \text{ and ticket} = K_A\{K_{AB}\} \\ B & \longrightarrow & A: \text{ticket}, \text{traffic secured with } K_{AB} \end{array}$$

- This protocol only achieves a shared key K_{AB} between Alice and Bob
- Real-world protocols implement additional services, such as:
  - only a valid user can make a key request
  - K_{AB} expires after some given time period
  - freshness of key by Bob
  - freshness of ticket by Alice
  - freshness of key request by TTP

Diversified keys, AKA Key Derivation

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: $K_C = F(K_M, \mathrm{Id}_C)$.

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

$$\begin{array}{l} C \longrightarrow T: \mathrm{Id}_C \quad (T \text{ checks } \mathrm{Id}_C \text{ is in range, and computes } K_C) \\ T \longrightarrow C: N \\ C \longrightarrow T: F(K_C, N) \end{array}$$

- 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: $K_S = F(K_C, \mathrm{Nr}_C)$

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

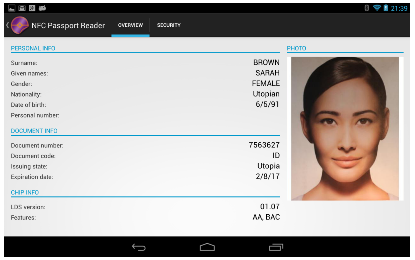
- Passports contain a (thick) plastic page, with embedded:
  - photo of cardholder + authenticity marks
  - chip + antenna
  - at bottom: 2-line Machine Readable Zone (MRZ) containing, date-of-issuance, BSN, document nr. etc.
- Essence of Basic Access Control (BAC):
  - cryptographic key for chip communication, 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



21:39

NFC Passport Reader OVERVIEW SECURITY

PERSONAL INFO

Surname:	BROWN
Given names:	SARAH
Gender:	FEMALE
Nationality:	Utopian
Date of birth:	6/5/91
Personal number:	

DOCUMENT INFO

Document number:	7563627
Document code:	ID
Issuing state:	Utopia
Expiration date:	2/8/17

CHIP INFO

LDS version:	01.07
Features:	AA, BAC

PHOTO

- 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

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

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

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

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:

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

- 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 $K_{\text{enc}}\{m\},\ F(K_{\text{mac}}, m)$
  - 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!