Part X

Protocols to do seemingly impossible and zero-knowledge protocols

PROTOCOLS to do SEEMINGLY IMPOSSIBLE

A **protocol** is an algorithm two (or more) parties have to follow to perform a communication/cooperation.

A cryptographical protocol is a protocol to achieve secure communication during some goal oriented cooperation.

In this chapter we first present several cryptographic protocols for such basic cryptographic primitives as **coin tossing**, **bit commitment** and **oblivious transfer.**

After that we deal with a variety of cryptographical protocols that allow to solve easily **seemingly unsolvable problems.**

Of special importance among them are so called **zero-knowledge protocols** we will deal with afterwards. They are counter-intuitive, though very powerful and very useful.

PRIMITIVES for CRYPTOGRAPHIC PROTOCOLS

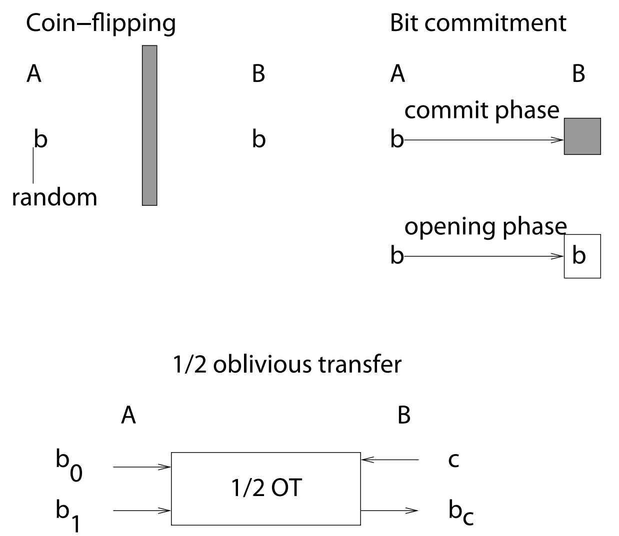
Cryptographic protocols are specifications how two parties, Alice and Bob, should prepare themselves for a communication and how they should behave during a communication in order to achieve their goal and be protected against an adversary.

In **coin-flipping protocols** Alice and Bob can flip a coin over a distance in such a way that neither of them can determine the outcome of the flip, but both can agree on the outcome in spite of the fact that they do not trust each other.

In **bit commitment protocols** Alice can choose a bit and get committed to it in the following sense: Bob has no way of learning Alice's commitment and Alice has no way of changing her commitment. Alice commits herself to a bit x using a *commit*(x) procedure, and reveals her commitment, if needed, using *open*(x) procedure.

In 1-out-2 **oblivious transfer protocols** Alice transmits two messages **first** and **second** to Bob who can chose to receive **first** or **second**, but cannot learn both, in such a way that Alice will have no idea which of them Bob will receive.

SCHEMES for PRIMITIVES of CRYPTOGRAPHIC PROTOCOLS



PROTOCOLS for COIN-FLIPPING/TOSSING BY PHONE

Coin-flipping by telephone:

Alice and Bob got divorced and they do not trust each other any longer. They want to decide, communicating by phone only, who gets the car.

Protocol 1 Alice sends Bob messages head and tail encrypted by a one-way function f. Bob guesses which one of them is encryption of head. Alice tells Bob whether his guess was correct. If Bob does not believe her, Alice sends f to Bob.

Protocol 2 Alice chooses two large primes p,q, sends Bob n = pq and keeps p, q secret.
Bob chooses randomly an integer $x \in \{1, \ldots, \frac{n}{2}\}$, sends Alice $y = x^2 \mod n$ and tells Alice: if you guess x correctly, car will be yours.
Alice computes four square roots $(x_1, n - x_1)$ and $(x_2, n - x_2)$ of x and

$$x_1' = min(x_1, n - x_1), x_2' = min(x_2, n - x_2).$$

Since $x \in \{1, \ldots, \frac{n}{2}\}$, then either $x = x_1'$ or $x = x_2'$.
Alice then guesses whether $x = x_1'$ or $x = x_2'$ and tells Bob her choice (for example by reporting the position and value of the leftmost bit in which x_1' and x_2' differ).
Bob tells Alice whether her guess was correct.
(Later, if necessary, Alice reveals p and q, and Bob reveals x.)

COIN TOSSING – requirements and problems

- **Basic requirements:** In any good coin tossing protocol both parties should influence the outcome and should accept the outcome. Both outcomes should have the same probability.
- Requirements for a coin tossing protocol are sometimes generalized as follows:
  - The outcome of the protocol is an element from the set {0, 1, reject}.
  - If both parties behave correctly, the outcome should be from the set {0, 1}.
  - If it is not the case that both parties behave correctly, the outcome should be reject.

Problem: In some coin tossing protocols one party can find out the outcome sooner than the second party. In such a case if she is not happy with the outcome she can disrupt the protocol – to produce reject or to say "I do not continue in performing the protocol". A way out is to require that in case of correct behavior no outcome should have probability $> \frac{1}{2}$.

COIN TOSSING USING a ONE-WAY FUNCTION

Protocol:

- **Alice chooses a one-way function f and informs Bob about the definition domain of f.**
- Bob chooses randomly r_1, r_2 from dom(f) and sends them to Alice.
- Alice sends to Bob one of the values $f(r_1)$ or $f(r_2)$.
- Bob announces Alice his guess which of the two values he received.
- Alice announces Bob whether his guess was correct (0) or not (1).
- If one needs to verify correctness, Alice should send to Bob the specification of f.

The protocol is computationally secure. Indeed, to cheat, Alice should be able to find, for randomly chosen r_1, r_2, such one-way function f that $f(r_1) = f(r_2)$.

BIT COMMITMENT PROTOCOLS (BCP)

Basic ideas and solutions I

In a bit commitment protocol Alice chooses a bit b and gets committed to b, in the following sense:

Bob has no way of knowing which commitment Alice has made, and Alice has no way of changing her commitment once she has made it; say after Bob announces his guess as to what Alice has chosen.

An example of a "pre-computer era" bit commitment protocol is that Alice writes her commitment on a paper, locks it in a box, sends the box to Bob and, later, in the opening phase, she sends also the key to Bob.

Complexity era solution I. Alice chooses a one-way function f and an even (odd) x if she wants to commit herself to 0 (1) and sends to Bob f(x) and f.

Problem: Alice may know an even x_1 and an odd x_2 such that $f(x_1) = f(x_2)$.

Complexity era solution II. Alice chooses a one-way function f, two random x_1, x_2 and a bit b she wishes to commit to, and sends to Bob $(f(x_1, x_2, b), x_1)$ - a commitment.

When times comes for Alice to reveal her bit, she sends to Bob f and the triple (x_1, x_2, b).

BIT COMMITMENT SCHEMES I

The basis of bit commitment protocols are bit commitment schemes:
A bit commitment scheme is a mapping $f : \{0,1\} \times X \rightarrow Y$, where X and Y are finite sets.

A commitment to a $b \in \{0,1\}$, or an encryption of b, is any value (called a blow) f(b, x) where x ∈ X.

Each bit commitment protocol has two phases:

Commitment phase: The sender sends a bit b he wants to commit to, in an encrypted form, to the receiver.

Opening phase: If required, the sender sends to the receiver additional information that enables the receiver to get b.

BIT COMMITMENT SCHEMES II

Each bit commitment scheme should have three properties:

Hiding (privacy): For no b ∈ $\{0,1\}$ and no $x \in X$, it is feasible for Bob to determine b from B = f(b, x).

Binding: Alice can "open" her commitment b, by revealing (opening) x and b such that B = f(b, x), but she should not be able to open a commitment (blow) B as both 0 and 1.

Correctness: If both, the sender and the receiver, follow the protocol, then the receiver will always learn (recover) the committed value b.

BIT COMMITMENT with ONE-WAY FUNCTIONS

Commitment phase:

- Alice and Bob choose a one-way function f
- Bob sends a randomly chosen r_1 to Alice
- Alice chooses random r_2 and her committed bit b and sends to Bob $f(r_1, r_2, b)$.

Opening phase:

- Alice sends to Bob r_2 and b
- Bob computes $f(r_1, r_2, b)$ and compares with the value he has already received.

HASH FUNCTIONS and COMMITMENTS

A commitment to a data w, without revealing w, using a hash function h, can be done as follows:

Commitment phase: To commit to a w choose a random r and make public h(wr).

Opening phase: reveal r and w.

For this application the hash function h has to be one-way: from h(wr) it should be unfeasible to determine wr.

TWO SPECIAL BIT COMMITMENT SCHEMES

Bit commitment scheme I. Let p, q be large primes, n = pq, m ∈ QNR(n), $X = Z_n^*$. Let n,m be public.

Commitment: $f(b, x) = m^b x^2 \mod n$ for a random x from X.

Since computation of quadratic residues is in general infeasible, this bit commitment scheme is hiding.

Since $m \in QNR(n)$, there are no x_1, x_2 such that $mx_1^2 = x_2^2 \mod n$ and therefore the scheme is binding.

Bit commitment scheme II. Let p be a large Blum prime, $X = Z_p* = Y$, α be a primitive element of Z_p^*.

$$f(b,x) = \alpha^x \mod p, \text{ if SLB(x) = b;}$$
$$= \alpha^{p-x} \mod p, \text{ if SLB(x)} \neq \text{b.}$$

where

$$\text{SLB(x)} = 0 \text{ if } x \equiv 0,1 \pmod 4;$$
$$= 1 \text{ if } x \equiv 2,3 \pmod 4.$$

Binding property of this bit commitment scheme follows from the fact that in the case of discrete logarithms modulo Blum primes there is no effective way to determine second least significant bit (SLB) of the discrete logarithm.

MAKING COIN TOSSING FROM BIT COMMITMENT

Each bit commitment scheme can be used to solve coin tossing problem as follows:

1. Alice tosses a coin, and commits itself to its outcome b_A (say to 0 (1) if the outcome is head (tail)) and sends the commitment to Bob.
2. Bob also tosses a coin and sends the outcome b_B to Alice.
3. Alice opens her commitment. to Bob (so he starts to know b_A)
4. Both Alice and Bob compute $b = b_A \oplus b_B$.

Observe that if at least one of the parties follows the protocol, that is it tosses a random coin, the outcome is indeed a random bit.

Note: Observe that after step 2 Alice will know what the outcome is, but Bob does not. So Alice can disrupt the protocol if the outcome is to be not good for her. This is a weak point of this protocol.

BASIC TYPES of HIDING and BINDING

If the hiding or the binding property of a commitment protocol depends on the complexity of a computational problem, we speak about computational hiding and computational binding.

In case, the binding or the hiding property does not depend on the complexity of a computational problem, we speak about unconditional hiding or unconditional binding.

A COMMITMENT SCHEME BASED on DISCRETE LOGARITHM

Alice wants to commit herself to an $m \in \{0, \ldots, q-1\}$.

Scheme setting:

Bob randomly chooses primes p and q such that

$$q|(p-1).$$

Bob chooses random generators $g \neq 1 \neq v$ of the subgroup G of order $q \in Z_n^*$.Bob sends p, q, g and v to Alice.

All following computations will be modulo p:

Commitment phase:

To commit to an $m \in \{0, \ldots, q-1\}$, Alice chooses a random $r \in Z_q$, and sends $c = g^r v^m$ to Bob.

Opening phase:

Alice sends r and m to Bob who then verifies whether c= $g^r v^m$.

COMMENTS

- If Alice, committed to an m, could open her commitment as $\bar{m} \neq m$, using some $\bar{r}$, then $g^r v^m = g^{\bar{r}} v^{\bar{m}}$ and therefore

$$\lg_g v = (r - \bar{r})(\bar{m} - m)^{-1}.$$

  Hence, Alice could compute $lg_g v$ of a randomly chosen element $v \in G$, what contradicts the assumption that computation of discrete logarithms in G is infeasible.
- Since g and v are generators of G, then g^r is a uniformly chosen random element in G, perfectly hiding v^m and m in $g^r v^m$, as in the encryption with ONE-TIME PAD cryptosystem.

BIT COMMITMENT using ENCRYPTIONS

Commit phase:

1. Bob generates a random string r and sends it to Alice
2. Alice commit herself to a bit b using a key k through an encryption

$$E_k(rb)$$

   and sends it to Bob.

Opening phase:

1. Alice sends the key k to Bob.
2. Bob decrypts the message to learn b and to verify r.

Comment: without Bob's random string r Alice could find a different key l such that $e_k(b) = e_l(\neg b)$.

COMMITMENTS and ELECTRONIC VOTING

Let com(r, m) $= g^r v^m$ denote commitment to m in the commitment scheme based on discrete logarithm. If $r_1, r_2, m_1, m_2 \in Z_n$, then $com(r_1, m_1) \times com(r_2, m_2) = com(r_1 + r_2, m_1 + m_2)$.
Commitment schemes with such a property are called **homomorphic commitment schemes.**
Homomorphic schemes can be used to cast yes-no votes of n voters $V_1, \ldots, V_n$, by the trusted authority TA for whom e_T and d_T are ElGamal encryption and decryption algorithms.
This works as follows: Each voter V_i chooses his vote $m_i \in \{0, 1\}$, a random $r_l \in \{0, \ldots, q-1\}$ and computes his voting commitment $c_l = com(r_i, m_i)$.Then V_i makes c_i public and sends $e_T(g^{r_i})$ to TA and TA computes

$$d_T\left(\prod_{i=1}^{n} e_T(g^{r_i})\right) = \prod_{i=1}^{n} g^{r_i} = g^r,$$

where $r = \sum_{i=1}^{n} r_i$, and makes public g^r.
Now, anybody can compute the result s of voting from publicly known c_i and g^r since

$$v^s = \frac{\prod_{i=1}^{n} c_i}{g^r},$$

with $s = \sum_{i=1}^{n} m_i$.
s can now be derived from v^s by computing $v^1, v^2, v^3, \ldots$ and comparing with v^s if the number of voters is not too large.

TRUST in CRYPTOGRAPHIC PROTOCOLS - deliberations

In any interaction between people, there is a certain level of risk, trust, and expected behaviour, that is implicit in the interchanges.

People may behave properly for a variety of reasons: fear from prosecution, desire to act in ethical manner due to social influences, and so on.

However, in cryptographic protocols trust has to be kept to the lowest possible level.

In any cryptographic protocol, if there is an absence of a mechanism for verifying, say authenticity, one must assume, as default, that other participants can be dishonest (if for no other reason than for self-preservation).

OBLIVIOUS TRANSFER (OT) PROBLEM

Story: **Alice knows a secret and wants to send secret to Bob in such a way that he gets secret with probability $\frac{1}{2}$, and he knows whether he got secret, but Alice has no idea whether he received secret.**(Or Alice has several secrets and Bob wants to buy one of them but he does not want Alice to know which one he bought.)

Oblivious transfer problem: **Design a protocol for sending a message from Alice to Bob in such a way that Bob receives the message with probability $\frac{1}{2}$ and "garbage" with the probability $\frac{1}{2}$.** Moreover, Bob knows whether he got the message or garbage, but Alice has no idea which one he got.

OBLIVIOUS TRANSFER PROTOCOL - continuation

Oblivious transfer problem: **Design a protocol for sending a message from Alice to Bob in such a way that Bob receives the message with probability $\frac{1}{2}$ and "garbage" with the probability $\frac{1}{2}$.** Moreover, Bob knows whether he got the message or garbage, but Alice has no idea which one he got.

An Oblivious transfer protocol:

1. Alice chooses two large primes p and q and sends $n = pq$ to Bob.
2. Bob chooses a random number x and sends $y = x^2 \mod n$ to Alice.
3. Alice computes four square roots $\pm x_1, \pm x_2$ of y (mod n) and sends one of them to Bob. (She can do it, but has no idea which of them is x.)
4. Bob checks whether the number he got is congruent to x. If yes, he has received no new information. Otherwise, Bob has two different square roots modulo n and can factor n. Alice has no way of knowing whether this is the case.

1-OUT-OF-2 OBLIVIOUS TRANSFER PROBLEM

The 1-out-of-2 oblivious transfer problem: Alice sends two messages to Bob in such a way that Bob can choose which of the messages he receives (but he cannot choose both), but Alice cannot learn Bob's decision.

A generalization of 1-out-of-2 oblivious transfer problem is two-party oblivious circuit evaluation problem:

Alice has a secret i and Bob has a secret j and they both know some function f.

At the end of protocol the following conditions should hold:

1. Bob knows the value f(i,j), but he does not learn anything about i.
2. Alice learns nothing about j and nothing about f(i,j).

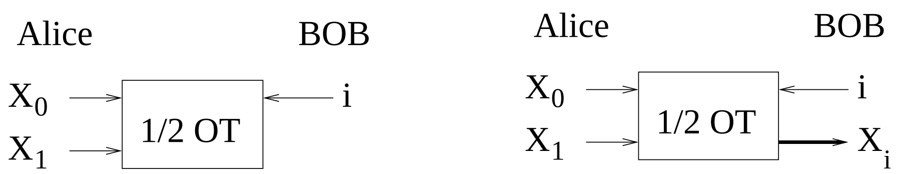
Note: The 1-out-of-2 oblivious transfer problem is the instance of the oblivious circuit evaluation problem for $i = (b_0, b_1), f(i, j) = b_j$.

1-out-2 OBLIVIOUS TRANSFER BOX

1-out-of-two oblivious transfer can be imagined as a box with three inputs and one output.

INPUTS: Alice inputs: X_0 and X_1;
.... Bob inputs a bit i

OUTPUT: Bob gets as the output: X_i



AN IMPLEMENTATION of OBLIVIOUS TRANSFER PROTOCOLS

- Alice generates two key pairs for a PKC P and sends both her public keys p_1, p_2 to Bob.
- Bob chooses a to-be random secret key k for a SKC S, encrypts it by one of Alice's public keys, p_1 or p_2 and sends the outcome to Alice.
- Alice uses her two secret keys to decrypt the message she received. One of the outcomes is garbage g, another one is k, but she does not know which one is k.
- Alice encrypts her two secret messages, one with k, another with g and sends them to Bob.
- Bob uses S with k to decrypt both messages he got and one of the attempts is successful. Alice has no idea which one.

HISTORY and POWER of OBLIVIOUS TRANSFER PROTOCOLS

- C. Crépeau (1988) showed that both versions of oblivious transfer are equivalent – a protocol for each version can be realized using any protocol for the other version, using a cryptographic reduction
- Original definition of the oblivious transfer is due to J. Halpern and M. O. Rabin (1983); 1-out-of-2 oblivious transfer suggested S. Even, O. Goldreich and A. Lempel in 1985.
- J. Kilian (1988) showed that oblivious transfers are very powerful protocols that allow **secure multiparty computation** of the value f(x, y) of any binary function f , where x is a secret value known only by Alice, and y is a secret value known only by Bob, in such a way that it holds:
  - Both, Alice and Bob, learn f(x, y)
  - Alice learns about y only as much as she can learn from x and f(x, y)
  - Bob learns about x only as much as he can learn from y and f(x, y)

BIT COMMITMENT from 1-out-2 oblivious transfer

Using 1-out-of-2 oblivious transfer box (OT-box) one can design a bit commitment scheme:

COMMITMENT PHASE:

1. Alice selects a random bit r and her commitment bit b;
2. Alice inputs $x_0 = r$ and $x_1 = r \oplus b$ into the OT-box.
3. Alice sends a message to Bob telling him it is his turn.
4. Bob selects a random bit c, inputs c into the OT-box and records the output x_c.

OPENING PHASE:

1. Alice sends r and b to Bob.
2. Bob checks to see if $x_c = r \oplus (bc)$

MENTAL POKER PLAYING by PHONE by Alice and Bob

Basic requirements (for playing poker with 52 cards):

- Initial hands (sets of 5 cards) of both players are equally likely.
- The initial hands of Alice and Bob are disjoint.
- Both players always know their own hands but not that of the opponent.
- Each player can detect eventual cheating of the other player.

A commutative cryptosystem is used with all functions kept secret.

Players agree on numbers $w_1, \ldots, w_{52}$ as the names of 52 cards.

Protocol:

1. Bob encrypts cards with e_B, and tells $e_B(w_1), \ldots, e_B(w_{52})$, in a randomly chosen order, to Alice.
2. Alice chooses five of the items $e_B(w_i)$ as Bob's hand and tells them Bob.
3. Alice chooses another five of $e_B(w_i)$, encrypts them with e_A and sends them to Bob.
4. Bob applies d_B to all five values $e_A(e_B(w_i))$ he got from Alice and sends $e_A(w_i)$ to Alice as Alice's hand. At this point both players have their hands and poker can start.

Remark: The cryptosystems that are used cannot be public-key in the normal sense. Otherwise Alice could compute $e_B(w_i)$ and deal with the cards accordingly – a good hand for B but slightly better for herself.

MENTAL POKER by PHONE with THREE PLAYERS

1. Alice encrypts 52 cards $w_1, \dots, w_{52}$ with e_A and sends encryptions, in a random order, to Bob.
2. Bob, who cannot decode the encryptions, chooses 5 of them, randomly. He encrypts them with e_B, and sends $e_B(e_A(w_i))$ to Alice and the remaining 47 encryptions $e_A(w_i)$ to Carol.
3. Carol, who cannot decode any of the encryptions, chooses five of them randomly, encrypts them also with her key and sends Alice $e_C(e_A(w_i))$.
4. Alice, who cannot read encrypted messages from Bob and Carol, decrypt them with her key and sends back to the senders,

   five $d_A(e_B(e_A(w_i))) = e_B(w_i)$ to Bob,

   five $d_A(e_C(e_A(w_i))) = e_C(w_i)$ to Carol.
5. Bob and Carol decrypt encryptions they got to learn their hands.
6. Carol chooses randomly 5 other messages $e_A(w_i)$ from the remaining 42 and sends them to Alice.
7. Alice decrypt messages to learn her hand.

Additional cards can be dealt with in a similar manner. If either Bob or Carol wants a card, they take an encrypted message $e_A(w_i)$ and go through the protocol with Alice. If Alice wants a card, whoever currently has the deck sends her a card.

ZERO-KNOWLEDGE PROOFS/PROTOCOLS

Loosely speaking, zero-knowledge proofs of an assertion are proofs that yield nothing beyond the validity of the assertion.

In other words, a verifier obtaining such a proof gains only conviction in the validity of the assertion.

One way to understand it is by saying that anything that can be efficiently computable from a zero-knowledge proof can also be efficiently computable under the belief/understanding that the assertion being proved is true.

There are various types of zero-knowledge protocols - of identity, of membership, of knowledge, ...

ZERO-KNOWLEDGE PROOFS and CRYPTOGRAPHY

Zero-knowledge proofs are fascinating and extremely useful cryptographic tools.

Their fascinating nature is due to their seemingly contradictorness: zero-knowledge proofs are both convincing and yet yield nothing beyond the assertion being proved.

Their applicability in cryptography is vast. For example, they are used to force malicious parties to behave honestly, according to a predetermined protocol, while maintaining privacy **i.e. the protocol may require communicating parties to provide zero-knowledge proofs of the correctness of their secret-based (privacy-protection) actions, without revealing these secrets.**

WHAT is a PROOF?

What is a proof?

- The concept of proof was one of main achievements of the Golden Era of Greek science/mathematics/geometry - 6th - 3rd century BC.
- After that the concept of proof was almost forgotten for more than 2000 years.
- A need to precise the concept of proof arose again at the very beginning of 20th century due to the existence very strange functions and paradoxes in set theory.
- Hilbert formalized the concept of proof. A sequence of statements each of which is either an axiom or can be derived from previous ones using one of the deduction rules - a proof should be checkable by machines.
- Later, it has turned out that such a concept of proof, producing "absolute truth", maybe sometimes much stronger than needed.
- By Manin: **Proof is whatever convinces me.**
- Zero-knowledge proofs and probabilistic proofs represent a new type of proofs – proofs that provide convincing evidence – so much convincing as needed.

ZERO-KNOWLEDGE PROOFS/PROTOCOLS - I.

Very informally, a zero-knowledge proof protocol allows one party, usually called PROVER, to convince another party, called VERIFIER, that PROVER has some knowledge (a secret, a proof of a theorem,...), or that something holds, without revealing to the VERIFIER **ANY** information about his knowledge (secret, proof,...) or how to show that.

In the rest of this chapter we present and illustrate very basic ideas of zero-knowledge proof protocols and their importance for cryptography.

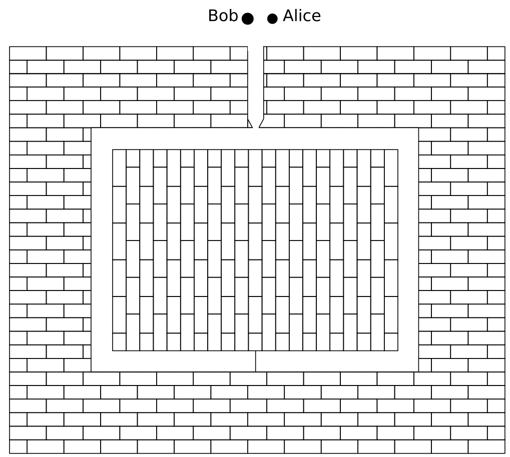
Zero-knowledge proof protocols are a special type of so-called interactive proof systems.

By a theorem we understand in the following a claim that a specific object has a specific property. For example, that a specific graph is 3-colorable.

AN ILLUSTRATIVE EXAMPLE

(A cave with a magic door opening on a secret word)

Alice knows a secret word opening the door in cave. How can she convince Bob about it without revealing this secret word?



ANOTHER ILLUSTRATION

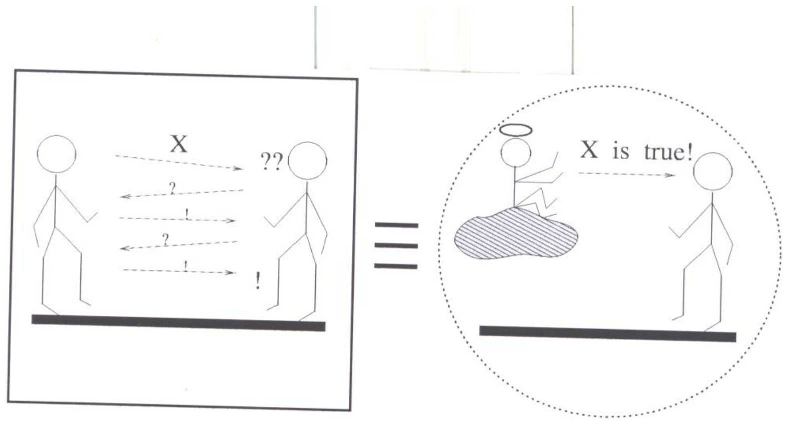


Figure 1: Zero-knowledge proofs – an illustration.

ZERO-KNOWLEDGE PROOFS/PROTOCOLS - II.

A zero-knowledge proof or protocol is an interactive process by which one party (the Prover) can convince another party (the Verifier) that a a particular statement is true, without conveying any additional information apart from the fact that the statement is indeed true.

For the case where the ability to prove the statement requires that the Prover has some secret information, zero-knowledge requirement implies that that the verifier will not be able to prove the statement to anyone else.

Notice that the notion of zero-knowledge applies only if the statement being proven is the fact that the Prover has a certain knowledge - a secret information. Otherwise, the statement would not be proven in zero-knowledge way, since at the end of the protocol the verifier would gain an additional information - namely the information that the prover has knowledge of the required secret information.

This is a particular case known as **zero-knowledge proof of knowledge**.

INTERACTIVE PROOF PROTOCOLS

In an interactive proof system there are two parties

- A (strong - all powerful) Prover, often called Peggy (a randomized algorithm that uses a private random number generator);
- A poor Verifier, often called Vic (a polynomial time randomized algorithm that uses a private random number generator).

Prover knows some secret, or a knowledge, or a fact about a specific object, and wishes to convince Vic, through a communication with him, that he has the above knowledge.

For example, both Prover and Verifier posses an input x and Prover wants to convince Verifier that x has a certain Property and that Prover knows how to prove that.

The interactive proof system consists of several rounds. In each round Prover and Verifier alternatively do the following.

1. Receive a message from the other party.
2. Perform a (private) computation.
3. Send a message to the other party.

Communication starts usually by a challenge of Verifier and a response of Prover.

At the end, Verifier either accepts or rejects Prover's attempts to convince Verifier.

INTERACTIVE PROOF SYSTEMS

An interactive proof protocol is said to be an interactive proof system for a secret/knowledge or a decision problem Π if the following properties are satisfied provided that Prover and Verifier posses an input x (or Prover has secret knowledge) and Prover wants to convince Verifier that x has certain properties and that Prover knows how to prove that (or that Prover knows the secret).

(Knowledge) Completeness: If x is a yes-instance of Π, or Peggy knows the secret, then Vic always accepts Peggy's "proof" for sure.

(Knowledge) Soundness: If x is a no-instance of Π, or Peggy does not know the secret, then Vic accepts Peggy's "proof" only with very small probability.

CHEATING

- If the Prover and the Verifier of an interactive proof system fully follow the protocol they are called honest Prover and honest Verifier.
- A Prover who does not know secret or proof and tries to convince the Verifier is called cheating Prover.
- A Verifier who does not follow the behaviour specified in the protocol is called a cheating Verifier.

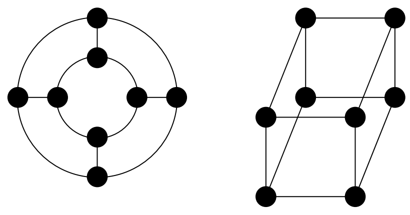
INTERACTIVE PROOF SYSTEMS INTUITIVELY

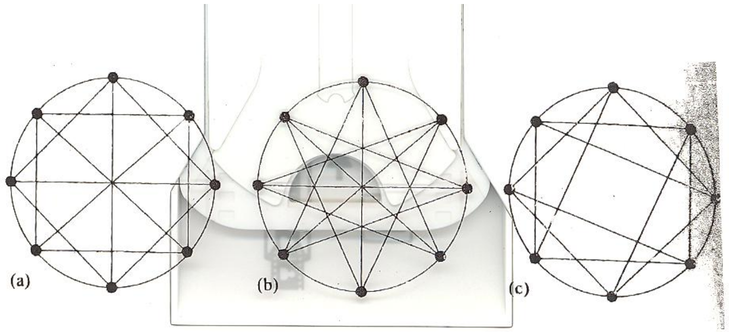
Loosely speaking, an interactive proof is a game between a computationally bounded verifier and a computationally unbounded prover whose goal is to convince the verifier of the validity of some assertion.

An interactive proof should allow the prover to convince the verifier of the validity of any true assertion, whereas no prover strategy may fool the verifier with not negligible probability to accept false assertions.

Intuitively, one may think about interactions between verifier and prover as consisting of "tricky" questions asked by the verifier to which the prover has to reply "convincingly".

GRAPH ISOMORPHISM - EXAMPLE





EXAMPLE – GRAPH NON-ISOMORPHISM

A simple interactive proof protocol exists for a computationally very hard graph non-isomorphism problem.

Input: Two graphs G_1 and G_2, with the set of nodes $\{1, \ldots, n\}$.

Protocol: Repeat n times the following steps:

1. Vic chooses randomly an integer $i \in \{1, 2\}$ and a permutation π of $\{1, \ldots, n\}$.Vic then computes the image H of G_i under the permutation π and sends H to Peggy.
2. Peggy determines the value j such that G_J is isomorphic to H, and sends j to Vic.
3. Vic checks to see if $i = j$.

Vic accepts Peggy's proof if $i = j$ in each of n rounds.

Completeness: If G_1 is not isomorphic to G_2, then probability that Vic accepts is 1 because Peggy will have no problem to answer correctly.

Soundness: If G_1 is isomorphic to G_2, then Peggy can deceive Vic if and only if she correctly guesses n times those i's Vic chooses randomly. The probability that this can happen is 2^{-n}.

Observe that Vic's computations can be performed in polynomial time (with respect to the size of graphs).

ZERO-KNOWLEDGE PROOFS

Informally speaking, an interactive proof systems has the property of being zero-knowledge if the Verifier, that interacts with the honest Prover of the system, learns nothing from their interaction beyond the validity of the statement being proved.

There are several variants of zero-knowledge protocols that differ in the specific way the notion of learning nothing is formalized.

In each variant it is viewed that a particular Verifier learns nothing if there exists a polynomial time simulator whose output is indistinguishable from the output of the Verifier after interacting with the Prover on any possible instance of the problem.

Different variants of zero-knowledge proof systems concern the strength of this distinguishability.In particular, perfect or statistical zero-knowledge refer to the situation where the simulator's output and the Verifier's output are indistinguishable in an information theoretic sense.

Computational zero-knowledge refer to the case there is no polynomial time distinguishability.

ZERO-KNOWLEDGE PROOF PROTOCOLS - VERY INFORMALLY

Very informally An interactive "proof protocol" at which a Prover tries to convince a Verifier about the truth of a statement, or about possession of a knowledge, is called "zero-knowledge" protocol if the Verifier does not learn from communication anything more except that the statement is true or that Prover has knowledge (secret) she claims to have.

Example The proof $n = 670592745 = 12345 \times 54321$ is not a zero-knowledge proof that n is not a prime.

ZERO-KNOWLEDGE PROOF PROTOCOLS - MORE FORMALLY

Informally, a zero-knowledge proof is an interactive proof protocol that provides **highly convincing evidence** that a statement is true or that Prover has certain knowledge (of a secret) and that Prover knows a (standard) proof of it while providing **not a single bit of information** about the proof (knowledge or secret). (In particular, Verifier who got convinced about the correctness of a statement cannot convince the third person about that.)

More formally A zero-knowledge proof of a theorem T is an interactive two party protocol, in which Prover is able to convince Verifier who follows the same protocol, by the overwhelming statistical evidence, that T is true, if T is indeed true, but no Prover is able to convince Verifier that T is true, if this is not so.

In addition, during interactions, Prover does not reveal to Verifier any other information, except whether T is true or not.Consequently, whatever Verifier can do after he gets convinced, he can do just believing that T is true.
Similar arguments hold for the case Prover possesses a secret.

FORMAL DEFINITION of ZERO-KNOWLEDGE

In the following definition both prover (P) and verifier (V) as well as a simulator (S) will be Turing machines.

An interactive proof system with (P, V) for a language L is zero-knowledge if for any polynomial time randomized verifier V there exists polynomial randomized simulator S such that

$$\forall x \in L$$

$$S(x) - -\{\text{ the value produced by the simulator S}\}$$

is undistinguishable from what can be obtained from the transcript of the communication between P and V for the input x.

AGE DIFFERENCE FINDING PROTOCOL

Alice and Bob want to find out who of them is older without disclosing any other information about their age.

The following protocol is based on a public-key cryptosystem, in which it is assumed that neither Bob nor Alice are older than 100 years.

Protocol Let age of Bob be j; and age of Alice be i.

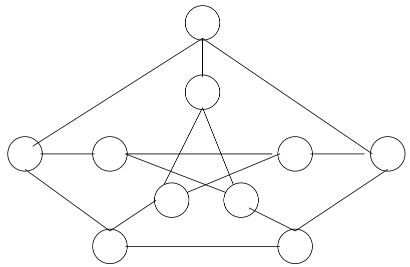
1. Bob chooses a random $x \in \{1, \ldots, 100\}$, computes $k = e_A(x)$ and sends to Alice s = k - j.
2. Alice first computes the numbers $y_u = d_A(s+u); 1 \leq u \leq 100$, then chooses a large random prime p and computes numbers
$$z_u = y_u \mod p, \quad 1 \leq u \leq 100 \qquad (*)$$
and verifies that for all $u \neq v$
$$|z_u - z_v| \geq 2 \text{ and } z_u \neq 0 \qquad (**)$$
(If this is not the case, Alice choose a new p, repeats computations in (*) and checks (**) again.)
Finally, Alice sends Bob the following sequence (order is important).
$$z_1, \ldots, z_i, z_{i+1} + 1, \ldots, z_{100} + 1, p$$
$$\text{as } z'_1, \ldots, z'_i, z'_{i+1}, \ldots, z'_{100}, p$$
3. Bob checks whether j-th number in the above sequence is congruent to x modulo p. If yes, Bob knows that $i \geq j$, otherwise $i < j$.
$$i \geq j \Rightarrow z'_j = z_j \equiv y_j = d_A(k) \equiv x \pmod p$$
$$i < j \Rightarrow z'_j = z_j + 1 \neq y_j = d_A(k) \equiv x \pmod p$$

MILLIONAIRE

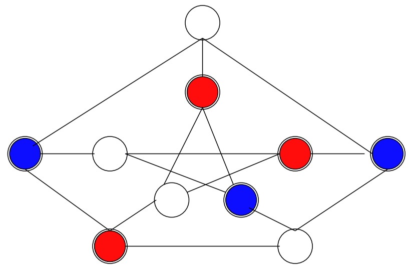
- The previous problem is ofter referred to as **Millionaire problem** that want to know who of them is richer without disclosing any additional information about their wealth.
- The problem is also often seen as an example of **two-party (multi-party) secure computation** at which both parties want to know some outcomes that depends on their inputs, but they do not want to disclose any information about their inputs.

3-COLORABILITY of GRAPHS - EXAMPLE

Are the nodes of the following graph colorable by three colors in such a way that no edge connects nodes of the same color?

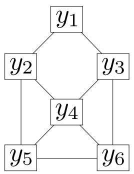


Yes, they are:



3-COLORABILITY of GRAPHS

With the following protocol Peggy can convince Vic that a particular graph G, known to both of them, is 3-colorable and that Peggy knows such a coloring, without revealing to Vic any information how such coloring looks.



(a)

1 red	e_1	$e_1(red) = y_1$
2 green	e_2	$e_2(green) = y_2$
3 blue	e_3	$e_3(blue) = y_3$
4 red	e_4	$e_4(red) = y_4$
5 blue	e_5	$e_5(blue) = y_5$
6 green	e_6	$e_6(green) = y_6$

(b)

Protocol: Peggy colors the graph G = (V, E) with colors (red, blue, green) and she performs with Vic $|E|^2$- times the following interactions, where $v_1, \ldots, v_n$ are nodes of V.

1. Peggy chooses a random permutation of colors, recolors G, and encrypts, for i = 1,2,. . . ,n, the color c_i of node v_i by an encryption procedure e_i – for each i different. Peggy then removes colors from nodes, labels the i-th node of G with cryptotext $y_i = e_i(c_i)$, and designs Table (b).
   Peggy finally shows Vic the graph with nodes labeled by cryptotexts.
2. Vic chooses an edge and asks Peggy to show him coloring of the corresponding nodes.
3. Peggy shows Vic entries of the table corresponding to the nodes of the chosen edge.
4. Vic performs desired encryptions to verify that nodes really have colors as shown.

A MORE CONCISE ZERO-KNOWLEDGE PROTOCOL

Common Input: A graph $G = (V, E)$, $V = \{1, \ldots, n\}$, $n = |V|$.
Peggy's Input: A coloring $\phi \rightarrow \{1, 2, 3\}$
Repeat $t|E|$ the following steps in order soundness error e^{-t}

- Peggy selects a random permutation π on $\{1, 2, 3\}$ and commits herself to Vic for all values $\pi(\phi(i))$
- Vic chooses randomly an edge $e = (j, k)$ and sends it to Peggy {asking her to show coloring of its nodes}
- Peggy decommit herself to reveal $\pi(j)$ and $\pi(k)$
- Vic checks whether colors are different and match the commitment received in the first step.

Zero-knowledge proofs for other **NP**-complete problems can be obtained using the standard reduction.

ZERO-KNOWLEDGE PROOF of HAMILTONIAN CYCLE

Peggy and Vic know a graph G. Peggy will prove to Vic that G has a Hamiltonian cycle (and that she knows how to draw Hamiltonian cycle in G) - cycle that passes through each node exactly once. To do that they perform several times the following rounds:

1. Peggy creates randomly a graph H isomorphic to G and commits herself to H before Vic.
2. Vic asks Peggy to do, randomly chosen, one of the following tasks:
   - Show isomorphism between G and H
   - Draw Hamiltonian cycle in H.
3. In case she is asked to show isomorphism, she open her commitment to make H public and show the required isomorphism.
4. In case she is asked to draw Hamiltonian cycle in H she opens her commitment and shows the Hamiltonian cycle. She can do that because she knows how to do that for G and knows isomorphism between G and H.

1. **Completeness** In case Peggy knows how to draw Hamiltonian cycle for G she always does well what Vic asks.
2. **Soundness**. In case G does not have Hamiltonian cycle, Peggy is able to do what Vic asks only with some small probability.
3. **Zero-knowledge**. None of the responses of Peggy reveals a Hamiltonian cycle of G. It reveals only either isomorphism or a Hamiltonian cycle of H. Vic would need to know both simultaneously in order to be able to draw a Hamiltonian cycle of G.

ZERO-KNOWLEDGE PROOFS in CRYPTOGRAPHIC PROTOCOLS

The fact that for a big class of statements there are zero-knowledge proofs can be used to design secure cryptographic protocols. (All languages in PSPACE have zero-knowledge proofs provided unbreakable encryptions exist (if one-way functions exist.)

A cryptographic protocol can be seen as a set of interactive programs to be executed by non-trusting parties.

Each party keeps secret her local input.

The protocol specifies the actions parties should take, depending on their local secrets and previous messages exchanged.

The main problem in this setting is how can a party verify that the other parties have really followed the protocol?

The way out: a party A can convince a party B that the transmitted message was completed according to the protocol without revealing its secrets.

An idea how to design a reliable protocol

1. Design a protocol under the assumption that all parties follow the protocol.
2. Transform protocol, using a method to make zero-knowledge proofs out of normal ones, into a protocol with communication based on zero-knowledge proofs, which preserves both correctness and privacy and works even if some parties have an adversary behavior.

HISTORY of ZERO-KNOWLEDGE PROOFS

Research in zero-knowledge proofs have been motivated by identification problems and an approach where one party wants to prove his identity by demonstrating some secret knowledge (say a password) but does not want that other parties learn anything about this knowledge.

The concept o zero-knowledge proofs was first published in 1985 by Shafi Goldwasser, Silvio Micali and Charles Rackoff.

Early version of their paper were from 1985 and were rejected three times from major conferences (FOCS83, STOC84, FOCS84).

The wide applicability of zero-knowledge proofs was first demonstrated in 1986 by Goldreich, Micali, Wigderson, who showed how to construct zero-knowledge proofs for any **NP**-set.

INTERACTIVE PROOF for QUADRATIC RESIDUA

Input: An integer $n = pq$, where p, q are primes and $x \in QR(n)$.

Protocol: Repeat $\lg n$ times the following steps:

1. Peggy chooses a random $v \in Z_n^*$ and sends to Vic
$$y = v^2 \mod n.$$
2. Vic sends to Peggy a random $i \in \{0, 1\}$.
3. Peggy computes a square root u of x and sends to Vic
$$z = u^i v \mod n.$$
4. Vic checks whether
$$z^2 \equiv x^i y \mod n.$$

Vic accepts Peggy's proof that x is QR if he succeeds in point 4 in each of $\lg n$ rounds.

Completeness: This is straightforward:

Soundness If x is not a quadratic residue, then Peggy can answer only one of two possible challenges (only if $i = 0$), because in such a case y is a quadratic residue if and only if xy is not a quadratic residue.This means that Peggy will be caught in any given round of the protocol with probability $\frac{1}{2}$.

The overall probability that prover deceives Vic is therefore $2^{-\lg n} = \frac{1}{n}$.

ZERO-KNOWLEDGE PROOF for QUADRATIC NON-RESIDUE

IMPLICATIONS

Zero-knowledge proof for quadratic residue has an interesting implication.

No efficient algorithm for deciding quadratic residuosity modulo an n is known when n's factorization is not given.

Moreover, all known **NP** proofs for this problem exhibit a factorization of n.

The existence of zero-knowledge proof for deciding residuosity indicates that adding interaction to the proving process may decrease the amount of knowledge that must be communicated in order to prove a theorem.

ZERO-KNOWLEDGE PROOF for GRAPH ISOMORPHISM

Input: Given are two graphs G_1 and G_2 with the set of nodes $\{1, \ldots, n\}$.
Repeat the following steps n times:

1. Peggy chooses a random permutation π of $\{1, \ldots, n\}$ and computes H to be the image of G_1 under the permutation π, and sends H to Vic.
2. Vic chooses randomly $i \in \{1, 2\}$ and sends it to Peggy. {This way Vic asks for isomorphism between H and G_i.}
3. Peggy creates a permutation ρ of $\{1, \ldots, n\}$ such that ρ specifies isomorphism between H and G_i and Peggy sends ρ to Vic.
   {If i = 1 Peggy takes $\rho = \pi$; if i = 2 Peggy takes $\rho = \sigma o \pi$, where σ is a fixed isomorphic mapping of nodes of G_2 to G_1.}
4. Vic checks whether H provides the isomorphism between G_i and H.
   Vic accepts Peggy's "proof" if H is the image of G_i in each of the n rounds.

Completeness. It is obvious that if G_1 and G_2 are isomorphic then Vic accepts with probability 1.

Soundness: If graphs G_1 and G_2 are not isomorphic, then Peggy can deceive Vic only if she is able to guess in each round the i Vic chooses and then sends as H the graph G_i. However, the probability that this happens is 2^{-n}.

Observe that Vic can perform all computations in polynomial time. However, why is this proof a zero-knowledge proof?

WHY is the last "PROOF" a "ZERO-KNOWLEDGE PROOF"?

Because Vic gets convinced, by the overwhelming statistical evidence, that graphs G_1 and G_2 are isomorphic, but he does not get any information ("knowledge") that would help him to create isomorphism between G_1 and G_2.
In each round of the proof Vic see isomorphism between H (a random isomorphic copy of G_1) and G_1 or G_2, (but not between both of them)!

However, Vic can create such random copies H of the graphs by himself and therefore it seems very unlikely that this can help Vic to find an isomorphism between G_1 and G_2.
Information that Vic can receive during the protocol, called transcript, contains:

- The graphs G_1 and G_2.
- All messages i transmitted during communications by Peggy and Vic.
- Random numbers r used by Peggy and Vic to generate their outputs.

Transcript has therefore the form

$$T = ((G_1, G_2); (H_1, i_1, r_1), \ldots, (H_n, i_n, r_n)).$$

The essential point, which is the basis for the formal definition of zero-knowledge proof, is that Vic can forge transcript, without participating in the interactive proof, that look like "real transcripts", if graphs are isomorphic, by means of the following forging algorithm called simulator.

SIMULATOR

A simulator for the previous graph isomorphism protocol.

- $T = (G_1, G_2)$,
- **for** j = 1 **to** n **do**
  - Chose randomly $i_j \in \{1, 2\}$.
  - Chose ρ_j to be a random permutation of $\{1, \ldots, n\}$.
  - Compute H_j to be the image of G_{i_j} under ρ_j;
  - Concatenate (H_j, i_j, ρ_j) at the end of T.

CONSEQUENCES and FORMAL DEFINITION

The fact that a simulator can forge transcripts has several important consequences.

- Anything Vic can compute using the information obtained from the transcript can be computed using only a forged transcript and therefore participation in such a communication does not increase Vic capability to perform any computation.
- Participation in such a proof does not allow Vic to prove isomorphism of G_1 and G_2.
- Vic cannot convince someone else that G_1 and G_2 are isomorphic by showing the transcript because it is indistinguishable from a forged one.

Formal definition of what this means that a forged transcript "looks like" a real one:
Definition Suppose that we have an interactive proof system for a decision problem Π and a polynomial time simulator S.
Denote by $\Gamma(x)$ the set of all possible transcripts that could be produced during the interactive proof communication for a yes-instance x.
Denote F(x) the set of all possible forged transcripts produced by the simulator S.
For any transcript $T \in \Gamma(x)$, let $p_\Gamma(T)$ denote the probability that T is the transcript produced during the interactive proof. Similarly, for $T \in F(x)$, let $p_F(T)$ denote the probability that T is the transcript produced by S.
If $\Gamma(x) = F(x)$ and, for any $T \in \Gamma(x)$, $p_\Gamma(T) = p_F(T)$, then we say that the interactive proof system is a zero-knowledge proof system.

APPENDIX

APPENDIX

WHAT IS A PROOF?

- A proof is whatever convinces me (M. Even).
- A nice proof makes us wiser (Yu. Manin).
- A proof is a sequence of statements each of them is either an axiom or follows from previous statements by am easy deduction rule - whether a to-be-proof is indeed a proof it should be checkeable by a computer. (A proof is therefore a computation process.)

HISTORY of PROOFS

- The concept of the proof (of a theorem from axioms) was introduced during the first golden era of mathematics, in Greece, 600-300 BC.
- Most of their proofs were actually proofs of correctness of geometric algorithms.
- After 300 BC, Greek's ideas concerning proofs were actually ignored for 2000 years.
- During the second golden era of mathematics, in 17th century, the concept of the proof did not play very important role. Famous was encouragement of those times "Go on, God will be with you" whenever rigour of some methods or correctness of some theorem was questioned.
- An understanding that proofs are important has developed again at the end of 19th century and especially at the beginning of 20th century because
  - a lot of counter-intuitive phenomena have appeared in mathematics (for example a function that is everywhere continuous but has nowhere derivative);
  - paradoxes have appeared in the set theory. - For example, Does there exist a set of all sets?

A PROBLEM And ITS SOLUTION

The term zero-knowledge is a bit misleading in case of "zero-knowledge proof of membership (in a language L).

The reason being that in the basic setting the Prover reveals one bit of knowledge to the Verifier (namely weather the input belong to L).

However, it is possible to resolve this problem by considering zero-knowledge proofs of knowledge about knowledge.

In such a setting the goal is not to prove that input is (or is not) in the given language, but that Prover knows whether the input is (or is not) in the language.