CRYPTOGRAPHY LECTURE, 2022

Computer Science and Algorithmics, PWr

Mirosław Kutyłowski

Authentication/Identification

eIDAS Regulation:

'electronic identification' means the process of using person identification data in electronic form uniquely representing either a natural or legal person, or a natural person representing a legal person;

'electronic identification means' means a material and/or immaterial unit containing person identification data and which is used for authentication for an online service ‘authentication’ means an electronic process that enables the electronic identification of a natural or legal person, or the origin and integrity of data in electronic form to be confirmed;

„identyfikacja elektroniczna” oznacza proces używania danych w postaci elektronicznej identyfikujących osobę, unikalnie reprezentujących osobę fizyczną lub prawną, lub osobę fizyczną reprezentującą osobę prawną;

„uwierzytelnianie” oznacza proces elektroniczny, który umożliwia identyfikację elektroniczną osoby fizycznej lub prawnej, lub potwierdzenie pochodzenia oraz integralności weryfikowanych danych w postaci elektronicznej;

Authentication:

what I have – hardware token

what I know - password

who I am - biometrics (details another lecture!)

Biometric authtentication

e.g. fingerprints

Fingerprints – crypto issues

how to store biometric data for comparison so that it cannot be used for impersonation?

a) small errors when reading biometric data

b) how to compare $\Delta +$ error with $\mathrm{Enc}_{\mathrm{PK}}(\Delta)$? - similarity of plaintexts must not be detectable

"Check on Chip" methods

Alice and Bob open a session:

- Alice must prove the person on the other side that she is Alice
- Bob must prove the person on the other side that he is Bob
- a session key must be established

Options:

i. Alice and Bob share a key

ii. no preshared keys, only public keys

Symmetric: based on preshared keys

challenge-response protocol for shared K:

i. Bob sends random nonce r_A to Alice

ii. Alice responds with $s = F(K, r_A)$ where F is a one-way function

iii. Bob checks whether s is correct

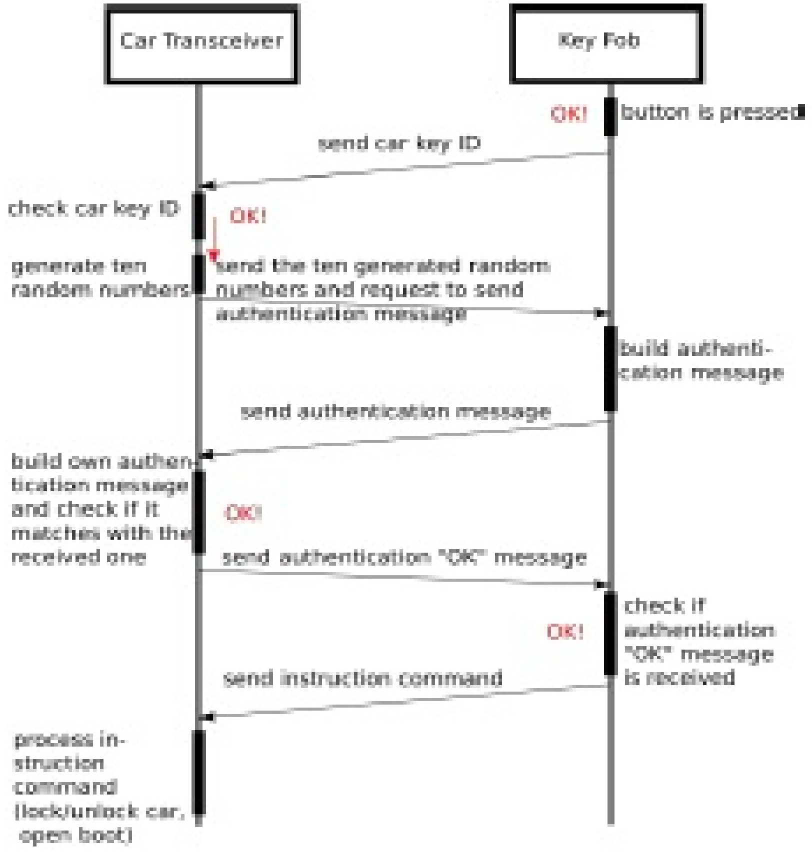
mutual authentication version:

– Alice nad Bob exchange random nonces r_A, r_B

– Alice sends $F(K, r_A, r_B)$, Bob sends $F(K, r_B, r_A)$

Example: keyless car

(pictures by T.Glovker, T. Mantere, M. Elmusrati



Car Transceiver
Key Fob
start button is pressed
OK!
request to send car key ID
read car key ID
generate ten random numbers
send car key ID, the ten generated random numbers and
request to send authentication message (A)
check car key ID
OK!
build authentica-tion message
generate ten random numbers
send authentication message from request (A), the ten generated random numbers and
request to send authentication message (B)
build own auth-entication mes-sage and check if it matches with the received one
OK!
build authenti-cation message
send authentication message from request (B)
build own auth-entication mes-sage and check if it matches with the received one
OK!
process start command

Dynamic shared key

each time when authentication suceeds the shared key is changed

- clones are detected
- problems of synchronization

Asymmetric authentication

- Alice holds a private key SK
- Bob knows a matching paublic key PK of Alice (e.g. from a certificate)
- a person proves that she knows SK in order to authenticate as Alice
  - Interactive Proof of Knowledge

Requirement:

no information on SK should be leaked during the authentication process

(otherwise Alice is no longer the only person that holds SK - impersonation becomes possible)

`already discussed: Alice signs a random challenge`

Schnorr Identification Protocol

Alice: secret key $\mathrm{SK} = x$, public key $\mathrm{PK} = g^x$

i. Alice chooses k at random, $r := g^k$

ii. Alice sends r to Verifier

iii. Verifier chooses c at random and sends to Alice

iv. Alice calculates $s := k - x \cdot c \bmod q$ and sends s to Verifier

v. Verifier checks that $g^s \cdot \mathrm{PK}^c = r$

observation: there is only one value s that would satisfy the test. In order to calculate it Alice must know x (and k)

Impersonation for Schnorr authentication?

Assume that an algorithm $\mathcal{A}$ can do it.

Use $\mathcal{A}$ to break Discrete Logarithm Problem

1. run $\mathcal{A}$: r received, challenge c chosen, s received
2. rerun $\mathcal{A}$ with the same randomness:
   - –. once r received choose a different challenge c'
   - –. response s' from $\mathcal{A}$

for (unknown) discrete logarithm k of r:

$$s = k - x \cdot c \bmod q$$

$$s' = k - x \cdot c' \bmod q$$

solve it for x

Simulating a transcript of interaction

Verifier can create a valid transcript o interaction without talking to Alice

$\Rightarrow$ Verifier cannot use a transcript to show that he has interacted with Alice

(privacy, data minimality etc)

Forging a transcript:

i. choose c and s at random

ii. calculate $r := g^s \cdot \mathrm{PK}^c$

(the same probability distribution of (r, c, s) as for genuine executions)

Consequences: zero-knowledge property

informally: executing the protocol **does not increase the chances** of Verifier to impersonate Alice

Why:

if protocol transcripts are required for the attack $\mathcal{A}$, then forge them himself

So: if impersonation is possible, then it is possible based on the public key only

(reduction to DLP)

Fiat-Shamir heuristics:

from an interactive proof of knowledge to a digital signature scheme:

– replace the random challenge by Hash of the elements exchanged so far

Fiat-Shamir protocol:

Alice knows square root s of v modulo RSA number n

interactive proof of knowledge of s:

i. Alice chooses r at random, $x := r^2 \bmod n$

ii. Alice sends r to Verifier

iii. Verifier chooses bit b at random

iv. if $b = 0$, then Alice has to present $a = r$, else Alice has to present $a = r \cdot s$

v. Verifier checks that $a^2 = x \bmod n$ (if $b = 0$) or that $a^2 = x \cdot v \bmod n$ (if $b = 1$)

probability to cheat successfully: 0.5, so the protocol repeated many times

or use Fiat-Shamir heuristic to reduce the number of messages

Stinson-Wu protocol

1. Verifier chooses x at random, computes $X := g^x$ and $Y := \mathrm{Hash}(A^x)$
2. Verifier sends X, Y to Alice
3. Alice computes $Z := X^a$ and aborts if $Y \neq \mathrm{Hash}(Z)$
4. Alice sends Z
5. Verifier accepts iff $Z = A^x$

Nice feature: Alice knows that Verifier knows x and can compute the answer himself

Password authentication

- significant challenge, since the entropy of passwords is low, it allows brute force attack

- a passive observer may try to derive the password used

- usually integrated as Password Authenticated Key Exchange (PAKE)

Jablon and his seminal protocol:

- never used due to a patent
- German authorithies developed their protocol to avoid the patent, now this protocol in ID documents (e.g. Polish personal ID: password= CAN number)

PACE

Chip(A)		Reader(B)
holds:		holds:
π - password		π, input from the document owner
$K_\pi := H(\pi\|\|0)$		$K_\pi := H(\pi\|\|0)$
choose $s \leftarrow \mathbb{Z}_q^*$ at random		
$z := \mathrm{Enc}(K_\pi, s)$	$\xrightarrow{\mathcal{G}, z}$	abort if $\mathcal{G}$ incorrect
		$s := \mathrm{Dec}(K_\pi, z)$
	DH2Point Start	
choose $x_A \leftarrow \mathbb{Z}_q^*$ at random		choose $x_B \leftarrow \mathbb{Z}_q^*$ at random
$X_A := g^{x_A}$	$\xleftarrow{X_B}$	$X_B := g^{x_B}$
abort if $X_B \notin \langle g \rangle \setminus \{1\}$	$\xrightarrow{X_A}$	abort if $X_A \notin \langle g \rangle \setminus \{1\}$
$h := X_B^{x_A}$		$h := X_A^{x_B}$
abort if $h = 1$		abort if $h = 1$
$\hat{g} := h \cdot g^s$		$\hat{g} := h \cdot g^s$
	DH2Point End	
choose $y_A \leftarrow \mathbb{Z}_q^*$ at random		choose $y_B \leftarrow \mathbb{Z}_q^*$ at random
$Y_A := \hat{g}^{y_A}$	$\xleftarrow{Y_B}$	$Y_B := \hat{g}^{y_B}$
	$\xrightarrow{Y_A}$	
$K := {Y_B}^{y_A}$		$K := {Y_A}^{y_B}$
$K_{\mathrm{Enc}} := H(K\|\|1)$		$K_{\mathrm{Enc}} := H(K\|\|1)$
$K_{\mathrm{MAC}} := H(K\|\|2)$, $K'_{\mathrm{MAC}} := H(K\|\|3)$		$K_{\mathrm{MAC}} := H(K\|\|2)$, $K'_{\mathrm{MAC}} := H(K\|\|3)$
$T_A := \mathrm{MAC}(K'_{\mathrm{MAC}}, (Y_B, \mathcal{G}, \hat{g}))$		$T_B := \mathrm{MAC}(K'_{\mathrm{MAC}}, (Y_A, \mathcal{G}, \hat{g}))$
	$\xleftarrow{T_B}$	
check correctness of T_B by recomputing it	$\xrightarrow{T_A}$	check correctness of T_A by recomputing it