CRYPTOGRAPHY LECTURE, 2022

Computer Science and Algorithmics, PWr

Mirosław Kutyłowski

Zero Knowledge Protocols

we are not talking about “ignorance” but

about a fundamental concept of not leaking information

Interactive proofs

Actors:

Peggy (Prover): tries to convince Victor about Φ

Victor (Verifier): check whether sentence Φ is true, possible outputs: `Accept`, `Reject`

Completeness:

if Φ is true, and Peggy and Victor follow the protocol, then output=`Accept`

Soundness:

if Φ is false, and Victor follows the protocol, then output=`Reject` with probability at least p (Peggy may attempt to cheat!)

Zero-knowledge property - informally:

Victor should not learn anything except that Φ is true

– seems to be impossible, Peggy and Victor exchange some information!

Example of a protocol where ZK is violated

challenge-response authentication:

1. Verifier chooses r at random
2. Prover creates a signature s of r
3. Verifier checks s with the public key P of Alice

$\Phi =$"Prover knows the secret key corresponding to P"

Completeness: ok

Soundness: based on unforgeability of signatures

ZKP: no! Verifier learns s that is unavailable without executing the protocol.

Verifier gains some knowledge from Prover!

The situation for Schnorr Identification

1. Alice chooses k at random, $r := g^k$
2. Alice sends r to Verifier
3. Verifier chooses c at random and sends to Alice
4. Alice calculates $s := k - x \cdot c \bmod q$ and sends s to Verifier
5. Verifier checks that $g^s \cdot \mathrm{PK}^c = r$

Question: does a protocol transcript (r, c, s) HELP to break x?

Answer: No: the attacker could create himself the triples (r, c, s) with exactly the same probability

(a weaker version would suffice: with probability distribution that is not distinguishable from the real one)

Simulator concept for computational Zero Knowledge

for each Verifier V^* there is a simulator S

such that the transcripts generated with $S \,\mathrm{on}\, V^*$ are computationally indistinguishable from

the transcripts generated during real executions with V^*

Simulator concept for perfect Zero Knowledge

for each Verifier V^* there is a simulator S

such that the transcripts generated with $S \, \mathrm{on} \, V^*$ have exactly the same probability distribution as

the transcripts generated during real executions with V^*

Example: graph isomorphism

sentence Φ: Peggy knows an isomorphism π between graphs G_0 and G_1

key issue: showing the isomorphism by Peggy would violate ZK as graph isomorphism is a hard problem and S cannot simulate it

Protocol ("left or right")

i. Peggy creates a graph G' together with an isomorphism $\rho\colon G_0 \to G'$

ii. given G', Victor chooses bit b at random

iii. if $b = 0$, then Peggy shows isomorphism between G_0 and G' $(\text{that is}, \rho)$, else she shows an isomorphism between G' and G_1 $(\pi \cdot \rho^{-1})$

Simulator for perfect Zero Knowledge

I. choose b' at random

II. create isomorphism $\rho\colon G_0 \rightarrow G'$ and G', if b'=0,

III. create isomorphism $\rho\colon G_1 \rightarrow G'$ and G', if b'=1

IV. simulate V^* until it presents b

V. if $b = b'$ then output transcript (G', b, ρ), else goto I.

Amplifying soundness

- if the correct output is Reject, then the protocol outputs Reject with pbb $\geq p$
- sometimes $p < 1$, e.g. $p = \frac{1}{2}$
- probability amplification: run the protocol k times, `Reject` if at least one run yields `Reject`

  $\rightarrow$ error probability $(1-p)^k$

Honest-verifier zero-knowledge

subtle issues: simulation concerns a verifier that follows the protocol

the situation of possibly dishonest verifier may be different

Proof-of-Knowledge

Language L,

- for each $v \in L$ there is a witness w such that $A(v, w) \Rightarrow v \in L$
- relation A is easy to evaluate

example

$L = \{(g^x, h^x) : x < q\}$ (equality of discrete logarithms)

witness for (g^w, h^w) is w

Zero-knowledge proof of Knowledge

special soundness via knowledge extractor:

if Peggy can run a protocol with (possibly dishonest) Victor, then she may run extractor to learn the witness

Proof of Knowledge: linear relation for discrete logarithms

Peggy knows $y_1 = g_1^{x_1}$ and $y_2 = g_2^{x_2}$ such that $a_1 \cdot x_1 + a_2 \cdot x_2 = b \bmod q$

Peggy has to prove that the discrete logs satisfy this equation. How?

- Peggy can prove that she knows $\log_{g_1} y_1$ (Schnorr identification protocol), ...
- how to prove the equality?
- if $g_1 = g_2 \Rightarrow$ it is easy, if $\log_{g_1} g_2$ is known $\Rightarrow$??

Protocol

i. Peggy chooses $v_{1,}v_2$ such that $a_1 \cdot v_1 + a_2 \cdot v_2 = b \bmod q$

ii. Peggy shows $t_1 := g^{v_1}, t_2 := g^{v_2}$

iii. Verifier chooses c at random

iv. Peggy calculates $r_1 := v_1 - x_1 \cdot c \bmod q$, $r_2 := v_2 - x_2 \cdot c \bmod q$

v. Verifier checks that $g_1^{r_1} \cdot y_1^c = t_1$ and $g_2^{r_2} \cdot y_2^c = t_2$

   and $a_1 \cdot r_1 + a_2 \cdot r_2 = b \cdot (1 - c) \bmod q$

Protocol completeness

i. Peggy chooses $v_{1,}v_2$ such that $a_1 \cdot v_1 + a_2 \cdot v_2 = b \bmod q$

ii. Peggy shows $t_1 {:=} g^{v_1}, t_2 {:=} g^{v_2}$

iii. Verifier chooses c at random

iv. Peggy calculates $r_1 := v_1 - x_1 \cdot c \bmod q$, $r_2 := v_2 - x_2 \cdot c \bmod q$

v. Verifier checks that $g_1^{r_1} \cdot y_1^c = t_1$ and $g_2^{r_2} \cdot y_2^c = t_2$

and $a_1 \cdot r_1 + a_2 \cdot r_2 = b \cdot (1 - c) \bmod q$

Extractor

i. Peggy chooses $v_{1,}v_2$ such that $a_1 \cdot v_1 + a_2 \cdot v_2 = b \bmod q$

ii. Peggy shows $t_1 : = g^{v_1}, t_2 : = g^{v_2}$

iii. Verifier chooses c at random

iv. Peggy calculates $r_1 := v_1 - x_1 \cdot c \bmod q$, $r_2 := v_2 - x_2 \cdot c \bmod q$

v. Verifier checks that $g_1^{r_1} \cdot y_1^c = t_1$ and $g_2^{r_2} \cdot y_2^c = t_2$

and $a_1 \cdot r_1 + a_2 \cdot r_2 = b \cdot (1 - c) \bmod q$

- . Run it twice with the same t_1, t_2
- . $r_1 - r_1' = (v_1 - x_1 \cdot c) - (v_1 - x_1 \cdot c') = x_1 \cdot (c' - c)$
- . similarly for x_2

Sigma protocols

a frequent form:

Peggy: commitment

Victor: challenge

Peggy: response

Victor: test and Accept/Reject

Proofs of knowledge for more complicated statements

e.g. "I know x such that $g^x = y$ and this x does not satisfy $h^x = z$ "

(this is "*Proof-of-knowledge of inequality of discrete logarithms*")

commitment: $(a_0, a_1, a_2) = (z^r h^{-r \cdot x}, y^{r_1} g^{r_2}, z^{r_1} h^{r_2})$ for random r, r_1, r_2

challenge: c

response: $(t_1, t_2) = (r_1 + c \cdot r, r_2 - c \cdot r \cdot x)$

test: $y^{t_1} g^{t_2} = a_1, \quad z^{t_1} h^{t_2} = a_2 \cdot a_0^{-c}$, and $a_0 \neq 1$

commitment: $(a_0, a_1, a_2) = (z^r h^{-r \cdot x}, y^{r_1} g^{r_2}, z^{r_1} h^{r_2})$ for random r, r_1, r_2

challenge: c

response: $(t_1, t_2) = (r_1 + c \cdot r, r_2 - c \cdot r \cdot x)$

test: $y^{t_1} g^{t_2} = a_1$, $z^{t_1} h^{t_2} = a_2 \cdot a_0^{-c}$, and $a_0 \neq 1$

Extractor

AND proofs

run two sigma protocols independently in parallel

Peggy: commitment 1, commitment 2

Victor: common challenge

Peggy: response 1, response 2

Victor: test 1, test 2

Example: "proof of knowledge of discrete log of y and of discrete log of z"

OR proof – typical trick for Sigma protocols

Peggy knows witness for sentence 2, but not for sentence 1:

i. commitment:

   a. Peggy runs simulator for sentence 1, gets transcript (a_1, c_1, r_1)

   b. Peggy creates a challenge a_2 for sentence 2

   commitemnt is $(a_{1,}a_2)$

ii. challenge: c

iii. Peggy splits c: $c = c_1 + c_2$, creates response r_2 for (a_2, c_2),

   final response $(c_{1,}r_{1,}c_{2,}r_2)$

iv. Victor separately checks $(r_{1,}c_1)$ and $(r_{2,}c_2)$, and that $c = c_1 + c_2$

NIZKP - Non-Interactive Zero-Knowledge Proof

replace the challenge by hash of the commitment

(the same idea as Fiat-Shamir heuristics but with no message to be signed under hash)