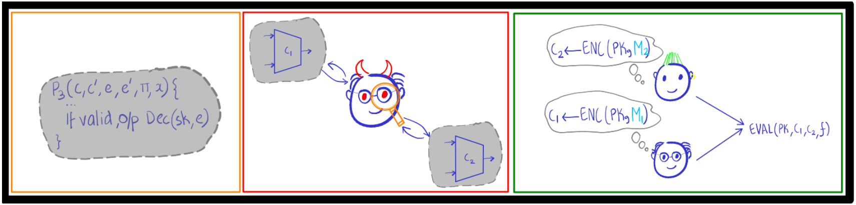


CS783: Theoretical Foundations of Cryptography

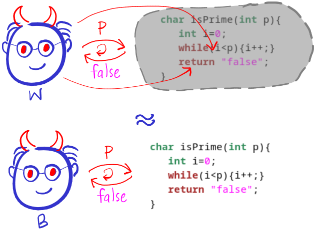
Lecture 24 (08/Nov/24)

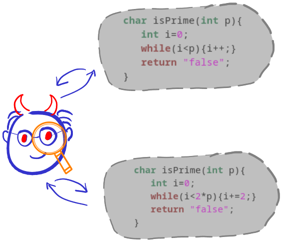
Instructor: Chethan Kamath

Recall from Last Two Lectures...

- Program obfuscation: "scramble/encrypt" a program such that
  1. functionality preserved
  2. hard to "reverse engineer"

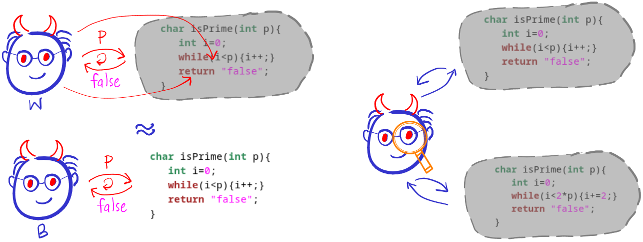
- Program obfuscation: "scramble/encrypt" a program such that
  1. functionality preserved
  2. hard to "reverse engineer"
- How to formalise "hard to reverse engineer"?
  - Lecture 22: Virtual black-box obfuscation (VBBO)
  - Lecture 23: Indistinguishability obfuscation (IO)





Recall from Last Two Lectures...

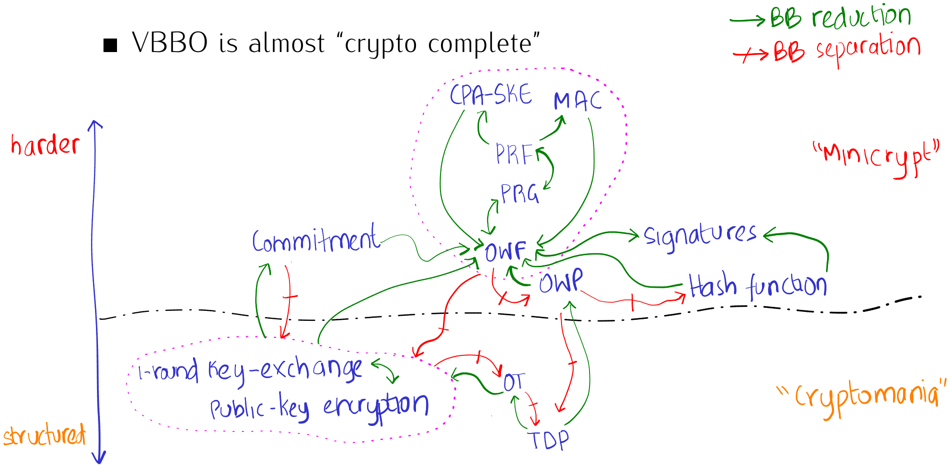
- Program obfuscation: "scramble/encrypt" a program such that
  1. functionality preserved
  2. hard to "reverse engineer"
- How to formalise "hard to reverse engineer"?
  - Lecture 22: Virtual black-box obfuscation (VBBO)
  - Lecture 23: Indistinguishability obfuscation (IO)



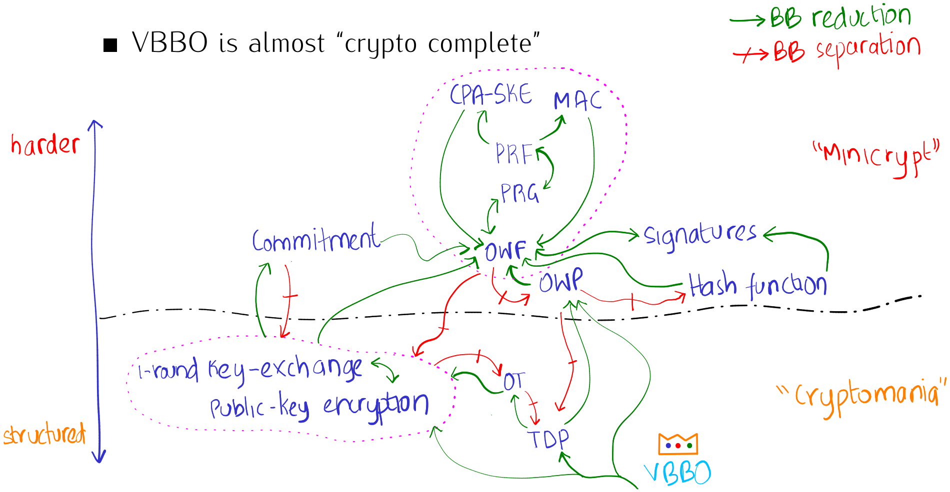
+ Bypassed black-box separations exploiting primitive's program
  - OWF $\xrightarrow{\text{VBBO}}$ OWP and PRG $\xrightarrow{\text{IO}}$ PKE

Recall from Last Two Lectures...

- VBBO is almost "crypto complete"

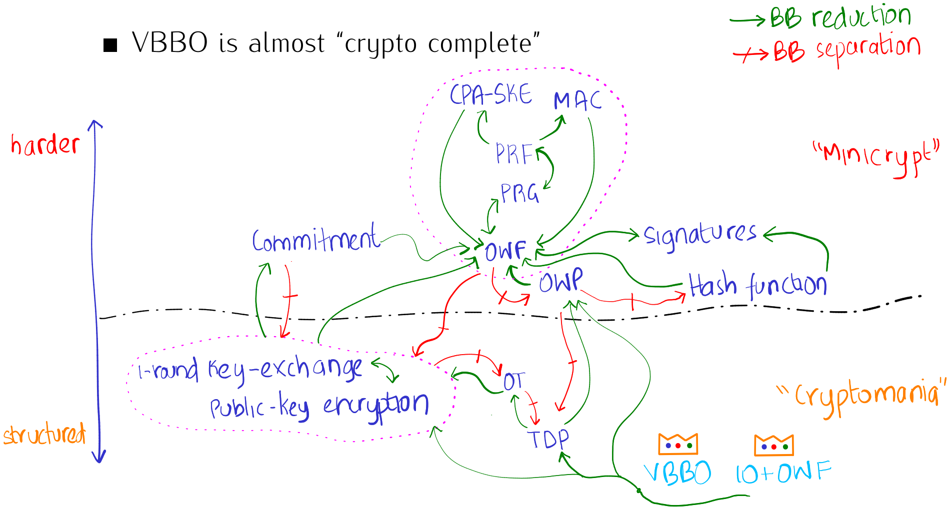


- VBBO is almost "crypto complete"



Recall from Last Two Lectures...

- VBBO is almost "crypto complete"



- IO + OWF also yields most of crypto!

Plan for Today's Lecture...

- VBBO for general programs is impossible

$$P^*_{\alpha,\beta,\gamma}(b,x) := \begin{cases} \Delta_{\alpha,\beta}(x) & \text{If } b=0 \\ S_{\alpha,\beta,\gamma}(x) & \text{If } b=1 \end{cases}$$

Plan for Today's Lecture...

- VBBO for general programs is impossible

$$P^*_{\alpha,\beta,\sigma}(b,x) := \begin{cases} \Delta_{\alpha,\beta}(x) & \text{if } b=0 \\ S_{\alpha,\beta,\sigma}(x) & \text{if } b=1 \end{cases}$$

- What about IO for general programs?
  - Boosting theorem for IO: fully homomorphic encryption (FHE) + IO for "shallow" circuits $\rightarrow$ IO for all circuits

Plan for Today's Lecture...

- VBBO for general programs is impossible

$$P^*_{\alpha,\beta,\gamma}(b,x) := \begin{cases} \Delta_{\alpha,\beta}(x) & \text{if } b=0 \\ S_{\alpha,\beta,\gamma}(x) & \text{if } b=1 \end{cases}$$

- What about IO for general programs?
  - Boosting theorem for IO: fully homomorphic encryption (FHE) + IO for "shallow" circuits $\rightarrow$ IO for all circuits
  - State of affairs for IO for "shallow" circuits

Plan for Today's Lecture...

1 Boosting IO Using FHE

2 Constructing IO for NC^1: What Do We Know?

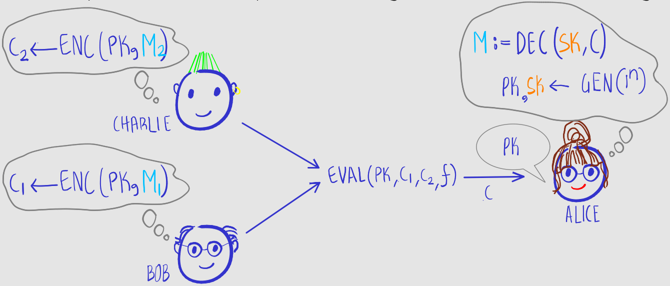
Recall... (Fully) Homomorphic Encryption (Lecture 19)

- Public-key encryption + public *evaluation* algorithm

- Public-key encryption + public *evaluation* algorithm

Defintion 1 (Homomorphic encryption (HE) for function class $\mathcal{F}$)

A PKE $\Sigma = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}) + \mathsf{Eval}$ *algorithm with following syntax*

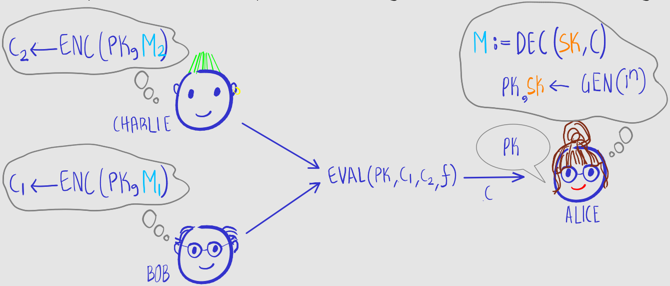


Recall... (Fully) Homomorphic Encryption (Lecture 19)

- Public-key encryption + public *evaluation* algorithm

Defintion 1 (Homomorphic encryption (HE) for function class $\mathcal{F}$)

A PKE $\Sigma = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}) + \mathsf{Eval}$ *algorithm with following syntax*



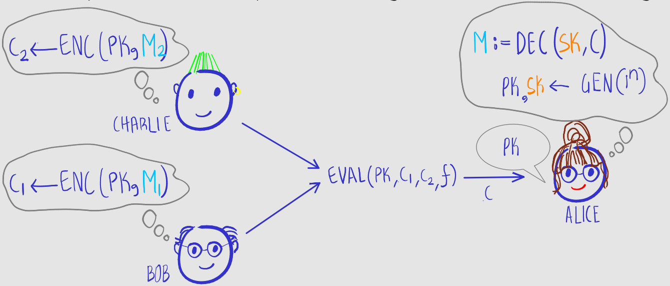
- *Compactness of evaluation:* $|c|$ *obtained from* Eval *independent of* $|f|$
- *Correctness of evaluation*

Recall... (Fully) Homomorphic Encryption (Lecture 19)

- Public-key encryption + public *evaluation* algorithm

Defintion 1 (Homomorphic encryption (HE) for function class $\mathcal{F}$)

A PKE $\Sigma = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}) + \mathsf{Eval}$ *algorithm with following syntax*



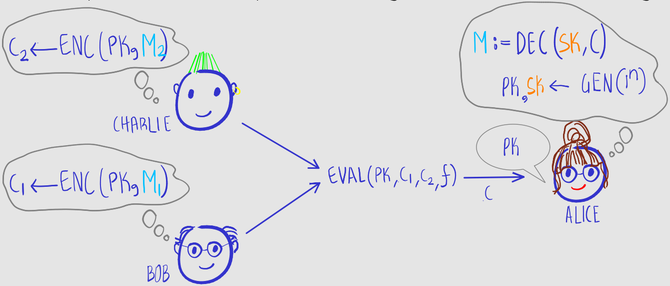
- *Compactness of evaluation:* $|c|$ *obtained from* Eval *independent of* $|f|$
- *Correctness of evaluation*

- *Fully* HE: $\mathcal{F}$=functions computable by poly.-sized circuits

- Public-key encryption + public *evaluation* algorithm

Defintion 1 (Homomorphic encryption (HE) for function class $\mathcal{F}$)

A PKE $\Sigma = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}) + \mathsf{Eval}$ *algorithm with following syntax*



- *Compactness of evaluation:* $|\mathsf{c}|$ *obtained from* Eval *independent of* $|f|$
- *Correctness of evaluation*

- *Fully* HE: $\mathcal{F}$=functions computable by poly.-sized circuits
- GSW construction: FHE that is secure assuming LWE

Recall… IO for Circuits (Lecture 23)

- Obfuscations of two *functionally-equivalent, same-sized* circuits are computationally indistinguishable

Recall... IO for Circuits (Lecture 23)

- Obfuscations of two *functionally-equivalent, same-sized* circuits are computationally indistinguishable

Defintion 2 (Indistinguishability obfuscator (IO) for circuit class $\mathcal{C}$)

A PPT algorithm Obf *that takes as input any circuit* $\mathsf{C} \in \mathcal{C}$ *and security parameter* n, *and outputs obfuscated circuit* C *such that:*

1. *Functionality preserved*
2. *Slowdown is polynomial*

Recall... IO for Circuits (Lecture 23)

- Obfuscations of two *functionally-equivalent, same-sized* circuits are computationally indistinguishable

Defintion 2 (Indistinguishability obfuscator (IO) for circuit class $\mathcal{C}$)

A PPT algorithm Obf *that takes as input any circuit* $\mathsf{C} \in \mathcal{C}$ *and security parameter* n, *and outputs obfuscated circuit* C *such that:*

1. *Functionality preserved*
2. *Slowdown is polynomial*
3. *IO security: for every functionally-equivalent, same-sized* $\mathsf{C}_1, \mathsf{C}_2 \in \mathcal{C}$ *and PPT* D, *the following is negligible:*

$$\left| \Pr_{\tilde{C}_1 \leftarrow \mathsf{Obf}(1^n, C_1)}\left[D(\tilde{C}_1) = 1\right] - \Pr_{\tilde{C}_2 \leftarrow \mathsf{Obf}(1^n, C_2)}\left[D(\tilde{C}_2) = 1\right] \right|$$

Boosting IO for NC^1 using FHE...

- Goal: Obf for NC^1 + FHE $\Pi \rightarrow \mathsf{Obf}'$ for all circuits
- High-level idea: use FHE to encrypt *circuit* and then use Obf to "decrypt-then-evaluate"

Boosting IO for NC^1 using FHE...

- Goal: Obf for NC^1 + FHE $\Pi \rightarrow \mathsf{Obf}'$ for all circuits
- High-level idea: use FHE to encrypt *circuit* and then use Obf to "decrypt-then-evaluate"
  - Use Obf to hide FHE's secret key

Boosting IO for NC^1 using FHE...

- Goal: Obf for NC^1 + FHE $\Pi \rightarrow \mathsf{Obf}'$ for all circuits
- High-level idea: use FHE to encrypt *circuit* and then use Obf to "decrypt-then-evaluate"
  - Use Obf to hide FHE's secret key
- Attempt 1:
  - $\mathsf{Obf}'(\mathsf{C})$ consists of the following:
    1. FHE ciphertext c of C under pk
    2. $\mathsf{Obf}(\mathsf{P}_1)$, where P_1 is following decrypt-then-evaluate function

Boosting IO for NC^1 using FHE...

- Goal: **Obf** for NC^1 + FHE $\Pi \rightarrow$ **Obf**$'$ for all circuits
- High-level idea: use FHE to encrypt *circuit* and then use **Obf** to "decrypt-then-evaluate"
  - Use **Obf** to hide FHE's secret key
- Attempt 1:
  - $\mathsf{Obf}'(\mathsf{C})$ consists of the following:
    1. FHE ciphertext c of C under pk
    2. $\mathsf{Obf}(\mathsf{P}_1)$, where P_1 is following decrypt-then-evaluate function

$P_1(c, x)\{$
  $C := Dec(sk, c)$ hardwired
  Output $C(x)$
$\}$

Boosting IO for NC^1 using FHE...

- Goal: **Obf** for NC^1 + FHE $\Pi \rightarrow$ **Obf′** for all circuits
- High-level idea: use FHE to encrypt *circuit* and then use **Obf** to "decrypt-then-evaluate"
  - Use **Obf** to hide FHE's secret key
- Attempt 1:
  - Obf′(C) consists of the following:
    1. FHE ciphertext c of C under pk
    2. Obf(P_1), where P_1 is following decrypt-then-evaluate function

$P_1(c, x)\{$
$\quad C := \mathrm{Dec}(sk, c)$ hardwired
$\quad \text{Output } C(x)$
$\}$

  - To evaluate (c, Obf(P_1)) on x, output Obf(P_1)(C, x)

Boosting IO for NC^1 using FHE...

- Goal: **Obf** for NC^1 + FHE $\Pi \rightarrow$ **Obf′** for all circuits
- High-level idea: use FHE to encrypt *circuit* and then use **Obf** to "decrypt-then-evaluate"
  - Use **Obf** to hide FHE's secret key
- Attempt 1:
  - Obf′(C) consists of the following:
    1. FHE ciphertext c of C under pk
    2. Obf(P_1), where P_1 is following decrypt-then-evaluate function

$P_1(c, x)\{$

$\quad C := \mathrm{Dec}(sk, c)$ → hardwired

$\quad$ Output $C(x)$

$\}$

  - To evaluate $(c, \mathrm{Obf}(P_1))$ on x, output $\mathrm{Obf}(P_1)(C, x)$

Problem: ?

Boosting IO for NC^1 using FHE...

- Goal: **Obf** for NC^1 + FHE $\Pi \rightarrow$ **Obf$'$** for all circuits

High-level idea: use FHE to encrypt *circuit* and then use **Obf** to "decrypt-then-evaluate"

  - Use **Obf** to hide FHE's secret key

- Attempt 1:
  - **Obf$'$**(C) consists of the following:
    1. FHE ciphertext c of C under pk
    2. **Obf**(P_1), where P_1 is following decrypt-then-evaluate function

```
P1(c, x) {
    C := Dec(sk, c)      hardwired (sk)
    Output C(x)
}
```

  - To evaluate (c, **Obf**(P_1)) on x, output **Obf**(P_1)(C, x)

⚠ Problem: **Obf** does not support evaluation of C

not necessarily in NC^1

Boosting IO using FHE...

- Attempt 2: let's exploit homomorphic evaluation
  - Obf′(C) consists of the following:
    1. FHE ciphertext c of C under pk
    2. Obf(P_2), where P_2 is the following decrypt-and-output function

$P_2(e)\{$

$\quad y := \mathrm{Dec}(\overset{\text{hardwired}}{sk}, e)$

$\quad$ Output y

$\}$

Boosting IO using FHE...

- Attempt 2: let's exploit homomorphic evaluation
  - Obf′(C) consists of the following:
    1. FHE ciphertext c of C under pk
    2. Obf(P_2), where P_2 is the following decrypt-and-output function

$P_2(e)\{$
$y := \mathrm{Dec}(sk, e)$ → hardwired
Output y
$\}$

  - To evaluate (c, Obf(P_2)) on x, homomorphically evaluate

$$e := \mathsf{Eval}(pk, c, \mathsf{U}(\cdot, x))$$

universal circuit
∀circuit C, input x:
$U(C,x) = C(x)$

and then output Obf(P_2)(e)

Boosting IO using FHE...

- Attempt 2: let's exploit homomorphic evaluation
  - Obf′(C) consists of the following:
    1. FHE ciphertext c of C under pk
    2. Obf(P$_2$), where $\mathsf{P_2}$ is the following decrypt-and-output function

```
P_2 (e) {
    y := Dec(sk, e)    -> hardwired
    Output y
}
```

  - To evaluate (c, Obf(P$_2$)) on x, homomorphically evaluate

$$e := \mathsf{Eval}(pk, c, \mathsf{U}(\cdot, x))$$

universal circuit
∀circuit C, input x:
U(C,x) = C(x)

and then output Obf(P$_2$)(e)

Problem: ?

- Attempt 2: let's exploit homomorphic evaluation
  - $\mathsf{Obf}'(\mathsf{C})$ consists of the following:
    1. FHE ciphertext c of C under pk
    2. $\mathsf{Obf}(\mathsf{P}_2)$, where P_2 is the following decrypt-and-output function

$P_2(e)\{$
  $y := \mathrm{Dec}(sk, e)$ hardwired
  Output y
$\}$

  - To evaluate $(c, \mathsf{Obf}(\mathsf{P}_2))$ on x, homomorphically evaluate

$$e := \mathsf{Eval}(pk, c, \mathsf{U}(\cdot, x))$$

universal circuit
$\forall$ circuit C, input x:
$U(C,x) = C(x)$

and then output $\mathsf{Obf}(\mathsf{P}_2)(e)$

⚠ Problem: insecure as P_2 decrypts all ciphertexts

Boosting IO using FHE...

- Solution: only decrypt certain "constrained" ciphertexts

Boosting IO using FHE...

Solution: only decrypt certain "constrained" ciphertexts

- $\mathsf{Obf}'(\mathsf{C})$ consists of the following:
  1. Two FHE ciphertexts (c, c') of C, under pk and pk'
  2. $\mathsf{Obf}(\mathsf{P_3})$, where $\mathsf{P_3}$ is verify-then-decrypt-and-output function

$P_3(c, c', e, e', \pi, x)\{$

$\}$

Boosting IO using FHE...

Solution: only decrypt certain "constrained" ciphertexts

- Obf′(C) consists of the following:
  1. Two FHE ciphertexts (c, c') of C, under pk and pk'
  2. Obf(P_3), where P_3 is verify-then-decrypt-and-output function

$\mathrm{Eval}(pk', c', U(\cdot, x))$

$\mathrm{Eval}(pk, c, U(\cdot, x))$

$P_3(c, c', e, e', \pi, x)\{$

$\}$

Solution: only decrypt certain "constrained" ciphertexts

- Obf′(C) consists of the following:
  1. Two FHE ciphertexts (c, c') of C, under pk and pk'
  2. Obf(P_3), where P_3 is verify-then-decrypt-and-output function

$\mathrm{Eval}(pk', c', U(\cdot, x))$

$\mathrm{Eval}(pk, c, U(\cdot, x))$

wire values in computing e, e'

$P_3(c, c', e, e', \pi, x)\{$

$\}$

Solution: only decrypt certain "constrained" ciphertexts

- Obf′(C) consists of the following:
  1. Two FHE ciphertexts (c, c') of C, under pk and pk'
  2. Obf(P_3), where P_3 is verify-then-decrypt-and-output function

Eval(pk′, c′, U(·, x))

Eval(pk, c, U(·, x))

wire values in computing e, e′

$P_3(c, c', e, e', \pi, x)\{$

Verify computation of e, e′ using π

If valid, o/p Dec(sk, e)

$\}$

hardwired

Boosting IO using FHE...

Solution: only decrypt certain "constrained" ciphertexts

- Obf′(C) consists of the following:
  1. Two FHE ciphertexts (c, c') of C, under pk and pk'
  2. Obf(P_3), where P_3 is verify-then-decrypt-and-output function

Eval(pk′, c′, U(·, x))

Eval(pk, c, U(·, x))

wire values in computing e, e′

$P_3(c, c', e, e', \pi, x)$ {
  Verify computation of e, e′ using π
  If valid, o/p Dec(sk, e)
}

hardwired

- To evaluate $\mathsf{Obf}'(\mathsf{C}) := (c, c', \mathsf{Obf}(\mathsf{P_3}))$ on input x
  1. Evaluate $e := \mathsf{Eval}(pk, c, \mathsf{U}(\cdot, x))$ and $e' := \mathsf{Eval}(pk', c', \mathsf{U}(\cdot, x))$
  2. Let π be wire values during computation of e and e'
  3. Output $\mathsf{Obf}(\mathsf{P_3})(c, c', e, e', \pi, x)$

Solution: only decrypt certain "constrained" ciphertexts

- $\mathsf{Obf}'(\mathsf{C})$ consists of the following:
  1. Two FHE ciphertexts (c, c') of C, under pk and pk'
  2. $\mathsf{Obf}(\mathsf{P}_3)$, where P_3 is verify-then-decrypt-and-output function

$\mathrm{Eval}(pk', c', U(\cdot, x))$

$\mathrm{Eval}(pk, c, U(\cdot, x))$

wire values in computing e, e'

$P_3(c, c', e, e', \pi, x)\{$

Verify computation of e, e' using π

If valid, o/p $\mathrm{Dec}(sk, e)$

hardwired

$\}$

- To evaluate $\mathsf{Obf}'(\mathsf{C}) := (c, c', \mathsf{Obf}(\mathsf{P}_3))$ on input x
  1. Evaluate $e := \mathsf{Eval}(pk, c, \mathsf{U}(\cdot, x))$ and $e' := \mathsf{Eval}(pk', c', \mathsf{U}(\cdot, x))$
  2. Let π be wire values during computation of e and e'
  3. Output $\mathsf{Obf}(\mathsf{P}_3)(c, c', e, e', \pi, x)$

Exercise 1

Show that verifying π can be carried out in NC^1

Boosting IO using FHE...

Theorem 1

If Obf *is IO for* NC^1 *and* Π *is an FHE then* Obf' *is IO for all circuits*

Boosting IO using FHE...

Theorem 1

If Obf *is IO for* NC^1 *and* Π *is an FHE then* Obf' *is IO for all circuits*

Proof Sketch (Hybrid argument).

Left world H_0 — Goal — Right world H_4

$(c_0, c_0'), \; P_3(c, c', e, e', \pi, x)\{ \; \ldots \;$ If valid, o/p $\mathrm{Dec}(sk, e) \; \}$ $\approx$ $(c_1, c_1'), \; P_3(c, c', e, e', \pi, x)\{ \; \ldots \;$ If valid, o/p $\mathrm{Dec}(sk, e) \; \}$

$\square$

Boosting IO using FHE...

Theorem 1

If Obf *is IO for* NC^1 *and* Π *is an FHE then* Obf' *is IO for all circuits*

Proof Sketch (Hybrid argument).

Left world H_0 Goal   Right world H_4

$(c_0, c_0'), \; P_3(c, c', e, e', \pi, x)\{ \; \ldots \; \text{If valid, o/p } Dec(sk, e) \; \} \;\approx\; (c_1, c_1'), \; P_3(c, c', e, e', \pi, x)\{ \; \ldots \; \text{If valid, o/p } Dec(sk, e) \; \}$

Hybrid H_1

$(c_0, c_1'), \; P_3(c, c', e, e', \pi, x)\{ \; \ldots \; \text{If valid, o/p } Dec(sk, e) \; \}$

□

Boosting IO using FHE...

Theorem 1

If Obf *is IO for* NC^1 *and* Π *is an FHE then* Obf' *is IO for all circuits*

Proof Sketch (Hybrid argument).

Left world H_0 $\approx$ Goal Right world H_4

$(c_0, c_0'),$ $P_3(c, c', e, e', \pi, x)\{$ ... If valid, o/p $\mathrm{Dec}(sk, e)$ $\}$

$\approx$

$(c_1, c_1'),$ $P_3(c, c', e, e', \pi, x)\{$ ... If valid, o/p $\mathrm{Dec}(sk, e)$ $\}$

Hybrid H_1

$\approx_{FHE}$

$(c_0, c_1'),$ $P_3(c, c', e, e', \pi, x)\{$ ... If valid, o/p $\mathrm{Dec}(sk, e)$ $\}$

Hybrid H_2

$(c_0, c_1'),$ $P_3(c, c', e, e', \pi, x)\{$ ... If valid, o/p $\mathrm{Dec}(sk', e)$ $\}$

$\square$

Boosting IO using FHE...

Theorem 1

If Obf *is IO for* NC^1 *and* Π *is an FHE then* Obf' *is IO for all circuits*

Proof Sketch (Hybrid argument).

Left world H_0

(c_0, c'_0), $P_3(c, c', e, e', \pi, x)\{$ … If valid, o/p $Dec(sk, e)$ $\}$

$\approx$ Goal

Right world H_4

(c_1, c'_1), $P_3(c, c', e, e', \pi, x)\{$ … If valid, o/p $Dec(sk, e)$ $\}$

Hybrid H_1

$\approx_{FHE}$

(c_0, c'_1), $P_3(c, c', e, e', \pi, x)\{$ … If valid, o/p $Dec(sk, e)$ $\}$

$\approx_{IO}$

Hybrid H_2

(c_0, c'_1), $P_3(c, c', e, e', \pi, x)\{$ … If valid, o/p $Dec(sk', e)$ $\}$

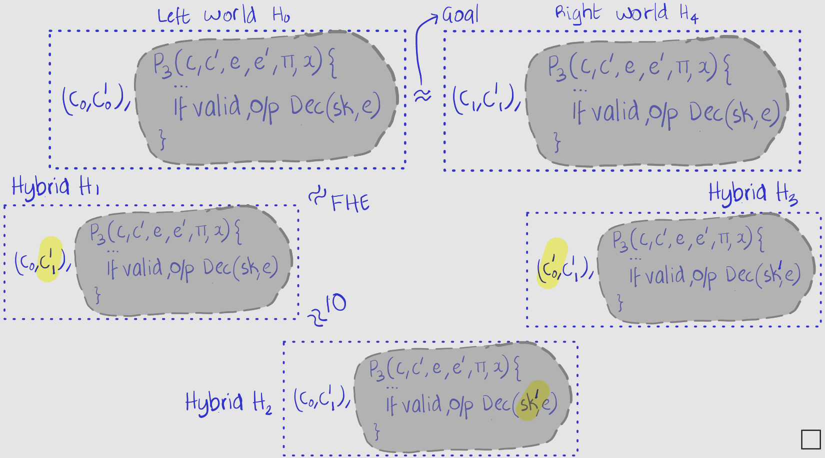
$\square$

Boosting IO using FHE...

Theorem 1

If Obf *is IO for* NC^1 *and* Π *is an FHE then* Obf$'$ *is IO for all circuits*

Proof Sketch (Hybrid argument).



$\square$

Boosting IO using FHE...

Theorem 1

If Obf *is IO for* NC^1 *and* Π *is an FHE then* Obf' *is IO for all circuits*

Proof Sketch (Hybrid argument).

Left world H_0

(c_0, c_0'), $P_3(c, c', e, e', \pi, x)\{$ ... If valid, o/p $\mathrm{Dec}(sk, e)$ $\}$

$\approx$ Goal

Right world H_4

(c_1, c_1'), $P_3(c, c', e, e', \pi, x)\{$ ... If valid, o/p $\mathrm{Dec}(sk, e)$ $\}$

Hybrid H_1

(c_0, c_1'), $P_3(c, c', e, e', \pi, x)\{$ ... If valid, o/p $\mathrm{Dec}(sk, e)$ $\}$

$\approx$ FHE

$\approx$ IO

Hybrid H_2

(c_0, c_1'), $P_3(c, c', e, e', \pi, x)\{$ ... If valid, o/p $\mathrm{Dec}(sk', e)$ $\}$

FHE $\approx$

Hybrid H_3

(c_0', c_1'), $P_3(c, c', e, e', \pi, x)\{$ ... If valid, o/p $\mathrm{Dec}(sk', e)$ $\}$

$\square$

Boosting IO using FHE...

Theorem 1

If Obf *is IO for* NC^1 *and* Π *is an FHE then* Obf' *is IO for all circuits*

Proof Sketch (Hybrid argument).

Left world H_0

$(c_0, c_0'),\ P_3(c, c', e, e', \pi, x)\{ \ \ldots$ If valid, o/p $\mathrm{Dec}(sk, e)$ $\}$

$\approx$ Goal

Right world H_4

$(c_1, c_1'),\ P_3(c, c', e, e', \pi, x)\{ \ \ldots$ If valid, o/p $\mathrm{Dec}(sk, e)$ $\}$

$\approx$ FHE

Hybrid H_1

$(c_0, c_1'),\ P_3(c, c', e, e', \pi, x)\{ \ \ldots$ If valid, o/p $\mathrm{Dec}(sk, e)$ $\}$

$\approx$ IO

Hybrid H_2

$(c_0, c_1'),\ P_3(c, c', e, e', \pi, x)\{ \ \ldots$ If valid, o/p $\mathrm{Dec}(sk', e)$ $\}$

FHE $\approx$

Hybrid H_3

$(c_0', c_1'),\ P_3(c, c', e, e', \pi, x)\{ \ \ldots$ If valid, o/p $\mathrm{Dec}(sk', e)$ $\}$

IO $\approx$

$\square$

Digression: Bootstrapping FHE

- Goal: HE Π for $\mathsf{NC}^1 \rightarrow$ FHE Π' for all circuits

Digression: Bootstrapping FHE

- Goal: HE Π for $\mathsf{NC}^1 \rightarrow$ FHE Π' for all circuits
  - Assumption: Π's Dec can be done in NC^1

Digression: Bootstrapping FHE

- Goal: HE Π for $\mathsf{NC}^1 \rightarrow$ FHE Π' for all circuits
  - Assumption: Π's Dec can be done in NC^1

Construction 1

- $\mathsf{Gen}'(1^n)$*:*
  - *Generate* $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$ *and compute* $c_{sk} := \mathsf{Enc}(pk, sk)$
  - *Output* $pk' := (pk, c_{sk})$ *as public key;* $sk' := sk$ *as secret key*
- Enc' *and* Dec' *are same as* Enc *and* Dec*, respectively*

Digression: Bootstrapping FHE

- Goal: HE Π for $\mathsf{NC}^1 \rightarrow$ FHE Π' for all circuits
  - Assumption: Π's Dec can be done in NC^1

Construction 1

- $\mathsf{Gen}'(1^n)$:
  - *Generate* $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$ *and compute* $c_{sk} := \mathsf{Enc}(pk, sk)$
  - *Output* $pk' := (pk, c_{sk})$ *as public key;* $sk' := sk$ *as secret key*
- Enc' *and* Dec' *are same as* Enc *and* Dec, *respectively*
- $\mathsf{Eval}'(pk, \mathrm{f}, c) := \mathsf{Eval}(pk, \mathrm{f}', c, c_{sk})$, *where*

$$f'(c, sk) \leftarrow Enc(pk, Dec((sk, c))$$

Digression: Bootstrapping FHE

- Goal: HE Π for $\mathsf{NC}^1 \rightarrow$ FHE Π' for all circuits
  - Assumption: Π's Dec can be done in NC^1

Construction 1

- $\mathsf{Gen}'(1^n)$:
  - *Generate* $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$ *and compute* $c_{sk} := \mathsf{Enc}(pk, sk)$
  - *Output* $pk' := (pk, c_{sk})$ *as public key;* $sk' := sk$ *as secret key*
- Enc' *and* Dec' *are same as* Enc *and* Dec*, respectively*
- $\mathsf{Eval}'(pk, \mathsf{f}, c) := \mathsf{Eval}(pk, \mathsf{f}', c, c_{sk})$*, where*

$$f'(c, sk) \leftarrow Enc(pk, Dec(sk, c))$$

Exercise 2

Show that Π' *is FHE for all circuits if* Π *is "circular secure" FHE for* NC^1

Plan for Today's Lecture

1. Boosting IO Using FHE

2. Constructing IO for NC^1: What Do We Know?

- Bilinear map:

$$\mathrm{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$$

such that for every $g_1, g_2 \in \mathbb{G}$ and $a, b \in \mathbb{Z}_p$,

$$\mathrm{e}(g_1^a, g_2^b) = \mathrm{e}(g_1, g_2)^{ab}$$

Multilinear Maps $\rightarrow$ IO for NC^1

- Bilinear map:

$$e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$$

such that for every $g_1, g_2 \in \mathbb{G}$ and $a, b \in \mathbb{Z}_p$,

$$e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$$

  - DDH easy in $\mathbb{G}$ (Homework 3, Problem 4)
  - Hardness assumption: bilinear version of DDH

Multilinear Maps $\rightarrow$ IO for NC^1

- Bilinear map:

$$e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$$

such that for every $g_1, g_2 \in \mathbb{G}$ and $a, b \in \mathbb{Z}_p$,

$$e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$$

  - DDH easy in $\mathbb{G}$ (Homework 3, Problem 4)
  - Hardness assumption: bilinear version of DDH
  - Can be constructed using "pairings" on elliptic curves

Multilinear Maps $\rightarrow$ IO for NC^1

- Bilinear map:

$$\mathrm{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$$

  such that for every $g_1, g_2 \in \mathbb{G}$ and $a, b \in \mathbb{Z}_p$,

$$\mathrm{e}(g_1^a, g_2^b) = \mathrm{e}(g_1, g_2)^{ab}$$

  - DDH easy in $\mathbb{G}$ (Homework 3, Problem 4)
  - Hardness assumption: bilinear version of DDH
  - Can be constructed using "pairings" on elliptic curves

- Multilinear map: extension to multiple "levels"
  - Multilinear maps with roughly logarithmic levels $\rightarrow$ IO for NC^1

Multilinear Maps $\rightarrow$ IO for NC^1

- Bilinear map:

$$e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$$

such that for every $g_1, g_2 \in \mathbb{G}$ and $a, b \in \mathbb{Z}_p$,

$$e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$$

  - DDH easy in $\mathbb{G}$ (Homework 3, Problem 4)
  - Hardness assumption: bilinear version of DDH
  - Can be constructed using "pairings" on elliptic curves

- Multilinear map: extension to multiple "levels"
  - Multilinear maps with roughly logarithmic levels $\rightarrow$ IO for NC^1

⚠ Problem: we don't know how to construct even trilinear maps

- All proposals of multilinear maps were later broken

LWE + "Local" PRG + Bilinear Maps $\rightarrow$ IO for NC^1

+ Recent result relaxes the assumptions to
  1. Learning with errors (LWE)
  2. Bilinear maps
  3. "Local" PRG: each output bit of the PRG only depends only on a "few" input bits

LWE + "Local" PRG + Bilinear Maps $\rightarrow$ IO for NC^1

+ Recent result relaxes the assumptions to
  1. Learning with errors (LWE)
  2. Bilinear maps
  3. "Local" PRG: each output bit of the PRG only depends only on a "few" input bits

− Construction is complex

LWE + “Local” PRG + Bilinear Maps $\rightarrow$ IO for NC^1

+ Recent result relaxes the assumptions to
  1. Learning with errors (LWE)
  2. Bilinear maps
  3. “Local” PRG: each output bit of the PRG only depends only on a “few” input bits

− Construction is complex

- Open:
  - LWE $\rightarrow$IO for NC^1
  - Simpler constructions from stronger assumptions

To Recap Module IV...

- We started with black-box separations: OWF$\not\rightarrow$OWP
- Program obfuscation and its applications
  - Potentially bypass black-box separations via non-black-box constructions

To Recap Module IV...

- We started with black-box separations: OWF$\nrightarrow$OWP
- Program obfuscation and its applications
  - Potentially bypass black-box separations via non-black-box constructions
  - Obfuscation has its limitations: e.g., OWF+IO$\nrightarrow$ CRHF

To Recap Module IV...

- We started with black-box separations: OWF $\not\rightarrow$ OWP
- Program obfuscation and its applications
  - Potentially bypass black-box separations via non-black-box constructions
  - Obfuscation has its limitations: e.g., OWF+IO $\not\rightarrow$ CRHF

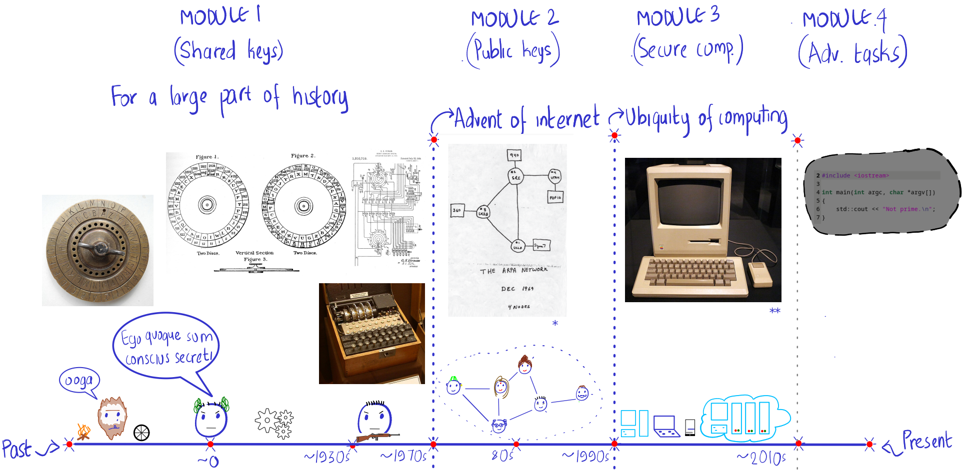
```
C(x){
   /*Hardwired: pk, m */
   if(G(x)=pk) output m
   else output "⊥"
}
```

IO for NC^1 $\xrightarrow{\text{FHE}}$ IO for P/poly

FHE for NC^1 $\longrightarrow$ FHE for P/poly

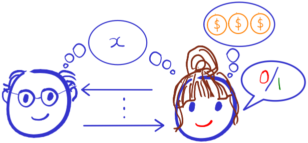
- Key tools/ideas: how to hide secrets using IO, punctured programming, bootstrapping/boosting
- Takeaways: separations are useful (they pin point our limits)

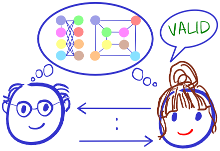
To Recap Module IV...



Spring'25: Introduction to Probabilistic Proof Systems

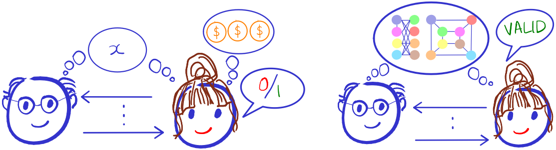
- Module III: Interactive proofs (IP), zero-knowledge proofs (ZKP)





Spring'25: Introduction to Probabilistic Proof Systems

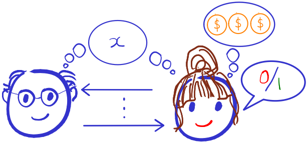
- Module III: Interactive proofs (IP), zero-knowledge proofs (ZKP)

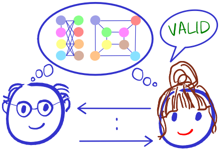


- Module I: Interactive proof (IP)
  - Class **AM**, sumcheck protocol, set lower bound protocol

Spring'25: Introduction to Probabilistic Proof Systems

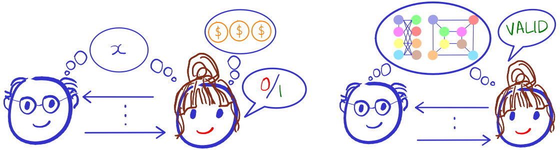
- Module III: Interactive proofs (IP), zero-knowledge proofs (ZKP)





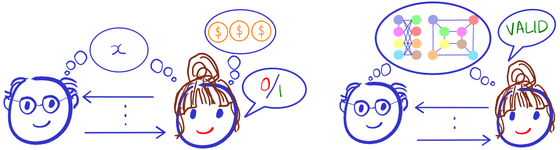
- Module I: Interactive proof (IP)
  - Class **AM**, sumcheck protocol, set lower bound protocol
- Module II: Probabilistically-checkable proof (PCP) and ZKP
  - Walsh-Hadamard PCP, More on ZKP

- Module III: Interactive proofs (IP), zero-knowledge proofs (ZKP)



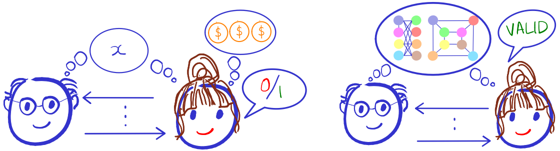
- Module I: Interactive proof (IP)
  - Class **AM**, sumcheck protocol, set lower bound protocol
- Module II: Probabilistically-checkable proof (PCP) and ZKP
  - Walsh-Hadamard PCP, More on ZKP
- Module III: Cryptographic proofs
  - Succinct arguments, SNARGs and NIZK

- Module III: Interactive proofs (IP), zero-knowledge proofs (ZKP)



- Module I: Interactive proof (IP)
  - Class **AM**, sumcheck protocol, set lower bound protocol
- Module II: Probabilistically-checkable proof (PCP) and ZKP
  - Walsh-Hadamard PCP, More on ZKP
- Module III: Cryptographic proofs
  - Succinct arguments, SNARGs and NIZK
- Module IV: Some recent results
  - Connections to complexity, batch proofs etc.

- Module III: Interactive proofs (IP), zero-knowledge proofs (ZKP)



- Module I: Interactive proof (IP)
  - Class **AM**, sumcheck protocol, set lower bound protocol
- Module II: Probabilistically-checkable proof (PCP) and ZKP
  - Walsh-Hadamard PCP, More on ZKP
- Module III: Cryptographic proofs
  - Succinct arguments, SNARGs and NIZK
- Module IV: Some recent results
  - Connections to complexity, batch proofs etc.

- Will send a link to course website via Moodle

References

1. The boosting result for IO is from [GGH+13]. The presentation here is from Lecture 13 of Mark Zhandry's COS597C course (Fall 16).
2. The bootstrapping result for FHE is due to Gentry [Gen09]
3. The construction of IO from multilinear maps can be found in [GGH+13]; the second construction can be found in [JLS21].

Craig Gentry.
Fully homomorphic encryption using ideal lattices.
In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.

Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters.
Candidate indistinguishability obfuscation and functional encryption for all circuits.
In *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013.

Aayush Jain, Huijia Lin, and Amit Sahai.
Indistinguishability obfuscation from well-founded assumptions.
In Samir Khuller and Virginia Vassilevska Williams, editors, *53rd ACM STOC*, pages 60–73. ACM Press, June 2021.