IV054 CHAPTER 10: Protocols to do seemingly impossible

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

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

In this chapter we deal with a variety of cryptographic protocols that allow to solve seemingly unsolvable problems.

We present several cryptographic protocols for such basic cryptographic primitives as bit commitment and oblivious transfer.

Of special importance are zero-knowledge protocols we discuss in second half of this chapter.

COIN-FLIPPING BY PHONE PROTOCOLS-EXAMPLE

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 a random number $y \in \{1,\ldots, n/2\}$, sends Alice $x = y^2 \bmod n$ and tells Alice: if you will guess *y correctly*, car is yours.

Alice computes four square roots $(x_1, n - x_1)$ and $(x_2, n - x_2)$ of x.

Let

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

Since $y \in \{1,\ldots,n/2\}$, Alice knows that either $y = x_1'$ or $y = x_2'$.

Alice then guesses whether $y = x_1'$ or $y = x_2'$ 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 y.)

COIN TOSSING

- In any coin tossing protocol both parties should influence outcome and should accept the outcome. Both outcomes should have the same probability.
- Requirements for a good 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 second party and in such a case can disrupt the protocol - to produce *reject*. A way out is to require that in case of correct behavior no outcome should have probability > 1/2.

IV054 COIN TOSSING USING ONE-WAY FUNCTION f

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 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 a one-way function f that $f(r_1) = f(r_2)$.

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) $f(b, x)$, $x \in X$.

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 $b \in \{0,1\}$ 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 Function

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.

TWO SPECIAL BIT COMMITMENT SCHEMES

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

Commitment: $f(b, x) = m^b x^2 \bmod n$ *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 $mx_1^2 = x_2^2 \bmod n$ and therefore the scheme is binding.

Bit commitment scheme II. p is a large Blum prime, $X = \{0,1,\ldots, p-1\} = Y$, α is a primitive element of Z_p^*.

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

where

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

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 heads = 0, tails = 1) 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.
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 knows what will be the outcome, but Bob 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.

IV054 A commitment scheme based on discret 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

$$q \mid (p-1).$$

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.

Commitment phase:

To commit to an $m \in \{0,\ldots,q-1\}$, Alice chooses a random $r \in \{0,\ldots,q-1\}$, 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 $\overline{m} \neq m$, then $g^r v^m = g^{\bar{r}} v^{\overline{m}}$ and therefore

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

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

$$E_k(\mathbf{r}b)$$

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 $e_k(b)=e_l(\neg b)$.

Let $com(r, m) = g^r v^m$ denote commitment to m in the commitment scheme based on discrete logarithm. If $r_1, r_2, m_1, m_2 \in \{0,\ldots,q-1\}$, 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 use 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.

Each voter V_i chooses his vote $m_i \in \{0,1\}$, a random $r_i \in \{0,\ldots, q-1\}$ and computes his voting commitment $c_i = com(r_i, m_i)$. Then V_i makes c_i public and sends $e_T(g^{r_i})$ to TA and TA computes

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

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

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 unethical 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 autencity, 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 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 that Alice knows 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 1/2 and "garbage" with the probability 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 $y = x^2 \bmod n$ to Alice.

(3) Alice computes four square roots $\pm x_1, \pm x_2$ of $y \pmod n$ 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 imagine as a box with three inputs and one output.

INPUTS: Alice inputs: x_0 and x_1;
……Bob inputs a bit c

OUTPUT: Bob gets as the output: x_c

Implementation of oblivious transfer

- Alice generates two key pairs for a PKC *P* and sends their public keys to Bob.
- Bob chooses a to-be random secret key *k* for a SKC *S*, encrypts it by one of Alice's public keys and sends it to Alice.
- Alice uses her two secret keys to decrypt the message she received. One of outcome is garbage *g*, another one is *k*, but she does not know which one.
- 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.

Power of Oblivious Transfers

- 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 olivious 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 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 so much she can learn from x and f(x, y)
  - Bob learns about x only so much 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 bit commitment:

COMMITMENT PHASE:

1.Alice selects a random bit r and her commitment bit b;
2. Alice inputs x_0 = r and x_1 =r xor 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 xor bc

IV054 Mental poker playing by phone - two players

Basic requirements:

- All hands (sets of 5 cards) are equally likely.
- The hands of Alice and Bob are disjoint.
- Both players know their own hand 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 shuffles cards, encrypts them 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 hands and tells them Bob.

(3) Alice chooses another five of $e_B(w_i)$, encrypt them with e_A and sends to Bob.

(4) Bob applies d_B to five values $e_A(e_B(w_i))$ he got from Alice and sends $e_A(w_i)$ to Alice as Alice's hands.

Remarque: The cryptosystem that is 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 with three players

1. Alice encrypts 52 cards $w_1,\ldots,w_{52}$ with e_A and sends them in a random order to Bob.
2. Bob, who cannot read the cards, 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 encrypted cards $e_A(w_i)$ to Carol.
3. Carol, who cannot read any of the messages, chooses five at random, encrypts them 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 the messages 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 hands.

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 proof protocols

To the most important primitives for cryptographic protocols, and at the same time very counterintuitive primitives, belong so-called zero-knowledge proof protocols (of knowledge).

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

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

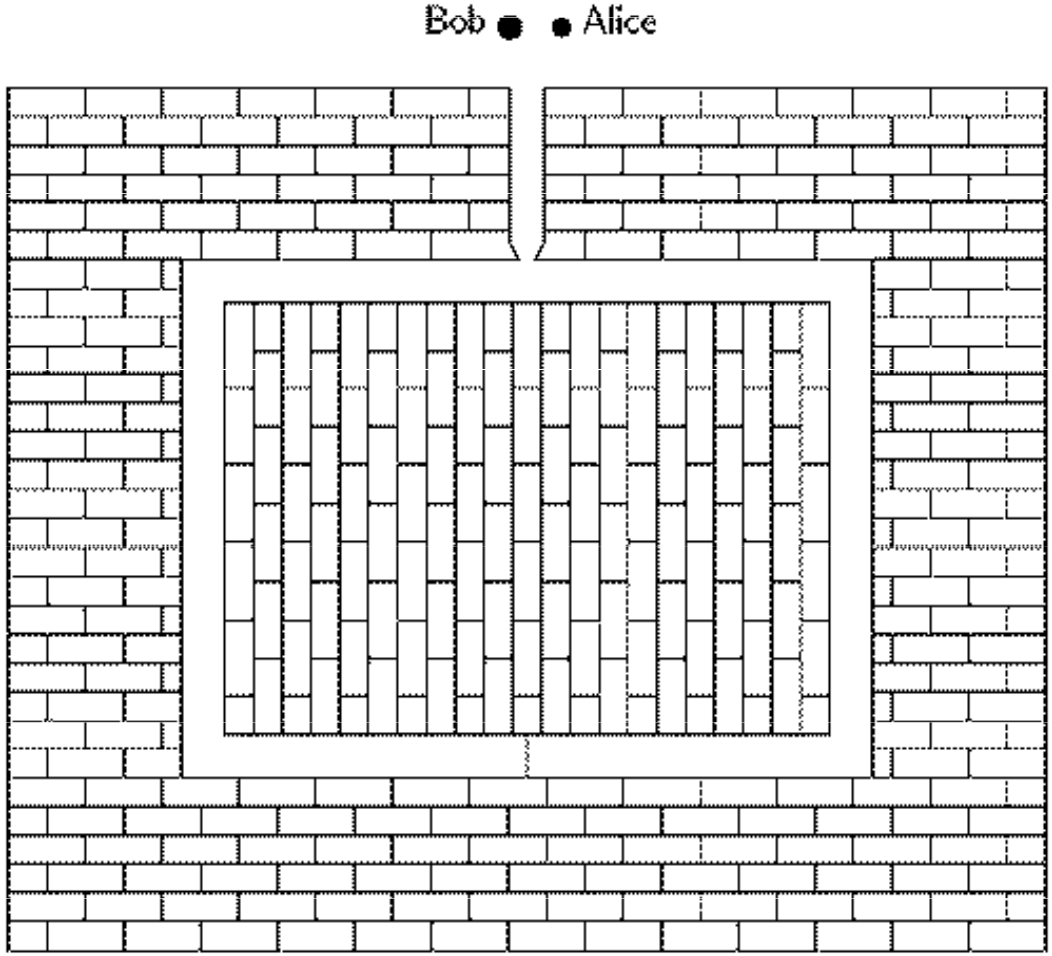
Zero-knowledge proof protocols are a special type of so-called interactive proof systems.

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-collorable.

Illustrative example

(A cave with a 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?



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 the 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 instant of the problem.

The 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.

INTERACTIVE PROOF PROTOCOLS

In an interactive proof system there are two parties

- An (all powerful) Prover, often called Peggy (a randomized algorithm that uses a private random number generator);
- A (little (polynomially) powerful) 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 proof 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.

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 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 clearly 1.

Soundness: If G_1 is isomorphic to G_2, then Peggy can deceive Vic if and only if she correctly guesses n times the i Vic choosed randomly. Probability that this happens is 2^{-n}.

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

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.

Assume that Prover and Verifier posses an input *x* (or Prover has secret knowledge) and Prover wants to convince Verifier that *x* has a certain properties and that Prover knows how to proof 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.

IV054 Zero-knowledge proof protocols 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.

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 correctnes 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 overhelming statistical evidence, that *T* is true, if *T* is indeed true, but no Prover is not able to convince Verifier that *T* is true, if this is not so. In additions, 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 posseses a secret.

AGE DIFFERENCE FINDING PROTOCOLS

Alice and Bob wants to find out who 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 Age of Bob: j, age of Alice: i.

1. Bob choose a random x, computes $k = e_A(x)$ and sends 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

$$z_u = y_u \bmod p, \qquad 1 \leq u \leq 100 \qquad (*)$$

and verifies that for all $u \neq v$

$$|z_u - z_v| \geq 2 \text{ and } z_u \neq 0 \qquad (**)$$

(If this it not the case, Alice choose a new p, repeats computations in (*) and checks (**) again.)

Finally, Alice sends Bob the following sequence (order is important).

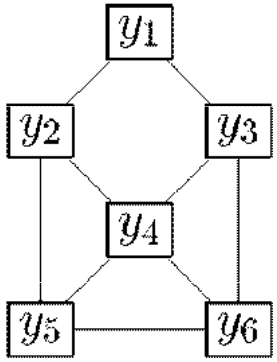
$$z_1,\ldots,z_i,\quad z_{i+1}+1,\ldots,z_{100}+1,\quad p$$
$$z'_1,\ldots,z'_i,\quad z'_{i+1},\ldots,z'_{100}$$

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 $i < j$.

$$i \geq j \Rightarrow z'_J = z_J \equiv y_J = d_A(k) \equiv x \pmod p$$
$$i < j \Rightarrow z'_J = z_J + 1 \not\equiv y_J = d_A(k) \equiv x \pmod p$$

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 choose a random permutation of colors, recolors G, and encrypts, for $i = 1,2,\ldots,n$, 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 encryptions to verify that nodes really have colors as shown.

Zero-knowledge proofs and 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 NP have zero-knowledge proofs.)

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

Each party keeps secret a 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 known methods how to make zero-knowledge proofs out of normal ones, into a protocol in which communication is based on zero-knowledge proofs, preserves both correctness and privacy and works even if some parties display an adversary behavior.

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

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

Completeness 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 1/2 .

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

Zero-knowledge proof for graph isomorphism

Input: 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 $\rho = \sigma \circ \pi$, 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 $i_i \in \{1,2\}$.
  - Chose ρ_i to be a random permutation of $\{1,\ldots,n\}$.
  - Compute H_i to be the image of G_{iJ} 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 what does it mean 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.

$\Gamma(x) = F(x)$ 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.

Proof for graph isomorphism protocol

Theorem The interactive proof system for *Graph isomorphism* is a perfect zero-knowledge proof if Vic follows protocol.

Proof Let G_1 and G_2 be isomorphic. A transcript (real or forged) contains triplets (H_J, i_J, ρ_J).

The set R of such triplets contains $2n!$ elements (because each pair i, ρ uniquely determines H and there are $n!$ permutation ρ.

In each round of the simulator each triplet occurs with the same probability, that is all triplets have probability $\frac{1}{(2n!)^n}$.

Let us now try to determine probability that a triplet (H, i, ρ) occurs at a *j-th* round of the interactive proof.

i is clearly chosen with the same probability. Concerning ρ this is either randomly chosen permutation π or a composition π with a fixed permutation. Hence all triplets (H, i, ρ) have the same probability $\frac{1}{(2n!)^n}$.

The next question is whether the above graph isomorphism protocol is zero-knowledge also if Vic does not follow fully the protocol.

The case Vic does not follow protocol

It is usually much more difficult to show that an interactive proof system is zero-knowledge even if Vic does not follow the protocol.

In the case of graph isomorphism protocol the only way Vic can deviate from the protocol is that *i* he does not choose in a completely random way.

The way around this difficulty is to prove that, no matter how a “cheating” Vic deviates from the protocol, there exists a polynomial-time simulator that will produce forged transcripts that “look like” the transcript *T* of the communication produced by Peggy and (the cheating) Vic during the interactive proof.

As before, the term “looks like" is formalized by requiring that two probability distributions are the same.

Definition Suppose that we have an interactive proof system for a decision problem Π.

Let *V** be any polynomial time probabilistic algorithm that a (possibly cheating) Verifier uses to generate his challenges.

The case Vic does not follow protocol

Denote by $\Gamma(V^*, x)$ the set of all possible transcripts that could be produced as a result of Peggy and V^* carrying out the interactive proof with a yes-instance x of Π.

Suppose that for every such V^* there exists an expected polynomial time probabilistic algorithm $S^* = S^*(V^*)$ (the simulator) which will produce a forged transcript.

Denote by $F(V^*, x)$ the set of possible forged transcripts.

For any transcript $T \in \Gamma(V^*, x)$, let $p_{\Gamma,V^*}(T)$ denote the probability that T is the transcript produced by V^* taking part in the interactive proof.

Similarly, for $T \in F(x)$, let $p_{F,V^*}(T)$ denote the probability that T is the (forged) transcript produced by S^*.

If $\Gamma(V^*, x) = F(V^*, x)$ and for any $T \in \Gamma(V^*, x)$, $p_{F,V^*}(T) = p_{\Gamma,V^*}(T)$, then the interactive proof system is said to be a perfect zero-knowledge protocol.

ADDITIONS

- It can be proved that the graph isomorphism protocol is zero-knowledge even in the case Vic cheats.
- If, in an interactive proof system, the probability distributions specified by the protocols with Vic and with simulator are the same, then we speak about perfect zero-knowledge proof system.
- If, in an interactive proof system, the probability distributions specified by the protocols with Vic and with simulator are computationally indistinguishable in polynomial time , then we speak about computationally zero-knowledge proof system.