ARIEL UNIVERSITY

אוניברסיטת אריאל בשומרון

Foundations of Modern Cryptography

Edward A. Hirsch

https://edwardahirsch.github.io/edwardahirsch

Neapolis University Pafos

Lecture 7: November 27, 2024

Lecture Plan

- Bit commitment (commit to a stake without disclosing it).

Lecture Plan

- Bit commitment (commit to a stake without disclosing it).
- Oblivious transfer (open one black box out of two without disclosing which one).

Lecture Plan

- Bit commitment (commit to a stake without disclosing it).
- Oblivious transfer (open one black box out of two without disclosing which one).
- Secure distributed computation (parties learn nothing about the others' inputs).

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

Task: Bit (or Bid) Commitment

Alice

bid, key

Public

Task: Bit (or Bid) Commitment

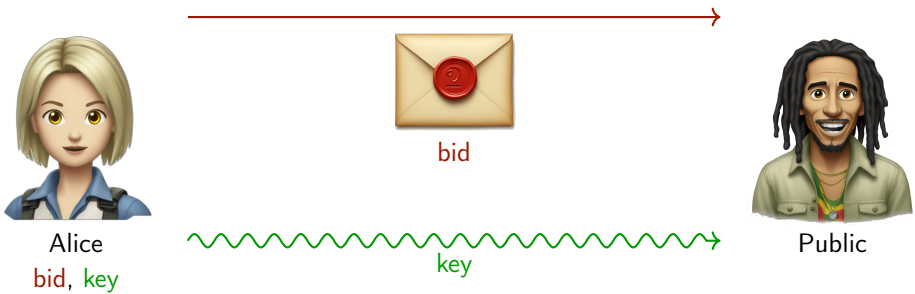
Alice

bid, key

bid

Public

Task: Bit (or Bid) Commitment



Alice's bid will be obvious to everybody after (but not before) opening the "envelope".

She should be unable to change her bid retroactively.

Pictures courtesy AI Emojis Generator

(Non-interactive) bit commitment: Definition

Definition (Bit commitment: non-interactive version)

- Polynomial-time $A(1^n, \beta) = (\texttt{code}, \texttt{key})$, where $\beta \in \{0, 1\}$.
- Polynomial-time B:
  - $B(\beta, \texttt{code}, \texttt{key}) = 1$ if $(\texttt{code}, \texttt{key}) \leftarrow A(1^n, \beta)$.
  - For code generated this way from β and any adversary $\tilde{B}$,

$$\mathbf{Pr}\{\tilde{B}(\beta, \texttt{code}, -) = 1\} < \frac{1}{2} + \varepsilon(n).$$

  - For any x and k not generated this way (or generated from $\beta' \neq \beta$),

$$\mathbf{Pr}\{B(\beta, x, k) = 1\} < \varepsilon(n)$$

    In particular, A cannot change its bit after she sent code by selecting a different key.

Non-interactive bit commitment scheme based on one-way permutation

Consider owp f, hardcore predicate B.

$A(1^n, \beta)$:
— Pick random $r \leftarrow U_n$
— Send $\texttt{code} = (f(r), B(r) \oplus \beta)$
— $(\dots)$
— Send $\texttt{key} = r$.

Non-interactive bit commitment scheme based on one-way permutation

Consider owp f, hardcore predicate B.

$A(1^n, \beta)$:
— Pick random $r \leftarrow U_n$
— Send $\texttt{code} = (f(r), B(r) \oplus \beta)$
— $(\dots)$
— Send $\texttt{key} = r$.

$B(\beta, \texttt{code}, \texttt{key})$:
— Let $\texttt{code} = (y, b)$.
— Check $f(\texttt{key}) = y$ and $b = B(\texttt{key}) \oplus \beta$.

Non-interactive bit commitment scheme based on one-way permutation

Consider owp f, hardcore predicate B.

$A(1^n, \beta)$:
— Pick random $r \leftarrow U_n$
— Send $\texttt{code} = (f(r), B(r) \oplus \beta)$
— $(\dots)$
— Send $\texttt{key} = r$.

$B(\beta, \texttt{code}, \texttt{key})$:
— Let $\texttt{code} = (y, b)$.
— Check $f(\texttt{key}) = y$ and $b = B(\texttt{key}) \oplus \beta$.

Security:

- If someone can reveal β, they reveal $B(r)$ from $f(r)$.
- There is no other key $k \neq \texttt{key}$ s.t. $f(k) = y$, it simply does not exist (f is injective)!

Non-interactive bit commitment scheme based on one-way permutation

Consider owp f, hardcore predicate B.

$A(1^n, \beta)$:
— Pick random $r \leftarrow U_n$
— Send $\texttt{code} = (f(r), B(r) \oplus \beta)$
— $(\dots)$
— Send $\texttt{key} = r$.

$B(\beta, \texttt{code}, \texttt{key})$:
— Let $\texttt{code} = (y, b)$.
— Check $f(\texttt{key}) = y$ and $b = B(\texttt{key}) \oplus \beta$.

Security:

- If someone can reveal β, they reveal $B(r)$ from $f(r)$.
- There is no other key $k \neq \texttt{key}$ s.t. $f(k) = y$, it simply does not exist (f is injective)!

Remark

There are more efficient multi-round protocols.
In particular, we can just use a PRG instead of a one-way permutation.

Task: Oblivious Transfer

(1,2)-OT

Alice

bids β_0, β_1, keys

Public

index i

Task: Oblivious Transfer

(1,2)-OT

Alice

bids β_0, β_1, keys

β_0 inside

β_1 inside

Public

index i

Task: Oblivious Transfer

(1,2)-OT

Alice

bids β_0, β_1, keys

β_0 inside

β_1 inside

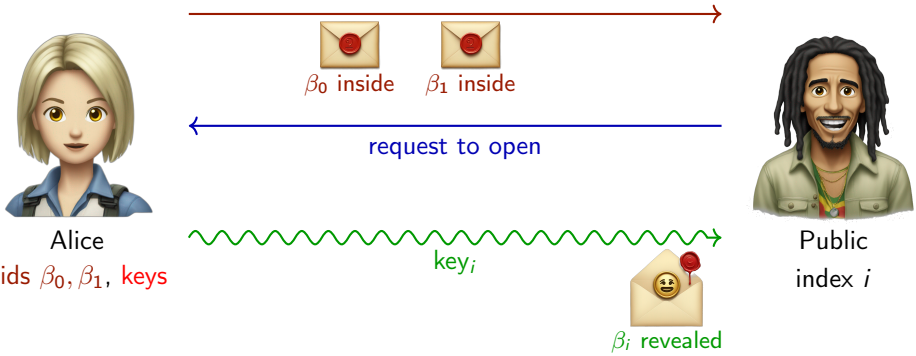
request to open

Public

index i

Task: Oblivious Transfer

(1,2)-OT



Bob opens the "selected" envelope i, and only it.

Alice has no idea which envelope was opened.

Pictures courtesy AI Emojis Generator

(1,2)–Oblivious Transfer: Definition

Definition ((1,2)–Oblivious Transfer, (1,2)–OT)

(1,2)–OT scheme is a protocol with polynomial-time participants $A(\beta_0, \beta_1)$ and $B(i)$, where secret bits $\beta_0, \beta_1 \in \{0,1\}$, index $i \in \{0,1\}$. After the protocol conclusion:

- B knows β_i.
- for any adversary B' (playing instead of B) the probability to guess β_{1-i} is $< \frac{1}{2} + \epsilon(n)$;
- for any adversary A' (playing instead of A) the probability to guess i is $< \frac{1}{2} + \epsilon(n)$.

(1,2)–OT using extended tdpf

Extended tdpf (e, s, s', d) (with hardcore predicate B) contains an additional sampler s':

- $d(s'(r'))$ is hard to compute without d even if we know r',
- $s'(r')$ is distributed similarly to $e(s(r))$ even if we know d,

(1,2)–OT using extended tdpf

Extended tdpf (e, s, s', d) (with hardcore predicate B) contains an additional sampler s':

- $d(s'(r'))$ is hard to compute without d even if we know r',
- $s'(r')$ is distributed similarly to $e(s(r))$ even if we know d,

Consider semi-honest participants (follow the protocol, but can compute something in addition).

(1,2)–OT using extended tdpf

Extended tdpf (e, s, s', d) (with hardcore predicate B) contains an additional sampler s':

- $d(s'(r'))$ is hard to compute without d even if we know r',
- $s'(r')$ is distributed similarly to $e(s(r))$ even if we know d,

Consider semi-honest participants (follow the protocol, but can compute something in addition).

Scheme:

1. Alice generates (e, s, s', d).
2. Alice sends (e, s, s').

(1,2)–OT using extended tdpf

Extended tdpf (e, s, s', d) (with hardcore predicate B) contains an additional sampler s':

- $d(s'(r'))$ is hard to compute without d even if we know r',
- $s'(r')$ is distributed similarly to $e(s(r))$ even if we know d,

Consider semi-honest participants (follow the protocol, but can compute something in addition).

Scheme:

1. Alice generates (e, s, s', d).
2. Alice sends (e, s, s').
3. Bob sends $a_i = e(s(r))$ and $a_{1-i} = s'(r')$.

(1,2)–OT using extended tdpf

Extended tdpf (e, s, s', d) (with hardcore predicate B) contains an additional sampler s':

- $d(s'(r'))$ is hard to compute without d even if we know r',
- $s'(r')$ is distributed similarly to $e(s(r))$ even if we know d,

Consider semi-honest participants (follow the protocol, but can compute something in addition).

Scheme:

1. Alice generates (e, s, s', d).
2. Alice sends (e, s, s').
3. Bob sends $a_i = e(s(r))$ and $a_{1-i} = s'(r')$.
4. Alice sends $c_0 = \beta_0 \oplus B(d(a_0))$ and $c_1 = \beta_1 \oplus B(d(a_1))$.
5. Bob computes $\beta_i = B(s(r)) \oplus c_i$.

Task: Secure Distributed Function Evaluation

Alice

a

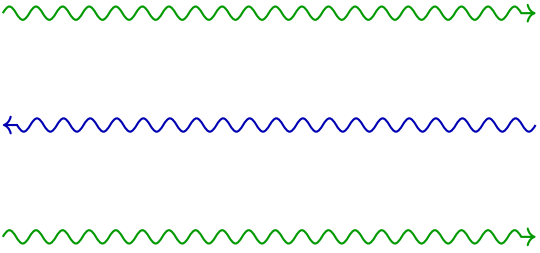
Bob

b

Task: Secure Distributed Function Evaluation

Alice

a



Bob

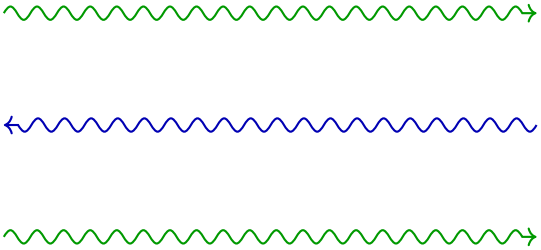
b

Task: Secure Distributed Function Evaluation

Alice

a

$f(a, b)$



Bob

b

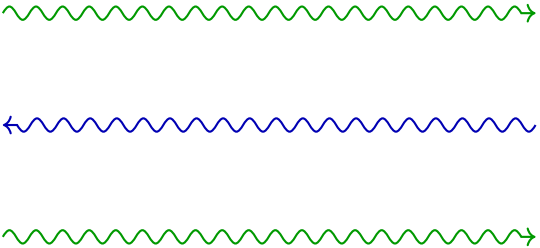
$f(a, b)$

Task: Secure Distributed Function Evaluation

Alice

a

$f(a, b)$



Bob

b

$f(a, b)$

Alice (Bob) does not learn from this protocol about b (a, resp.) more than $f(a, b)$.

Secure Distributed Function Evaluation: Definition

Consider a polynomial-time computable function f with half arguments given to A and half to B:

$$f(a_1, \ldots, a_m, b_1, \ldots, b_m)$$

Alice and Bob need to compute the result without revealing a_i's and b_i's to each other.

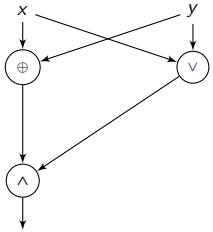
For any adversary A', any function g, and oblivious adversary $\tilde{A}$

$$\mathbf{Pr}\{A'(\vec{a}, \text{protocol}) \text{ computes also } g(\vec{a}, \vec{b})\} \leqslant \mathbf{Pr}\{\tilde{A}(\vec{a}, f(\vec{a}, \vec{b})) \text{ computes } g(\vec{a}, \vec{b})\} + \epsilon(n),$$

and the same for Bob.

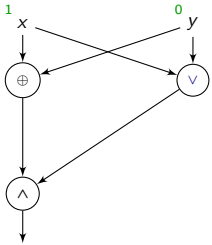
Yao's scheme for Secure Distributed Function Evaluation

- f is computed by a Boolean circuit



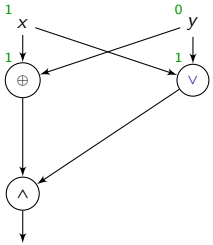
Yao's scheme for Secure Distributed Function Evaluation

- f is computed by a Boolean circuit



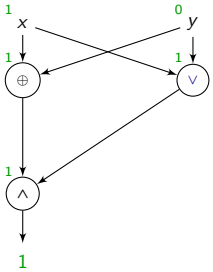
Yao's scheme for Secure Distributed Function Evaluation

- f is computed by a Boolean circuit



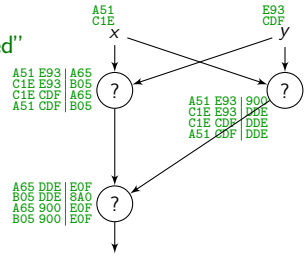
Yao's scheme for Secure Distributed Function Evaluation

- f is computed by a Boolean circuit



Yao's scheme for Secure Distributed Function Evaluation

- f is computed by a Boolean circuit
- Alice encrypts each gate of the circuit: input and output bits are "encoded"

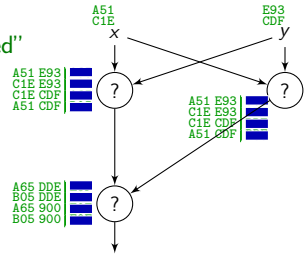


Yao's scheme for Secure Distributed Function Evaluation

- f is computed by a Boolean circuit
- Alice encrypts each gate of the circuit: input and output bits are "encoded"
- Pick symmetric keys $u_0, u_1, v_0, v_1, w_0, w_1$, encrypt the outputs, send:

0	0	1	$\longmapsto$	u_0	v_0	$E_{u_0}(E_{v_0}(w_1))$
0	1	0		u_0	v_1	$E_{u_0}(E_{v_1}(w_0))$
1	0	1		u_1	v_0	$E_{u_1}(E_{v_0}(w_1))$
1	1	0		u_1	v_1	$E_{u_1}(E_{v_1}(w_0))$

} reshuffle!

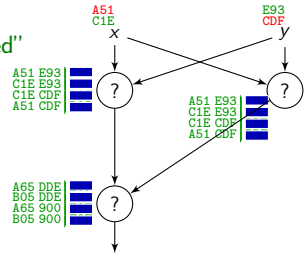


Yao's scheme for Secure Distributed Function Evaluation

- f is computed by a Boolean circuit
- Alice encrypts each gate of the circuit: input and output bits are "encoded"
- Pick symmetric keys $u_0, u_1, v_0, v_1, w_0, w_1$, encrypt the outputs, send:

0	0	1	u_0	v_0	$E_{u_0}(E_{v_0}(w_1))$
0	1	0	u_0	v_1	$E_{u_0}(E_{v_1}(w_0))$
1	0	1	u_1	v_0	$E_{u_1}(E_{v_0}(w_1))$
1	1	0	u_1	v_1	$E_{u_1}(E_{v_1}(w_0))$

$\longmapsto$ reshuffle!



- Send specific codewords for the inputs $a_1, \ldots, a_m$

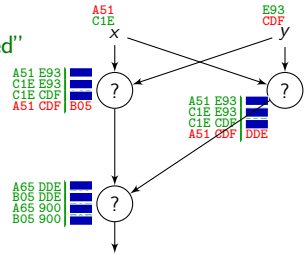
- Now Bob can simulate the circuit (in private!) and get the encrypted output

Yao's scheme for Secure Distributed Function Evaluation

- f is computed by a Boolean circuit
- Alice encrypts each gate of the circuit: input and output bits are "encoded"
- Pick symmetric keys $u_0, u_1, v_0, v_1, w_0, w_1$, encrypt the outputs, send:

0	0	1	u_0	v_0	$E_{u_0}(E_{v_0}(w_1))$
0	1	0	u_0	v_1	$E_{u_0}(E_{v_1}(w_0))$
1	0	1	u_1	v_0	$E_{u_1}(E_{v_0}(w_1))$
1	1	0	u_1	v_1	$E_{u_1}(E_{v_1}(w_0))$

$\longmapsto$ reshuffle!



- Send specific codewords for the inputs $a_1, \ldots, a_m$

- Now Bob can simulate the circuit (in private!) and get the encrypted output

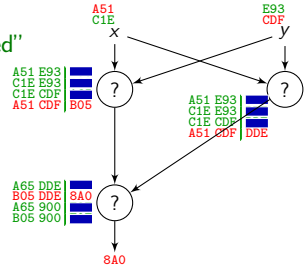
Yao's scheme for Secure Distributed Function Evaluation

- f is computed by a Boolean circuit
- Alice encrypts each gate of the circuit: input and output bits are "encoded"
- Pick symmetric keys $u_0, u_1, v_0, v_1, w_0, w_1$, encrypt the outputs, send:

0	0	1	$\longmapsto$	u_0	v_0	$E_{u_0}(E_{v_0}(w_1))$
0	1	0		u_0	v_1	$E_{u_0}(E_{v_1}(w_0))$
1	0	1		u_1	v_0	$E_{u_1}(E_{v_0}(w_1))$
1	1	0		u_1	v_1	$E_{u_1}(E_{v_1}(w_0))$

reshuffle!

- Send specific codewords for the inputs $a_1, \ldots, a_m$



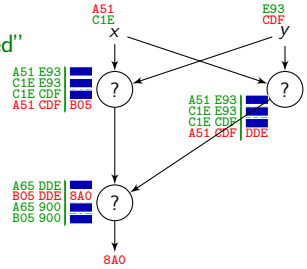
- Now Bob can simulate the circuit (in private!) and get the encrypted output

Yao's scheme for Secure Distributed Function Evaluation

- f is computed by a Boolean circuit
- Alice encrypts each gate of the circuit: input and output bits are "encoded"
- Pick symmetric keys $u_0, u_1, v_0, v_1, w_0, w_1$, encrypt the outputs, send:

0	0	1	u_0	v_0	$E_{u_0}(E_{v_0}(w_1))$
0	1	0	u_0	v_1	$E_{u_0}(E_{v_1}(w_0))$
1	0	1	u_1	v_0	$E_{u_1}(E_{v_0}(w_1))$
1	1	0	u_1	v_1	$E_{u_1}(E_{v_1}(w_0))$

$\longmapsto$ reshuffle!



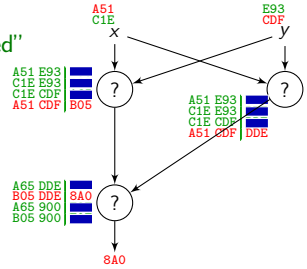
- Send specific codewords for the inputs $a_1, \ldots, a_m$
- Bob has
  - encrypted circuit
  - codewords for Alice's inputs
- Bob gets codewords for his inputs from Alice using OT
- Now Bob can simulate the circuit (in private!) and get the encrypted output

Yao's scheme for Secure Distributed Function Evaluation

- f is computed by a Boolean circuit
- Alice encrypts each gate of the circuit: input and output bits are "encoded"
- Pick symmetric keys $u_0, u_1, v_0, v_1, w_0, w_1$, encrypt the outputs, send:

0	0	1	u_0	v_0	$E_{u_0}(E_{v_0}(w_1))$
0	1	0	u_0	v_1	$E_{u_0}(E_{v_1}(w_0))$
1	0	1	u_1	v_0	$E_{u_1}(E_{v_0}(w_1))$
1	1	0	u_1	v_1	$E_{u_1}(E_{v_1}(w_0))$

$\longmapsto$ reshuffle!



- Send specific codewords for the inputs $a_1, \ldots, a_m$
- Bob has
  - encrypted circuit
  - codewords for Alice's inputs
- Bob gets codewords for his inputs from Alice using OT
- Now Bob can simulate the circuit (in private!) and get the encrypted output
- Alice decrypts the output (e.g., bit commitment or (1,2)-OT again)

Takeout

- Non-interactive bit commitment from owp.
- 1-out-of-2 oblivious transfer from extended tdpf.
- Secure distributed function evaluation.

Takeout

- Non-interactive bit commitment from owp.
- 1-out-of-2 oblivious transfer from extended tdpf.
- Secure distributed function evaluation.

Coming next:

- *Exercises and Repeat.*
- *Zero-knowledge protocols*
  (forces parties to be semi-honest: ensure the bank computes your partner's balance correctly!).
- *Blockchains.*