# Theoretical obfuscation

Alice Pellet-Mary

LIP, ENS de Lyon

Fridaycon, Quarkslab
May 17, 2019

# Obfuscation

An obfuscator should:

- render the code of a program unintelligible;
- while preserving functionality and efficiency.

# Overview of the talk

1. Definition
2. Candidates
   - Security
   - Practicability
3. Example of construction of an obfuscator

# Outline of the talk

# What is a program?

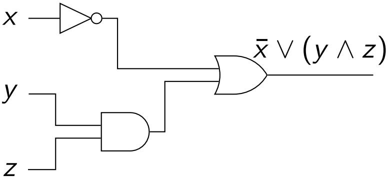
- C/C++/Python/$\cdots$ code;

# What is a program?

- C/C++/Python/$\cdots$ code;
- Turing machine;

# What is a program?

- C/C++/Python/$\cdots$ code;
- Turing machine;
- Boolean circuit;

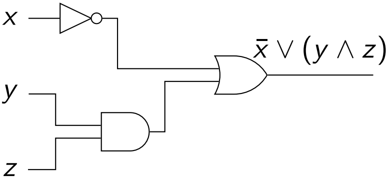



**Notation**

$\mathcal{C}$ = class of all polynomial size boolean circuits

# What is a program?

- C/C++/Python/$\cdots$ code;
- Turing machine;
- Boolean circuit;
- Branching programs;




**Notation**

$\mathcal{C}$ = class of all polynomial size boolean circuits

# Virtual Black Box (VBB) obfuscation

**Recall**

$\mathcal{C}$ = class of all polynomial size boolean circuits

A VBB obfuscator $\mathcal{O} : \mathcal{C} \to \mathcal{C}$ should satisfy

# Virtual Black Box (VBB) obfuscation

**Recall**

$\mathcal{C}$ = class of all polynomial size boolean circuits

A VBB obfuscator $\mathcal{O} : \mathcal{C} \to \mathcal{C}$ should satisfy

- (Functionality) For all $C \in \mathcal{C}$, $\mathcal{O}(C) \equiv C$;

# Virtual Black Box (VBB) obfuscation

**Recall**

$\mathcal{C}$ = class of all polynomial size boolean circuits

A VBB obfuscator $\mathcal{O} : \mathcal{C} \to \mathcal{C}$ should satisfy

- (Functionality) For all $C \in \mathcal{C}$, $\mathcal{O}(C) \equiv C$;
- (Efficiency) For all $C \in \mathcal{C}$, $|\mathcal{O}(C)| \leq p(|C|)$ for some polynomial $p$;

# Virtual Black Box (VBB) obfuscation

**Recall**

$\mathcal{C}$ = class of all polynomial size boolean circuits

A VBB obfuscator $\mathcal{O} : \mathcal{C} \to \mathcal{C}$ should satisfy

- (Functionality) For all $C \in \mathcal{C}$, $\mathcal{O}(C) \equiv C$;
- (Efficiency) For all $C \in \mathcal{C}$, $|\mathcal{O}(C)| \leq p(|C|)$ for some polynomial $p$;
- (Virtual Black Box security) For all PPT $\mathcal{A}$, there exists a PPT Sim s.t. for all $C \in \mathcal{C}$,

$$\left| \mathbb{P}\left[\mathcal{A}(\mathcal{O}(C)) = 1\right] - \mathbb{P}\left[\mathrm{Sim}^{C}(1^{|C|}) = 1\right] \right| \leq \text{negl}.$$

# Virtual Black Box (VBB) obfuscation

**Recall**

$\mathcal{C}$ = class of all polynomial size boolean circuits

A VBB obfuscator $\mathcal{O} : \mathcal{C} \to \mathcal{C}$ should satisfy

- (Functionality) For all $C \in \mathcal{C}$, $\mathcal{O}(C) \equiv C$;
- (Efficiency) For all $C \in \mathcal{C}$, $|\mathcal{O}(C)| \leq p(|C|)$ for some polynomial $p$;
- (Virtual Black Box security) For all PPT $\mathcal{A}$, there exists a PPT Sim s.t. for all $C \in \mathcal{C}$,

$$\left| \mathbb{P}\left[\mathcal{A}(\mathcal{O}(C)) = 1\right] - \mathbb{P}\left[\mathrm{Sim}^{C}(1^{|C|}) = 1\right] \right| \leq \text{negl}.$$

VBB obfuscation is impossible to achieve [BGI$^+$01]

[BGI+01] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan and K. Yang. On the (im) possibility of obfuscating programs, Crypto.

# Indistinguishability Obfuscation (iO)

An indistinguishability obfuscator $\mathcal{O} : \mathcal{C} \to \mathcal{C}$ should satisfy

# Indistinguishability Obfuscation (iO)

An indistinguishability obfuscator $\mathcal{O} : \mathcal{C} \to \mathcal{C}$ should satisfy

- (Functionality) For all $C \in \mathcal{C}$, $\mathcal{O}(C) \equiv C$;

# Indistinguishability Obfuscation (iO)

An indistinguishability obfuscator $\mathcal{O} : \mathcal{C} \rightarrow \mathcal{C}$ should satisfy

- (Functionality) For all $C \in \mathcal{C}$, $\mathcal{O}(C) \equiv C$;
- (Efficiency) For all $C \in \mathcal{C}$, $|\mathcal{O}(C)| \leq p(|C|)$ for some polynomial $p$;

# Indistinguishability Obfuscation (iO)

An indistinguishability obfuscator $\mathcal{O} : \mathcal{C} \to \mathcal{C}$ should satisfy

- (Functionality) For all $C \in \mathcal{C}$, $\mathcal{O}(C) \equiv C$;
- (Efficiency) For all $C \in \mathcal{C}$, $|\mathcal{O}(C)| \leq p(|C|)$ for some polynomial $p$;
- (indistinguishability) For all $C_1, C_2 \in \mathcal{C}$ with $C_1 \equiv C_2$,

$$\mathcal{O}(C_1) \simeq_c \mathcal{O}(C_2).$$

# Why is iO useful (1)

iO achieves “best possible” obfuscation

# Why is iO useful (1)

iO achieves “best possible” obfuscation

**Proof:**

- let $\mathcal{O}$ be an iO obfuscator and $\mathcal{O}'$ be another obfuscator

# Why is iO useful (1)

iO achieves "best possible" obfuscation

**Proof:**

- let $\mathcal{O}$ be an iO obfuscator and $\mathcal{O}'$ be another obfuscator
- for any $C \in \mathcal{C}$, $\mathcal{O}(C) \simeq_c \mathcal{O}(\mathcal{O}'(C))$

# Why is iO useful (1)

iO achieves “best possible” obfuscation

**Proof:**

- let $\mathcal{O}$ be an iO obfuscator and $\mathcal{O}'$ be another obfuscator
- for any $C \in \mathcal{C}$, $\mathcal{O}(C) \simeq_c \mathcal{O}(\mathcal{O}'(C))$
- $\mathcal{O}(\mathcal{O}'(C))$ reveals less info than $\mathcal{O}'(C)$

# Why is iO useful (1)

iO achieves "best possible" obfuscation

**Proof:**

- let $\mathcal{O}$ be an iO obfuscator and $\mathcal{O}'$ be another obfuscator
- for any $C \in \mathcal{C}$, $\mathcal{O}(C) \simeq_c \mathcal{O}(\mathcal{O}'(C))$
- $\mathcal{O}(\mathcal{O}'(C))$ reveals less info than $\mathcal{O}'(C)$
- $\mathcal{O}(C)$ reveals less info than $\mathcal{O}'(C)$

# Why is iO useful (1)

iO achieves "best possible" obfuscation

**Proof:**

- let $\mathcal{O}$ be an iO obfuscator and $\mathcal{O}'$ be another obfuscator
- for any $C \in \mathcal{C}$, $\mathcal{O}(C) \simeq_c \mathcal{O}(\mathcal{O}'(C))$
- $\mathcal{O}(\mathcal{O}'(C))$ reveals less info than $\mathcal{O}'(C)$
- $\mathcal{O}(C)$ reveals less info than $\mathcal{O}'(C)$

Informally: anything revealed by $\mathcal{O}(C)$ is revealed by any $C' \equiv C$

# Why is iO useful (2)

Many cryptographic constructions from iO:
functional encryption, deniable encryption, NIKZs, oblivious transfer, ...

# Why is iO useful (2)

Many cryptographic constructions from iO:
functional encryption, deniable encryption, NIKZs, oblivious transfer, . . .

**Example:** black box decryption (symmetric setting)

# Why is iO useful (2)

Many cryptographic constructions from iO:
functional encryption, deniable encryption, NIKZs, oblivious transfer, ...

**Example:** black box decryption (symmetric setting)

Recall: anything revealed by $\mathcal{O}(C)$ is revealed by any $C' \equiv C$

# Why is iO useful (2)

Many cryptographic constructions from iO:
functional encryption, deniable encryption, NIKZs, oblivious transfer, ...

**Example:** black box decryption (symmetric setting)

Recall: anything revealed by $\mathcal{O}(C)$ is revealed by any $C' \equiv C$

- take ($\texttt{Setup}$,$\texttt{Enc}$,$\texttt{Dec}$) your favourite SKE scheme

# Why is iO useful (2)

Many cryptographic constructions from iO:
functional encryption, deniable encryption, NIKZs, oblivious transfer, ...

**Example:** black box decryption (symmetric setting)

Recall: anything revealed by $\mathcal{O}(C)$ is revealed by any $C' \equiv C$

- take (Setup,Enc,Dec) your favourite SKE scheme
- Setup':
  - $sk_1 \leftarrow \texttt{Setup}(),\ \ sk_2 \leftarrow \texttt{Setup}()$
  - output $sk' = (sk_1, sk_2)$

# Why is iO useful (2)

Many cryptographic constructions from iO:
functional encryption, deniable encryption, NIKZs, oblivious transfer, ...

**Example:** black box decryption (symmetric setting)

Recall: anything revealed by $\mathcal{O}(C)$ is revealed by any $C' \equiv C$

- take (`Setup`,`Enc`,`Dec`) your favourite SKE scheme
- `Setup'`:
  - $sk_1 \leftarrow \texttt{Setup}()$, $sk_2 \leftarrow \texttt{Setup}()$
  - output $sk' = (sk_1, sk_2)$
- `Enc'`$(m, sk')$:
  - $c_1 \leftarrow \texttt{Enc}(m, sk_1)$, $c_2 \leftarrow \texttt{Enc}(m, sk_2)$
  - output $(c_1, c_2)$

# Why is iO useful (2)

Many cryptographic constructions from iO:
functional encryption, deniable encryption, NIKZs, oblivious transfer, ...

**Example:** black box decryption (symmetric setting)

Recall: anything revealed by $\mathcal{O}(C)$ is revealed by any $C' \equiv C$

- take ($\texttt{Setup}$,$\texttt{Enc}$,$\texttt{Dec}$) your favourite SKE scheme
- `Setup'`:
  - $sk_1 \leftarrow \texttt{Setup}()$, $sk_2 \leftarrow \texttt{Setup}()$
  - output $sk' = (sk_1, sk_2)$
- `Enc'`$(m, sk')$:
  - $c_1 \leftarrow \texttt{Enc}(m, sk_1)$, $c_2 \leftarrow \texttt{Enc}(m, sk_2)$
  - output $(c_1, c_2)$
- `Dec'`:
  - $C_1(c_1, c_2) = \texttt{Dec}(sk_1, c_1)$ ($sk_1$ hardcoded in $C_1$)
  - $C_2(c_1, c_2) = \texttt{Dec}(sk_2, c_2)$ ($sk_2$ hardcoded in $C_2$)

# Why is iO useful (2)

Many cryptographic constructions from iO:
functional encryption, deniable encryption, NIKZs, oblivious transfer, ...

**Example:** black box decryption (symmetric setting)

Recall: anything revealed by $\mathcal{O}(C)$ is revealed by any $C' \equiv C$

- take $(\texttt{Setup}, \texttt{Enc}, \texttt{Dec})$ your favourite SKE scheme
- Setup':
  - $sk_1 \leftarrow \texttt{Setup}()$, $sk_2 \leftarrow \texttt{Setup}()$
  - output $sk' = (sk_1, sk_2)$
- Enc'$(m, sk')$:
  - $c_1 \leftarrow \texttt{Enc}(m, sk_1)$, $c_2 \leftarrow \texttt{Enc}(m, sk_2)$
  - output $(c_1, c_2)$
- Dec':
  - $C_1(c_1, c_2) = \texttt{Dec}(sk_1, c_1)$ ($sk_1$ hardcoded in $C_1$)
  - $C_2(c_1, c_2) = \texttt{Dec}(sk_2, c_2)$ ($sk_2$ hardcoded in $C_2$)

$C_1 \equiv C_2 \Rightarrow C = \mathcal{O}(C_1) \simeq_c \mathcal{O}(C_2)$ does not reveal $sk_1$ or $sk_2$

# Outline of the talk

# Disclaimer

We only have candidate iO

(no construction based on standard cryptographic assumptions)

# Three main categories

- Branching program obfuscators

# Three main categories

- Branching program obfuscators
  - needs bootstrapping via fully homomorphic encryption

# Three main categories

- Branching program obfuscators
  - needs bootstrapping via fully homomorphic encryption
  - security proofs in some idealized models . . .
  - . . . but many attacks

# Three main categories

- Branching program obfuscators
  - needs bootstrapping via fully homomorphic encryption
  - security proofs of VBB in some idealized models ...
  - ... but many attacks

# Three main categories

- Branching program obfuscators
  - needs bootstrapping via fully homomorphic encryption
  - security proofs of VBB in some idealized models ...
  - ... but many attacks
- Circuit obfuscators
  - no need for bootstrapping

# Three main categories

- Branching program obfuscators
  - needs bootstrapping via fully homomorphic encryption
  - security proofs of VBB in some idealized models ...
  - ... but many attacks
- Circuit obfuscators
  - no need for bootstrapping
  - security proofs of VBB in some idealized models ...
  - ... but many attacks

# Three main categories

- Branching program obfuscators
  - needs bootstrapping via fully homomorphic encryption
  - security proofs of VBB in some idealized models ...
  - ... but many attacks
- Circuit obfuscators
  - no need for bootstrapping
  - security proofs of VBB in some idealized models ...
  - ... but many attacks
- Obfuscation via functional encryption
  - try to find the weakest primitive implying iO
  - some attacks and impossibility results (not well understood yet)
  - most of them are not instantiable

# Security

Branching program and circuit obfuscators use multilinear maps.
All the candidate multilinear maps we know suffer from weaknesses.

# Security

Branching program and circuit obfuscators use multilinear maps.
All the candidate multilinear maps we know suffer from weaknesses.

| | number of candidates | still standing classically | still standing quantumly |
|---|---|---|---|
| Branching program iO | $\approx 20$ | $\approx 10$ | 3 |
| Circuit iO | $\approx 8$ | $\approx 8$ | 0 |

All attacks rely on the underlying multilinear map

# Restricted functionalities

- point functions

$$f_y(x) = 1 \text{ iff } x = y$$

# Restricted functionalities

- point functions

$$f_y(x) = 1 \text{ iff } x = y$$

- conjunctions

$$f(x_1, \dots, x_n) = \bigwedge_{i \in I} y_i \quad (\text{with } y_i = x_i \text{ or } \bar{x}_i)$$

# Restricted functionalities

- point functions

$$f_y(x) = 1 \text{ iff } x = y$$

- conjunctions

$$f(x_1, \ldots, x_n) = \bigwedge_{i \in I} y_i \quad (\text{with } y_i = x_i \text{ or } \bar{x}_i)$$

- compute-and-compare functions

$$f_{g,y}(x) = 1 \text{ iff } g(x) = y$$

# Restricted functionalities

- point functions

$$f_y(x) = 1 \text{ iff } x = y$$

- conjunctions

$$f(x_1, \ldots, x_n) = \bigwedge_{i \in I} y_i \quad (\text{with } y_i = x_i \text{ or } \bar{x}_i)$$

- compute-and-compare functions

$$f_{g,y}(x) = 1 \text{ iff } g(x) = y$$

VBB obfuscators based on RLWE

# Practicability

| function obfuscated | security parameter $\lambda$ | size obfuscated program | obfuscation time | evaluation time | security assumption | reference |
|---|---|---|---|---|---|---|
| AES | 128 | 18 700 TB | | $10^{10}$ mults of $10^8$ bits integers | none - | [YLX17] |
| one-round key-exchange with 4 users | 52 | 4.8 GB | 2h20 | $\leq$ 1 min | none - | [CP18] |
| $A_1^{x_1} \times \cdots \times A_{20}^{x_{20}}$ | 80 | | 80 h | 25 min | none | [HHSSD17] |
| $x_1 \wedge \bar{x}_4 \wedge \cdots \wedge x_{32}$ | 53 | | 6.2 min | 32ms | entropic RLWE | [CDCG+18] |
| $x_1 \wedge \bar{x}_4 \wedge \cdots \wedge x_{64}$ | 73 | | 6.7h | 2.4s | entropic RLWE | [CDCG+18] |

# Outline of the talk

# Branching programs

A branching program is a way of representing a function (like a Turing machine, or a circuit).

# Branching programs

A branching program is a way of representing a function (like a Turing machine, or a circuit).

## A Branching Program (BP) is a collection of

- $2\ell$ matrices $A_{i,b}$ (for $i \in \{1, \dots, \ell\}$ and $b \in \{0, 1\}$),
- two vectors $A_0$ and $A_{\ell+1}$,
- a function $\text{inp} : \{1, \dots, \ell\} \to \{1, \dots, r\}$ (where $r$ is the size of the input).

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $\text{inp}(i)$ | 1 | 1 | 2 | 1 | 3 | 2 |

$x = 0 \quad 1 \quad 1$

$$A_0 \quad \begin{matrix} A_{1,1} \\ A_{1,0} \end{matrix} \quad \begin{matrix} A_{2,1} \\ A_{2,0} \end{matrix} \quad \begin{matrix} A_{3,1} \\ A_{3,0} \end{matrix} \quad \begin{matrix} A_{4,1} \\ A_{4,0} \end{matrix} \quad \begin{matrix} A_{5,1} \\ A_{5,0} \end{matrix} \quad \begin{matrix} A_{6,1} \\ A_{6,0} \end{matrix} \quad A_7$$

# Branching programs

A branching program is a way of representing a function (like a Turing machine, or a circuit).

## A Branching Program (BP) is a collection of

- $2\ell$ matrices $A_{i,b}$ (for $i \in \{1, \ldots, \ell\}$ and $b \in \{0, 1\}$),
- two vectors $A_0$ and $A_{\ell+1}$,
- a function $\text{inp} : \{1, \ldots, \ell\} \to \{1, \ldots, r\}$ (where $r$ is the size of the input).

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $\text{inp}(i)$ | 1 | 1 | 2 | 1 | 3 | 2 |

$x = 0 \quad 1 \quad 1$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $A_0$ | $A_{1,1}$ | $A_{2,1}$ | $A_{3,1}$ | $A_{4,1}$ | $A_{5,1}$ | $A_{6,1}$ | $A_7$ |
| | $A_{1,0}$ | $A_{2,0}$ | $A_{3,0}$ | $A_{4,0}$ | $A_{5,0}$ | $A_{6,0}$ | |

# Branching programs

A branching program is a way of representing a function (like a Turing machine, or a circuit).

**A Branching Program (BP) is a collection of**

- $2\ell$ matrices $A_{i,b}$ (for $i \in \{1, \dots, \ell\}$ and $b \in \{0, 1\}$),
- two vectors $A_0$ and $A_{\ell+1}$,
- a function $\mathrm{inp} : \{1, \dots, \ell\} \to \{1, \dots, r\}$ (where $r$ is the size of the input).

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $\mathrm{inp}(i)$ | 1 | 1 | 2 | 1 | 3 | 2 |

$$x = 0 \quad 1 \quad 1$$
$$\uparrow$$

$$A_0 \times \begin{matrix} A_{1,1} & A_{2,1} & A_{3,1} & A_{4,1} & A_{5,1} & A_{6,1} \\ A_{1,0} & A_{2,0} & A_{3,0} & A_{4,0} & A_{5,0} & A_{6,0} \end{matrix} \quad A_7$$

# Branching programs

A branching program is a way of representing a function (like a Turing machine, or a circuit).

**A Branching Program (BP) is a collection of**

- $2\ell$ matrices $A_{i,b}$ (for $i \in \{1, \dots, \ell\}$ and $b \in \{0,1\}$),
- two vectors $A_0$ and $A_{\ell+1}$,
- a function $\text{inp} : \{1, \dots, \ell\} \to \{1, \dots, r\}$ (where $r$ is the size of the input).

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| inp($i$) | 1 | 1 | 2 | 1 | 3 | 2 |

$x = 0 \quad 1 \quad 1$
$\uparrow$

$$A_0 \times \begin{matrix} A_{1,1} \\ A_{1,0} \end{matrix} \times \begin{matrix} A_{2,1} \\ A_{2,0} \end{matrix} \quad \begin{matrix} A_{3,1} \\ A_{3,0} \end{matrix} \quad \begin{matrix} A_{4,1} \\ A_{4,0} \end{matrix} \quad \begin{matrix} A_{5,1} \\ A_{5,0} \end{matrix} \quad \begin{matrix} A_{6,1} \\ A_{6,0} \end{matrix} \quad A_7$$

# Branching programs

A branching program is a way of representing a function (like a Turing machine, or a circuit).

**A Branching Program (BP) is a collection of**

- $2\ell$ matrices $A_{i,b}$ (for $i \in \{1, \dots, \ell\}$ and $b \in \{0, 1\}$),
- two vectors $A_0$ and $A_{\ell+1}$,
- a function $\text{inp} : \{1, \dots, \ell\} \to \{1, \dots, r\}$ (where $r$ is the size of the input).

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $\text{inp}(i)$ | 1 | 1 | 2 | 1 | 3 | 2 |

$x = 0 \quad 1 \quad 1$

$\uparrow$

$$A_0 \times \begin{matrix} A_{1,1} \\ A_{1,0} \end{matrix} \times \begin{matrix} A_{2,1} \\ A_{2,0} \end{matrix} \times \begin{matrix} A_{3,1} \\ A_{3,0} \end{matrix} \quad \begin{matrix} A_{4,1} \\ A_{4,0} \end{matrix} \quad \begin{matrix} A_{5,1} \\ A_{5,0} \end{matrix} \quad \begin{matrix} A_{6,1} \\ A_{6,0} \end{matrix} \quad A_7$$

# Branching programs

A branching program is a way of representing a function (like a Turing machine, or a circuit).

**A Branching Program (BP) is a collection of**

- $2\ell$ matrices $A_{i,b}$ (for $i \in \{1, \dots, \ell\}$ and $b \in \{0, 1\}$),
- two vectors $A_0$ and $A_{\ell+1}$,
- a function $\text{inp} : \{1, \dots, \ell\} \to \{1, \dots, r\}$ (where $r$ is the size of the input).

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $\text{inp}(i)$ | 1 | 1 | 2 | 1 | 3 | 2 |

$$x = 0 \quad 1 \quad 1$$
$$\uparrow$$

$$A_0 \times \begin{matrix} A_{1,1} \\ A_{1,0} \end{matrix} \times \begin{matrix} A_{2,1} \\ A_{2,0} \end{matrix} \times \begin{matrix} A_{3,1} \\ A_{3,0} \end{matrix} \times \begin{matrix} A_{4,1} \\ A_{4,0} \end{matrix} \quad \begin{matrix} A_{5,1} \\ A_{5,0} \end{matrix} \quad \begin{matrix} A_{6,1} \\ A_{6,0} \end{matrix} \quad A_7$$

# Branching programs

A branching program is a way of representing a function (like a Turing machine, or a circuit).

## A Branching Program (BP) is a collection of

- $2\ell$ matrices $A_{i,b}$ (for $i \in \{1, \dots, \ell\}$ and $b \in \{0, 1\}$),
- two vectors $A_0$ and $A_{\ell+1}$,
- a function $\text{inp} : \{1, \dots, \ell\} \to \{1, \dots, r\}$ (where $r$ is the size of the input).

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $\text{inp}(i)$ | 1 | 1 | 2 | 1 | 3 | 2 |

$x = 0 \quad 1 \quad 1$

$\uparrow$

$$A_0 \times \begin{matrix} A_{1,1} \\ A_{1,0} \end{matrix} \times \begin{matrix} A_{2,1} \\ A_{2,0} \end{matrix} \times \begin{matrix} A_{3,1} \\ A_{3,0} \end{matrix} \times \begin{matrix} A_{4,1} \\ A_{4,0} \end{matrix} \times \begin{matrix} A_{5,1} \\ A_{5,0} \end{matrix} \quad \begin{matrix} A_{6,1} \\ A_{6,0} \end{matrix} \quad A_7$$

# Branching programs

A branching program is a way of representing a function (like a Turing machine, or a circuit).

## A Branching Program (BP) is a collection of

- $2\ell$ matrices $A_{i,b}$ (for $i \in \{1, \ldots, \ell\}$ and $b \in \{0, 1\}$),
- two vectors $A_0$ and $A_{\ell+1}$,
- a function $\mathrm{inp} : \{1, \ldots, \ell\} \to \{1, \ldots, r\}$ (where $r$ is the size of the input).

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $\mathrm{inp}(i)$ | 1 | 1 | 2 | 1 | 3 | 2 |

$x = 0 \quad 1 \quad 1$

$\uparrow$

$$A_0 \times \begin{matrix} A_{1,1} \\ A_{1,0} \end{matrix} \times \begin{matrix} A_{2,1} \\ A_{2,0} \end{matrix} \times \begin{matrix} A_{3,1} \\ A_{3,0} \end{matrix} \times \begin{matrix} A_{4,1} \\ A_{4,0} \end{matrix} \times \begin{matrix} A_{5,1} \\ A_{5,0} \end{matrix} \times \begin{matrix} A_{6,1} \\ A_{6,0} \end{matrix} \quad A_7$$

# Branching programs

A branching program is a way of representing a function (like a Turing machine, or a circuit).

## A Branching Program (BP) is a collection of

- $2\ell$ matrices $A_{i,b}$ (for $i \in \{1, \ldots, \ell\}$ and $b \in \{0,1\}$),
- two vectors $A_0$ and $A_{\ell+1}$,
- a function $\text{inp} : \{1, \ldots, \ell\} \to \{1, \ldots, r\}$ (where $r$ is the size of the input).

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $\text{inp}(i)$ | 1 | 1 | 2 | 1 | 3 | 2 |

$x = 0 \quad 1 \quad 1$

$$A_0 \times \begin{matrix} A_{1,1} \\ A_{1,0} \end{matrix} \times \begin{matrix} A_{2,1} \\ A_{2,0} \end{matrix} \times \begin{matrix} A_{3,1} \\ A_{3,0} \end{matrix} \times \begin{matrix} A_{4,1} \\ A_{4,0} \end{matrix} \times \begin{matrix} A_{5,1} \\ A_{5,0} \end{matrix} \times \begin{matrix} A_{6,1} \\ A_{6,0} \end{matrix} \times A_7$$

# Branching programs

A branching program is a way of representing a function (like a Turing machine, or a circuit).

## A Branching Program (BP) is a collection of

- $2\ell$ matrices $A_{i,b}$ (for $i \in \{1, \dots, \ell\}$ and $b \in \{0, 1\}$),
- two vectors $A_0$ and $A_{\ell+1}$,
- a function inp : $\{1, \dots, \ell\} \to \{1, \dots, r\}$ (where $r$ is the size of the input).

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| inp($i$) | 1 | 1 | 2 | 1 | 3 | 2 |

$$x = 0 \quad 1 \quad 1$$

$$A_0 \times \begin{matrix} A_{1,1} \\ A_{1,0} \end{matrix} \times \begin{matrix} A_{2,1} \\ A_{2,0} \end{matrix} \times \begin{matrix} A_{3,1} \\ A_{3,0} \end{matrix} \times \begin{matrix} A_{4,1} \\ A_{4,0} \end{matrix} \times \begin{matrix} A_{5,1} \\ A_{5,0} \end{matrix} \times \begin{matrix} A_{6,1} \\ A_{6,0} \end{matrix} \times A_7 \begin{matrix} = 0 \to 0 \\ \neq 0 \to 1 \end{matrix}$$

# Cryptographic multilinear maps

## Definition: $\kappa$-multilinear map

Different levels of encodings, from 1 to $\kappa$.
Denote by $\mathrm{Enc}(a, i)$ a level-$i$ encoding of the message $a$.

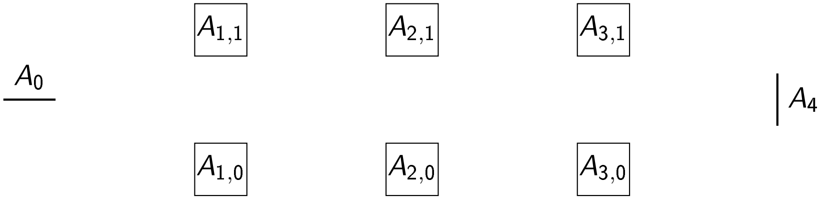
**Addition:** $\mathrm{Add}(\mathrm{Enc}(a_1, i), \mathrm{Enc}(a_2, i)) = \mathrm{Enc}(a_1 + a_2, i)$.

**Multiplication:** $\mathrm{Mult}(\mathrm{Enc}(a_1, i), \mathrm{Enc}(a_2, j)) = \mathrm{Enc}(a_1 \cdot a_2, i + j)$.

**Zero-test:** $\text{Zero-test}(\mathrm{Enc}(a, \kappa)) = \text{True iff } a = 0$.

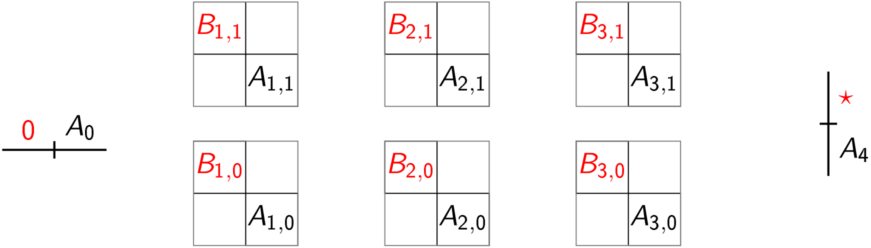
# Simple obfuscator

- **Input:** A branching program
- Randomize the branching program
  - Add random diagonal blocks
  - Killian's randomization
  - Multiply by random (non zero) bundling scalars
- Encode the matrices using GGH13
- **Output:** The encoded matrices and vectors



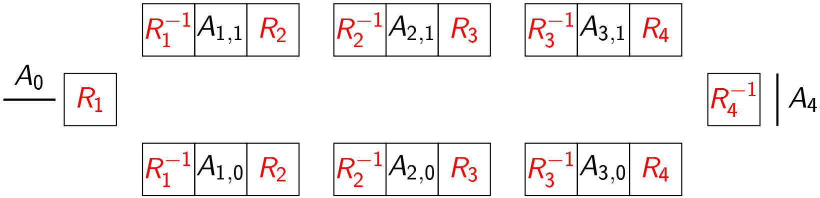
# Simple obfuscator

- **Input:** A branching program
- Randomize the branching program
  - Add random diagonal blocks
  - Killian's randomization
  - Multiply by random (non zero) bundling scalars
- Encode the matrices using GGH13
- **Output:** The encoded matrices and vectors



# Simple obfuscator

- **Input:** A branching program
- Randomize the branching program
  - Add random diagonal blocks
  - Killian's randomization
  - Multiply by random (non zero) bundling scalars
- Encode the matrices using GGH13
- **Output:** The encoded matrices and vectors



# Simple obfuscator

- **Input:** A branching program
- Randomize the branching program
  - Add random diagonal blocks
  - Killian's randomization
  - Multiply by random (non zero) bundling scalars
- Encode the matrices using GGH13
- **Output:** The encoded matrices and vectors

$$\alpha_{1,1} \times \boxed{A_{1,1}} \quad \alpha_{2,1} \times \boxed{A_{2,1}} \quad \alpha_{3,1} \times \boxed{A_{3,1}}$$

$$\underline{A_0} \qquad\qquad\qquad\qquad\qquad\qquad \big| A_4$$

$$\alpha_{1,0} \times \boxed{A_{1,0}} \quad \alpha_{2,0} \times \boxed{A_{2,0}} \quad \alpha_{3,0} \times \boxed{A_{3,0}}$$

# Simple obfuscator

- **Input:** A branching program
- Randomize the branching program
  - Add random diagonal blocks
  - Killian's randomization
  - Multiply by random (non zero) bundling scalars
- Encode the matrices using GGH13
- **Output:** The encoded matrices and vectors

$\widetilde{A_{1,1}}$ $\widetilde{A_{2,1}}$ $\widetilde{A_{3,1}}$

$\widetilde{A_0}$ $\widetilde{A_4}$

$\widetilde{A_{1,0}}$ $\widetilde{A_{2,0}}$ $\widetilde{A_{3,0}}$

# Simple obfuscator

- **Input:** A branching program
- Randomize the branching program
  - Add random diagonal blocks
  - Killian's randomization
  - Multiply by random (non zero) bundling scalars
- Encode the matrices using GGH13
- **Output:** The encoded matrices and vectors

$\mathrm{Enc}(\widetilde{A_{1,1}})$ $\mathrm{Enc}(\widetilde{A_{2,1}})$ $\mathrm{Enc}(\widetilde{A_{3,1}})$

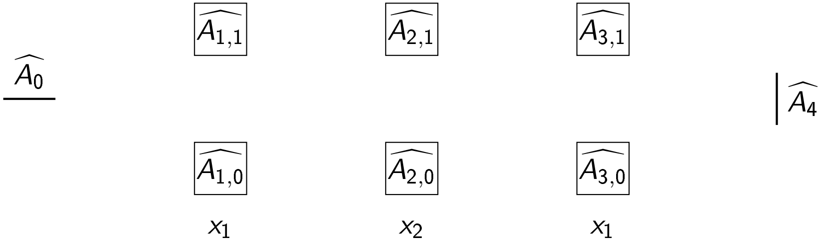
$\mathrm{Enc}(\widetilde{A_0})$ $\mathrm{Enc}(\widetilde{A_4})$

$\mathrm{Enc}(\widetilde{A_{1,0}})$ $\mathrm{Enc}(\widetilde{A_{2,0}})$ $\mathrm{Enc}(\widetilde{A_{3,0}})$

# Mixed-input attack

## Notations

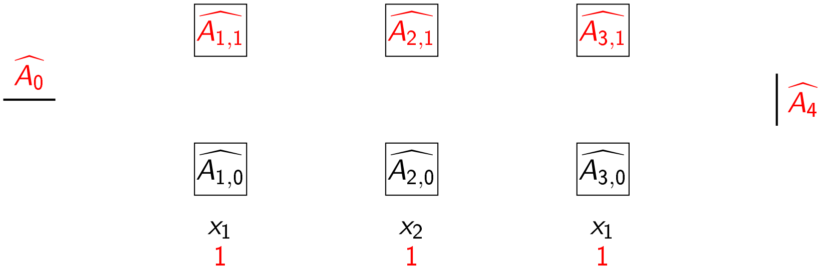
- $A_{i,b}$ input branching program
- $\widetilde{A_{i,b}}$ after randomisation
- $\widehat{A_{i,b}}$ after encoding with GGH13 map (output of the iO)



# Mixed-input attack

## Notations

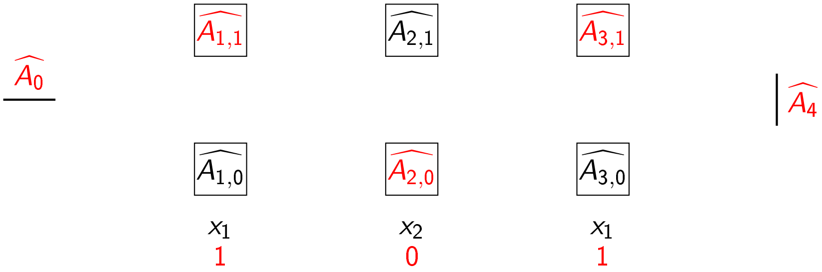
- $A_{i,b}$ input branching program
- $\widetilde{A_{i,b}}$ after randomisation
- $\widehat{A_{i,b}}$ after encoding with GGH13 map (output of the iO)



# Mixed-input attack

## Notations

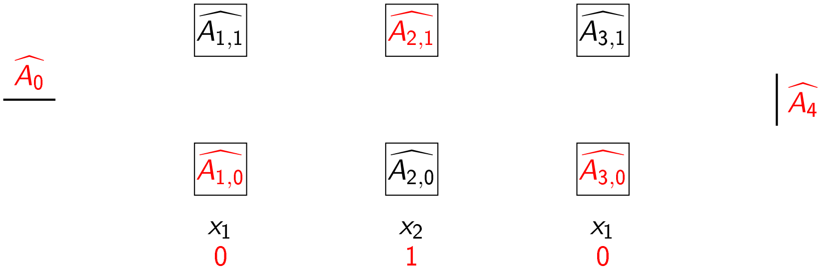
- $A_{i,b}$ input branching program
- $\widetilde{A_{i,b}}$ after randomisation
- $\widehat{A_{i,b}}$ after encoding with GGH13 map (output of the iO)



# Mixed-input attack

## Notations

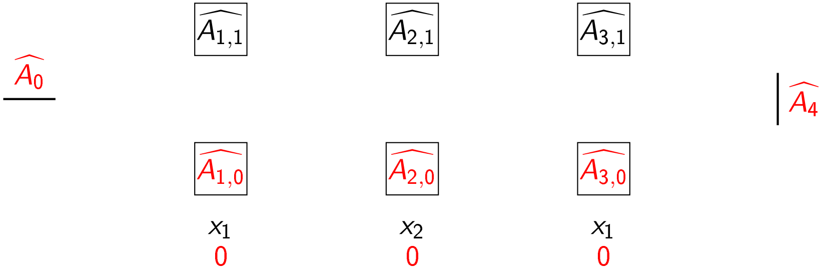
- $A_{i,b}$ input branching program
- $\widetilde{A_{i,b}}$ after randomisation
- $\widehat{A_{i,b}}$ after encoding with GGH13 map (output of the iO)



# Mixed-input attack

## Notations

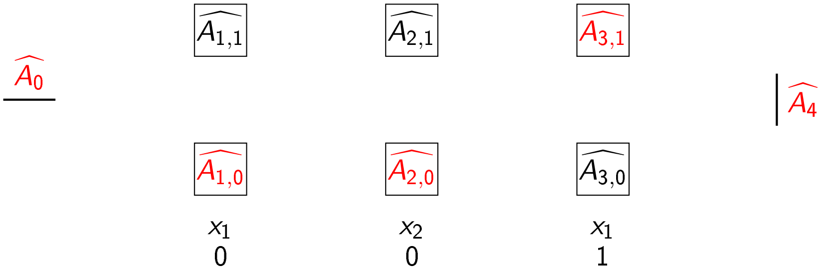
- $A_{i,b}$ input branching program
- $\widetilde{A_{i,b}}$ after randomisation
- $\widehat{A_{i,b}}$ after encoding with GGH13 map (output of the iO)



# Mixed-input attack

## Notations

- $A_{i,b}$ input branching program
- $\widetilde{A_{i,b}}$ after randomisation
- $\widehat{A_{i,b}}$ after encoding with GGH13 map (output of the iO)



# Mixed-input attack

## Notations

- $A_{i,b}$ input branching program
- $\widetilde{A_{i,b}}$ after randomisation
- $\widehat{A_{i,b}}$ after encoding with GGH13 map (output of the iO)

$$\mathrm{Enc}(\widetilde{A_0}, 1) \quad \begin{matrix} \mathrm{Enc}(\boxed{\widetilde{A_{1,1}}}, 1) & \mathrm{Enc}(\boxed{\widetilde{A_{2,1}}}, 1) & \mathrm{Enc}(\boxed{\widetilde{A_{3,1}}}, 1) \\ \mathrm{Enc}(\boxed{\widetilde{A_{1,0}}}, 1) & \mathrm{Enc}(\boxed{\widetilde{A_{2,0}}}, 1) & \mathrm{Enc}(\boxed{\widetilde{A_{3,0}}}, 1) \\ x_1 & x_2 & x_1 \\ 0 & 0 & 1 \end{matrix} \quad \Big| \, \mathrm{Enc}(\widetilde{A_4}, 1)$$

# Preventing mixed-input attacks

- In the randomization phase $\Rightarrow$ not in this talk
- Using the mmap $\Rightarrow$ straddling set system

# Preventing mixed-input attacks

- In the randomization phase $\Rightarrow$ not in this talk
- Using the mmap $\Rightarrow$ straddling set system

**Mmap degree:** $\kappa = 5$

$$\mathrm{Enc}(\widetilde{A_0}, 1) \quad \begin{matrix} \mathrm{Enc}(\boxed{\widetilde{A_{1,1}}}, 1) & \mathrm{Enc}(\boxed{\widetilde{A_{2,1}}}, 1) & \mathrm{Enc}(\boxed{\widetilde{A_{3,1}}}, 1) \\ \mathrm{Enc}(\boxed{\widetilde{A_{1,0}}}, 1) & \mathrm{Enc}(\boxed{\widetilde{A_{2,0}}}, 1) & \mathrm{Enc}(\boxed{\widetilde{A_{3,0}}}, 1) \\ x_1 & x_2 & x_1 \end{matrix} \quad \Big| \, \mathrm{Enc}(\widetilde{A_4}, 1)$$

# Preventing mixed-input attacks

- In the randomization phase $\Rightarrow$ not in this talk
- Using the mmap $\Rightarrow$ straddling set system

**Mmap degree:** $\kappa = 6$

$$\underline{\mathrm{Enc}(\widetilde{A_0}, 1)}$$

$$\mathrm{Enc}(\boxed{\widetilde{A_{1,1}}}, 1) \quad \mathrm{Enc}(\boxed{\widetilde{A_{2,1}}}, 1) \quad \mathrm{Enc}(\boxed{\widetilde{A_{3,1}}}, 2)$$

$$\mathrm{Enc}(\boxed{\widetilde{A_{1,0}}}, 2) \quad \mathrm{Enc}(\boxed{\widetilde{A_{2,0}}}, 1) \quad \mathrm{Enc}(\boxed{\widetilde{A_{3,0}}}, 1)$$

$$x_1 \qquad x_2 \qquad x_1$$

$$\Big|\, \mathrm{Enc}(\widetilde{A_4}, 1)$$

# Preventing mixed-input attacks

- In the randomization phase $\Rightarrow$ not in this talk
- Using the mmap $\Rightarrow$ straddling set system

**Mmap degree:** $\kappa = 6$

$\mathrm{Enc}(\widetilde{A_0}, 1)$

$\mathrm{Enc}(\boxed{\widetilde{A_{1,1}}}, 1)$ $\mathrm{Enc}(\boxed{\widetilde{A_{2,1}}}, 1)$ $\mathrm{Enc}(\boxed{\widetilde{A_{3,1}}}, 2)$

$\mathrm{Enc}(\boxed{\widetilde{A_{1,0}}}, 2)$ $\mathrm{Enc}(\boxed{\widetilde{A_{2,0}}}, 1)$ $\mathrm{Enc}(\boxed{\widetilde{A_{3,0}}}, 1)$

$\mathrm{Enc}(\widetilde{A_4}, 1)$

| $x_1$ | $x_2$ | $x_1$ |
|---|---|---|
| 0 | 0 | 1 |

# Preventing mixed-input attacks

- In the randomization phase ⇒ not in this talk
- Using the mmap ⇒ straddling set system

**Mmap degree:** $\kappa = 6$

$$\mathrm{Enc}(\widetilde{A_0}, 1) \quad \mathrm{Enc}(\boxed{\widetilde{A_{1,1}}}, 1) \quad \mathrm{Enc}(\boxed{\widetilde{A_{2,1}}}, 1) \quad \mathrm{Enc}(\boxed{\widetilde{A_{3,1}}}, 2) \quad \Big| \ \mathrm{Enc}(\widetilde{A_4}, 1)$$

$$\mathrm{Enc}(\boxed{\widetilde{A_{1,0}}}, 2) \quad \mathrm{Enc}(\boxed{\widetilde{A_{2,0}}}, 1) \quad \mathrm{Enc}(\boxed{\widetilde{A_{3,0}}}, 1)$$

| $x_1$ | $x_2$ | $x_1$ |
|---|---|---|
| 0 | 0 | 1 |

Total level: 7 ⇒ cannot zero-test

# What to remember

+ iO would be very useful (at least for theory) . . .

# What to remember

+ iO would be very useful (at least for theory) . . .

− . . . but no constructions from standard assumptions yet

# What to remember

+ iO would be very useful (at least for theory) ...
– ... but no constructions from standard assumptions yet
– ... even insecure constructions are very inefficient

# What to remember

- \+ iO would be very useful (at least for theory) ...
- − ... but no constructions from standard assumptions yet
- − ... even insecure constructions are very inefficient
- \+ maybe for restricted class of functions efficiency and security are possible

# What to remember

- \+ iO would be very useful (at least for theory) ...
- − ... but no constructions from standard assumptions yet
- − ... even insecure constructions are very inefficient
- \+ maybe for restricted class of functions efficiency and security are possible

Questions?

# References I


Boaz Barak, Oded Goldreich, Rusell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang.
On the (im) possibility of obfuscating programs.
In Crypto 2001, pages 1–18. Springer, 2001.

David Bruce Cousins, Giovanni Di Crescenzo, Kamil Doruk Gür, Kevin King, Yuriy Polyakov, Kurt Rohloff, Gerard W Ryan, and Erkay Savas.
Implementing conjunction obfuscation under entropic ring lwe.
In 2018 IEEE Symposium on Security and Privacy (SP), pages 354–371. IEEE, 2018.

Jean-Sébastien Coron and Hilder VL Pereira.
On kilian's randomization of multilinear map encodings.
ePrint, 2018.

Shai Halevi, Tzipora Halevi, Victor Shoup, and Noah Stephens-Davidowitz.
Implementing bp-obfuscation using graph-induced encoding.
In SIGSAC, pages 783–798. ACM, 2017.

Dingfeng Ye, Peng Liu, and Jun Xu.
How fast can we obfuscate using ideal graded encoding schemes.
ePrint, 2017.