ARIEL UNIVERSITY

אוניברסיטת אריאל בשומרון

Foundations of Modern Cryptography

Edward A. Hirsch

https://edwardahirsch.github.io/edwardahirsch

Neapolis University Pafos

Lecture 9: December 12, 2024

Lecture Plan

- How ZK helps to force malicious participants to be semi-honest.
- (maybe) Clarifications regarding cryptographic primitives.

If the screen seems frozen and I do not respond,
please call me in Telegram.

Task: Zero-Knowledge Proof

Alice

x, proof that $x \in L$

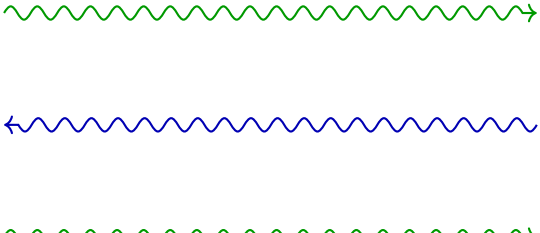
Bob

x

Task: Zero-Knowledge Proof

Alice

x, proof that $x \in L$



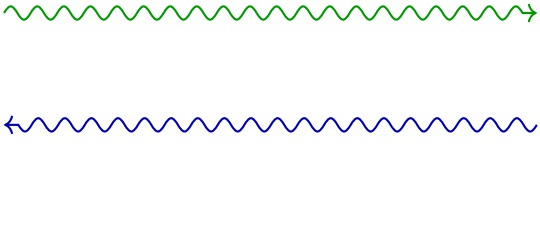
Bob

x

Task: Zero-Knowledge Proof

Alice

x, proof that $x \in L$



Bob

x

convinced that $x \in L$

Pictures courtesy AI Emojis Generator

Task: Zero-Knowledge Proof

Alice

x, proof that $x \in L$

Bob

x

convinced that $x \in L$

Bob gets convinced iff $x \in L$ (with certain probability of error).
Bob does not learn from this protocol about Alice's proof more than the fact $x \in L$.

Strong ZK: Bob also gets convinced that Alice indeed knows some proof that Bob could verify.

In the previous lecture we have seen such proofs for **NP** languages.

Pictures courtesy AI Emojis Generator

Plan

- Select public random string together to ensure it's generated uniformly.

Plan

- Select public random string together to ensure it's generated uniformly.
- Delegate computation for unknown input (perhaps bounded by prior commitment), while having some public "check-sum" information.

Plan

- Select public random string together to ensure it's generated uniformly.
- Delegate computation for unknown input (perhaps bounded by prior commitment), while having some public "check-sum" information.
- Persuade that a string is in the image of a function without revealing the preimage.

Plan

- Select public random string together to ensure it's generated uniformly.
- Delegate computation for unknown input (perhaps bounded by prior commitment), while having some public "check-sum" information.
- Persuade that a string is in the image of a function without revealing the preimage.
- Delegate computation for unknown input (perhaps bounded by prior commitment), while having some secret "check-sum" information and requesting to recalculate it.

Plan

- Select public random string together to ensure it's generated uniformly.
- Delegate computation for unknown input (perhaps bounded by prior commitment), while having some public "check-sum" information.
- Persuade that a string is in the image of a function without revealing the preimage.
- Delegate computation for unknown input (perhaps bounded by prior commitment), while having some secret "check-sum" information and requesting to recalculate it.
- Delegate sampling according to a polynomial-time samplable distribution.

Plan

- Select public random string together to ensure it's generated uniformly.
- Delegate computation for unknown input (perhaps bounded by prior commitment), while having some public "check-sum" information.
- Persuade that a string is in the image of a function without revealing the preimage.
- Delegate computation for unknown input (perhaps bounded by prior commitment), while having some secret "check-sum" information and requesting to recalculate it.
- Delegate sampling according to a polynomial-time samplable distribution.
- Input commitment with provably random keys and with optional opening.

Plan

- Select public random string together to ensure it's generated uniformly.
- Delegate computation for unknown input (perhaps bounded by prior commitment), while having some public "check-sum" information.
- Persuade that a string is in the image of a function without revealing the preimage.
- Delegate computation for unknown input (perhaps bounded by prior commitment), while having some secret "check-sum" information and requesting to recalculate it.
- Delegate sampling according to a polynomial-time samplable distribution.
- Input commitment with provably random keys and with optional opening.
- Show it's enough for Yao's secure 2-party computation protocol.

Forcing uniform randomness: tossing a coin together

Failed attempt:

- Alice selects $a \in U_1$, Bob selects $b \in U_1$.
- They exchange the bits.
- Now $a \oplus b$ is distributed as U_1.

Forcing uniform randomness: tossing a coin together

Failed attempt:

- Alice selects $a \in U_1$, Bob selects $b \in U_1$.
- They exchange the bits.
- Now $a \oplus b$ is distributed as U_1.

The one who sends the bit the first looses control: the other one (say, Bob) can cheat, $b := 1 \oplus a$.

Forcing uniform randomness: tossing a coin together

Failed attempt:

- Alice selects $a \in U_1$, Bob selects $b \in U_1$.
- They exchange the bits.
- Now $a \oplus b$ is distributed as U_1.

The one who sends the bit the first looses control: the other one (say, Bob) can cheat, $b := 1 \oplus a$.

Correct scheme:

- Alice selects $a \in U_1$, Bob selects $b \in U_1$.
- Alice sends a commitment $C_s(a)$ for a keeping the key s in secret.
- Bob sends b.
- Alice sends s.
- They treat $a \oplus b$ as a randomly generated bit.

Forcing uniform randomness: tossing a coin together

Failed attempt:

- Alice selects $a \in U_1$, Bob selects $b \in U_1$.
- They exchange the bits.
- Now $a \oplus b$ is distributed as U_1.

The one who sends the bit the first looses control: the other one (say, Bob) can cheat, $b := 1 \oplus a$.

Correct scheme:

- Alice selects $a \in U_1$, Bob selects $b \in U_1$.
- Alice sends a commitment $C_s(a)$ for a keeping the key s in secret.
- Bob sends b.
- Alice sends s.
- They treat $a \oplus b$ as a randomly generated bit.

Exercise

Prove formally that $a \oplus b$ is distributed almost as U_1. In particular, the so-generated strings are computationally indistinguishable from U_n.

Delegating the computation, simple version

TASK: delegate computing $f(x)$ without knowing x; public checksum

- Alice knows x, Bob knows $h(x)$.

Delegating the computation, simple version

TASK: delegate computing $f(x)$ without knowing x; public checksum

- Alice knows x, Bob knows $h(x)$.
- Alice sends $f(x)$ (matching $h(x)$) to Bob without revealing x.
- Bob gets convinced he received a correct value.

Delegating the computation, simple version

TASK: delegate computing $f(x)$ without knowing x; public checksum

- Alice knows x, Bob knows $h(x)$.
- Alice sends $f(x)$ (matching $h(x)$) to Bob without revealing x.
- Bob gets convinced he received a correct value.

SCHEME:

- $L = \{(u, v) \mid \exists x \; u = f(x), \; v = h(x)\} \in \mathbf{NP}$.

Delegating the computation, simple version

TASK: delegate computing $f(x)$ without knowing x; public checksum

- Alice knows x, Bob knows $h(x)$.
- Alice sends $f(x)$ (matching $h(x)$) to Bob without revealing x.
- Bob gets convinced he received a correct value.

SCHEME:

- $L = \{(u, v) \mid \exists x \; u = f(x), \; v = h(x)\} \in \mathbf{NP}$.
- Alice knows (u, v) because she knows x and $f, h \in \mathbf{P}$.
- Bob knows (u, v) because he receives it.
- Thus we can use the CZK protocol from the previous lecture.

Exercise

Prove it formally ...

Hit the Image: Provably send the value without revealing the preimage

TASK: send a value in $\mathrm{Im}\, f$ without revealing the preimage

- Alice generates the value of $f(x)$ and sends it to Bob.
- Bob gets convinced he received a value in $\mathrm{Im}\, f$ while learning nothing about x.

Hit the Image: Provably send the value without revealing the preimage

TASK: send a value in $\operatorname{Im} f$ without revealing the preimage

- Alice generates the value of $f(x)$ and sends it to Bob.
- Bob gets convinced he received a value in $\operatorname{Im} f$ while learning nothing about x.

SCHEME:

- $L = \operatorname{Im} f$.

Hit the Image: Provably send the value without revealing the preimage

TASK: send a value in $\operatorname{Im} f$ without revealing the preimage

- Alice generates the value of $f(x)$ and sends it to Bob.
- Bob gets convinced he received a value in $\operatorname{Im} f$ while learning nothing about x.

SCHEME:

- $L = \operatorname{Im} f$.
- They both know f.
- Thus we can use the Strong CZK protocol from the previous lecture.
- Bonus: Bob is also convinced that Alice knows (at least one) x.

Exercise

Prove it formally ...

Delegating the computation, full version

TASK: delegate computing $f(x)$ without knowing x, private checksum

- Alice knows x, Bob knows y (which is maybe $h(x)$ and maybe not).

Delegating the computation, full version

TASK: delegate computing $f(x)$ without knowing x, private checksum

- Alice knows x, Bob knows y (which is maybe $h(x)$ and maybe not).
- Alice sends $f(x)$ and $h(x)$ to Bob without revealing x.

Delegating the computation, full version

TASK: delegate computing $f(x)$ without knowing x, private checksum

- Alice knows x, Bob knows y (which is maybe $h(x)$ and maybe not).
- Alice sends $f(x)$ and $h(x)$ to Bob without revealing x.
- If $h(x) = y$, Bob gets convinced that Alice has x and he received the correct value, and does not learn anything about x.

Delegating the computation, full version

TASK: delegate computing $f(x)$ without knowing x, private checksum

- Alice knows x, Bob knows y (which is maybe $h(x)$ and maybe not).
- Alice sends $f(x)$ and $h(x)$ to Bob without revealing x.
- If $h(x) = y$, Bob gets convinced that Alice has x and he received the correct value, and does not learn anything about x.
- If $h(x) \neq y$, Bob still does not learn anything about x beyond $f(x), h(x)$.

SCHEME:

- Alice sends $f(x), h(x)$ as an image point of (f, h).

Delegating the computation, full version

TASK: delegate computing $f(x)$ without knowing x, private checksum

- Alice knows x, Bob knows y (which is maybe $h(x)$ and maybe not).
- Alice sends $f(x)$ and $h(x)$ to Bob without revealing x.
- If $h(x) = y$, Bob gets convinced that Alice has x and he received the correct value, and does not learn anything about x.
- If $h(x) \neq y$, Bob still does not learn anything about x beyond $f(x), h(x)$.

SCHEME:

- Alice sends $f(x), h(x)$ as an image point of (f, h).
- $L = \{(u, v) \mid \exists x \; u = f(x), \; v = h(x)\} \in \mathbf{NP}$.
- Thus we can use the Strong CZK protocol from the previous lecture (and then Bob checks $y = h(x)$).

Delegating the computation, full version

TASK: delegate computing $f(x)$ without knowing x, private checksum

- Alice knows x, Bob knows y (which is maybe $h(x)$ and maybe not).
- Alice sends $f(x)$ and $h(x)$ to Bob without revealing x.
- If $h(x) = y$, Bob gets convinced that Alice has x and he received the correct value, and does not learn anything about x.
- If $h(x) \neq y$, Bob still does not learn anything about x beyond $f(x), h(x)$.

SCHEME:

- Alice sends $f(x), h(x)$ as an image point of (f, h).
- $L = \{(u, v) \mid \exists x \; u = f(x), \; v = h(x)\} \in \mathbf{NP}$.
- Thus we can use the Strong CZK protocol from the previous lecture (and then Bob checks $y = h(x)$).
- If Bob did not know $h(x)$, this fact does not help him to reveal anything but $h(x)$.

Exercise

Prove it formally ...

Forcing polynomial-time samplable randomness

We write $C_t(u)$ to denote a vector of commitments $C_{t^{(i)}}(u^{(i)})$.

TASK: Alice works as a sampler, Bob trusts her but doesn't learn the seed

Forcing polynomial-time samplable randomness

We write $C_t(u)$ to denote a vector of commitments $C_{t^{(i)}}(u^{(i)})$.

TASK: Alice works as a sampler, Bob trusts her but doesn't learn the seed

- $s\colon \{0,1\}^\ell \to \ldots$ is polynomial-time computable, where $\ell = \ell(n)$.
- Alice generates uniformly random $r \leftarrow U_\ell$ and sends $s(r)$ to Bob.
- Bob does not learn r and gets convinced the distribution was $s(U_\ell)$.

Forcing polynomial-time samplable randomness

We write $C_t(u)$ to denote a vector of commitments $C_{t^{(i)}}(u^{(i)})$.

TASK: Alice works as a sampler, Bob trusts her but doesn't learn the seed

- $s\colon \{0,1\}^\ell \to \dots$ is polynomial-time computable, where $\ell = \ell(n)$.
- Alice generates uniformly random $r \leftarrow U_\ell$ and sends $s(r)$ to Bob.
- Bob does not learn r and gets convinced the distribution was $s(U_\ell)$.

SCHEME:

- Alice chooses $t_1, \dots, t_\ell \in U_n$ and $r' \in U_\ell$,
  sends commitments $C_t(r')$ persuading Bob using the "image" protocol.

Forcing polynomial-time samplable randomness

We write $C_t(u)$ to denote a vector of commitments $C_{t^{(i)}}(u^{(i)})$.

TASK: Alice works as a sampler, Bob trusts her but doesn't learn the seed

- $s\colon \{0,1\}^\ell \to \ldots$ is polynomial-time computable, where $\ell = \ell(n)$.
- Alice generates uniformly random $r \leftarrow U_\ell$ and sends $s(r)$ to Bob.
- Bob does not learn r and gets convinced the distribution was $s(U_\ell)$.

SCHEME:

- Alice chooses $t_1, \ldots, t_\ell \in U_n$ and $r' \in U_\ell$,
  sends commitments $C_t(r')$ persuading Bob using the "image" protocol.
- Alice and Bob jointly generate $r'' \leftarrow U_\ell$.

Forcing polynomial-time samplable randomness

We write $C_t(u)$ to denote a vector of commitments $C_{t^{(i)}}(u^{(i)})$.

TASK: Alice works as a sampler, Bob trusts her but doesn't learn the seed

- $s\colon \{0,1\}^\ell \to \ldots$ is polynomial-time computable, where $\ell = \ell(n)$.
- Alice generates uniformly random $r \leftarrow U_\ell$ and sends $s(r)$ to Bob.
- Bob does not learn r and gets convinced the distribution was $s(U_\ell)$.

SCHEME:

- Alice chooses $t_1, \ldots, t_\ell \in U_n$ and $r' \in U_\ell$,
  sends commitments $C_t(r')$ persuading Bob using the "image" protocol.
- Alice and Bob jointly generate $r'' \leftarrow U_\ell$.
- Alice computes $r := r' \oplus r''$.
- Bob delegates the computation of $s(r)$ to Alice and gets the value:
  $f(r', t, r'') = s(r' \oplus r'') \longleftarrow$ Alice computes,
  $h(r', t, r'') = (C_t(r'), r'') \longleftarrow$ Bob already knows.

Exercise

Prove it formally ...

All-confident input commitment

We write $C_s(u)$ meaning a vector of commitments $C_{s^{(i)}}(u^{(i)})$.

TASK: Alice commits to bits of x and proves it did so even without opening the envelopes (and more!)

All-confident input commitment

We write $C_s(u)$ meaning a vector of commitments $C_{s^{(i)}}(u^{(i)})$.

TASK: Alice commits to bits of x and proves it did so even without opening the envelopes (and more!)

- Alice has x, she generates random commitment keys r.
- Alice sends the commitments to Bob.

All-confident input commitment

We write $C_s(u)$ meaning a vector of commitments $C_{s^{(i)}}(u^{(i)})$.

TASK: Alice commits to bits of x and proves it did so even without opening the envelopes (and more!)

- Alice has x, she generates random commitment keys r.
- Alice sends the commitments to Bob.
- Bob gets convinced that she did it well and r were uniformly random strings.

All-confident input commitment

We write $C_s(u)$ meaning a vector of commitments $C_{s^{(i)}}(u^{(i)})$.

TASK: Alice commits to bits of x and proves it did so even without opening the envelopes (and more!)

- Alice has x, she generates random commitment keys r.
- Alice sends the commitments to Bob.
- Bob gets convinced that she did it well and r were uniformly random strings.
- He can open some envelopes or abstain from it.
- He does not learn anything about the other bids and keys.

All-confident input commitment

We write $C_s(u)$ meaning a vector of commitments $C_{s^{(i)}}(u^{(i)})$.

TASK: Alice commits to bits of x and proves it did so even without opening the envelopes (and more!)

- Alice has x, she generates random commitment keys r.
- Alice sends the commitments to Bob.
- Bob gets convinced that she did it well and r were uniformly random strings.
- He can open some envelopes or abstain from it.
- He does not learn anything about the other bids and keys.

Key point: commitment is a function of the bid and the key.

All-confident input commitment

We write $C_s(u)$ meaning a vector of commitments $C_{s^{(i)}}(u^{(i)})$.

TASK: Alice commits to bits of x and proves it did so even without opening the envelopes (and more!)

- Alice has x, she generates random commitment keys r.
- Alice sends the commitments to Bob.
- Bob gets convinced that she did it well and r were uniformly random strings.
- He can open some envelopes or abstain from it.
- He does not learn anything about the other bids and keys.

Key point: commitment is a function of the bid and the key.

SCHEME:

- A sends "pre-commitment" $C_s(x)$ for a random s // *image*

All-confident input commitment

We write $C_s(u)$ meaning a vector of commitments $C_{s^{(i)}}(u^{(i)})$.

TASK: Alice commits to bits of x and proves it did so even without opening the envelopes (and more!)

- Alice has x, she generates random commitment keys r.
- Alice sends the commitments to Bob.
- Bob gets convinced that she did it well and r were uniformly random strings.
- He can open some envelopes or abstain from it.
- He does not learn anything about the other bids and keys.

Key point: commitment is a function of the bid and the key.

SCHEME:

- A sends "pre-commitment" $C_s(x)$ for a random s // *image*
- A and B generate Alice-known r, t and B gets $C_t(r)$ // *polynomial-time sampling*

All-confident input commitment

We write $C_s(u)$ meaning a vector of commitments $C_{s^{(i)}}(u^{(i)})$.

TASK: Alice commits to bits of x and proves it did so even without opening the envelopes (and more!)

- Alice has x, she generates random commitment keys r.
- Alice sends the commitments to Bob.
- Bob gets convinced that she did it well and r were uniformly random strings.
- He can open some envelopes or abstain from it.
- He does not learn anything about the other bids and keys.

Key point: commitment is a function of the bid and the key.

SCHEME:

- A sends "pre-commitment" $C_s(x)$ for a random s *// image*
- A and B generate Alice-known r, t and B gets $C_t(r)$ *// polynomial-time sampling*
- A sends $C_r(x)$ while Bob holds A's commitments for [some] r and x and can check they have been used in the computation *// delegated computing $C_r(x)$ matching all the three*
  - Bob learns nothing about x or r
  - final commitments to parts of x can be verified upon request
  - their secret keys are random (guaranteed by the sampling protocol: they are the seeds)

Exercise...

Forcing malicious adversaries to behave semi-honestly

Input commitment. Alice and Bob do commit to their (secret) inputs using "provably random" keys.

Forcing malicious adversaries to behave semi-honestly

Input commitment. Alice and Bob do commit to their (secret) inputs using "provably random" keys.

We can now control they perform some action based on specific data.

Forcing malicious adversaries to behave semi-honestly

Input commitment. Alice and Bob do commit to their (secret) inputs using "provably random" keys.

We can now control they perform some action based on specific data.

Randomness commitment. Alice and Bob do commit to their (secret) randomness using "provably random" keys.

Forcing malicious adversaries to behave semi-honestly

Input commitment. Alice and Bob do commit to their (secret) inputs using "provably random" keys.
We can now control they perform some action based on specific data.

Randomness commitment. Alice and Bob do commit to their (secret) randomness using "provably random" keys.
Randomness is just additional data. We are now confident it's uniformly random.

Forcing malicious adversaries to behave semi-honestly

Input commitment. Alice and Bob do commit to their (secret) inputs using "provably random" keys.
We can now control they perform some action based on specific data.

Randomness commitment. Alice and Bob do commit to their (secret) randomness using "provably random" keys.
Randomness is just additional data. We are now confident it's uniformly random.

Computation. (Viewed from Bob.)
Alice's input a, randomness r_a, commitment keys s_a, s'_a, messages m_a.
Bob's input b, randomness r_b, commitment keys s_b, s'_b, messages m_b.

Forcing malicious adversaries to behave semi-honestly

Input commitment. Alice and Bob do commit to their (secret) inputs using "provably random" keys.
We can now control they perform some action based on specific data.

Randomness commitment. Alice and Bob do commit to their (secret) randomness using "provably random" keys.
Randomness is just additional data. We are now confident it's uniformly random.

Computation. (Viewed from Bob.)
Alice's input a, randomness r_a, commitment keys s_a, s'_a, messages m_a.
Bob's input b, randomness r_b, commitment keys s_b, s'_b, messages m_b.
Note that they will never open the envelopes!

Forcing malicious adversaries to behave semi-honestly

Input commitment. Alice and Bob do commit to their (secret) inputs using "provably random" keys.
We can now control they perform some action based on specific data.

Randomness commitment. Alice and Bob do commit to their (secret) randomness using "provably random" keys.
Randomness is just additional data. We are now confident it's uniformly random.

Computation. (Viewed from Bob.)
Alice's input a, randomness r_a, commitment keys s_a, s'_a, messages m_a.
Bob's input b, randomness r_b, commitment keys s_b, s'_b, messages m_b.
Note that they will never open the envelopes!

- Bob delegates Alice's computations f to Alice (f generates her messages).
- Arguments of her computations: a, r_a, s_a, s'_a, m_b.
("Alice's view" of the protocol.)

Forcing malicious adversaries to behave semi-honestly

Input commitment. Alice and Bob do commit to their (secret) inputs using "provably random" keys.
We can now control they perform some action based on specific data.

Randomness commitment. Alice and Bob do commit to their (secret) randomness using "provably random" keys.
Randomness is just additional data. We are now confident it's uniformly random.

Computation. (Viewed from Bob.)
Alice's input a, randomness r_a, commitment keys s_a, s'_a, messages m_a.
Bob's input b, randomness r_b, commitment keys s_b, s'_b, messages m_b.
Note that they will never open the envelopes!

- Bob delegates Alice's computations f to Alice (f generates her messages).
- Arguments of her computations: a, r_a, s_a, s'_a, m_b.
  ("Alice's view" of the protocol.)
- Bob checks these computations against h,
  which outputs A's public commitments for the input and randomness:
  $h(a, r_a, s_a, s'_a, m_b) = (C_{s_a}(a), C_{s'_a}(r_a), m_b)$.
- Bob holds the target value of h to check the computations.

Forcing malicious adversaries to behave semi-honestly

Input commitment. Alice and Bob do commit to their (secret) inputs using "provably random" keys.
We can now control they perform some action based on specific data.

Randomness commitment. Alice and Bob do commit to their (secret) randomness using "provably random" keys.
Randomness is just additional data. We are now confident it's uniformly random.

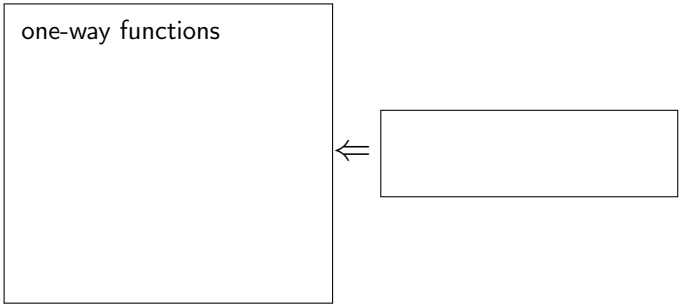
Computation. (Viewed from Bob.)
Alice's input a, randomness r_a, commitment keys s_a, s'_a, messages m_a.
Bob's input b, randomness r_b, commitment keys s_b, s'_b, messages m_b.
Note that they will never open the envelopes!

- Bob delegates Alice's computations f to Alice (f generates her messages).
- Arguments of her computations: a, r_a, s_a, s'_a, m_b.
  ("Alice's view" of the protocol.)
- Bob checks these computations against h,
  which outputs A's public commitments for the input and randomness:
  $h(a, r_a, s_a, s'_a, m_b) = (C_{s_a}(a), C_{s'_a}(r_a), m_b)$.
- Bob holds the target value of h to check the computations.

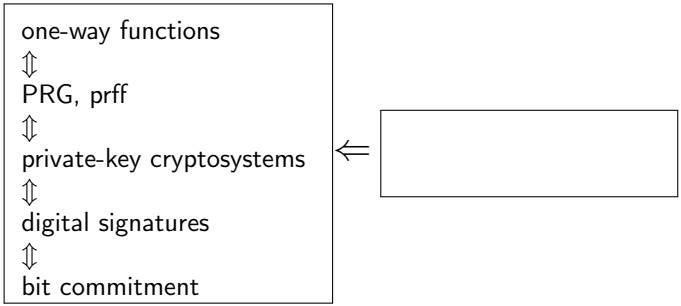
Exercise

Prove it formally ☺

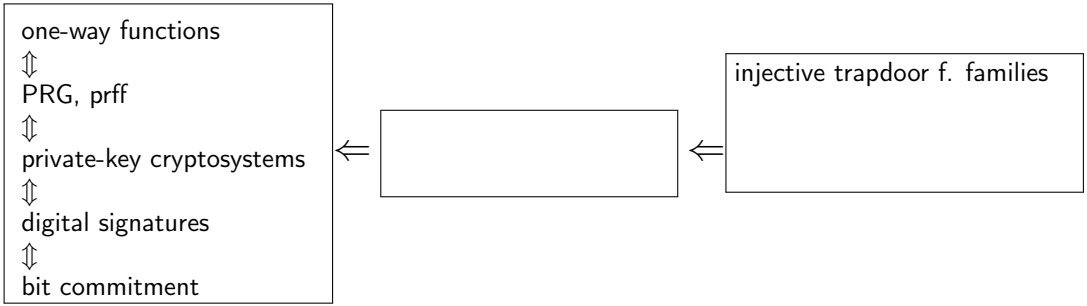
Clarifications regarding cryptographic primitives



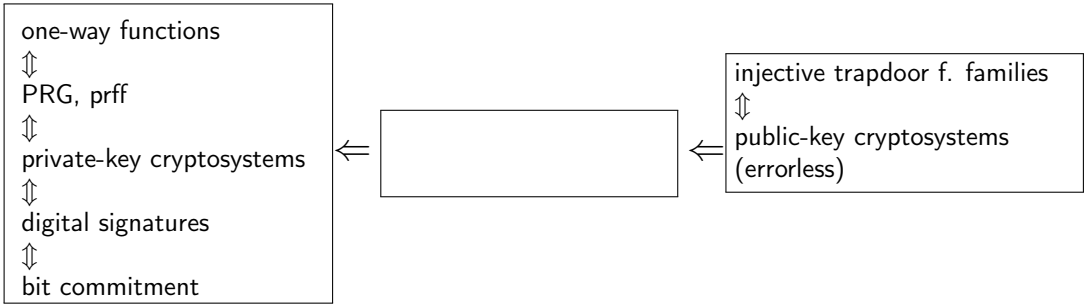
Clarifications regarding cryptographic primitives



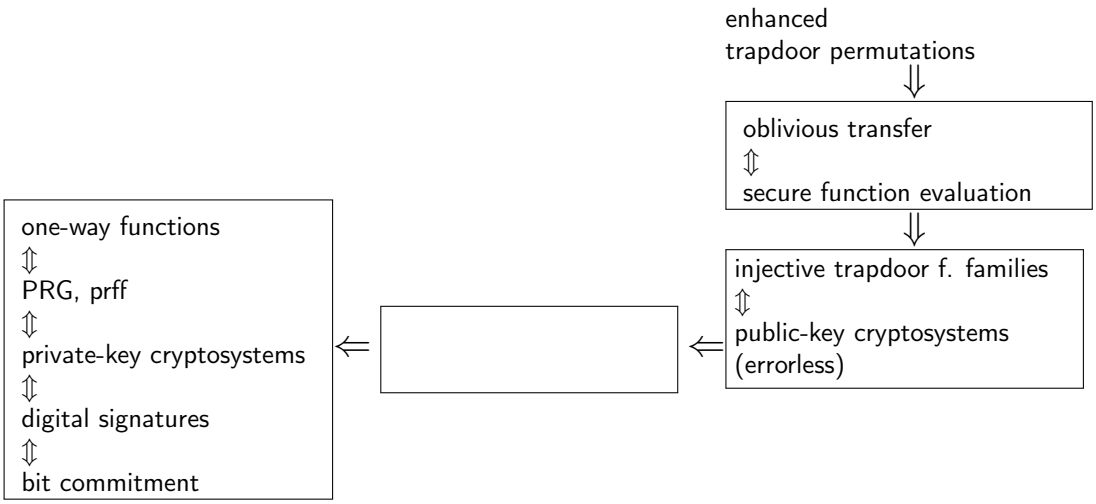
Clarifications regarding cryptographic primitives



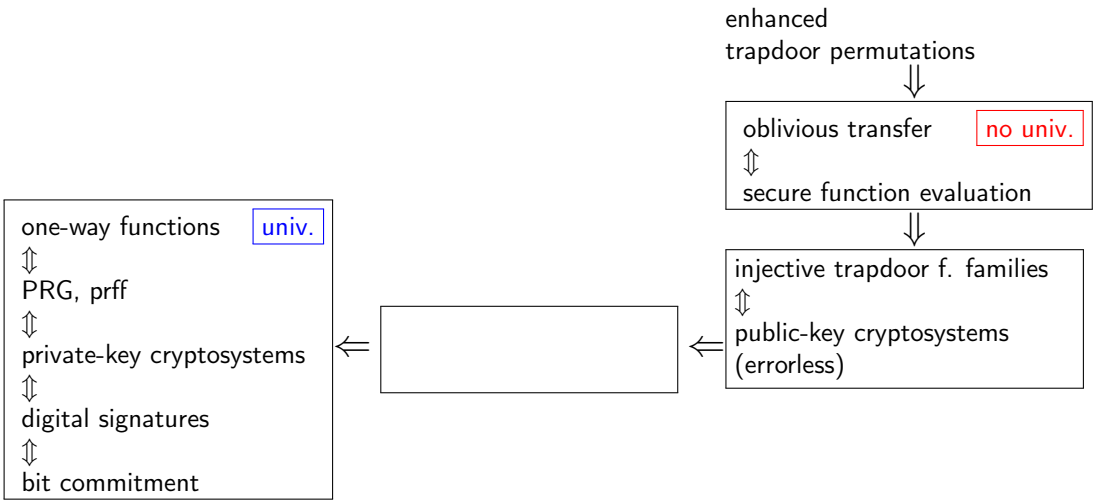
Clarifications regarding cryptographic primitives



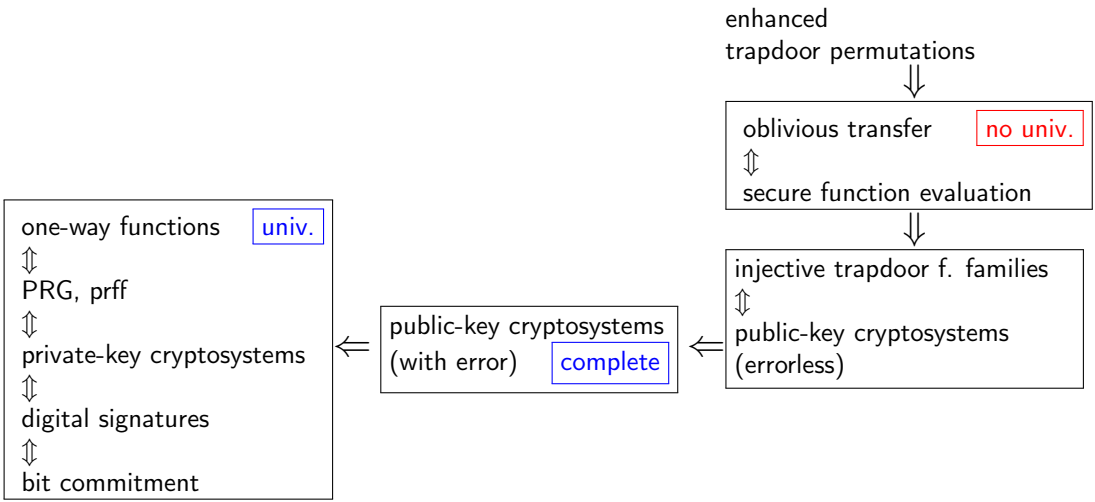
Clarifications regarding cryptographic primitives



Clarifications regarding cryptographic primitives



Clarifications regarding cryptographic primitives



Takeout

- Secure two-party computation with malicious adversaries.
- In particular, how to delegate computations with partially unknown data.
- Cryptographic primitives: what implies what.

Takeout

- Secure two-party computation with malicious adversaries.
- In particular, how to delegate computations with partially unknown data.
- Cryptographic primitives: what implies what.

Coming next:

- *(maybe) Exercises and Repeat.*
- *Blockchains.*
- *(maybe) The real grounds of the security in modern cryptography.*
- *(maybe) A complete (universal) public-key cryptosystem.*