# PA Cybersecurity/IPP.CS.M1/MSc&T-CTD

## Introduction to Cryptology (INF558)

F. MORAIN

## Real-world cryptography

I. Remote access.

II. Smart cards.

III. Secure tunneling.

IV. Covid-19: a scientific and societal problem



## The need for protocols

???

We have primitives like encryption and signature.
And now, what?



## I. Remote access

**Problem:** getting access to a computer (or a web account).

- password based solutions;
- identification tokens.

2017: Newfusion (Belgique); Three Square Market (USA)

RSA-SECURE-ID please!



## A) Password based solutions

**Evolution in time/complexity:**

- naive way: stored in clear; easy but very weak security.
- the old way: `/etc/passwd`, etc. At a time were few machines existed and internet was in infancy. Revamped with shadow passwords.
- new way: SSH.

## Passwords: attacks

- **Dictionnary attack:** try all passwords in your file. Can be a real dictionnary and/or specialized given context via social engineering:
  - company, lab, X, ...;
  - young people vs. old people;
- **Brute force:** add to the preceding rules for adding special characters, e.g. $i \mapsto 1$, etc.
- **Many programs exist:** John the ripper, Hashcat, etc. cf. INF565.



## In real life

Building really (a lot of) strong passwords is boring for any human; random passwords cannot be memorized.

**10 most common passwords** (CNN Business from UK's National Cyber Security Centre, 2019.04.22):

- 123456 (23.2M),
- 123456789 (7.7M),
- qwerty (3M),
- password (3M),
- 111111, 12345678, abc123, 1234567, password1, 12345. Good guesses: Ashley, Michael; Liverpool, Chelsea, Arsenal, manutd, cowboys1 (from NFL!); Sunday, August; etc.



## More fun

`https://arstechnica.com/information-technology/2019/10/forum-cracks-the-vintage-passwords-of-ken-thompson`

Leah Neukirchen reported finding a source tree for BSD version 3, circa 1980

```
Dennis Ritchie       [Unix]          dmac
Stephen R. Bourne    [BSD]           bourne
Eric Schmidt         [now Alphabet]  wendy!!!
Stuart Feldman       [make]          axolotl
Brian W. Kernighan   [Unix]          /.,/.,
Ken Thompson         [Unix]          p/q2-q4!
```



## Protecting passwords

**Good practice:**
`https://www.ssi.gouv.fr/guide/mot-de-passe/`

- **Easy rule:** more than 12 characters from different types (lowercase, uppercase, digits, special characters);
- Two methods:
  - phonetic: *I bought eight CD's for 100 euros this afternoon* $\mapsto$ Ib8CD%Eta.
  - first letters: *Old soldiers never die, they simply fade away* $\mapsto$ OsndtsFa.

When you need to handle a lot of passwords: use a special program (such as `keepass`) that protects passwords as in a safe.

## The Unix case

**Principle:** store $F(m)$ where $F$ is difficult to invert; $F(m)$ is computed each time.

In `UNIX` (historical), modified DES:

- salt: 12 *random* bits (2 chars in `[a-zA-Z0-9./]`) when the passwd is created; they modify the `E` box in DES, using $2^{12} = 4096$ variants ($\Rightarrow$ any dictionary attack must be $\times 4096$);
- password truncated at 8 chars $\rightarrow$ 7 least significant bits per char $\rightarrow$ DES key (56 bits) $k \rightarrow r = DES_k^{(25)}(0_{64})$;
- $(s, r) \rightarrow 2 + 11$ printable characters stored in `/etc/passwd`.



## Unix: shadow passwds

Stored in `/etc/shadow`, only accessible by `root`:

```
root:$1$Etg2ExUZ$F9NTP7omafhKIlqaBMqng1:......
```

where

- `$1` means MD5 was used (on my laptop, SHA-512);
- `Etg2ExUZ` is the salt;
- `F9NTP7omafhKIlqaBMqng1` $= MD5(salt||passwd)$.

Check:

```
unix% openssl passwd -1 -salt Etg2ExUZ redhat
$1$Etg2ExUZ$F9NTP7omafhKIlqaBMqng1
```

Even with `salt`, dictionnary attack still possible.

But: passwords are sent in clear and can be reused.



## Semi-weak: One time passwords (Lamport)

**Rem.** OTP (RFC 1938), Skey, etc.

**Principle:** $f$ one way function (e.g., MD5).

- Oscar computes $x_0, x_1 = f(x_0), \ldots, x_n = f(x_{n-1})$, gives it to Alice and keeps $x_n$;
- when Alice wants to connect, Oscar asks for $x_{n-1}$, Alice sends it;
- Oscar checks whether $f(x_{n-1}) = x_n$; if yes, Oscar keeps $x_{n-1}$ for the next connection.

**Advantage:** a given password is used only once.

**Drawback:** heavy load on the user; doesn't prevent a theft of connection.



## B) Adding authentication

Until now, one arrow:

$$\text{Alice} \rightarrow \text{Oscar} : \text{passwd}$$

and Oscar grants access or not.

No authentication of Oscar w.r.t. Alice, or Alice w.r.t. Oscar.

What can we do?

Alice has to prove she has a secret that Oscar can verify and vice versa.

## Challenge/answer in the symmetric world

Prerequisite: Alice and Oscar have a common secret symmetric key $K$.

Intuitive protocol:

$$\begin{array}{lcll} A & \rightarrow & O: & r_A \\ A & \leftarrow & O: & E_K(r_A, r_O) \\ A & \rightarrow & O: & r_O \end{array}$$

But reflection attack (too symmetric protocol):

$$\begin{array}{ll} A \rightarrow E: & r_A \\ \quad A \leftarrow E: & r_A \\ \quad A \rightarrow E: & E_K(r_A, r'_A) \\ A \leftarrow E: & E_K(r_A, r'_A) \\ A \rightarrow E: & r'_A \end{array}$$

$\Rightarrow$ sharing a common secret value is not enough!
$\Leftrightarrow$ concurrency models



## SSH (1/2)

**Sketch:**

$$\begin{array}{lcll} A & \rightarrow & O: & \text{authentication request} & (0) \\ A & \leftarrow & O: & \text{host key} + \text{server key} & (1) \\ A & \rightarrow & O: & \text{encrypted session key} & (2) \\ A & \leftarrow & O: & \text{OK} & (3) \end{array}$$

(1): host key (RSA $Pub_O$) is fixed and binds a key to an IP address; server key (RSA $Pub_s$) changes regularly; Alice checks that the (host key, IP) is known (`.ssh/known_hosts`), can accept a new one.

(2): generates a 256-bit random number $r$, chooses a block cipher and returns $Pub_s(Pub_O(r))$.

(3): Oscar decrypts $r$ and starts an encrypted connection with Alice, using $r$ (session key never re-used).



## SSH (2/2)

If required: RSA authentication of Alice; requires a data base of public keys.

$$\begin{array}{lcll} A & \rightarrow & O: & \text{authentication request} & (0) \\ A & \leftarrow & O: & Pub_A(r) & (0') \\ A & \rightarrow & O: & H(r) & (0'') \end{array}$$

(0’): Alice decrypts $r$; in fact, its secret key is stored on the local computer, protected via a passphrase.

(0”): any hash function.

**Characteristics:**

- No certificate, but preconfiguration needed (data bases of hosts/keys, data bases of users);
- easy to make it run.



## C) How to distribute keys? (1/2)

**Initial authentication:**

- easy face-to-face (ID card);
- initial sheet of paper and quick change;
- check strength *a posteriori*.

## How to distribute keys? (2/2)

The best (only) way is to use another communication channel.

Ideal version of the universe: everyone $A$ has a pair $(\text{Pub}_A, \text{Priv}_A)$; the public key is stored in a directory that is universally accessible.

**But:**

- Who maintains the directory? (add/delete/etc.)
- How do you access it (in a secure/authenticated way)?
- How do you distribute it? How does one validate the copies?
- How does one register? Off-line request.

Some cases where it more or less works: local or proprietary networks (GSM), in which some trusted authority exists (CA, TPC/TTP, etc.); LDAP.



## Certificates, PKI

A certification authority delivers certificates (electronic passports): $\text{Certif}(F) = S_{ac}(\text{"François"}||\text{Pub}_F)$; $V_{ac}$ is universally available.

**Typically:** François presents his certificate; Bob is convinced of the authenticity of François's public key.

**Certificate chains:** necessary if Alice and Bob do not share the same CA. Managing this can be heavy.

**Alternative:** Web of trust (PGP, gnupg, etc.)

$\Longleftrightarrow$ replace trust in one authority by trust in a lot of people.

Each user associates a confidence level to each public key he detains. He becomes an endorser for other actors.



## X.509 (ISO)

```
certificate ::= SIGNED SEQUENCE (
  signature AlgorithmIdentifier,
  issuer Name,
  validity Validity ::= SEQUENCE (
      notBefore UTCTime,
      notAfter  UTCTime)
  subject Name,
  subjectPublicKeyInfo SubjectPublicKeyInfo ::= SE
      algorithm AlgorithmIdentifier,
      subjectPublicKey BIT STRING))
```

Beware: certificates protected by MD5 are no longer secure (attacks of A. K. Lenstra, X. Wang and B. de Weger – 2005).



## Statistics on certificates

(Lenstra, Hughes, Augier, Bos, Kleinjung, Wachter, CRYPTO 2012)

6,185,372 distinct X.509 certificates :

- containing 6,185,230 RSA, 141 DSA, 1 ECDSA;
- 47 % with expiration date $> 2011$;
- 77.7 % use SHA-1 or better
- RSA:
  - eight certificates have $e = 1$, two have $e$ even;
  - 266, 729 certificates contain an RSA modulus shared with another certificate;
  - 1200 pairs of $N$-values sharing a distinct prime factor in common.

$\Rightarrow$ take care to poor seeding of the random number generator; use NIST's recommendation (FIPS-186-3, 2009).

See also: Heninger/Durumeric/Wustrow/Halderman (USENIX 2012).

## Keys in the wild

(Bernstein, Chang, Cheng, Chou, Heninger, Lange, van Someren – 2013)

Taiwan issued Citizen Digital Certificate for $> 2$ million citizens (1024-bit RSA keys).

**Registration:**
citizen goes to government registration office; a government official places the smart ID card into a registration device; the device prompts the card to generate a new RSA-key and the public key is incorporated into a certificate to be signed by government's agency MOICA [1].

Claimed to be secure, etc.

[1]Min. Of Interior Certif. Auth.



## Taiwan (cont'd)

But... 184 keys can be factored:

- 103 with $p$ shared, 81 presenting randomness-generation failures;
- 103 keys leading to 119 different primes; examining the shared primes gave hints as to the fabrication of primes: 46 times $NextPrime(2^{511} + 2^{510})$; 7 times

  0xc92424922492924992494924492424922492 92\
  4992494924492424922492924992494924492424\

  92249292499249492449242492249292499249492449 2424e5



## II. Smart cards

1984: France is the first country to conduct field trials of microprocessor chip cards embedded in plastic bank cards.
By 1994: B0' (even with PIN only, fraud dramatically reduced).

**Principles:**

- B(ank) has $S_B, V_B$;
- C(ard) has $Data = name, etc., S_B(Data)$ and PIN (3456).
- T(erminal) has $V_B$.

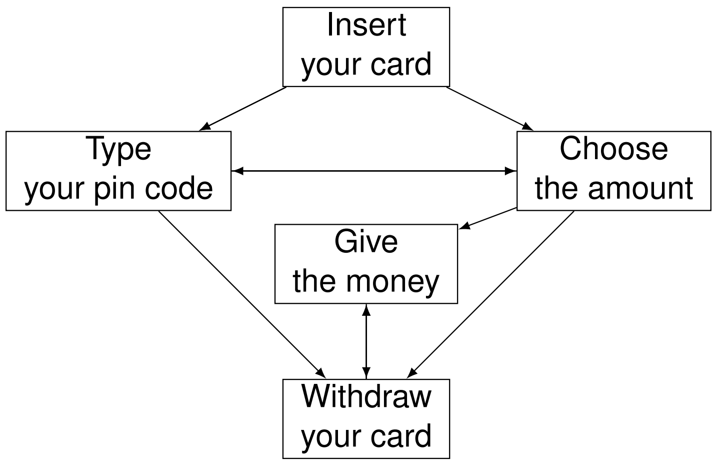
**Protocol:**

$$\begin{array}{llll} T & \rightarrow & A: & \text{authentication request} & (1)\\ T & \leftarrow & C: & Data, S_B(Data) & (2)\\ T & \rightarrow & A: & \text{code?} & (3)\\ T & \leftarrow & A: & 3456 & (4)\\ T & \rightarrow & C: & 3456 & (5)\\ T & \leftarrow & C: & \text{ok} & (6) \end{array}$$

(2): $T$ checks $V_B(S_B(Data)) = Data$.



## ATM's automata



## Attacks (1/2)

Suppose we can substitute $C'$ for $C$ at the end:

$$
\begin{array}{llll}
T & \rightarrow & E: & \text{authentication request} & (1) \\
T & \leftarrow & C: & Data, S_B(Data) & (2) \\
T & \rightarrow & E: & \text{code?} & (3) \\
T & \leftarrow & E: & 6789 & (4) \\
\\
T & \rightarrow & C': & 6789 & (5) \\
T & \leftarrow & C': & \text{ok} & (6)
\end{array}
$$

Then money is debited from $C$ and not $C'$.

Realistic attack? Yes if I can program my own smart card and put the data I want in it.



## Attacks (2/2)

Yes card: if $S_B$ broken (The Humpich case)

$$
\begin{array}{lllll}
T & \rightarrow & E: & \text{authentication request} & (1) \\
T & \leftarrow & C: & XXX, S_B(XXX) & (2) \\
T & \rightarrow & E: & \text{code?} & (3) \\
T & \leftarrow & E: & 0000 & (4) \\
T & \rightarrow & C: & 0000 & (5) \\
T & \leftarrow & C: & \text{ok} & (6)
\end{array}
$$



## Europay MasterCard Visa (EMV)

Need for global interoperability.

Issuer: RSA key pair generated and certified by the Payment System Certification Authority

If public key technology supported by the card: RSA key pair must be generated and certified by the issuer.

**Two authentication modes:**

- SDA (*Static Data Authentication*); too close to the old one.
- DDA (*Dynamic Data Authentication*).



## Static Data Authentication (SDA)

$$
\begin{array}{lllll}
T & \rightarrow & A: & \text{authentication request} & (1) \\
T & \leftarrow & C: & S_S(Pub_B), Data, S_B(Data) & (2) \\
T & \rightarrow & A: & \text{code?} & (3) \\
T & \leftarrow & A: & 3456 & (4) \\
T & \rightarrow & C: & 3456 & (5) \\
T & \leftarrow & C: & \text{ok} & (6)
\end{array}
$$

(2): $T$ gets $V_S(S_S(Pub_B)) = Pub_B$.

Still possible to build a YesCard...

# Dynamic Data Authentication (DDA)

$C$ has $Pub_C$ and $Priv_C$.

$$\begin{array}{llll} T & \rightarrow & A: & \text{authentication request} & (1)\\ T & \leftarrow & C: & S_S(Pub_B), S_S(Pub_C), Data, S_B(Data) & (2)\\ T & \rightarrow & C: & N_T & (3)\\ T & \leftarrow & C: & D_C[Priv_C](N_T) & (4)\\ T & \rightarrow & A: & \text{code?} & (5)\\ T & \leftarrow & A: & 3456 & (6)\\ T & \rightarrow & C: & E_C[Pub_C](3456) & (7)\\ T & \leftarrow & C: & \text{ok} & (8) \end{array}$$

(3)-(4): $T$ challenges $C$ by sending a nonce $N_T$ that must be decrypted (signed) by $C$. This prevents a YesCard to be used at that point.

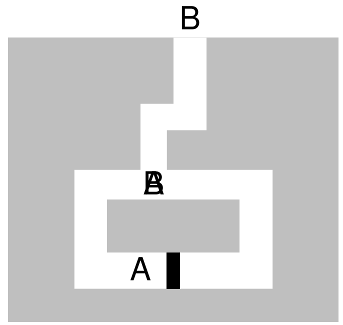
(8): $C$ decrypts $D_C[Priv_C](E_C[Pub_C](3456)) = 3456$.



# Zeroknowledge proofs

**Idea:** prove one's identity without revealing any secret.

Answer: Ali Baba's cave (Guillou–Quisquater).




**Ex.** Fiat-Shamir, Guillou-Quisquater, Schnorr, etc.



# Fiat-Shamir

Prerequisite: Alice chooses two secret prime numbers $p$ and $q$, forms $N = pq$, $s \in_R [1, N[$ and computes $v = s^2 \bmod N$.

Public key: $(N, v)$.

Private key: $s$.

Protocol: Bob wants to verify Alice's identity, for instance by being convinced that Alice possesses $p$ and $q$.

1. Alice chooses $r \in_R [1, N[$ and sends $x = r^2 \bmod N$ to Bob.
2. Bob sends a random bit $b$ ($b = 0$ or $1$) to Alice.
3. If $b = 0$, Alice sends $y = r$; if $b = 1$, she sends back $y = rs \bmod N$.
4. If $b = 0$, Bob checks that $y^2 \equiv x \bmod N$ (Alice knows $\sqrt{x}$); if $b = 1$, Bob checks that $y^2 = xv \bmod N$ (Alice knows $\sqrt{v}$).



# How can Alice cheat?

She must guess what Bob is going to ask her in advance:

• if she guesses that Bob is going to send her $b = 0$, Alice prepares $x = r^2 \bmod N$ and sends $x$, then $r$; but she cannot have the answer to $b = 1$, since she doesn't know $s$.

• if she guesses that Bob is going to send her $b = 1$, Alice prepares $x \equiv r^2/v \bmod N$ and then sends $x$, followed by $r$. But Alice has no answer for $b = 0$ since she doesn't know $s$.

With probability 1/2, Alice convinces Bob: the protocol is repeated $t$ times, and Bob is convinced with cheating probability $\leq 1/2^t$.

**Rem.** It can be shown that the protocol does not disclose any secret information on $s$ during the execution (real difficult theorem) $\Leftrightarrow$ zeroknowledge.

## III. Secure tunneling

### A) The Diffie-Hellman protocol

Prerequisite: $G = \langle g \rangle$ (typically $(\mathbb{Z}/p\mathbb{Z})^*$).

Protocol:

$$\begin{array}{ccccc} A & \rightarrow & B & : & g^a \\ A & \leftarrow & B & : & g^b \end{array}$$

Actions: in the end, $A$ and $B$ possess $g^{ab}$.

Properties: security is based on the DH problem.

Paparazzi-in-the-middle attack: (*Mafia attack* or *(wo)man-in-the-middle*).

$$\begin{array}{ccccc} \text{Alice} & & \text{Charlie} & & \text{Bob} \\ g^a & \rightarrow & g^c & \rightarrow & \\ & \leftarrow & g^c & \leftarrow & g^b \\ g^{ac} & & g^{ac}, g^{bc} & & g^{bc} \end{array}$$

$\Rightarrow$ Alice and Bob must authenticate themselves!



## First repair

Prerequisite: each user has a pair $(S, V)$.

Protocol:

$$\begin{array}{cccccr} A & \rightarrow & B & : & \text{Cert}_A, g^x & (1) \\ A & \leftarrow & B & : & \text{Cert}_B, g^y, S_B(g^y, g^x) & (2) \\ A & \rightarrow & B & : & \text{Cert}_A, S_A(g^x, g^y) & (3) \end{array}$$

Common key: $k = g^{xy}$.

(2): one adds its own share to guarantee freshness.

**Prop.** protocol BADH is not consistent (bad link principal/key).
*Proof.* Eve doesn't touch the first two messages, and replaces (3) by:

$$\begin{array}{cccccr} E & \rightarrow & B & : & \text{Cert}_E, S_E(g^x, g^y) & (3) \end{array}$$

In the sequel, all message arriving from Alice, encrypted with key $k$ at Bob will be considered as coming from Eve (e.g., Bob is a bank, Alice, Eve are clients). $\square$



## Better variants (1/2)

Station-to-station (Diffie/van Oorschot/Wiener, 1992)

Prerequisite: each user has a pair $(S, V)$.

Protocol:

$$\begin{array}{cccccr} A & \rightarrow & B & : & \text{Cert}_A, g^x & (1) \\ A & \leftarrow & B & : & \text{Cert}_B, g^y, E_k(S_B(g^y, g^x)), \quad k = g^{xy} & (2) \\ A & \rightarrow & B & : & \text{Cert}_A, E_k(S_A(g^x, g^y)) & (3) \end{array}$$

Idea: Alice and Bob prove each other they know $k$.

**Rem.** one needs an external proof binding $A$ to $V_A$; otherwise, Eve registers $V_A$ in her name ($V_A$ is public...).



## Better variants (2/2)

ISO:

Protocol:

$$\begin{array}{cccccr} A & \rightarrow & B & : & A, g^x & (1) \\ A & \leftarrow & B & : & B, g^y, S_B(g^y, g^x, A) & (2) \\ A & \rightarrow & B & : & S_A(g^x, g^y, B) & (3) \end{array}$$

**Property:** the preceding attacks are countered.

# B) SSL/TLS

Principle: low level compression/encryption of all applications using TCP/IP.

**Rem.** initiated by and used in browsers (Netscape).

**Overview:**

- negotiation of encryption algorithms;
- optional authentication of principals and key exchange;
- all communications are encrypted after the first phase;
- compression of data;
- integrity of data by MAC.



# Some SSL protocols

**Two protocols among others:**

- Handshake protocol : negotiation of algorithms.
- Record protocol : encapsulation layer, using valid keys and algorithms.

**Applications:** Secure POP, secure IMAP.

**Implementations:** OpenSSL, Netscape. Cf. Apache-ssl. Beware of old US versions (40 bits). Cf `www.fortify.net`.



# TLS with pictures (excerpts from RFC 2246)

client_version
client_random
session_id
cipher_suites
compression_methods

Client | Server

server_version
server_random
cipher_suite
compression_meth

ClientHello $\longrightarrow$

$\longleftarrow$ ServerHello

$\longleftarrow$ Certificate* (X509)

$\longleftarrow$ ServerKeyExchange*

$\longleftarrow$ CertificateRequest*

$\longleftarrow$ ServerHelloDone

Certificate* $\longrightarrow$

ClientKeyExchange $\longrightarrow$

CertificateVerify* $\longrightarrow$

[ChangeCipherSpec] $\longrightarrow$

verify_data "client finished" Finished $\longrightarrow$

$\longleftarrow$ [ChangeCipherSpec]

$\longleftarrow$ Finished verify_data "server finished"

application $\longleftrightarrow$ application



# Cipher suites

Example of **cipher_suite:** `TLS_RSA_WITH_RC4_128_MD5`.

| Key Exchange Algorithm | Certificate Key Type |
|---|---|
| RSA | |
| DHE_DSS | DSS public key |
| DHE_RSA | RSA signature key |
| DH_DSS | |
| DH_RSA | |

**Pb:** just too many of these! Tedious to implement them all, security problem.

# Using RSA

| Client | | Server |
|---|---|---|
| | $\longleftarrow$ | Certificate* (X509): $\ni$ Pub$_S$ |
| | $\longleftarrow$ | ServerK~~ey~~Exc~~h~~ange* |
| | $\longleftarrow$ | CertificateRequest* |
| | $\longleftarrow$ | ServerHelloDone |
| Certificate*: $\ni$ Pub$_C$ | $\longrightarrow$ | |
| ClientKeyExchange: Pub$_S(pms)$ | $\longrightarrow$ | |
| CertificateVerify*: $S_C$(handshake msgs) | $\longrightarrow$ | |



# Using Diffie-Hellman

| Client | | Server |
|---|---|---|
| | $\longleftarrow$ | Certificate* (X509) |
| | $\longleftarrow$ | ServerKeyExchange*: $p$, $g$, $g^X$ |
| | $\longleftarrow$ | CertificateRequest* |
| | $\longleftarrow$ | ServerHelloDone |
| Certificate* | $\longrightarrow$ | |
| ClientKeyExchange: $g^Y$ | $\longrightarrow$ | |
| CertificateVerify* | $\longrightarrow$ | |



# Miscellaneous computations

```
PRF(secret, label, seed) = P_MD5(S1, label+seed)
                           XOR P_SHA-1(S2, label+s
```

with `secret = S1 || S2` (with potential overlap in such a way that $|S1| = |S2|$).

RSA: `pre_master_secret`=random.

DH: `pre_master_secret`= $g^{XY}$.

```
master_secret=PRF(pre_master_secret, "master_secre
   ClientHello.random+ServerHello.random)[0..47];
```

```
verify_data = PRF(master_secret, finished_label,
            MD5(handshake_messages)
           +SHA-1(handshake_messages))[0..11];
```

with `finished_label = Sender.server` or `Sender.client`.



# The record layer

Client and server both compute new quantities from what they got in the handshake phase:

```
key_block = PRF(master_secret, "key expansion",
                server_random, client_random);
```

from which a MAC key $K_m$ is deduced, as well as an encryption key $K_s$ (for a symmetrical system).

What is really exchanged is: $E_{K_s}(M, \mathrm{HMAC}_{K_m}(M))$.

## Bleichenbacher vs. PKCS # 1 v1.5

(*Public Key Cryptography Standard*, RSA, Inc.)

**Input:** $2^{8(k-1)} \leq N < 2^{8k}$; $M$ of length $mLen \leq k - 11$.

**Output:** $C$ of length $k$.

Encryption:
1. If $mLen > k - 11$ then error.
2. Build $EM = 0x00||0x02||PS||0x00||M$, with $PS$ a string of at least 8 random non-zero bytes.
3. Compute $m =$ OS2IP$(EM)$; $c = m^e \bmod N$; $C =$ I2OSP$(c)$.

Decryption:
1. If $length(C) > k$ then error.
2. Compute $c =$ OS2IP$(C)$; $m = c^d \bmod N$; $EM =$ I2OSP$(m)$.
3. Rebuild $EM = o_1||o_2||ps||o_3||x$. If $o_1 \neq 0x00$ or $o_2 \neq 0x02$ or $o_3 \neq 0x00$ or $ps$ has less than 8 bytes, then error; else return $x$.



## Bleichenbacher's attack (CRYPTO'98)

In some implementations (SSL): if an attacker can have access to the exact error among these three, she can use the system as an oracle.

| | Client | | Server |
|---|---|---|---|
| | | $\longleftarrow$ | Certificate* (X509): $\ni$ Pub$_S$ |
| | | $\longleftarrow$ | ServerK~~ey~~Exc~~h~~ange* |
| | | $\longleftarrow$ | CertificateRequest* |
| | | $\longleftarrow$ | ServerHelloDone |
| Certificate*: $\ni$ Pub$_C$ | | $\longrightarrow$ | |
| ClientKeyExchange: Pub$_S(pms)$ | | $\longrightarrow$ | |
| | | $\longleftarrow$ | Error |



## Bleichenbacher's attack: end

If $B = 2^{8(k-2)}$, the oracle answers: $2B \leq m < 3B$.

To decrypt $y = f(w)$: look for $s$ s.t. $yf(s) = f(ws)$ has the right formatting.

For these, one deduces $2B \leq (ws) \bmod N < 3B$, which limits the possible values for $w$.

$\Rightarrow$ replace

```
if(! goodFormatForMessage(m))
    send_error("bad format");
```

by

```
ok = goodFormatForMessage(m);
if(ok){
    // remaining code
}
if(! ok)
    kill_connection();
```



## Manger's attack – CRYPTO'01

Timing attack on the preceding scheme. Replace it with:

```
ok = goodFormatForMessage(m);
// remaining code
if(! ok) kill_connection();
```

$\Rightarrow$ Do not turn a program into an oracle!

$\Rightarrow$ Good cryptography is orthogonal to good software engineering!!

## A glimpse of TLS 1.3

(See e.g., `https://www.bortzmeyer.org/8446.html`.)

- RFC 8446: 2018/08, after a lot of work, debates, strong oppositions, etc. First draft in 2014...
- New protocol, not compatible with TLS 1.2 (beware of downgrade attacks!).
- List of symmetric cryptography drastically reduced: keep authenticated encryption algorithms only (e.g., MAC-then-Encrypt, etc.).
- Forward secrecy using ephemeral keys (complicates the work of BB's; harder debugging).
- Elliptic curves used; DSA/RSA withdrawn.
- etc.



## IV. Covid-19: a scientific and societal problem

**How can we help health workers inform people who were close to an infected person?**

**General idea:** everybody has a smartphone, so write an app!

**But:**

- you must respect privacy;
- the app must not be used for something else (by a user or authorities, see GDPR);
- minimize the quantity of private data and their treatment (classical problem for CNIL, etc.).



## Attackers

- Classical attacker: can spy, modify, delete all messages on the network (including mobile apps).
- Priviledged user: honnest but curious who might use several sources to learn things he should not know.
- High-level attacker: who can analyze national traffic, etc.



## Desired security

**The app should not:**

- trace users;
- deanonymize the users;
- build the social graph;
- identify sick people;
- enable a false sickness declaration (otherwise, forces a useless quarantine for contacts);
- send a false infection message to someone.

## Typical scenario

These people cross each other in RER-B, then Rey is diagnosed to have the Covid.
The other travelers are warned by the app on their smartphones!



## First idea

- all inhabitants of a country have a smathphone with activated GPS;
- the ministry for health stores `(name, date, position)` in a big file;
- if Rey catches covid-19, the ministry gets informed (by Rey herself? Her doctor?) and can find all *contacts* of Rey in the last 14 days to warn them.

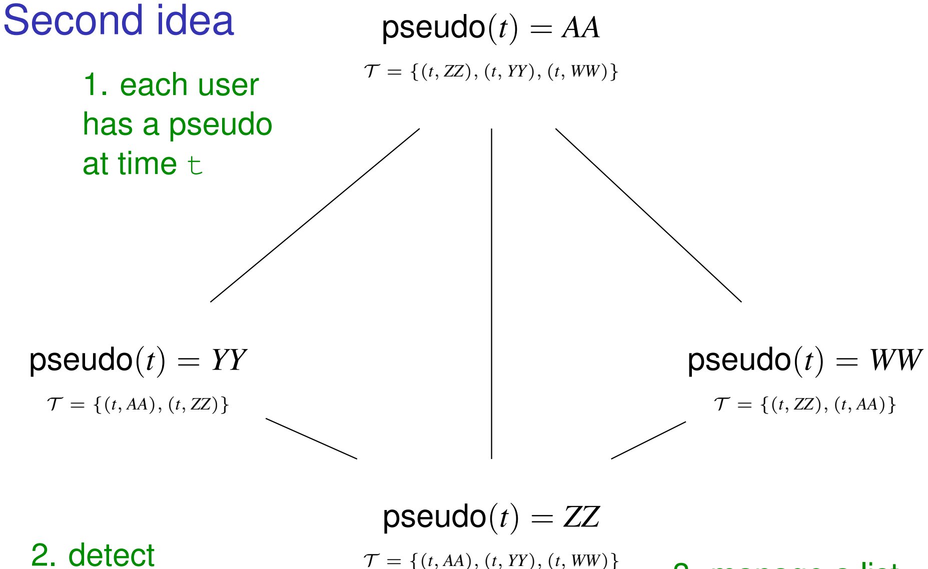
**Problems:**

- HUGE quantity of data;
- HUGE weight on the confidentiality of the file (at least replace `name` by `pseudonym`);
- no respect for privacy (the *social graph* is easy to build).



## Second idea

1. each user has a pseudo at time `t`




2. detect your neighbors using bluetooth
3. manage a list of contacts



## Detecting contacts using bluetooth

*Everything Is Better*
*With Bluetooth*
Dr. S. Cooper

Bluetooth link (2.4 GHz):

- very weak energy consumption;
- very short range (radius $\approx 10$ m);
- weak transmission rate (ok for stereo sound);
- very cheap and light;
- with a Low Energy version (consumption / 20 ou 100);
- weak basic security.

Not clear it works fine (range, obstacles, etc.).

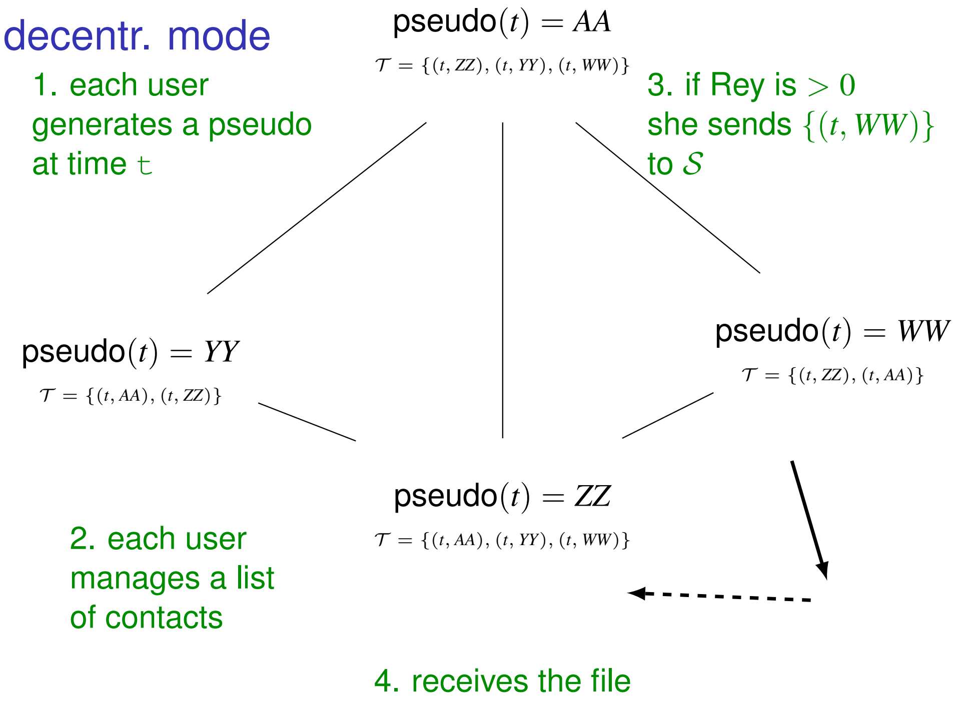
## Two modes: centralized and decentralized

Close models, but differences on
- infrastructure;
- creation of ephemeral pseudos;
- the way the user is warned.

**Examples:**
- centralized mode: ROBERT, NTK, TraceTogether (Singapore), ...
- decentralized mode: DP3T, Apple/Google, ...



## decentr. mode






## Tracing users

**In centralized mode:** if a bad guy has access to the pseudo building algorithm of $\pi_U$, he can link alltransactions to $\pi_U$. Then link $\pi_U$ to $U$.

**In decentralized mode:** local attack, a bit less disastrous (cf. ref. [3]).

Both modes are in danger in case of coercition or theft of the smartphone.



## Identifying infected users

**In decentralized mode:** see ref. [4]
- Paparazzi: $A$ intercepts the identifiers of $B$ and searches for them in the server's list.
- Insurance: a company (hotel, etc.) collects bluetooth identifiers and real identifiers.
- Malicious app intercepts identifiers.
- Pair this with video watching.
- ...

**In centralized mode:** one may create a phony account $A$ only contacting $B$.

# Conclusions

**Centralized or decentralized?**

- attacks in both cases;
- this is not the panacea (hybrid model?);
- more cryptography may help (DESIRE);
- hot topic for the crypto/security communities.

**Other societal problems:**

- Digital divide.
- How do one convince people?
- Open Source apps for looking for bugs by world-wide specialists (and also to try to counter the conspiracy theories).

**And now (22/10/20):** *Tous Anti Covid* – same protocol, new wrapping.



# Some links

1. French Académie Des Sciences `https://www.academie-sciences.fr/pdf/rapport/2020_05_07_Tracage_Expert.pdf`. See also the (pluridisciplinary videos with real science in them!).
2. Fraunhofer's report: `https://eprint.iacr.org/2020/489.pdf`.
3. S. Vaudenay's article: `https://eprint.iacr.org/2020/531`.
4. `https://risques-tracage.fr/`: a dozen generic attack scenarii.
5. Round Table during Eurocrypt 2020 meeting `https://www.youtube.com/watch?v=Xt4P8E_Y-xc`