An introduction to Zero Knowledge Proofs

Vincenzo Iovino, University of Luxembourg

June 25, 2019

NP, BPP, . . .

NP

A decision problem is the problem of deciding if a string x belongs to some language (set of strings) L.
A language is in NP if there is a relation $\mathcal{R}$ and a polynomial $p(\cdot)$ such that $x \in L$ if and only if there is a witness $y, |y| \leq p(|x|)$ such that $\mathcal{R}(x, y) = 1$.
Example: Sudoku (or your favorite game) is in NP because if I give you an alleged solution y to an instance x of the Sudoku, you can easily check that y is indeed a solution.

BPP

A language $L \in \mathtt{BPP}$ if there is a probabilistic polynomial-time algorithm (PPT) $\mathcal{A}$ such that:

- For any $x \in L$, $\mathcal{A}(x) = 1$ with probability $\geq 2/3$.
- For any $x \notin L$, $\mathcal{A}(x) = 1$ with probability $\leq 1/3$.

That is, BPP languages are easy to decide. We will be thus interested in non-BPP languages.

Interactive algorithms and protocols

- Each party in our protocol has a public input shared by each other party, a private input, a private memory, can use randomness (flip random coins) and interacts with other parties by exchanging messages on some shared memory.

Interactive algorithms and protocols

- Each party in our protocol has a public input shared by each other party, a private input, a private memory, can use randomness (flip random coins) and interacts with other parties by exchanging messages on some shared memory.
- This can be seen as proceeding in rounds.

Interactive algorithms and protocols

- Each party in our protocol has a public input shared by each other party, a private input, a private memory, can use randomness (flip random coins) and interacts with other parties by exchanging messages on some shared memory.
- This can be seen as proceeding in rounds.
- A party P starts the protocol on input the public input and its private input and performs some computation (possibly flipping random coins) and terminates its turn leaving some string on the shared memory..

Interactive algorithms and protocols

- Each party in our protocol has a public input shared by each other party, a private input, a private memory, can use randomness (flip random coins) and interacts with other parties by exchanging messages on some shared memory.
- This can be seen as proceeding in rounds.
- A party P starts the protocol on input the public input and its private input and performs some computation (possibly flipping random coins) and terminates its turn leaving some string on the shared memory..
- Then, another party Q observes that party P ended its round, reads the message P left on the shared memory and takes the turn continuing as before, etc.

Interactive algorithms and protocols

- Each party in our protocol has a public input shared by each other party, a private input, a private memory, can use randomness (flip random coins) and interacts with other parties by exchanging messages on some shared memory.
- This can be seen as proceeding in rounds.
- A party P starts the protocol on input the public input and its private input and performs some computation (possibly flipping random coins) and terminates its turn leaving some string on the shared memory..
- Then, another party Q observes that party P ended its round, reads the message P left on the shared memory and takes the turn continuing as before, etc.
- In the end, each party outputs some string that is the result of its local computation in the last round.

Interactive algorithms and protocols

- Each party in our protocol has a public input shared by each other party, a private input, a private memory, can use randomness (flip random coins) and interacts with other parties by exchanging messages on some shared memory.
- This can be seen as proceeding in rounds.
- A party P starts the protocol on input the public input and its private input and performs some computation (possibly flipping random coins) and terminates its turn leaving some string on the shared memory..
- Then, another party Q observes that party P ended its round, reads the message P left on the shared memory and takes the turn continuing as before, etc.
- In the end, each party outputs some string that is the result of its local computation in the last round.
- Denote by T_i the state of the shared memory after each round i. A transcript of the protocol is the sequence of the states T_i's.

Interactive Proofs

An interactive proof $\Pi = (\mathcal{P}, \mathcal{V})$ for $\mathtt{NP}$ language L with witness relation $\mathcal{R}_L$ satisfies:

Completeness

For any pair $(x, w) \in \mathcal{R}_L$, the probability (taken over the random choices of $\mathcal{P}$ and $\mathcal{V}$) that at the end of the interaction $\mathcal{V}(x)$ outputs 1 (i.e., *accepts* x) after interacting with $\mathcal{P}(x, w)$ is 1.

Statistical or computational Soundness

For any, possibly dishonest, prover $\mathcal{P}^\star$, any $x \notin L$, the probability (taken over the random choices of $\mathcal{P}^\star$ and $\mathcal{V}$) that at the end of the interaction $\mathcal{V}$ accepts x is negligible in $|x|$.

Proofs useful only for hard languages

If a language is in $\mathtt{BPP}$, then there is no need for a ZK proof of membership in L because a verifier can check if an input $x \in L$ by itself. Interaction is usually useful only for non-$\mathtt{BPP}$ languages.

Interactive Zero-Knowledge Proofs

An interactive proof $\Pi = (\mathcal{P}, \mathcal{V})$ for $\mathtt{NP}$ language L can be additionally HVZK or ZK:

Honest-Verifier Zero-Knowledge (HVZK)

There exists a PPT simulator algorithm Sim that takes as input instance $x \in L$ and outputs a transcript that has the same distribution as a honest transcript of the execution of $\mathcal{V}(x)$ with $Prover(x, w)$, for any witness w to x.

Zero-Knowledge (ZK)

For any, possibly dishonest, PPT verifier $\mathcal{V}^{\star}$, there exists a PPT simulator Sim (that can depend on $\mathcal{V}^{\star}$) with the above property. Output of Sim can be statistically or computationally indistinguishable from honest transcript and in such case we talk about statistical or computational ZK.

Conflict between ZK and soundness and non-interactivity

ZK clashes with perfect soundness

If there exists a ZK proof with perfect soundness, the simulator can be used to decide L: run Sim on input x to get a transcript and outputs the decision that the verifier would take from this transcript.

ZK clashes with non-interaction

There is no one-message ZK proof even with statistical soundness.

Nevertheless, we will see that non-interactive ZK proofs are possible in a special model that is of practical relevance.

Σ-protocols [Cramer, Damgard, Schoenmakers '94]

Special case of public-coin HVZK proofs

Σ-protocol for NP language L with witness relation $\mathcal{R}_L$:

- 3-round public-coin: transcript (a, c, z)
- **Perfect Completeness**
- **Special Soundness**:

  given x and accepting transcripts (a, c, z) and (a, c', z') for x with $c \neq c'$:

  one can efficiently compute w s.t. $(x, w) \in \mathcal{R}_L$.

- **Perfect Special HVZK**:
  Sim takes $x \in L$ and challenge c and outputs an accepting conversation (a, c, z) for x

Example: Sigma protocol for DH tuple

- **Relation $\mathcal{R}$ for DH tuples**
  - We work in a group of prime order p, e.g., the group of quadratic residues modulo a prime $q \stackrel{\triangle}{=} 2p + 1$.
  - $(g, h, u, v) \in \mathcal{R}$ iff $\exists$ w s.t. $u = g^w$ and $v = h^w$.
  - Useful in many applications
- **Protocol**
  - *Prover* chooses a random r and sends $a = g^r, b = h^r$.
  - $\mathcal{V}$ sends a random c
  - *Prover* sends $z = r + cw \mod q$.
  - $\mathcal{V}$ accepts iff $g^z = au^c$ and $h^z = bv^c$.

Example: Sigma protocol for DH tuple

- **Completeness:** Straightforward.
- **Special soundness:**
  - Given (a, b, c, z),(a, b, c', z'), we have $g^z = au^c, g^{z'} = au^{c'}, h^z = bv^c, h^{z'} = bv^{c'}$ and so (can be seen that)
    $w = (z - z')/(c - c') \mod q$.
- **Special HVZK:**
  - Given (g, h, u, v) and c, choose random z and compute
    - $a = g^z u^{-c}$.
    - $b = h^z v^{-c}$.
- **Note:** no *additional* computational assumption.

Basic properties of sigma protocols

Basic properties of sigma protocols

- **Any sigma protocol is an interactive proof with soundness error 2^{-t}, with t the bit length of the challenge**
  - This is because special soundness implies that if $x \notin L$, for each first message a, there is at most one challenge c such that, for some z, (a, c, z) is an accepting transcript for x. Since c is a uniformly chosen string of length t, the soundness error is thus 2^{-t}.

Basic properties of sigma protocols

- **Any sigma protocol is an interactive proof with soundness error 2^{-t}, with t the bit length of the challenge**
  - This is because special soundness implies that if $x \notin L$, for each first message a, there is at most one challenge c such that, for some z, (a, c, z) is an accepting transcript for x. Since c is a uniformly chosen string of length t, the soundness error is thus 2^{-t}.
- **Properties of sigma protocols are invariant under parallel composition**

Basic properties of sigma protocols

- **Any sigma protocol is an interactive proof with soundness error 2^{-t}, with t the bit length of the challenge**
  - This is because special soundness implies that if $x \notin L$, for each first message a, there is at most one challenge c such that, for some z, (a, c, z) is an accepting transcript for x. Since c is a uniformly chosen string of length t, the soundness error is thus 2^{-t}.
- **Properties of sigma protocols are invariant under parallel composition**
- **Any sigma protocol is a proof of knowledge with error 2^{-t}**

Basic properties of sigma protocols

- **Any sigma protocol is an interactive proof with soundness error 2^{-t}, with t the bit length of the challenge**
  - This is because special soundness implies that if $x \notin L$, for each first message a, there is at most one challenge c such that, for some z, (a, c, z) is an accepting transcript for x. Since c is a uniformly chosen string of length t, the soundness error is thus 2^{-t}.
- **Properties of sigma protocols are invariant under parallel composition**
- **Any sigma protocol is a proof of knowledge with error 2^{-t}**
  - The difference between the probability that Prove* convinces $\mathcal{V}$ and the probability that Ext obtains a witness is at most 2^{-t}

AND, OR and compound statements of sigma protocols

AND, OR and compound statements of sigma protocols

- **AND of multiple statements:** run all in parallel using the same challenge for all

AND, OR and compound statements of sigma protocols

- **AND of multiple statements:** run all in parallel using the same challenge for all
- **OR of two statements**

AND, OR and compound statements of sigma protocols

- **AND of multiple statements:** run all in parallel using the same challenge for all
- **OR of two statements**
  - *Prover* has a witness, w.l.o.g., for x_0 but not for x_1.

AND, OR and compound statements of sigma protocols

- **AND of multiple statements:** run all in parallel using the same challenge for all
- **OR of two statements**
  - *Prover* has a witness, w.l.o.g., for x_0 but not for x_1.
  - *Prover* chooses a random c_1 and runs SIM to get (a_1, c_1, z_1).

AND, OR and compound statements of sigma protocols

- **AND of multiple statements:** run all in parallel using the same challenge for all
- **OR of two statements**
  - *Prover* has a witness, w.l.o.g., for x_0 but not for x_1.
  - *Prover* chooses a random c_1 and runs SIM to get (a_1, c_1, z_1).
  - *Prover* computes first message a_0 by running the prover for the original statement on input (x_0, w_0), and sends (a_0, a_1) to the verifier.

AND, OR and compound statements of sigma protocols

- **AND of multiple statements:** run all in parallel using the same challenge for all
- **OR of two statements**
  - *Prover* has a witness, w.l.o.g., for x_0 but not for x_1.
  - *Prover* chooses a random c_1 and runs SIM to get (a_1, c_1, z_1).
  - *Prover* computes first message a_0 by running the prover for the original statement on input (x_0, w_0), and sends (a_0, a_1) to the verifier.
  - $\mathcal{V}$ sends a single challenge c to the prover.

AND, OR and compound statements of sigma protocols

- **AND of multiple statements:** run all in parallel using the same challenge for all
- **OR of two statements**
  - *Prover* has a witness, w.l.o.g., for x_0 but not for x_1.
  - *Prover* chooses a random c_1 and runs SIM to get (a_1, c_1, z_1).
  - *Prover* computes first message a_0 by running the prover for the original statement on input (x_0, w_0), and sends (a_0, a_1) to the verifier.
  - $\mathcal{V}$ sends a single challenge c to the prover.
  - *Prover* chooses c_0 s.t. c_0 XOR $c_1 = c$.

AND, OR and compound statements of sigma protocols

- **AND of multiple statements:** run all in parallel using the same challenge for all
- **OR of two statements**
  - *Prover* has a witness, w.l.o.g., for x_0 but not for x_1.
  - *Prover* chooses a random c_1 and runs SIM to get (a_1, c_1, z_1).
  - *Prover* computes first message a_0 by running the prover for the original statement on input (x_0, w_0), and sends (a_0, a_1) to the verifier.
  - $\mathcal{V}$ sends a single challenge c to the prover.
  - *Prover* chooses c_0 s.t. c_0 XOR $c_1 = c$.
  - *Prover* already has z_1 and can compute z_0 using the witness and sends c_0, c_1, z_0, z_1 to the verifier that checks that both (a_0, c_0, z_0) and (a_1, c_1, z_1) are accepting transcripts.

AND, OR and compound statements of sigma protocols

- **AND of multiple statements:** run all in parallel using the same challenge for all
- **OR of two statements**
  - *Prover* has a witness, w.l.o.g., for x_0 but not for x_1.
  - *Prover* chooses a random c_1 and runs SIM to get (a_1, c_1, z_1).
  - *Prover* computes first message a_0 by running the prover for the original statement on input (x_0, w_0), and sends (a_0, a_1) to the verifier.
  - $\mathcal{V}$ sends a single challenge c to the prover.
  - *Prover* chooses c_0 s.t. c_0 XOR $c_1 = c$.
  - *Prover* already has z_1 and can compute z_0 using the witness and sends c_0, c_1, z_0, z_1 to the verifier that checks that both (a_0, c_0, z_0) and (a_1, c_1, z_1) are accepting transcripts.
  - **Soundness:**

AND, OR and compound statements of sigma protocols

- **AND of multiple statements:** run all in parallel using the same challenge for all
- **OR of two statements**
  - *Prover* has a witness, w.l.o.g., for x_0 but not for x_1.
  - *Prover* chooses a random c_1 and runs SIM to get (a_1, c_1, z_1).
  - *Prover* computes first message a_0 by running the prover for the original statement on input (x_0, w_0), and sends (a_0, a_1) to the verifier.
  - $\mathcal{V}$ sends a single challenge c to the prover.
  - *Prover* chooses c_0 s.t. c_0 XOR $c_1 = c$.
  - *Prover* already has z_1 and can compute z_0 using the witness and sends c_0, c_1, z_0, z_1 to the verifier that checks that both (a_0, c_0, z_0) and (a_1, c_1, z_1) are accepting transcripts.
  - **Soundness:**
    - *Prover* doesnt know a witness for both statements, so can only answer for a single challenge.

AND, OR and compound statements of sigma protocols

- **AND of multiple statements:** run all in parallel using the same challenge for all
- **OR of two statements**
  - *Prover* has a witness, w.l.o.g., for x_0 but not for x_1.
  - *Prover* chooses a random c_1 and runs SIM to get (a_1, c_1, z_1).
  - *Prover* computes first message a_0 by running the prover for the original statement on input (x_0, w_0), and sends (a_0, a_1) to the verifier.
  - $\mathcal{V}$ sends a single challenge c to the prover.
  - *Prover* chooses c_0 s.t. c_0 XOR $c_1 = c$.
  - *Prover* already has z_1 and can compute z_0 using the witness and sends c_0, c_1, z_0, z_1 to the verifier that checks that both (a_0, c_0, z_0) and (a_1, c_1, z_1) are accepting transcripts.
  - **Soundness:**
    - *Prover* doesnt know a witness for both statements, so can only answer for a single challenge.
    - This means that c defines a single challenge c' that is either c_0 or c_1 depending on which witness the prover knows, like in a regular proof.

AND, OR and compound statements of sigma protocols

- **AND of multiple statements:** run all in parallel using the same challenge for all
- **OR of two statements**
  - *Prover* has a witness, w.l.o.g., for x_0 but not for x_1.
  - *Prover* chooses a random c_1 and runs SIM to get (a_1, c_1, z_1).
  - *Prover* computes first message a_0 by running the prover for the original statement on input (x_0, w_0), and sends (a_0, a_1) to the verifier.
  - $\mathcal{V}$ sends a single challenge c to the prover.
  - *Prover* chooses c_0 s.t. c_0 XOR $c_1 = c$.
  - *Prover* already has z_1 and can compute z_0 using the witness and sends c_0, c_1, z_0, z_1 to the verifier that checks that both (a_0, c_0, z_0) and (a_1, c_1, z_1) are accepting transcripts.
  - **Soundness:**
    - *Prover* doesnt know a witness for both statements, so can only answer for a single challenge.
    - This means that c defines a single challenge c' that is either c_0 or c_1 depending on which witness the prover knows, like in a regular proof.
- **Can be generalized to any monotone formula [CDS94]**

The Fiat-Shamir (FS) transform applied to Σ-protocols

- FS transform turns an Σ-protocol into a non-interactive ZK argument (NIZK).
- **To prove a statement x:**
  - Suppose to have a good hash function H.
  - Generate a, compute $c = H(a, x)$, compute z.
  - Send (a, c, z)
- **To verify a proof (a, c, z) for statement x:**
  - Verifier checks that $c = H(a, x)$ and that (a, c, z) is an accepted transcript for the sigma protocol.

The Fiat-Shamir (FS) transform applied to Σ-protocols

- FS transform turns an Σ-protocol into a non-interactive ZK argument (NIZK).
- **To prove a statement** x**:**
  - Suppose to have a good hash function H.
  - Generate a, compute $c = H(a, x)$, compute z.
  - Send (a, c, z)
- **To verify a proof** (a, c, z) **for statement** x**:**
  - Verifier checks that $c = H(a, x)$ and that (a, c, z) is an accepted transcript for the sigma protocol.

Programmable RO model

The non-interactive version of the previous proof system for DH tuples is not known to be ZK. Given statement $x = (g, h, u, v)$, if you choose random c, z and compute $a = g^z u^{-c}, b = h^z v^{-c}$, with very low probability $H((a, b), x) = c$.
Trick: the proof of ZK is in a model where the simulator can "program" the RO, i.e., can set $H((a, b), x) = c$ at its like. That is, the ZK property is proven with a respect to a *different* hash functions than the one used in the actual protocol.

Proofs for Circuit Satisfiability

- We construct a non-interactive ZK (in the programmable RO model) argument for Boolean Circuit Satisfiability.

Proofs for Circuit Satisfiability

- We construct a non-interactive ZK (in the programmable RO model) argument for Boolean Circuit Satisfiability.
- Assume the circuits consist only of NAND gates.

Proofs for Circuit Satisfiability

- We construct a non-interactive ZK (in the programmable RO model) argument for Boolean Circuit Satisfiability.
- Assume the circuits consist only of NAND gates. A NAND gate with input wires w_0, w_0 outputs $w_2 = \neg(w_0 \wedge w_1)$.

Proofs for Circuit Satisfiability

- We construct a non-interactive ZK (in the programmable RO model) argument for Boolean Circuit Satisfiability.
- Assume the circuits consist only of NAND gates. A NAND gate with input wires w_0, w_0 outputs $w_2 = \neg(w_0 \wedge w_1)$.
- The prover does know the circuit and a witness to the satisfiability of the circuit whereas the verifier does know only the circuit. The witness consists of a Boolean assignment to the input wires of the circuit.

Proofs for Circuit Satisfiability

- We construct a non-interactive ZK (in the programmable RO model) argument for Boolean Circuit Satisfiability.
- Assume the circuits consist only of NAND gates. A NAND gate with input wires w_0, w_0 outputs $w_2 = \neg(w_0 \wedge w_1)$.
- The prover does know the circuit and a witness to the satisfiability of the circuit whereas the verifier does know only the circuit. The witness consists of a Boolean assignment to the input wires of the circuit.
- The prover has to convince the verifier that the circuit has a satisfying assignment without leaking information about the assignment.

Proofs for Circuit Satisfiability

- We construct a non-interactive ZK (in the programmable RO model) argument for Boolean Circuit Satisfiability.
- Assume the circuits consist only of NAND gates. A NAND gate with input wires w_0, w_0 outputs $w_2 = \neg(w_0 \wedge w_1)$.
- The prover does know the circuit and a witness to the satisfiability of the circuit whereas the verifier does know only the circuit. The witness consists of a Boolean assignment to the input wires of the circuit.
- The prover has to convince the verifier that the circuit has a satisfying assignment without leaking information about the assignment.
- Boolean Circuit satisfiability is $\mathtt{NP}$-complete, so by $\mathtt{NP}$-reductions, we can construct a proof for any other $\mathtt{NP}$ relation.

Proofs for Circuit Satisfiability (2)

- We use exponential El Gamal encryption:
  - The public key $\mathrm{pk} = (g, h = g^w)$ and the secret key is w.-
  - The encryption of some message m in some "small" message space $\mathcal{M}$ with respect to pk is $(c_1 = g^r, c_2 = h^r \cdot g^m)$.
  - To decrypt a ciphertext $(c_1 = g^r, c_2 = h^r \cdot g^m)$, compute $c_2/c_1^w = g^m$ and extract m by brute force.

Proofs for Circuit Satisfiability (2)

- We use exponential El Gamal encryption:
  - The public key $\mathrm{pk} = (g, h = g^w)$ and the secret key is w.-
  - The encryption of some message m in some "small" message space $\mathcal{M}$ with respect to pk is $(c_1 = g^r, c_2 = h^r \cdot g^m)$.
  - To decrypt a ciphertext $(c_1 = g^r, c_2 = h^r \cdot g^m)$, compute $c_2/c_1^w = g^m$ and extract m by brute force.
- The NIZK for DH can be used to prove that a ciphertext (c_1, c_2) for public key pk decrypts to m by showing that the tuple $(g, c_1, \mathrm{pk} = g^w, c_2/g^m = c_1^w)$ is DH for witness w.

Proofs for Circuit Satisfiability (2)

- We use exponential El Gamal encryption:
  - The public key $\mathsf{pk} = (g, h = g^w)$ and the secret key is w.-
  - The encryption of some message m in some "small" message space $\mathcal{M}$ with respect to pk is $(c_1 = g^r, c_2 = h^r \cdot g^m)$.
  - To decrypt a ciphertext $(c_1 = g^r, c_2 = h^r \cdot g^m)$, compute $c_2/c_1^w = g^m$ and extract m by brute force.
- The NIZK for DH can be used to prove that a ciphertext (c_1, c_2) for public key pk decrypts to m by showing that the tuple $(g, c_1, \mathsf{pk} = g^w, c_2/g^m = c_1^w)$ is DH for witness w.
- Using OR proofs, we have a NIZK to prove that a ciphertext decrypts to m_1 or m_2 and in particular a NIZK to prove that a ciphertext decrypts to a bit.

Proofs for Circuit Satisfiability (2)

- We use exponential El Gamal encryption:
  - The public key $\mathsf{pk} = (g, h = g^w)$ and the secret key is w.-
  - The encryption of some message m in some "small" message space $\mathcal{M}$ with respect to pk is $(c_1 = g^r, c_2 = h^r \cdot g^m)$.
  - To decrypt a ciphertext $(c_1 = g^r, c_2 = h^r \cdot g^m)$, compute $c_2/c_1^w = g^m$ and extract m by brute force.
- The NIZK for DH can be used to prove that a ciphertext (c_1, c_2) for public key pk decrypts to m by showing that the tuple $(g, c_1, \mathsf{pk} = g^w, c_2/g^m = c_1^w)$ is DH for witness w.
- Using OR proofs, we have a NIZK to prove that a ciphertext decrypts to m_1 or m_2 and in particular a NIZK to prove that a ciphertext decrypts to a bit.
- Exponential El Gamal is homomorphic, i.e., if I have two ciphertexts ct_1 and ct_2 encrypting resp. m_1 and m_2, I can "multiply" them together to get encryption of $m_1 + m_2$.

Proofs for Circuit Satisfiability (3)

- The prover creates an El Gamal public key pk and associates a ciphertext to each wire of the circuit in the following way.

Proofs for Circuit Satisfiability (3)

- The prover creates an El Gamal public key pk and associates a ciphertext to each wire of the circuit in the following way.
- To the i-th input wire corresponding to a bit w_i of the witness, the prover associates a ciphertext encrypting w_i.

Proofs for Circuit Satisfiability (3)

- The prover creates an El Gamal public key pk and associates a ciphertext to each wire of the circuit in the following way.
- To the i-th input wire corresponding to a bit w_i of the witness, the prover associates a ciphertext encrypting w_i.
- Each wire of the circuit that is not an input wire is an output wire of some gate. The prover evaluates the circuit at each gate and associates to the output wire of a gate the encryption of the corresponding bit.

Proofs for Circuit Satisfiability (3)

- The prover creates an El Gamal public key pk and associates a ciphertext to each wire of the circuit in the following way.
- To the i-th input wire corresponding to a bit w_i of the witness, the prover associates a ciphertext encrypting w_i.
- Each wire of the circuit that is not an input wire is an output wire of some gate. The prover evaluates the circuit at each gate and associates to the output wire of a gate the encryption of the corresponding bit.
- To each output wire of a gate and to each input wire of the circuit, the prover also adds an OR proof of the fact that the associated ciphertext decrypts to 0 or 1, i.e., it is a bit.

Proofs for Circuit Satisfiability (3)

- The prover creates an El Gamal public key pk and associates a ciphertext to each wire of the circuit in the following way.
- To the i-th input wire corresponding to a bit w_i of the witness, the prover associates a ciphertext encrypting w_i.
- Each wire of the circuit that is not an input wire is an output wire of some gate. The prover evaluates the circuit at each gate and associates to the output wire of a gate the encryption of the corresponding bit.
- To each output wire of a gate and to each input wire of the circuit, the prover also adds an OR proof of the fact that the associated ciphertext decrypts to 0 or 1, i.e., it is a bit.
- Let t be a ciphertext encrypting the integer -2. For each gate with ciphertexts $\mathrm{ct}_0, \mathrm{ct}_1$ associated to its input wires and ciphertext ct_2 associated to its output wire, the prover also adds an OR proof of the fact that the ciphertext $G \triangleq \mathrm{ct}_0 * \mathrm{ct}_1 * \mathrm{ct}_2^2 * t$ decrypts to 0 or 1, i.e., that $w_0 + w_1 + 2w2 - 2 \in \{0, 1\}$.

Proofs for Circuit Satisfiability (3)

- The prover creates an El Gamal public key pk and associates a ciphertext to each wire of the circuit in the following way.
- To the i-th input wire corresponding to a bit w_i of the witness, the prover associates a ciphertext encrypting w_i.
- Each wire of the circuit that is not an input wire is an output wire of some gate. The prover evaluates the circuit at each gate and associates to the output wire of a gate the encryption of the corresponding bit.
- To each output wire of a gate and to each input wire of the circuit, the prover also adds an OR proof of the fact that the associated ciphertext decrypts to 0 or 1, i.e., it is a bit.
- Let t be a ciphertext encrypting the integer -2. For each gate with ciphertexts $\mathrm{ct}_0, \mathrm{ct}_1$ associated to its input wires and ciphertext ct_2 associated to its output wire, the prover also adds an OR proof of the fact that the ciphertext $G \triangleq \mathrm{ct}_0 * \mathrm{ct}_1 * \mathrm{ct}_2^2 * t$ decrypts to 0 or 1, i.e., that $w_0 + w_1 + 2w2 - 2 \in \{0, 1\}$.
- Finally, the prover shows that the output gate decrypts to 1, i.e., that the circuit is satisfied by the assignment.

Proofs for Circuit Satisfiability (4)

- Soundness: Using the homomorphic property of El Gamal and the above fact, the verifier can check the consistency as follows.

Proofs for Circuit Satisfiability (4)

- Soundness: Using the homomorphic property of El Gamal and the above fact, the verifier can check the consistency as follows.
- Fact: If w_0, w_1 are the values corresponding to the input wires of a gate and w_2 is the value corresponding to its output wire, it is easy to see that w_0, w_1, w_2 are a valid assignment (i.e., $w_2 = \neg(w_0 \wedge w_1)$) iff $w_0 + w_1 + 2w_2 - 2 \in \{0, 1\}$ and $w_0, w_1, w_2 \in \{0, 1\}$.

Proofs for Circuit Satisfiability (4)

- Soundness: Using the homomorphic property of El Gamal and the above fact, the verifier can check the consistency as follows.
- Fact: If w_0, w_1 are the values corresponding to the input wires of a gate and w_2 is the value corresponding to its output wire, it is easy to see that w_0, w_1, w_2 are a valid assignment (i.e., $w_2 = \neg(w_0 \wedge w_1)$) iff $w_0 + w_1 + 2w_2 - 2 \in \{0, 1\}$ and $w_0, w_1, w_2 \in \{0, 1\}$.
- The verifier verifies (1) that the ciphertext associated to each input wire and to any other output wire encrypts a bit.

Proofs for Circuit Satisfiability (4)

- Soundness: Using the homomorphic property of El Gamal and the above fact, the verifier can check the consistency as follows.
- Fact: If w_0, w_1 are the values corresponding to the input wires of a gate and w_2 is the value corresponding to its output wire, it is easy to see that w_0, w_1, w_2 are a valid assignment (i.e., $w_2 = \neg(w_0 \wedge w_1)$) iff $w_0 + w_1 + 2w_2 - 2 \in \{0, 1\}$ and $w_0, w_1, w_2 \in \{0, 1\}$.
- The verifier verifies (1) that the ciphertext associated to each input wire and to any other output wire encrypts a bit.
- If ct_0 and ct_1 are the ciphertexts associated to the input wires of a gate encrypting resp. w_0 and w_1, and ct_2 is the ciphertext encrypting w_2 associated to the output wire of the gate, the verifier can compute using the homomorphic properties of El Gamal the ciphertext G encrypting $w_0 + w_1 + 2w_2 - 2$ and (2) verify that it decrypts to a bit, i.e., that $w_0 + w_1 + 2w_2 - 2 \in \{0, 1\}$.

Proofs for Circuit Satisfiability (4)

- Soundness: Using the homomorphic property of El Gamal and the above fact, the verifier can check the consistency as follows.
- Fact: If w_0, w_1 are the values corresponding to the input wires of a gate and w_2 is the value corresponding to its output wire, it is easy to see that w_0, w_1, w_2 are a valid assignment (i.e., $w_2 = \neg(w_0 \wedge w_1)$) iff $w_0 + w_1 + 2w_2 - 2 \in \{0, 1\}$ and $w_0, w_1, w_2 \in \{0, 1\}$.
- The verifier verifies (1) that the ciphertext associated to each input wire and to any other output wire encrypts a bit.
- If ct_0 and ct_1 are the ciphertexts associated to the input wires of a gate encrypting resp. w_0 and w_1, and ct_2 is the ciphertext encrypting w_2 associated to the output wire of the gate, the verifier can compute using the homomorphic properties of El Gamal the ciphertext G encrypting $w_0 + w_1 + 2w_2 - 2$ and (2) verify that it decrypts to a bit, i.e., that $w_0 + w_1 + 2w_2 - 2 \in \{0, 1\}$.
- By the previous Fact and (1) and (2), the verifier has the assurance that the ciphertext associated to each wire respects the computation with respect to the input wires.

Proofs for Circuit Satisfiability (4)

- Soundness: Using the homomorphic property of El Gamal and the above fact, the verifier can check the consistency as follows.
- Fact: If w_0, w_1 are the values corresponding to the input wires of a gate and w_2 is the value corresponding to its output wire, it is easy to see that w_0, w_1, w_2 are a valid assignment (i.e., $w_2 = \neg(w_0 \wedge w_1)$) iff $w_0 + w_1 + 2w_2 - 2 \in \{0, 1\}$ and $w_0, w_1, w_2 \in \{0, 1\}$.
- The verifier verifies (1) that the ciphertext associated to each input wire and to any other output wire encrypts a bit.
- If ct_0 and ct_1 are the ciphertexts associated to the input wires of a gate encrypting resp. w_0 and w_1, and ct_2 is the ciphertext encrypting w_2 associated to the output wire of the gate, the verifier can compute using the homomorphic properties of El Gamal the ciphertext G encrypting $w_0 + w_1 + 2w_2 - 2$ and (2) verify that it decrypts to a bit, i.e., that $w_0 + w_1 + 2w_2 - 2 \in \{0, 1\}$.
- By the previous Fact and (1) and (2), the verifier has the assurance that the ciphertext associated to each wire respects the computation with respect to the input wires.
- Finally, the verifier checks that the ciphertext associated with the output wire of the circuit decrypts to 1, thus the circuit is satisfiable.

Exercise

Exercise

Prove that the previous NIZK is ZK.

Exercise

Exercise

Prove that the previous NIZK is ZK.

Bonus: Using the previous NIZK for Circuit Satisfiability and the Sikoba's compilers from programs to circuits, can we give a a ZK proof that the previous NIZK is ZK?

Conclusions

- We introduced the powerful concept of ZK proof systems.

Conclusions

- We introduced the powerful concept of ZK proof systems.
- We presented efficient ZK proof systems for practical purposes.

Conclusions

- We introduced the powerful concept of ZK proof systems.
- We presented efficient ZK proof systems for practical purposes.
- We showed how a specific and efficient proof system can be turned into a general proof system to prove correctness of computation.

Conclusions

- We introduced the powerful concept of ZK proof systems.
- We presented efficient ZK proof systems for practical purposes.
- We showed how a specific and efficient proof system can be turned into a general proof system to prove correctness of computation.

Thank you for your attention!

For additional questions: vinciovino@gmail.com