Lecture 10
Commitment Schemes and Zero-Knowledge Protocols

Stefan Dziembowski
University of Rome
La Sapienza

**BiSS 2009
Bertinoro International
Spring School
2-6 March 2009**

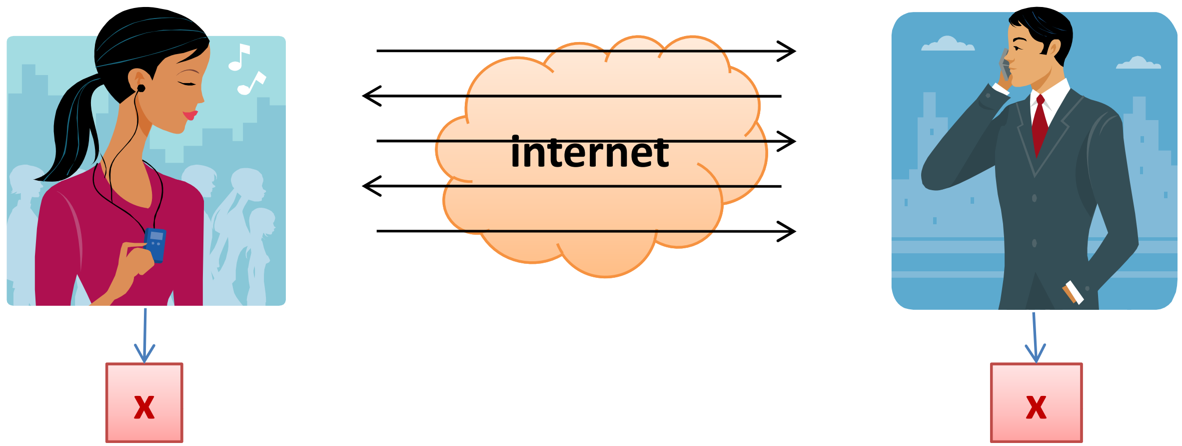
Plan

1. Coin-flipping by telephone
2. Commitment schemes
   1. definition
   2. construction based on QRA
   3. construction based on discrete log
   4. construction based on PRG
3. Zero-knowledge (ZK)
   1. motivation and definition
   2. ZK protocol for graph isomorphism
   3. ZK protocol for Hamiltonian cycles
   4. applications

Coin-flipping by telephone [Blum'81]

privacy and authenticity is not a problem

Suppose Alice and Bob are connected by a secure internet link:



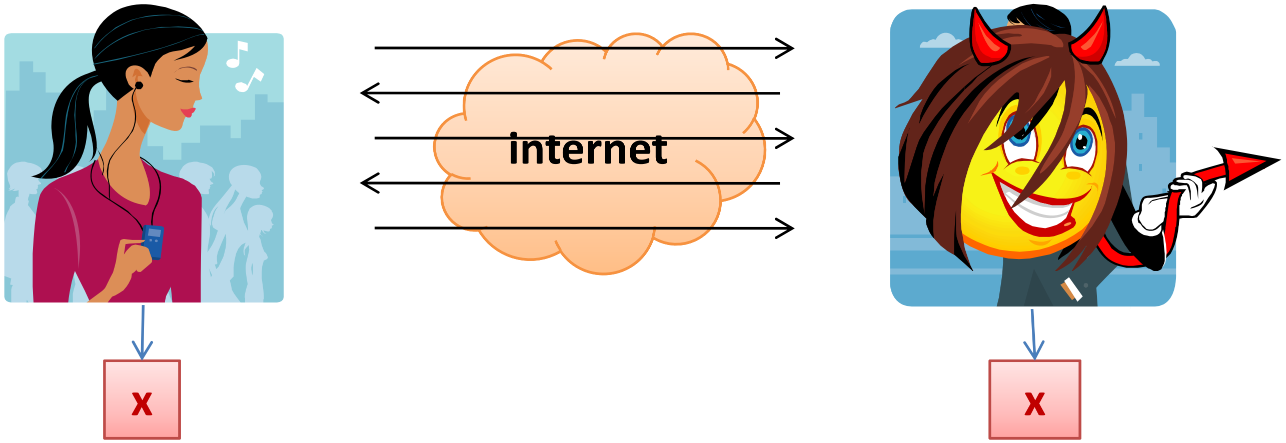
The goal of Alice and Bob is to toss a coin.

In other words:
They want to execute some protocol **π** in such a way that
at the end of the execution they both output the same bit **x** distributed uniformly over **{0,1}** .

How to define security? [1/2]

Let us just stay at an informal level...

From the point of view of Alice:

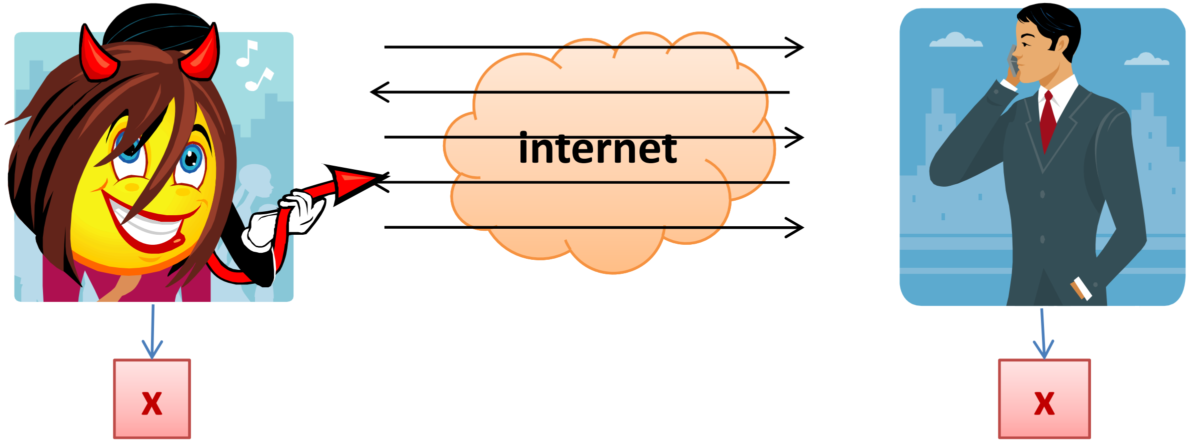


even if **Bob** is **cheating** (i.e.: he doesn't follow the protocol):

if the protocol terminates successfully, then **x** has a uniform distribution

How to define security? [2/2]

The same holds from the point of view of Bob



even if Alice is **cheating** (i.e.: he doesn’t follow the protocol):

if the protocol terminates successfully, then **x** has a uniform distribution

Note the difference

Unlike what we saw on the previous lectures:

the enemy can be one of the parties

(**not** an external entity)

A cheating party is sometimes called a **corrupted** party, or a **malicious** party.

We will see many other examples of this later!

How to solve this problem?

Idea

Remember the old game:

rock-paper-scissors?

		Alice		
				
		draw	Alice wins	Bob wins
Bob		Bob wins	draw	Alice wins
		Alice wins	Bob wins	draw

Let's simplify this game

		Alice	
		A=0	A=1
Bob	B=0	Alice wins	Bob wins
	B=1	Bob wins	Bob wins

In other words: **Alice** wins iff **A xor B = 0**.

Another way to look at it

Bob
has an input A

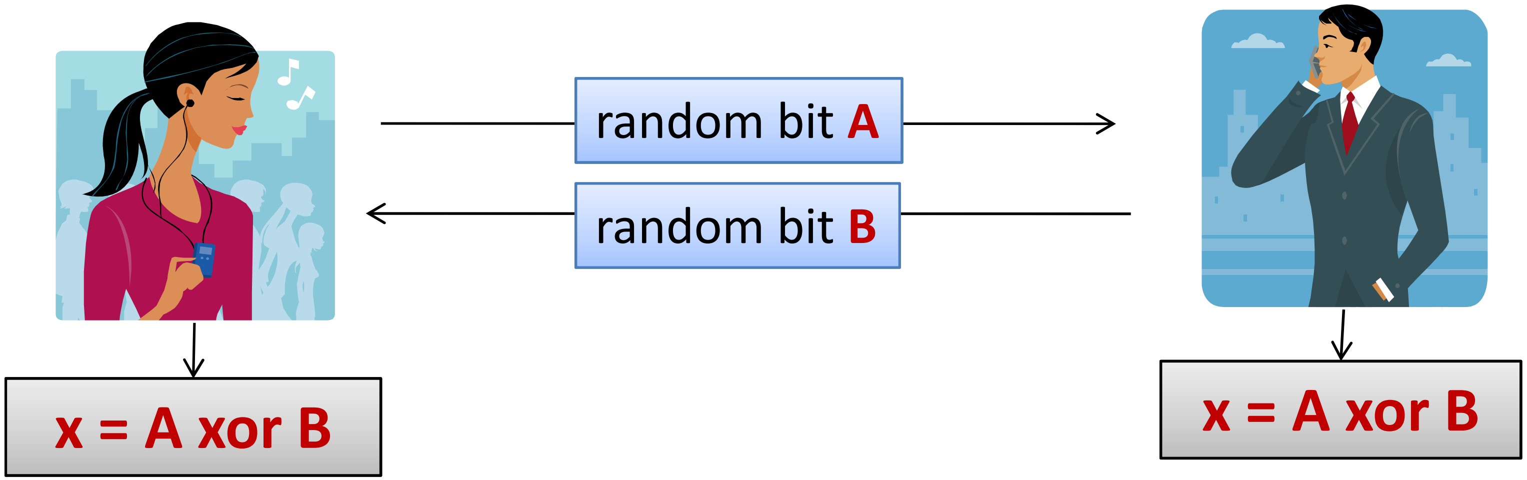
Alice
has an input B

they should jointly compute

x = A xor B

(in a secure way)

What to do?



Problem:

A and B should be sent at the same time

(e.g. if A is sent before B then a malicious Bob can set B := x xor A, where x is chosen by him).

How to guarantee this?

Seems hard:

the internet is not synchronous...

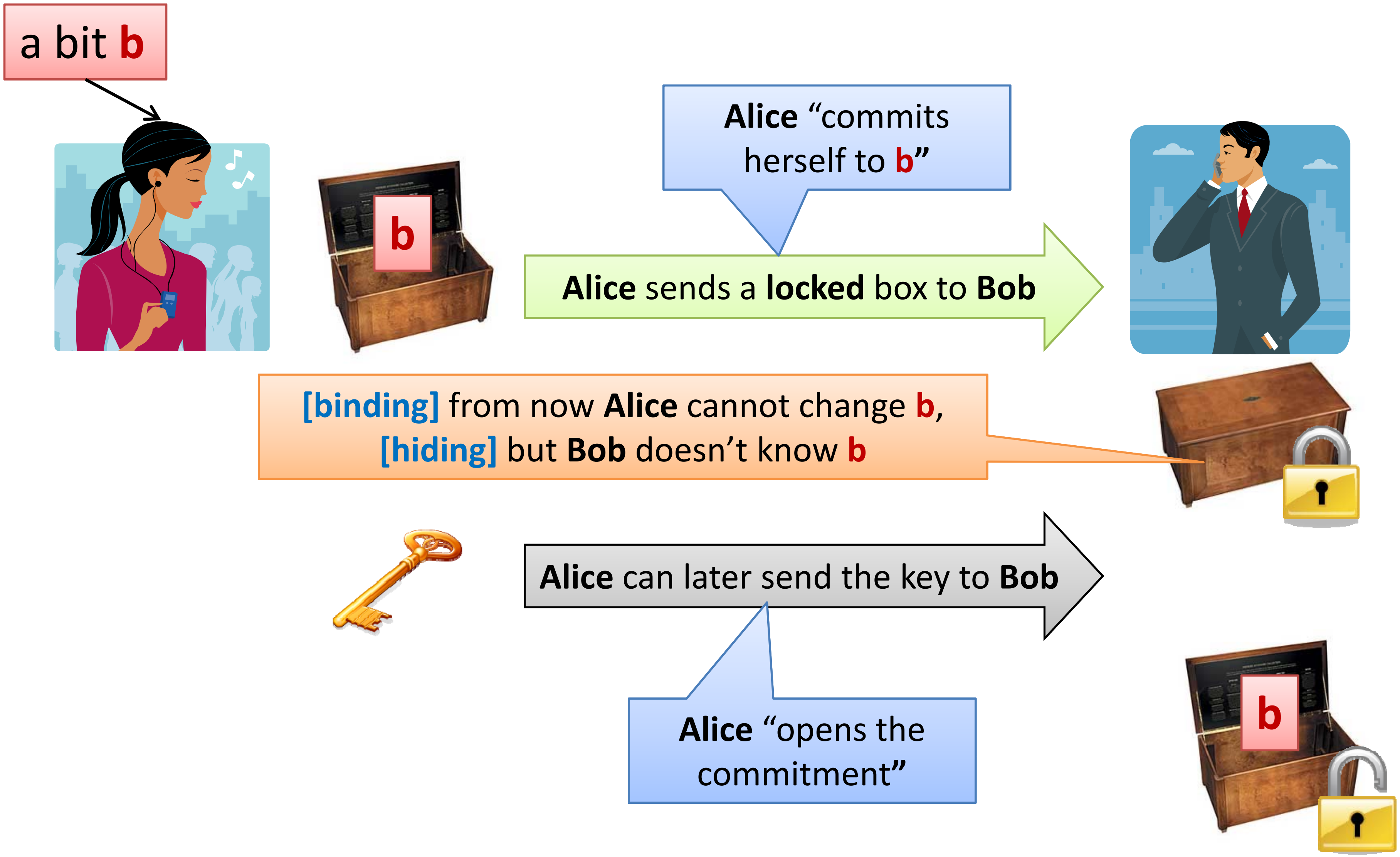
A solution:

bit commitments

Plan

1. Coin-flipping by telephone
2. Commitment schemes
    1. definition
    2. construction based on QRA
    3. construction based on discrete log
    4. construction based on PRG
3. Zero-knowledge (ZK)
    1. motivation and definition
    2. ZK protocol for graph isomorphism
    3. ZK protocol for Hamiltonian cycles
    4. Applications

Commitment schemes – an intuition



Commitment schemes – a functional definition

A **commitment scheme** is a protocol executed between Alice and Bob consisting of two phases: **commit** and **open**.

In the **commit** phase:

- **Alice** takes some input bit **b**.
- **Bob** takes no input.

In the **open** phase:

- **Alice** outputs nothing
- **Bob** outputs **b**, or **error**

Security requirements - informally

[binding]
After the **commit** phase there exists at most one value **b** that can be open in the **open** phase.

[hiding]
As long as the **open** phase did not start **Bob** has no information about **b**.

How to define security formally?

Not so trivial – remember that the parties can misbehave arbitrarily.

We do not present a complete definition here.

(The hiding property can be defined using the "**indistinguishability**" principle.)

The definition depends on some options.

1. What is the computational power of a **cheating Alice**?
2. What is the computational power of a **cheating Bob**?

The computational power of the adversary

If a cheating Alice can be infinitely powerful, we say that the protocol is **unconditionally binding**.

Otherwise it is **computationally binding**.

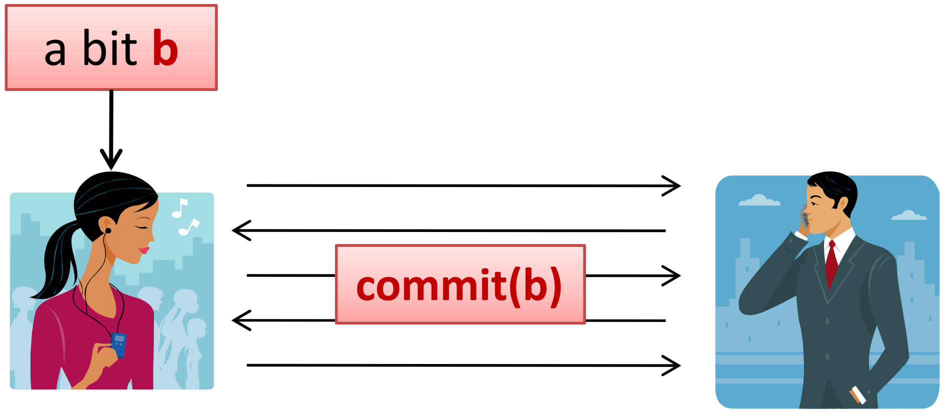
If a cheating Bob can be infinitely powerful, we say that the protocol is **unconditionally hiding**.

Otherwise it is **computationally hiding**.

Of course, to be formal we would need to introduce a security parameter...

Unconditionally hiding and binding commitment schemes do not exist

Proof (intuition)



There are two options:

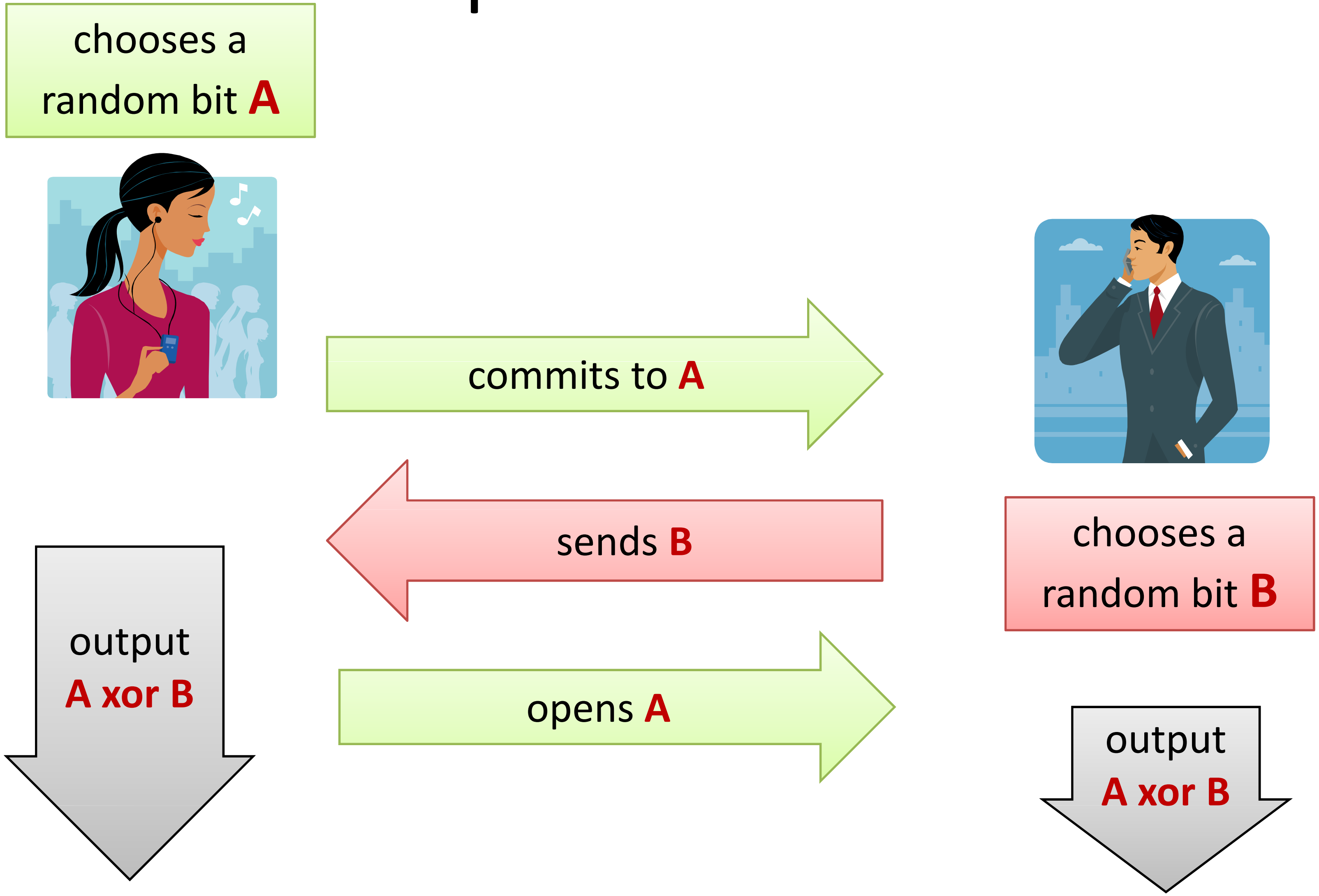
1. there exists a way to open **1-b**, or

   in this case Alice can cheat

2. there doesn't exist such a way

   in this case Bob can learn **b**

So, how does it solve the coin-flipping problem?



Problem

Alice can always refuse to send the last message.

This is unavoidable (there has to be the **last message** in the protocol).

But they can use a convention:
if **Alice** didn't send the last message – she lost!

Plan

1. Coin-flipping by telephone
2. Commitment schemes
    1. definition
    2. construction based on QRA
    3. construction based on discrete log
    4. construction based on PRG
3. Zero-knowledge (ZK)
    1. motivation and definition
    2. ZK protocol for graph isomorphism
    3. ZK protocol for Hamiltonian cycles
    4. Applications

A construction based on **QRA**

a bit **b**

selects a random **RSA** modulus **N=pq**.

Let

$$x := \begin{cases} \text{a random } QR_N & \text{if } b = 0 \\ \text{a random } Z^{+}_N \setminus QR_N & \text{if } b = 1 \end{cases}$$

to commit to **b** **Alice** sends **(N,x)**

to open a commitment **Alice** sends **(p,q)**.

if **N ≠ pq**.
then Bob outputs **error**

if **x** is a QR_N outputs **b=0**
if **x** is not a QR_N outputs **b=1**

This commitment scheme is unconditionally binding

Why?

Suppose **Alice** has sent **(N,x)** to **Bob**.

What can **Bob** output at the end of the opening phase?

There exists the following options:

- **N** is not an **RSA** modulus – in this case **Bob** will always output **error**,
- **x** is a QR_N – in this case **Bob** can only output **0** or **error**,
- **x** is not a QR_N – in this case **Bob** can only output **1** or **error**.

This commitment scheme is computationally hiding, assuming QRA holds

Proof (intuition)

To distinguish between **b=0** and **b=1** a malicious **Bob** would need to distinguish QR_N from the other elements of Z^{+}_{N} ...

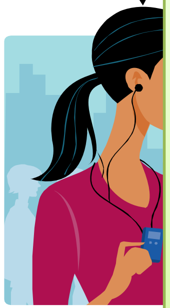
Plan

1. Coin-flipping by telephone
2. Commitment schemes
    1. definition
    2. construction based on QRA
    3. construction based on discrete log
    4. construction based on PRG
3. Zero-knowledge (ZK)
    1. motivation and definition
    2. ZK protocol for graph isomorphism
    3. ZK protocol for Hamiltonian cycles
    4. applications

A construction based on discrete log

Selects
p – a random prime,
g – a generator of QR_p
s – a random element of QR_p

a bit **b**



checks if **p** is prime and
g and **s** are in QR_p
selects a random **y** from $Z_{(p-1)/2}$

Let

$$x := \begin{cases} g^y & \text{if } b = 0 \\ s \cdot g^y & \text{if } b = 1 \end{cases}$$

p,g,s

to commit to **b** **Alice** sends **x**

to open a commitment **Alice** sends **y**.

if $x = g^y$ outputs **b=0**
if $x = s \cdot g^y$ outputs **b=1**

This commitment scheme is computationally binding, assuming that the discrete log is hard in QR_p

Proof (intuition)

To be able to open the commitment in two ways, a **cheating Alice** needs to know **y** and **y'** such that there exists **x** such that

$$g^{y} = x = s \cdot g^{y'}$$

But this means that $g^{y-y'} = s$. So, she would know the discrete log of **s**.

This commitment scheme is unconditionally hiding

Why?

It is easy to see that **x** is just a random element of $\mathbf{QR_p}$

Plan

1. Coin-flipping by telephone
2. Commitment schemes
    1. definition
    2. construction based on QRA
    3. construction based on discrete log
    4. construction based on PRG
3. Zero-knowledge (ZK)
    1. motivation and definition
    2. ZK protocol for graph isomorphism
    3. ZK protocol for Hamiltonian cycles
    4. applications

A construction based on PRGs [Naor'91]

G : $\{0,1\}^L \rightarrow \{0,1\}^{3L}$ -- a PRG

a bit **b**

selects
X $\in \{0,1\}^{3L}$ – a random string

X

selects
Z $\in \{0,1\}^L$ – a random string

$$Y := \begin{cases} G(Z) \text{ xor } X & \text{if } b = 0 \\ G(Z) & \text{if } b = 1 \end{cases}$$

to commit to **b** **Alice** sends **Y**

to open a commitment **Alice** sends **Z**.

if **Y = G(Z) xor X** outputs **b=0**
if **Y = G(Z)** outputs **b=1**

This commitment scheme is unconditionally binding

Proof (intuition)

To be able to open the commitment in two ways, a **cheating Alice** needs to find **Z** and **Z'** such that there exists **Y** such that:

$$G(Z) \text{ xor } X = Y = G(Z')$$

This means that **G(Z) xor G(Z') = X**.

How many **X**'s have the property that
there exist **Z** and **Z'** such that **G(Z) xor G(Z') = X** ?
By the counting argument: at most $(2^L)^2 = 2^{2L}$.

Therefore, the probability that a **random** $X \in \{0,1\}^{3L}$ has this property is at most $2^{2L} / 2^{3L} = 2^{-L}$.

QED

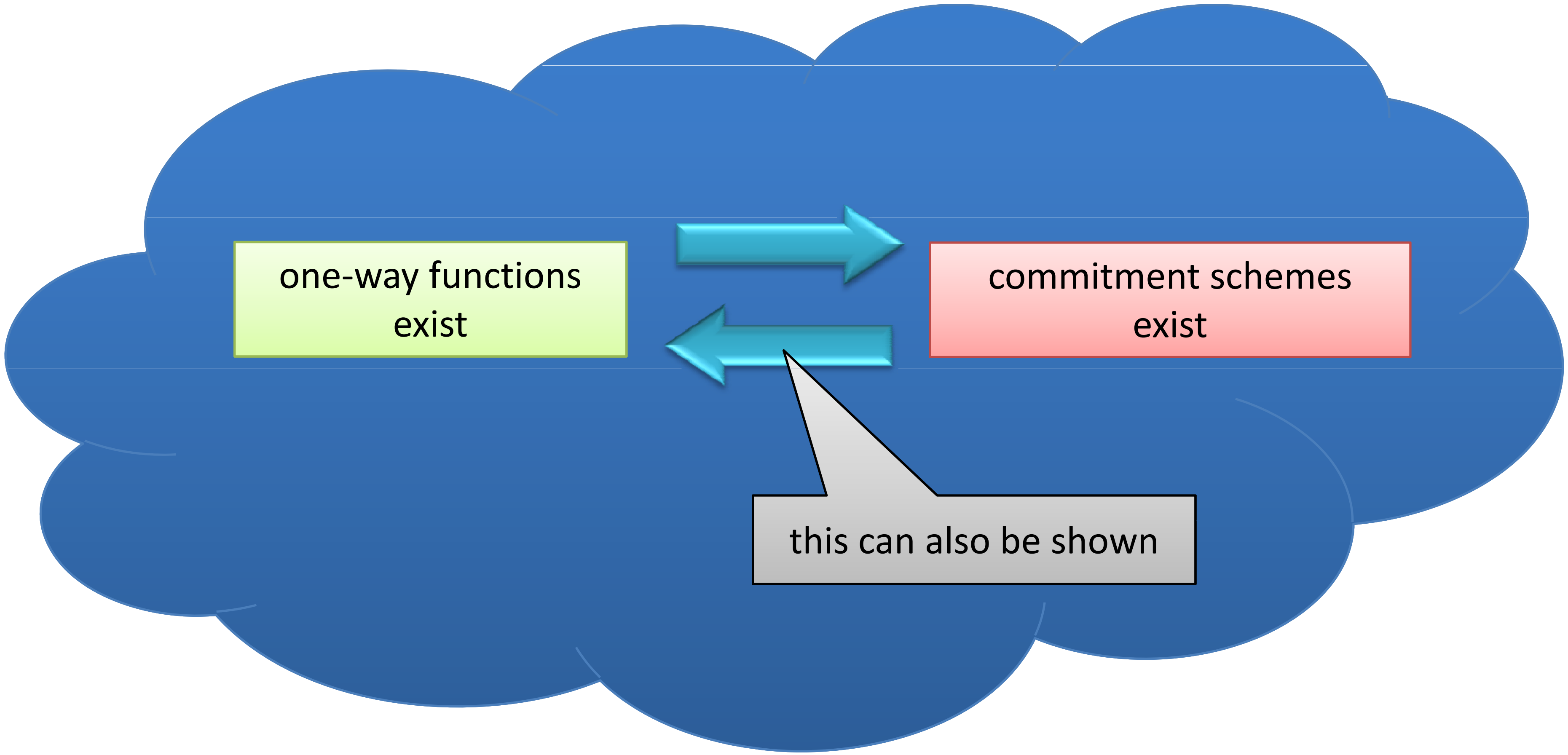
This commitment scheme is computationally hiding, assuming G is a secure PRG

Why?

Obviously, if, instead of **G(Z)** **Alice** uses a completely random string **R**, then the scheme is secure against a **cheating Bob**.

If a scheme behaved differently with **R** and with **G(Z)**, then a **cheating Bob** could be used as a distinguisher for **G**.

Moral



Commitment schemes are a part of **Minicrypt**!

Plan

1. Coin-flipping by telephone
2. Commitment schemes
    1. definition
    2. construction based on QRA
    3. construction based on discrete log
    4. construction based on PRG
3. Zero-knowledge (ZK)
    1. motivation and definition
    2. ZK protocol for graph isomorphism
    3. ZK protocol for Hamiltonian cycles
    4. applications

Zero-knowledge (ZK)

We will now talk about the **zero-knowledge proofs**.

Informally:
A proof of some statement **φ** is **zero-knowledge**, if it doesn't reveal any information (other than that **φ** holds).

We will now explain what it means...

A motivating example: public-key identification

(Enc,Dec) – a **public key** encryption scheme

(pk,sk) – a **(public key, private key)** pair of **Alice**

Everybody the knows **pk** can verify the identity of **Alice**

pk

Alice

Take a random message **m**

c := Enc(pk, m)

m’ := Dec(sk,c)

Check if **m = m’**

Is it secure?

(we didn't define security, so this is just an informal question)

To impersonate **Alice** one needs to be able to decrypt **c** without the knowledge of **m**.

What does the verifier learn about **sk**?

If the verifier follows the protocol – he doesn't learn anything that he didn't know before (he already knows **m**).

But what if the verifier is malicious?

Alice acts as a decryption oracle!
(so he learns something that he didn't know)

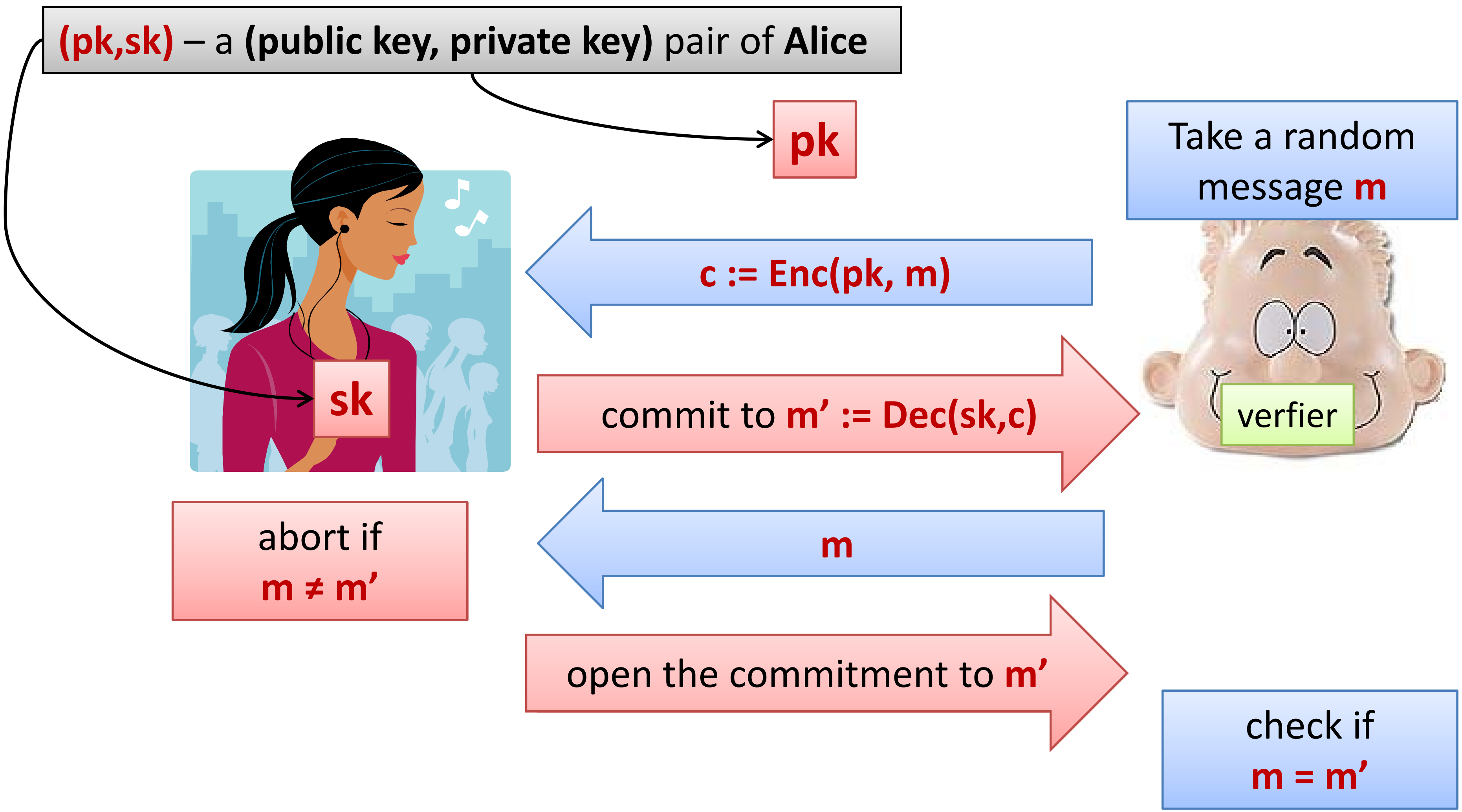
is it a problem – depends on the application

A question

Is it possible to design a protocol where

- a verifier learns nothing,
- besides of the fact that he is talking to Alice?

A new variant of the protocol



Can a malicious verifier learn something from this protocol?

Intuition:

No, because he

doesn't learn **m'**

(he already knows **m'**).

Can this be proven formaly?

Yes!

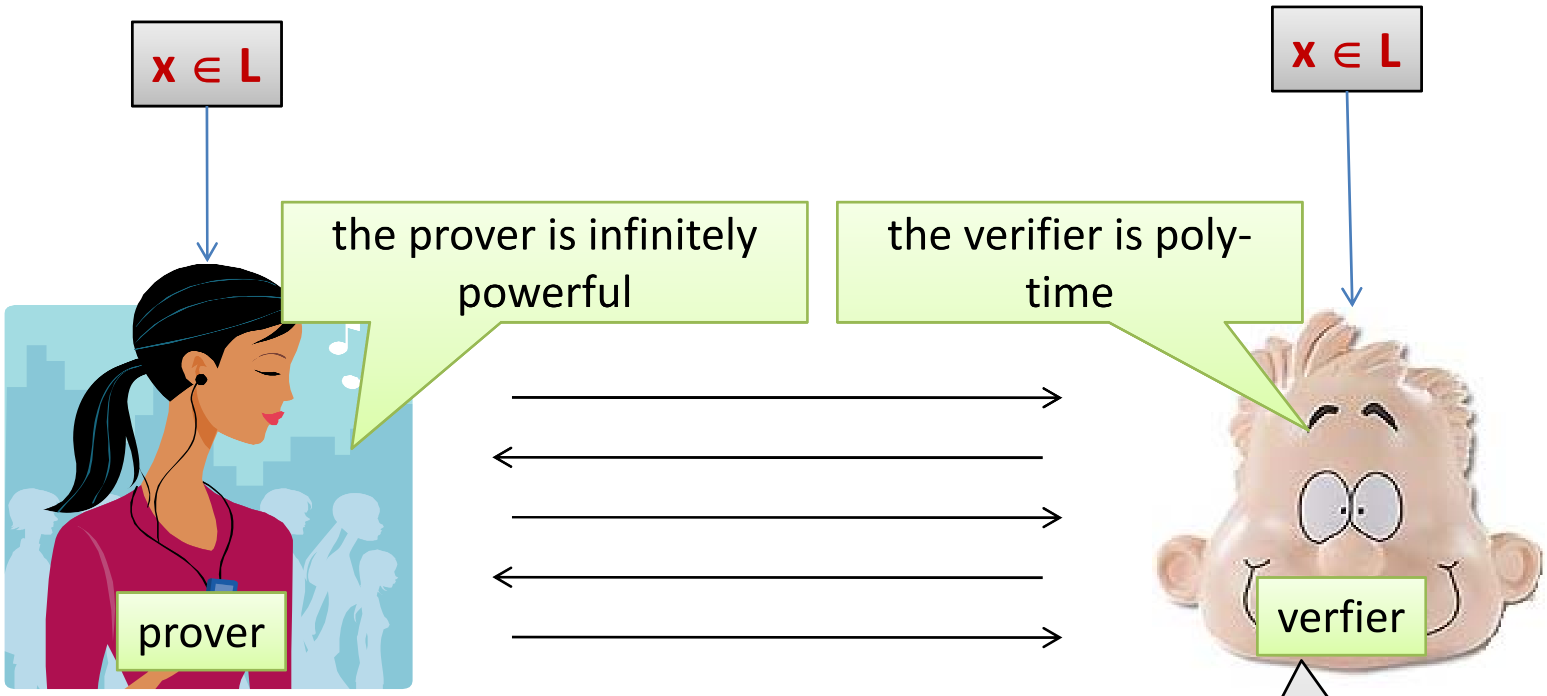
But we first need to

define what it means that "the verifier learns nothing".

This will lead us to the concept of **zero knowledge**

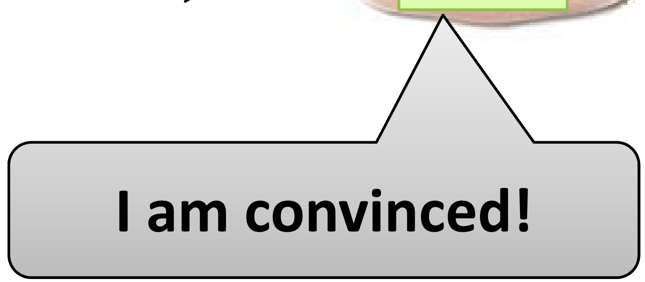
The general picture

L – some language (usually not in **P**)



Two main properties:

1. soundness
2. zero-knowledge

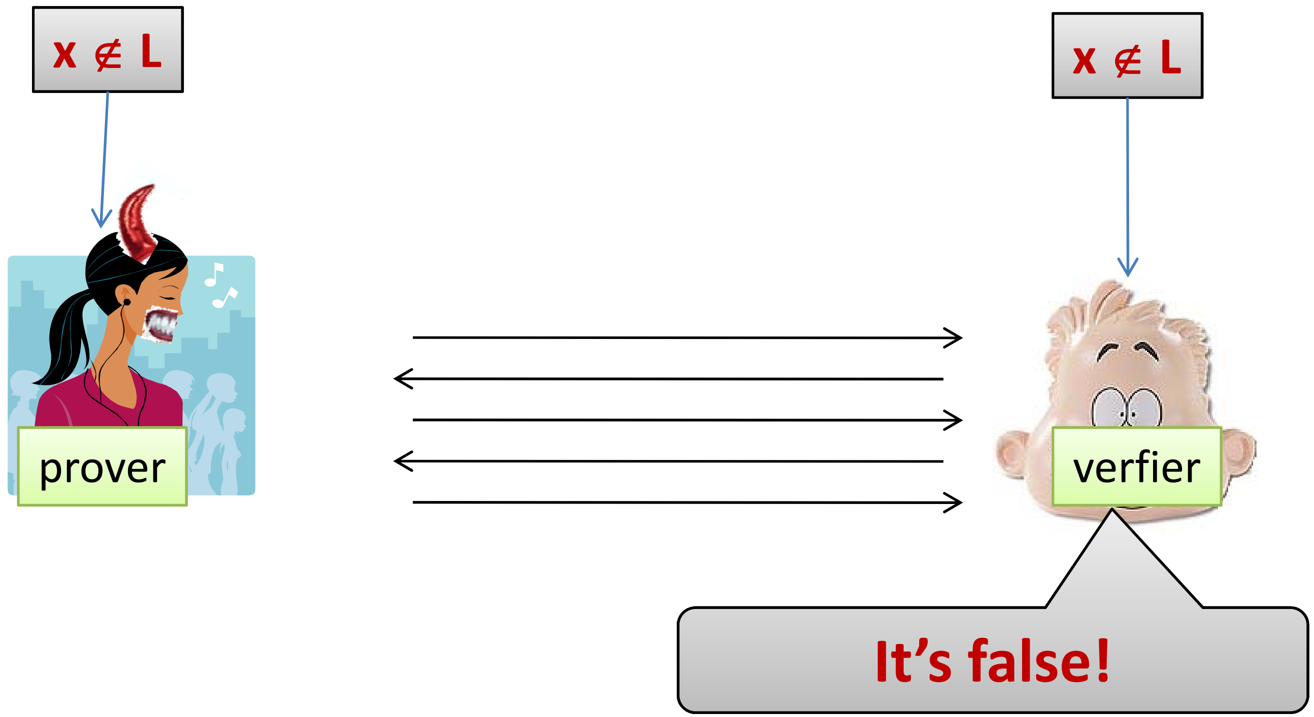


Soundness - informally

A cheating prover cannot convince the verifier that

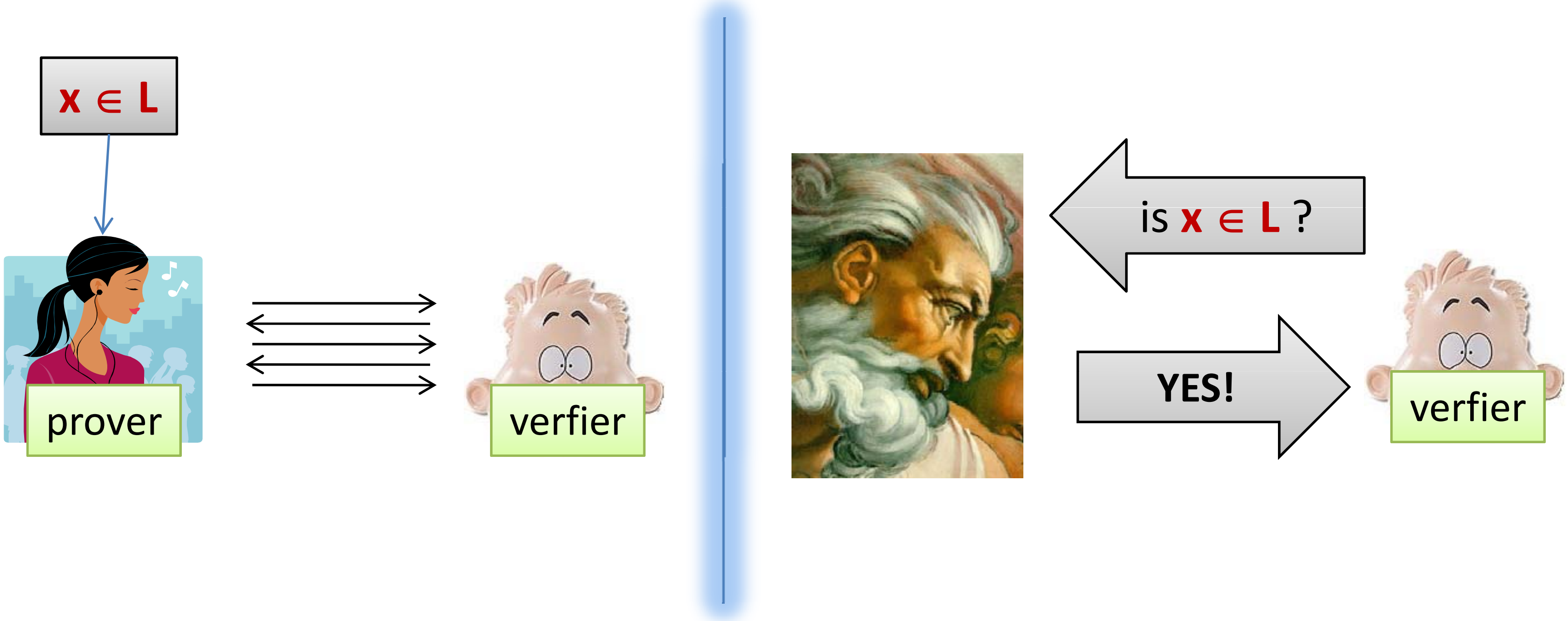
$x \in L$

if it is not true (negligible error probability is allowed)



Zero Knowledge

The only thing that the verifier should learn is that **x ∈ L**

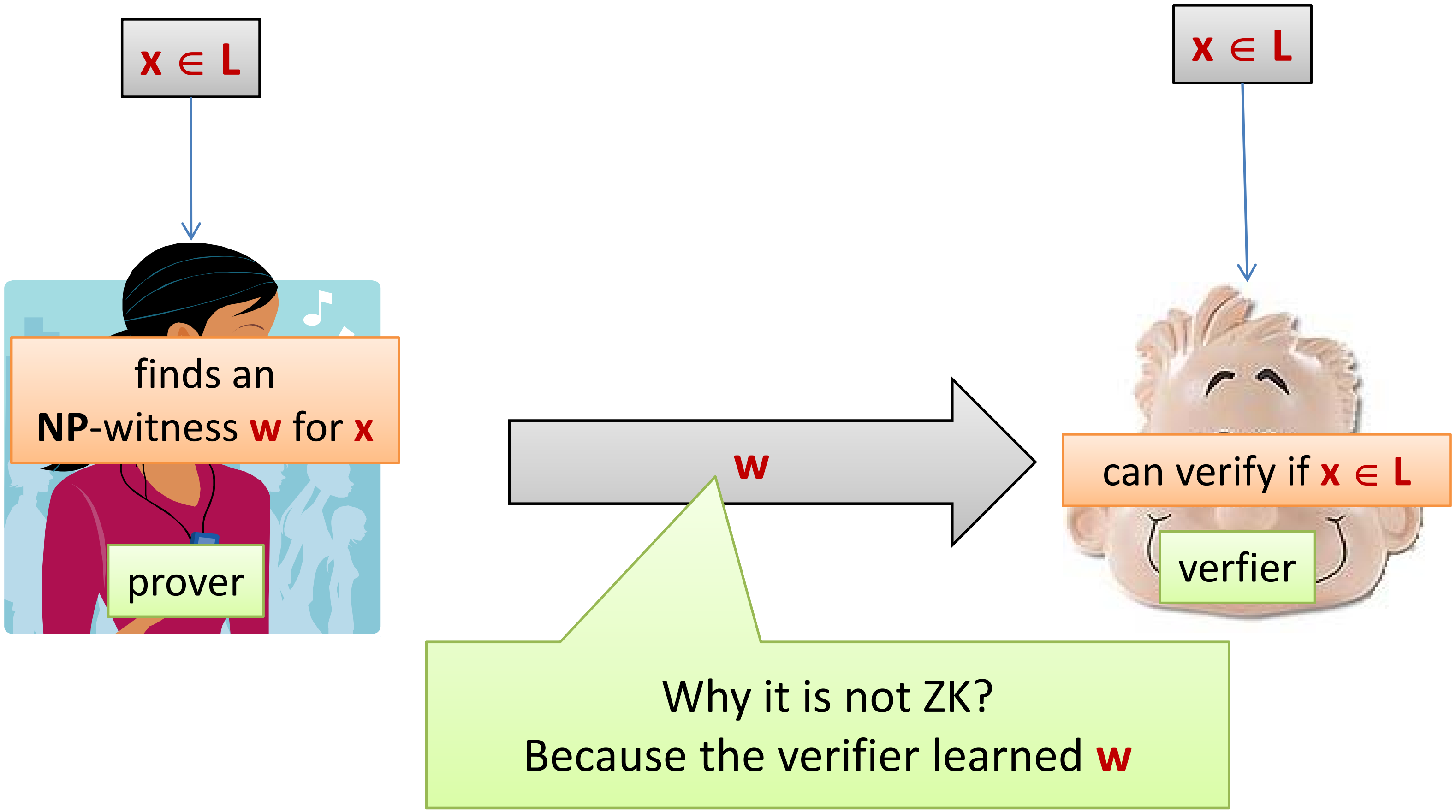


This should hold even if the verifier doesn't follow the protocol.

(again: we allow some negligible error)

An example of a protocol that is **not** Zero Knowledge

L – some NP-complete language



Notation

Suppose we are given a protocol consisting of two randomized machines **P** and **V**.

Suppose **P** and **V** take some common input **x**, and then **V** outputs **yes** or **no**.

We say that **(P,V) accepts x** if **V** outputs **yes**. Otherwise we say that it rejects **x**.

View(P,V,x) – a random variable denoting the "view of **V**", i.e.:

1. the random input of **V** and the input **x**,
2. the transcript of the communication.

Zero-knowledge proofs

A pair **(P, V)** is a **zero-knowledge proof system** for **L** if it satisfies the following conditions:

- **P** has an infinite computing power and **V** is poly-time.
- **Completeness**: If **x** $\in$ **L**, then the probability that **(P, V)** rejects **x** is negligible in the length of **x**.
- **Soundness**: If **x** $\notin$ **L** then for **any prover P***, the probability that **(P*, V)** accepts **x** is negligible in the length of **x.**
- **Zero-Knowledge**: "a cheating **V** should not learn anything except of the fact that **x** $\in$ **L**"

How to define it formally?

“a cheating **V*** should not learn anything more than fact that **$x \in L$**”

“What a cheating **V*** can learn can be simulated without interacting with **P**”

Definition (main idea)

For every (even malicious) poly-time **V*** there exists an (expected) poly-time machine **S** such that
$\{View(P,V^*,x)\}_{x \in L}$ is “***indistinguishable from***” $\{S(x)\}_{x \in L}$

What does it mean?

Indistinguishability

Let

$$\alpha = \{A(x)\}_{x \in L} \text{ and } \beta = \{B(x)\}_{x \in L}$$

be two sets of distributions.

α and β are **computationally indistinguishable** if for every poly-time **D** there exists a negligible function ε such that for every $x \in L$

$$|P(D(x, A(x)) = 1) - P(D(x, B(x)) = 1)| < \varepsilon(|x|) \qquad (*)$$

α and β are **statistically indistinguishable** if **(*)** holds also for infinitely powerful **D**.

α and β are **perfectly indistinguishable** if **(*)** holds also for infinitely powerful **D**, and $\varepsilon = 0$.

“a cheating **V** should not learn anything besides of the fact that **$x \in L$**”

Definition (a bit more formally)

For every (even malicious) poly-time **V^*** there exists an (expected) poly-time machine **S** such that

$$\{View(P,V^*,x)\}_{x \in L}$$

is computationally indistinguishable from $\{S(x)\}_{x \in L}$

This is a definition of a **computational** **zero-knowledge**.

By changing the “**computational** indistinguishability” into

- “**statistical** indistinguishability” we get a **statistical** **zero-knowledge**
- “**perfect** indistinguishability” we get a **perfect** **zero-knowledge**

Plan

1. Coin-flipping by telephone
2. Commitment schemes
    1. definition
    2. construction based on QRA
    3. construction based on discrete log
    4. construction based on PRG
3. Zero-knowledge (ZK)
    1. motivation and definition
    2. ZK protocol for graph isomorphism
    3. ZK protocol for Hamiltonian cycles
    4. applications

Graph isomorphism

A **graph** is a pair **(V,E)**, where **E** is a binary symmetric relation on **V**.

A **graph isomorphism between (V,E) and (V',E')** is a bijection:

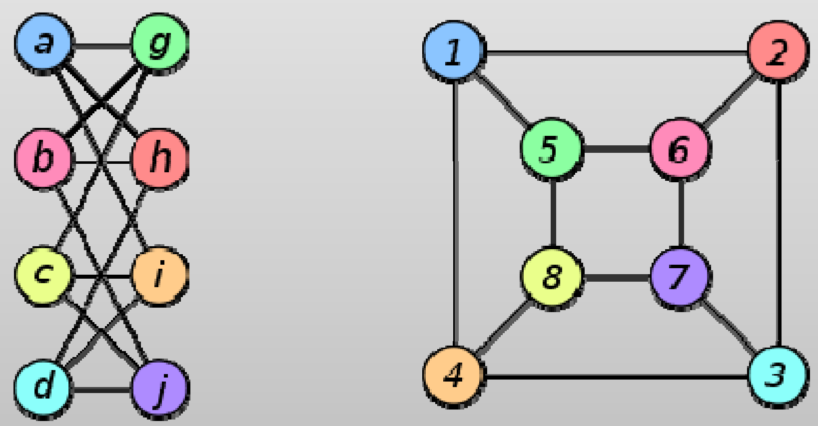
$$\phi : V \rightarrow V'$$

such that

$$(e_1,e_2) \in E \text{ iff } (\phi(e_1), \phi(e_2)) \in E$$

Graphs **G** and **H** are **isomorphic** if there exists an isomorphism between them.

Example



isomorphism:

f(a) = 1
f(b) = 6
f(c) = 8
f(d) = 3
f(g) = 5
f(h) = 2
f(i) = 4
f(j) = 7

Hardness of graph isomorphism

No poly-time algorithm for the graph isomorphism problem is known.

Without loss of generality we will consider only isomorphisms between **(V,E)** and **(V',E')**, where

$$V = V' = \{1,\dots,n\} \text{ (for some } n\text{).}$$

That is, a bijection:

$$\phi : V \to V'$$

is a permutation of the set **{1,...,n}**.

Notation

If **G = (V,E)** is a graph, and

π : V $\longrightarrow$ V is a permutation

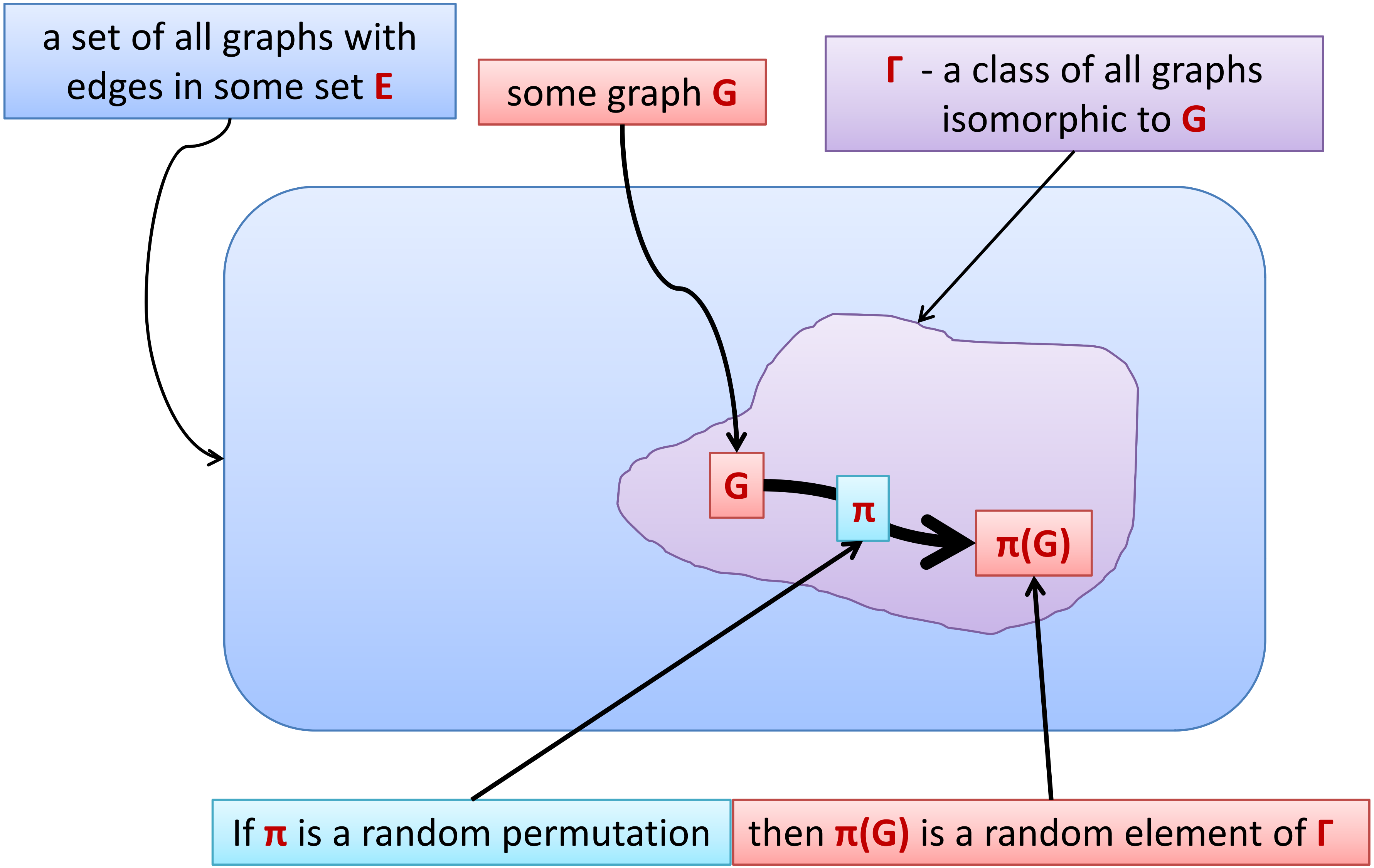
then by **π(G)** we mean a graph

G'=(V',E')

where

(a,b) $\in$ E iff (π(a), π(b)) $\in$ E'

A fact



a set of all graphs with edges in some set E
some graph G
Γ - a class of all graphs isomorphic to G
G
π
π(G)
If π is a random permutation
then π(G) is a random element of Γ

A zero knowledge proof of graph isomorphism – a wrong solution

input
two isomorphic graphs
G_0 and G_1

prover

computes the isomorphism ϕ between G_0 and G_1

ϕ

verifier

checks if ϕ is an isomorphism between G_0 and G_1

A zero knowledge proof of graph isomorphism

input
two isomorphic graphs G_0 and G_1

prover

selects a random permutation π

iterate **n** times:

→ $H := \pi(G_1)$

← a random $i \in \{0,1\}$

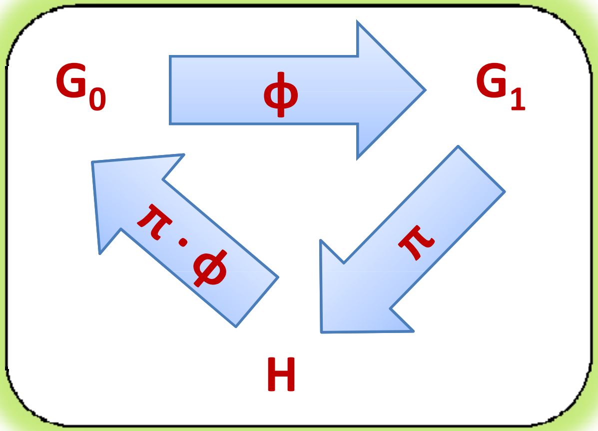
→ an isomorphism between **H** and G_i

verifier

accepts only if the answer is correct

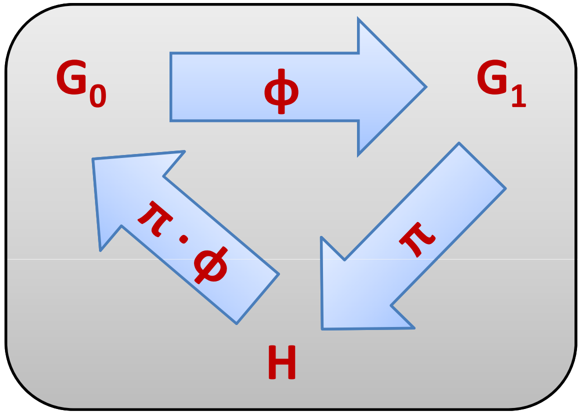
Note:
Prover does not need to be infinitely powerful, if he knows the isomorphism isomorphism ϕ between G_0 and G_i .

- if $i=1$ then he just sends π
- if $i=0$ then he sends $\pi \cdot \phi$



Why is this a zero-knowledge proof system?

- **Completeness**: trivial
- **Soundness**:
  Suppose G_0 and G_1 are **not** isomorphic



Then, one of the following has to hold:

- G_0 and H are **not** isomorphic
- H and G_1 are **not** isomorphic

probability that a verifier rejects is at least **0.5**.

Since the protocol is repeated **n** times, the probability that the verifier rejects is $1 - 0.5^n$

Setting $n := |G_0| + |G_1|$ we are done!

Zero-knowledge?

Intuitively, the zero-knowledge property comes from the fact that:

The only thing that verifier learns is a permutation between:

- **G_0** or **G_1**

  and

- **H** – a random permutation of **G_0** (which is also a random permutation of **G_1**).

In fact: we can show that this is a **perfect zero knowledge proof system**.

More formally

For every poly-time

malicious verifier **V***

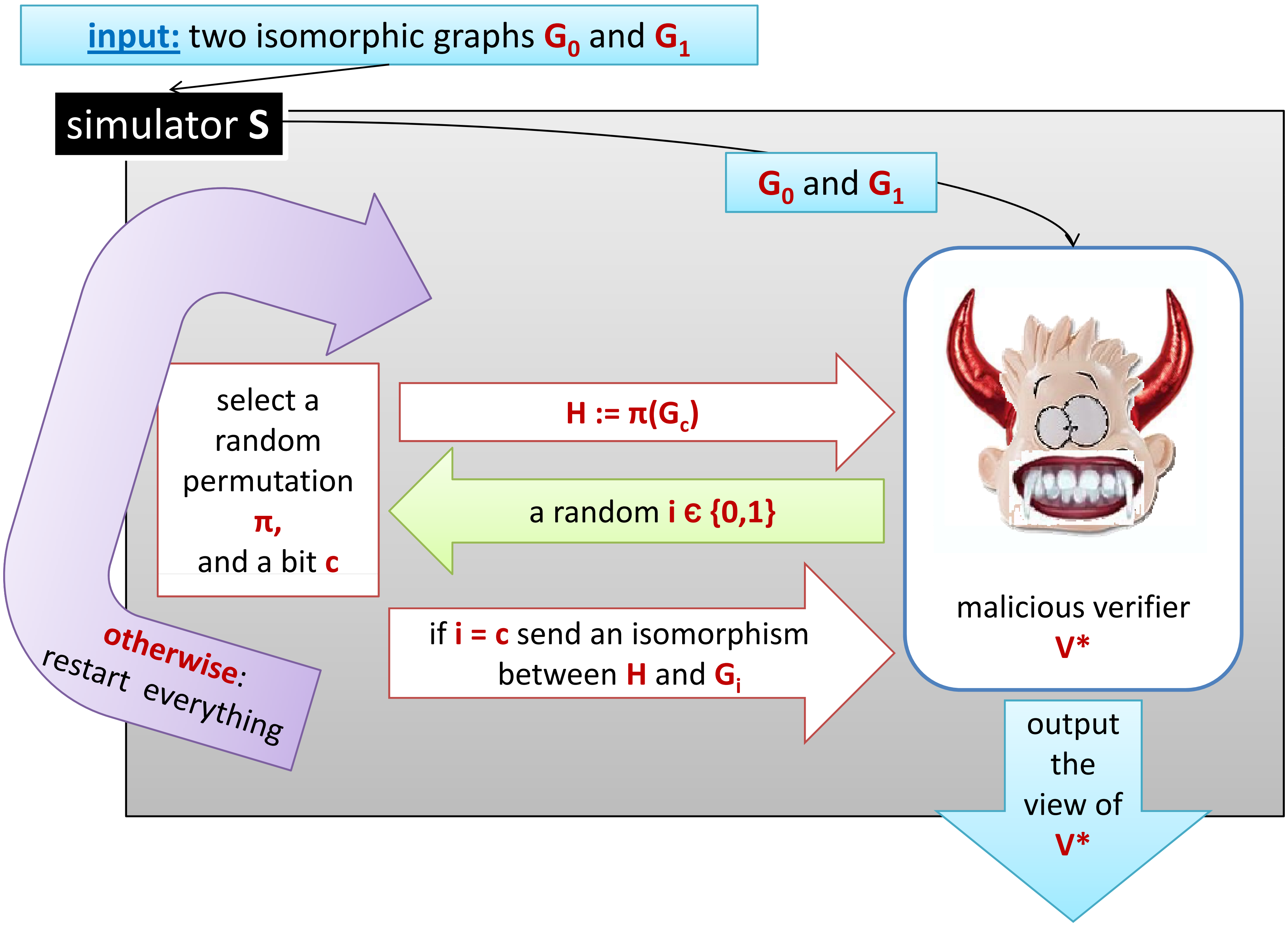
there exists an (expected) poly-time

simulator S

such that

$\{\mathrm{View}(P,V^*,x)\}_{x \in L}$

is perfectly indistinguishable from $\{S(x)\}_{x \in L}$



input: two isomorphic graphs G_0 and G_1
simulator S
G_0 and G_1
select a random permutation π, and a bit c
H := π(G_c)
a random i є {0,1}
if i = c send an isomorphism between H and G_i
otherwise: restart everything
malicious verifier V*
output the view of V*

The running time

First, observe, that the distribution of **H** doesn't depend on **c** (since it is uniform in the class of graphs isomorphic with G_0 and G_1)

Therefore the probability that **S** needs to restart **V*** is equal to **0.5**.

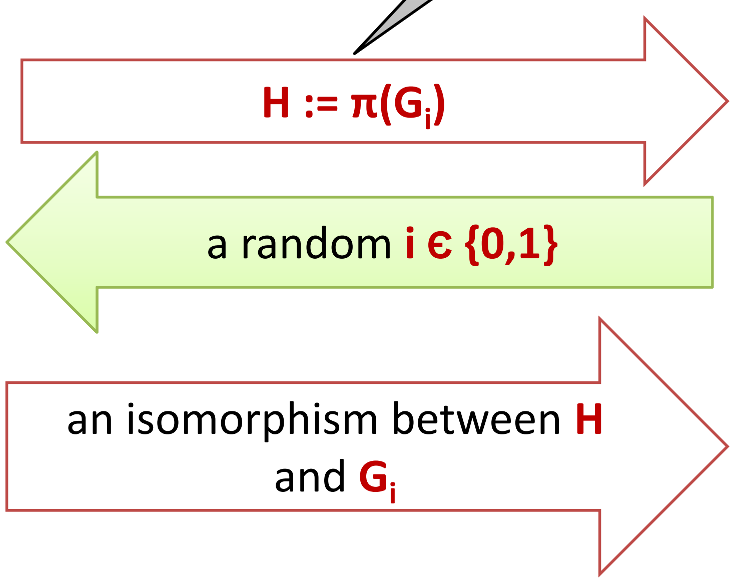
So the expected number of restarts is **2**.

Therefore, the running time is (expected) polynomial time.

Indistinguishability of the distributions

Suppose **i = c,** and hence we didn't restart.

In this case, the simulator simply simulated "perfectly" execution of **V*** against **P**.



QED

Plan

1. Coin-flipping by telephone
2. Commitment schemes
    1. definition
    2. construction based on QRA
    3. construction based on discrete log
    4. construction based on PRG
3. Zero-knowledge (ZK)
    1. motivation and definition
    2. ZK protocol for graph isomorphism
    3. ZK protocol for Hamiltonian cycles
    4. applications

What is provable in NP?

Theorem [Goldreich, Micali, Wigderson, 1986]

Assume that the one-way functions exist.

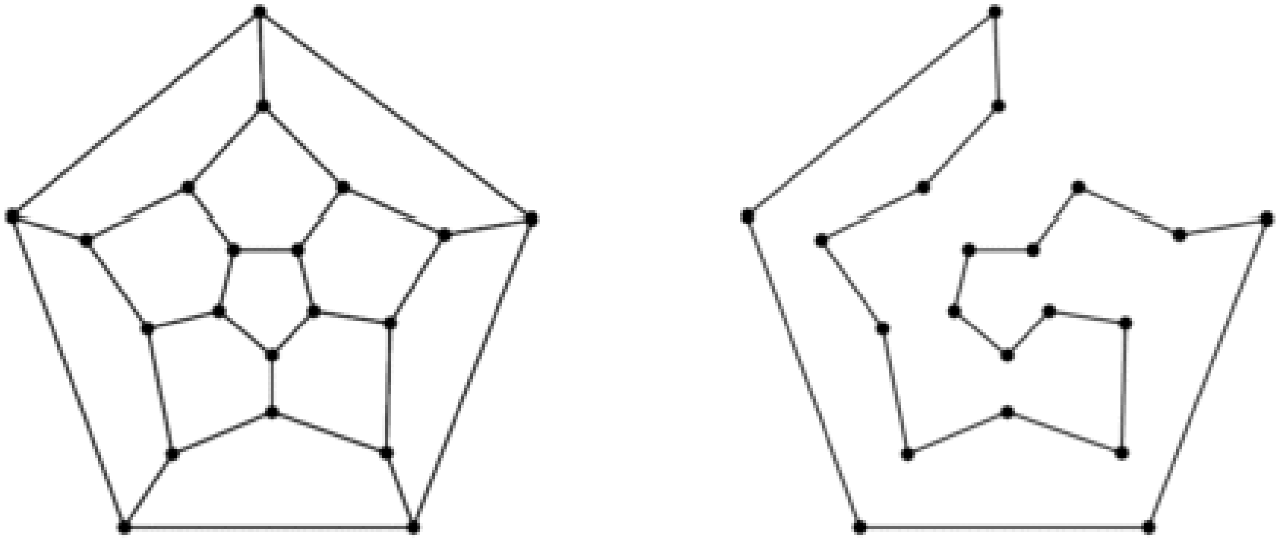
Then, every language **L є NP** has a computational zero-knowledge proof system.

How to prove it?

It is enough to show it for one **NP-complete** problem!

An NP-complete problem: Hamiltonian cycle

Example of a **Hamiltonian cycle**:



Hamiltonian graph – a graph that has a **Hamiltonian cycle**

L := {G : G is Hamiltonian}

How to construct a ZK proof that a graph **G** is Hamiltonian?

Of course:

sending the Hamiltonian cycle in a graph **G** to the verifier doesn't work.

> **H** is Hamiltonian
> iff
> **G** is Hamiltonian

Idea:

We permute the graph **G** randomly – let **H** be the permuted graph.

Then we prove that

1. the **H** is Hamiltonian,
2. **H** is a permutation of **G**.

The first idea:

input
a Hamiltonian graph **G**

prover

chooses a random permutation **π** and sets **H := π(G)**

random bit **i є {0,1}**

verifier

if **i = 0** sends **π**
otherwise sends a Hamiltonian cycle in **H**

Problem: Prove can choose his response depending on **i**.

Solution: use commitments

Remember, that we assumed that the one-way functions exist, so we are allowed to use commitments!

How to commit to a longer string?

Just commit to each bit separately...

Assume the vertices of the graph are natural numbers **{1,...,n}**

How to commit to a permutation of a graph?

Represent it as a string

How to commit to a graph?

Represent it as an **adjacency matrix**, and commit to each bit in the matrix separately.

input

a Hamiltonian graph **G**

prover

chooses a random permutation **π** and sets **H := π(G)**

iterate **n** times:

commit to **π**

commit to **H**

verifier

random bit **i є {0,1}**

if **i = 0** open all the commitments

otherwise opens only the edges that form a Hamiltonian cycle in **H**

Why is it a ZK proof?

Completeness: trivial

Soundness: If **G** is not Hamiltonian, then either
H is not Hamiltonian or **π** is not a permutation.

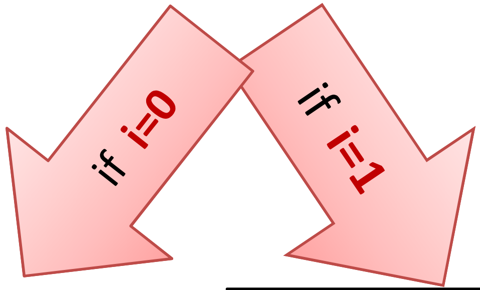
Therefore, to cheat with probability higher than **0.5** the prover needs to break the binding property of the commitment scheme.

If we use the commitment scheme of **Naor**, this probability is negligible, even against an infinitely-powerful adversary

Since the protocol is repeated **n** times, the probability that the verifier rejects is very close to $1 - 0.5^n$. Setting $n := |G|$ we are done!

Zero-Knowledge - intuition

“a cheating **V** should not learn anything besides of the fact that **x ϵ L**”



P “opens everything”, so **V** just learns a randomly permuted graph **G**.

P “opens only the edges that form a Hamiltonian cycle”, so **V** just learns a randomly permuted cycle of vertices

Note, that this gives us only computational indistinguishability. This is because the commitment scheme is only computationally binding.

Observation

The honest prover doesn't need to be infinitely powerful, if he receives the **NP**-witness as an additional input!

Corollary

"Everything that is provable is provable in Zero Knowledge!"

Plan

1. Coin-flipping by telephone
2. Commitment schemes
    1. definition
    2. construction based on QRA
    3. construction based on discrete log
    4. construction based on PRG
3. Zero-knowledge (ZK)
    1. motivation and definition
    2. ZK protocol for graph isomorphism
    3. ZK protocol for Hamiltonian cycles
    4. applications

Example

Suppose, **Alice** knows a signature σ of Bob on some document $m=(m_1,m_2)$.

$$\sigma = Sign_{sk}(m)$$

She want to reveal the first part m_1 of m to **Carol**, and convince her that it was signed by **Bob**, while keeping m_2 and σ secret.

$L = \{m_1$**: there exists** m_2 **and** σ **such that** $Vrfy_{pk}((m_1,m_2),\sigma) = yes\}$

L is in **NP**. So (in principle) **Alice** can do it!

Another example

Alice has a document (signed by some public authority) saying:

"Alice was born on **DD-MM-YYYY**".

She can now prove in zero-knowledge that she is at least **18** years old (without revealing her exact age)

There are many other examples!

For instance:

Alice can show that some message **m** was signed by **Bob** or by **Carol**,

without revealing which was the case.

etc...

Other applications of **ZK**

- a building block in some other protocols

- zero-knowledge identification (e.g. a **Feige-Fiat-Shamir** protocol, based on quadratic residues)

Example

We show a zero-knowledge proof that some **x** is a quadratic residue modulo **N**.

How does it work?

Similarly to the proof that two graphs are isomorphic!

Fact

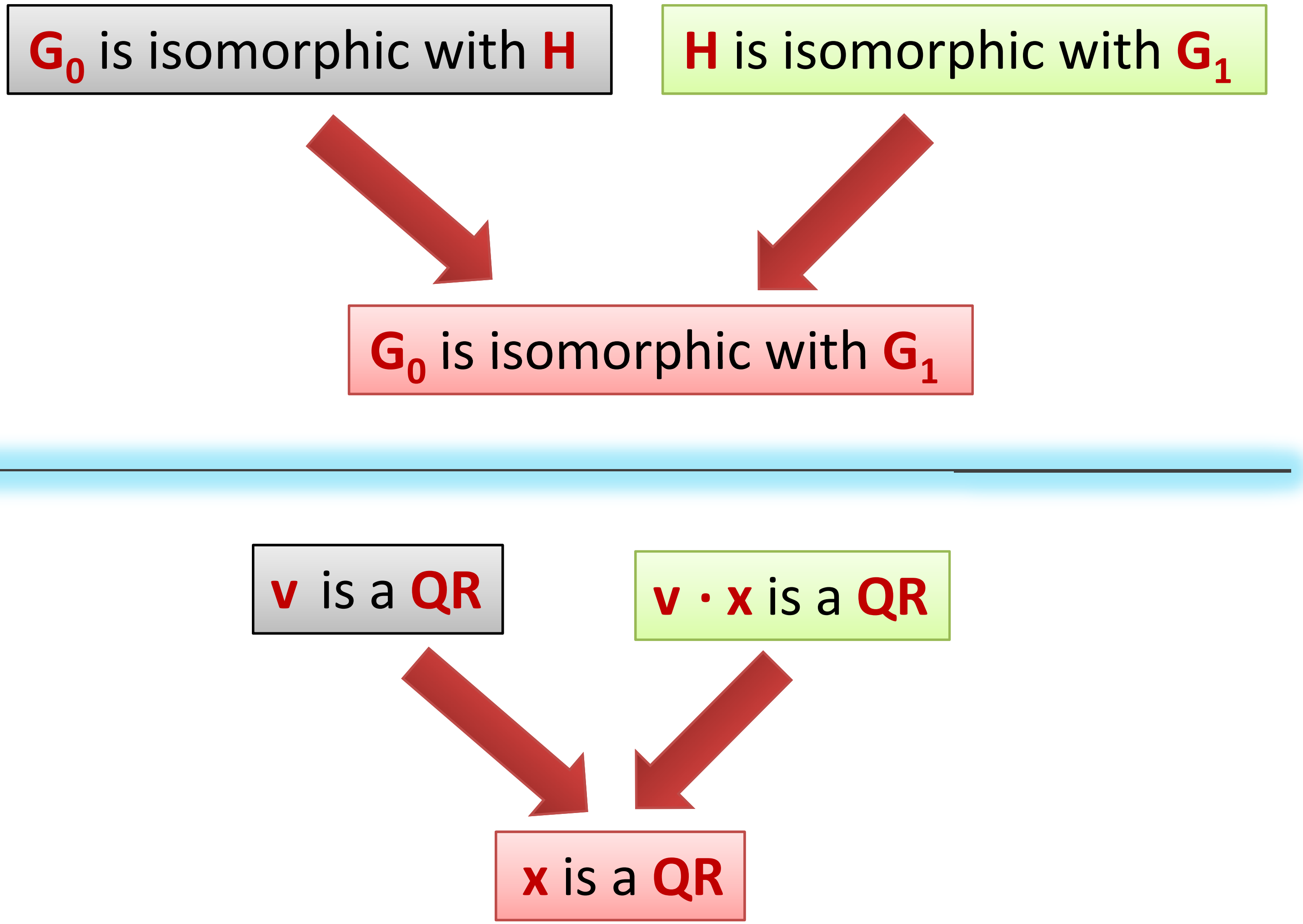
For $a,b \in Z_N^*$ we have:

- if $a \in QR_N$ and $b \in QR_N$ then $a \in QR_N$

and

- if $a \notin QR_N$ and $b \in QR_N$ then $ab \notin QR_N$

Main idea


G_0 is isomorphic with H
H is isomorphic with G_1
G_0 is isomorphic with G_1
v is a QR
$v \cdot x$ is a QR
x is a QR

RSA modulus **N**,
x in QR_N

y such that
$y^2 = x \bmod N$

chose a random
$u \in Z_N^*$

iterate **n** times:

$v := u^2 \bmod N$ →

← random bit $i \in \{0,1\}$

$w := u \cdot y^i \bmod N$ →

accept if
$v \cdot x^i = w^2 \pmod N$

Why is this a zero-knowledge proof system?

- **Completeness**: trivial
- **Soundness**: suppose that **x** is not a **QR_N**

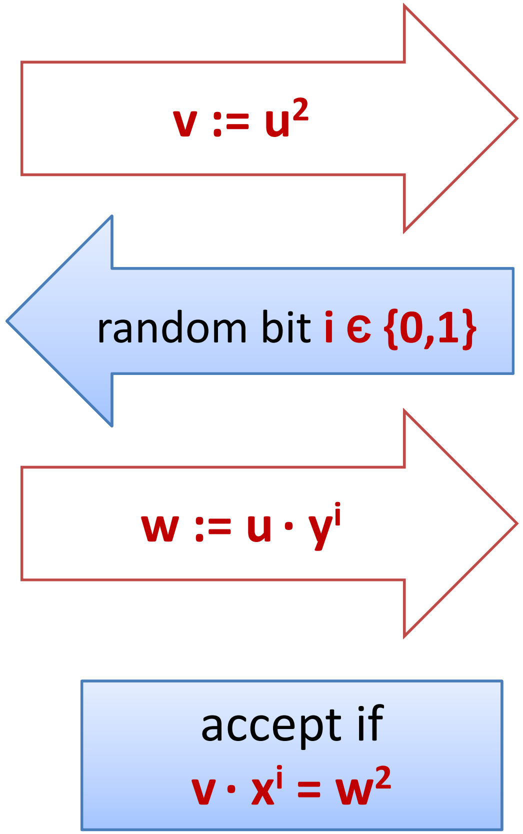
Then

- if **v** is a **QR_N** then the cheating prover will be caught when **$i=1$** since we cannot have

$$QR \cdot QNR = QR$$

- if **v** is a **QNR_N** the cheating prover gets caught when **$i=0$**.

So, the prover can cheat with probability at most **0.5** (in each iteration of the protocol).



Zero-knowledge - intuition

The only information that the verifier gets is:

$v := u^2$

and

- $w := u$ if $i=0$, or
- $w := u \cdot y$ if $i=1$.

This obviously gives him no information on y

This also gives him no information on y, since y is "encrypted" with u

Observation

In fact, the prover demonstrated not only that **x** in **QR_N**, but also that **she knows the square root of x.**

This is called a **zero-knowledge proof of knowledge**.

It can be defined formally!

Zero-knowledge public-key identification

The protocol on the previous slides can be used as a simple **zero-knowledge public-key identification scheme**:

- public key: **N, x**
- private key: **y** such that $y^2 = x \bmod N$

It's extension is called a **Feige-Fiat-Shamir** protocol.