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

1. Definition

2. Candidates
   - Security
   - Practicability

3. Example of construction of an obfuscator
What is a program?

- C/C++/Python/⋯ code;
What is a program?

- C/C++/Python/⋯ code;
- Turing machine;
What is a program?

- C/C++/Python/⋯ code;
- Turing machine;
- Boolean circuit;

![Boolean Circuit Diagram]

**Notation**

\[ C = \text{class of all polynomial size boolean circuits} \]
What is a program?

- C/C++/Python/⋯ code;
- Turing machine;
- Boolean circuit;
- Branching programs;

\[
x \lor (y \land z)
\]

Notation

\[C = \text{class of all polynomial size boolean circuits}\]
Virtual Black Box (VBB) obfuscation

Recall

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

A VBB obfuscator $\mathcal{O} : \mathcal{C} \rightarrow \mathcal{C}$ 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 $p$;
- **Virtual Black Box security**: For all PPT $A$, there exists a PPT $\text{Sim}$ such that for all $C \in \mathcal{C}$, $|\mathcal{O}[A(\mathcal{O}(C))] = 1 - \mathcal{O}[\text{Sim}(1|C|)] = 1| \leq \text{negl}$.
Virtual Black Box (VBB) obfuscation

Recall

\[ \mathcal{C} = \text{class of all polynomial size boolean circuits} \]

A VBB obfuscator \( \mathcal{O} : \mathcal{C} \rightarrow \mathcal{C} \) 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 $\mathcal{O} : \mathcal{C} \rightarrow \mathcal{C}$ 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 $p$;

Virtual Black Box (VBB) obfuscation

Recall

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

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

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

$$\left| \mathbb{P}[A(O(C)) = 1] - \mathbb{P}[\text{Sim}^C(1^{\|C\|}) = 1] \right| \leq \text{negl}.$$
Virtual Black Box (VBB) obfuscation

Recall

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

A VBB obfuscator $\mathcal{O} : \mathcal{C} \rightarrow \mathcal{C}$ 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 $p$;
- (Virtual Black Box security) For all PPT $A$, there exists a PPT $\text{Sim}$ s.t. for all $C \in \mathcal{C}$,

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

VBB obfuscation is impossible to achieve [BGI+01]

---

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**: For all $C_1, C_2 \in \mathcal{C}$ with $C_1 \equiv C_2$, $\mathcal{O}(C_1) \simeq C_2$. 

A. Pellet-Mary
Theoretical obfuscation

Fridaycon 7/22
Indistinguishability Obfuscation (iO)

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

- (Functionality) For all $C \in \mathcal{C}$, $O(C) \equiv C$;
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 $C \in \mathcal{C}$, $\mathcal{O}(C) \equiv_c \mathcal{O}(\mathcal{O}'(C))$
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 $C \in \mathcal{C}$, $\mathcal{O}(C) \sim_C \mathcal{O}(\mathcal{O}'(C))$.
- $\mathcal{O}(\mathcal{O}'(C))$ reveals less info than $\mathcal{O}'(C)$. 

A. Pellet-Mary
Theoretical obfuscation
Fridaycon 8/22
Why is iO useful (1)

iO achieves “best possible” obfuscation

Proof:

- let $O$ be an iO obfuscator and $O'$ be another obfuscator
- for any $C \in \mathcal{C}$, $O(C) \sim_c O(O'(C))$
- $O(O'(C))$ reveals less info than $O'(C)$
- $O(C)$ reveals less info than $O'(C)$
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 $C \in \mathcal{C}$, $\mathcal{O}(C) \cong_c \mathcal{O}(\mathcal{O}'(C))$
- $\mathcal{O}(\mathcal{O}'(C))$ 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, NIZKs, oblivious transfer, . . .
Why is iO useful (2)

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

**Example:** black box decryption (symmetric setting)
Why is iO useful (2)

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

Example: black box decryption (symmetric setting)

Recall: anything revealed by $O(C)$ is revealed by any $C' \equiv C$
Why is iO useful (2)

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

Example: black box decryption (symmetric setting)

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

- take $(\text{Setup}, \text{Enc}, \text{Dec})$ 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 $O(C)$ is revealed by any $C' \equiv C$

- take $(\text{Setup}, \text{Enc}, \text{Dec})$ your favourite SKE scheme
- Setup':
  - $sk_1 \leftarrow \text{Setup}()$, $sk_2 \leftarrow \text{Setup}()$
  - output $sk' = (sk_1, sk_2)$
Why is iO useful (2)
Many cryptographic constructions from iO:
functional encryption, deniable encryption, NIZKs, oblivious transfer, ...

Example: black box decryption (symmetric setting)
Recall: anything revealed by $\mathcal{O}(C)$ is revealed by any $C' \equiv C$

- take $(\text{Setup,Enc,Dec})$ your favourite SKE scheme
- Setup':
  - $sk_1 \leftarrow \text{Setup}(),$ $sk_2 \leftarrow \text{Setup}()$
  - output $sk' = (sk_1, sk_2)$
- Enc'$(m, sk')$:
  - $c_1 \leftarrow \text{Enc}(m, sk_1),$ $c_2 \leftarrow \text{Enc}(m, sk_2)$
  - output $(c_1, c_2)$
Why is iO useful (2)

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

Example: black box decryption (symmetric setting)

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

- take $(\text{Setup}, \text{Enc}, \text{Dec})$ your favourite SKE scheme
- Setup':
  - $sk_1 \leftarrow \text{Setup}()$, $sk_2 \leftarrow \text{Setup}()$
  - output $sk' = (sk_1, sk_2)$
- $\text{Enc}'(m, sk')$:
  - $c_1 \leftarrow \text{Enc}(m, sk_1)$, $c_2 \leftarrow \text{Enc}(m, sk_2)$
  - output $(c_1, c_2)$
- $\text{Dec}'$:
  - $C_1(c_1, c_2) = \text{Dec}(sk_1, c_1)$ ($sk_1$ hardcoded in $C_1$)
  - $C_2(c_1, c_2) = \text{Dec}(sk_2, c_2)$ ($sk_2$ hardcoded in $C_2$)
Many cryptographic constructions from iO: functional encryption, deniable encryption, NIKEZs, oblivious transfer, …

Example: black box decryption (symmetric setting)

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

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

$C_1 \equiv C_2 \Rightarrow C = O(C_1) \sim_c O(C_2)$ does not reveal $sk_1$ or $sk_2$
Outline of the talk

1. Definition

2. Candidates
   - Security
   - Practicability

3. Example of construction of an obfuscator
We only have candidate iO
(no construction based on standard cryptographic assumptions)
Three main categories

- Branching program obfuscators
  
- Circuit obfuscators
  
- 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
Three main categories

- Branching program obfuscators
  - needs bootstrapping via fully homomorphic encryption

- 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
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.

<table>
<thead>
<tr>
<th></th>
<th>number of candidates</th>
<th>still standing classically</th>
<th>still standing quantumly</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branching program iO</td>
<td>$\approx 20$</td>
<td>$\approx 10$</td>
<td>3</td>
</tr>
<tr>
<td>Circuit iO</td>
<td>$\approx 8$</td>
<td>$\approx 8$</td>
<td>0</td>
</tr>
</tbody>
</table>

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

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

1. Definition

2. Candidates
   - Security
   - Practicability

3. Example of construction of an obfuscator
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 $i \in \{1, \ldots, \ell\}$ and $b \in \{0, 1\}$),
- two vectors $A_0$ and $A_{\ell+1}$,
- a function $\text{inp} : \{1, \ldots, \ell\} \rightarrow \{1, \ldots, r\}$ (where $r$ is the size of the input).

---

### Example

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>inp($i$)</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

$x = 0 \ 1 \ 1$

$A_0 \quad A_{1,1} \quad A_{2,1} \quad A_{3,1} \quad A_{4,1} \quad A_{5,1} \quad A_{6,1} \quad A_7$

$A_{1,0} \quad A_{2,0} \quad A_{3,0} \quad A_{4,0} \quad A_{5,0} \quad A_{6,0}$
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

- 2ℓ 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 $\text{inp} : \{1, \ldots, \ell\} \rightarrow \{1, \ldots, r\}$ (where $r$ is the size of the input).

<table>
<thead>
<tr>
<th>$i$</th>
<th>$1$</th>
<th>$2$</th>
<th>$3$</th>
<th>$4$</th>
<th>$5$</th>
<th>$6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{inp}(i)$</td>
<td>$1$</td>
<td>$1$</td>
<td>$2$</td>
<td>$1$</td>
<td>$3$</td>
<td>$2$</td>
</tr>
</tbody>
</table>

$A_0$

\[
\begin{array}{ccccccc}
A_{1,0} & A_{1,1} & A_{2,0} & A_{2,1} & A_{3,0} & A_{3,1} & A_{4,0} & A_{4,1} & A_{5,0} & A_{5,1} & A_{6,0} & A_{6,1} & A_7 \\
\end{array}
\]

$x = 0 1 1$
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).

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>inp($i$)</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

$x = \begin{bmatrix} 0 & 1 & 1 \end{bmatrix}$

$A_0 \times A_{1,1} \quad A_{2,1} \quad A_{3,1} \quad A_{4,1} \quad A_{5,1} \quad A_{6,1} \quad A_7$
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 $\text{inp} : \{1, \ldots, \ell\} \rightarrow \{1, \ldots, r\}$ (where $r$ is the size of the input).

$$\begin{array}{c|c|c|c|c|c|c}
  i & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
  \text{inp}(i) & 1 & 1 & 2 & 1 & 3 & 2 \\
\end{array}$$

$$x = \begin{pmatrix} 0 & 1 & 1 \end{pmatrix}$$
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).

\[
\begin{array}{c|ccccccc}
 i & 1 & 2 & 3 & 4 & 5 & 6 \\
 \text{inp}(i) & 1 & 1 & 2 & 1 & 3 & 2 \\
\end{array}
\]

\[
x = \begin{bmatrix} 0 & 1 & 1 \\ \uparrow \end{bmatrix}
\]

\[
A_0 \times A_{1,1} \times A_{2,1} \times A_{3,1} = A_{4,1} A_{5,1} A_{6,1} A_7
\]
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 $\text{inp} : \{1, \ldots, \ell\} \rightarrow \{1, \ldots, r\}$ (where $r$ is the size of the input).

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>inp($i$)</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

$x = \begin{pmatrix} 0 & 1 & 1 \end{pmatrix}$

$$
A_0 \times A_{1,1} \times A_{1,0} \times A_{2,1} \times A_{2,0} \times A_{3,1} \times A_{3,0} \times A_{4,1} \times A_{4,0} \times A_{5,1} \times A_{5,0} \times A_{6,1} \times A_{6,0} \times A_7
$$
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).

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>inp($i$)</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

\[
A_0 \times A_{1,1} \times A_{2,1} \times A_{3,1} \times A_{4,1} \times A_{5,1} \times A_{6,1} \times A_{7,0}
\]
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 $i \in \{1, \ldots, \ell\}$ and $b \in \{0, 1\}$),
- two vectors $A_0$ and $A_{\ell+1}$,
- a function $\text{inp} : \{1, \ldots, \ell\} \rightarrow \{1, \ldots, r\}$ (where $r$ is the size of the input).

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{inp}(i)$</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

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

$A_0 \times A_{1,1} \times A_{2,1} \times A_{3,1} \times A_{4,1} \times A_{5,1} \times A_{6,1} \times A_{7}$
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 $i \in \{1, \ldots, \ell\}$ and $b \in \{0, 1\}$),
- two vectors $A_0$ and $A_{\ell+1}$,
- a function $\text{inp} : \{1, \ldots, \ell\} \rightarrow \{1, \ldots, r\}$ (where $r$ is the size of the input).

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{inp}(i)$</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

$A_0 \times A_{1,1} \times A_{2,1} \times A_{3,1} \times A_{4,1} \times A_{5,1} \times A_{6,1} \times A_7$
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 $i \in \{1, \ldots, \ell\}$ and $b \in \{0, 1\}$),
- two vectors $A_0$ and $A_{\ell+1}$,
- a function $\text{inp} : \{1, \ldots, \ell\} \rightarrow \{1, \ldots, r\}$ (where $r$ is the size of the input).

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{inp}(i)$</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

$$x = \begin{bmatrix} 0 & 1 & 1 \end{bmatrix}$$
Cryptographic multilinear maps

**Definition: \(\kappa\)-multilinear map**

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

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

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

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

\[
\begin{array}{ccc}
A_{0} & A_{1,1} & A_{2,1} & A_{3,1} \\
A_{1,0} & A_{2,0} & A_{3,0} \\
& & & A_{4}
\end{array}
\]
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

\[
\begin{array}{c|c}
0 & A_0 \\
\hline
B_{1,0} & \bar{A}_{1,0} \\
A_{1,0} & B_2,1 & A_{2,1} & B_3,1 & A_{3,1} \\
\end{array}
\]

\[
\begin{array}{c|c}
B_{1,1} & \bar{A}_{1,1} \\
A_{1,1} & B_2,1 & A_{2,1} & B_3,1 & A_{3,1} \\
\end{array}
\]

\[
\begin{array}{c|c}
\bar{B}_{1,0} & A_{1,0} \\
B_{2,0} & A_{2,0} & B_3,0 & A_{3,0} \\
\end{array}
\]

\[
\text{\(A_4 \star \)}
\]
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

\[
\begin{align*}
A_0 & \quad R_1 & \quad R_2 & \quad R_3 & \quad R_4 \\
R_1^{-1}A_{1,1} & \quad R_2 & \quad R_2^{-1}A_{2,1} & \quad R_3 & \quad R_3^{-1}A_{3,1} \\
R_1^{-1}A_{1,0} & \quad R_2 & \quad R_2^{-1}A_{2,0} & \quad R_3 & \quad R_3^{-1}A_{3,0} \\
R_4^{-1} & \quad A_4
\end{align*}
\]
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

\[
\begin{align*}
\alpha_{1,1} \times A_{1,1} & \quad \alpha_{2,1} \times A_{2,1} & \quad \alpha_{3,1} \times A_{3,1} \\
A_0 & \quad & \\
\alpha_{1,0} \times A_{1,0} & \quad \alpha_{2,0} \times A_{2,0} & \quad \alpha_{3,0} \times A_{3,0} \\
& & \quad A_4
\end{align*}
\]
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

\[
\begin{align*}
\widetilde{A}_0 & \quad \widetilde{A}_{1,1} & \quad \widetilde{A}_{2,1} & \quad \widetilde{A}_{3,1} \\
\widetilde{A}_{1,0} & \quad \widetilde{A}_{2,0} & \quad \widetilde{A}_{3,0} & \quad \widetilde{A}_4
\end{align*}
\]
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

\[
\text{Enc}(\overline{A}_0) \quad \text{Enc}(\overline{A}_1,0) \quad \text{Enc}(\overline{A}_1,1) \quad \text{Enc}(\overline{A}_2,0) \quad \text{Enc}(\overline{A}_2,1) \quad \text{Enc}(\overline{A}_3,0) \quad \text{Enc}(\overline{A}_3,1) \quad \text{Enc}(\overline{A}_4)
\]
Mixed-input attack

**Notations**

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

\[
\begin{align*}
\widehat{A}_0 & \\
\widehat{A}_{1,0} & & \widehat{A}_{2,0} & & \widehat{A}_{3,0} \\
& & \widehat{A}_{1,1} & & \widehat{A}_{2,1} & & \widehat{A}_{3,1} & & \widehat{A}_4 \\
x_1 & & x_2 & & x_1
\end{align*}
\]
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)

\[
\begin{align*}
\widehat{A}_0 & \quad \widehat{A}_{1,1} & \quad \widehat{A}_{2,1} & \quad \widehat{A}_{3,1} \\
\widehat{A}_{1,0} & \quad \widehat{A}_{2,0} & \quad \widehat{A}_{3,0} & \quad \widehat{A}_4 \\
\chi_1 & \quad \chi_2 & \quad \chi_1 & \quad 1 \\
1 & \quad 1 & \quad 1
\end{align*}
\]
Mixed-input attack

Notations

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

\[
\begin{align*}
\hat{A}_0 &\quad \hat{A}_{1,1} &\quad \hat{A}_{2,1} &\quad \hat{A}_{3,1} &\quad \hat{A}_4 \\
\hline
\hat{A}_{1,0} & A_{1,0} & A_{2,0} & A_{3,0} & A_{4,0} \\
\hline
x_1 & 1 & x_2 & 0 & x_1 & 1
\end{align*}
\]
Mixed-input attack

Notations

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

\[ \hat{A}_{0}, \hat{A}_{1,0}, \hat{A}_{2,0}, \hat{A}_{3,0}, \hat{A}_{1,1}, \hat{A}_{2,1}, \hat{A}_{3,1}, \hat{A}_{4} \]
### Mixed-input attack

#### Notations
- $A_{i,b}$ input branching program
- $\tilde{A}_{i,b}$ after randomisation
- $\hat{A}_{i,b}$ after encoding with GGH13 map (output of the iO)

\[
\begin{align*}
\hat{A}_0 & \quad \hat{A}_{1,1} & \quad \hat{A}_{2,1} & \quad \hat{A}_{3,1} \\
\hat{A}_{1,0} & \quad \hat{A}_{2,0} & \quad \hat{A}_{3,0} & \quad \hat{A}_4
\end{align*}
\]

$X_1$ 0  $X_2$ 0  $X_1$ 0
Mixed-input attack

Notations

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

\[
\begin{align*}
\hat{A}_0 & \quad \hat{A}_1, 1 \quad \hat{A}_2, 1 \\
\quad \quad \hat{A}_1, 0 \quad \hat{A}_2, 0 \quad \hat{A}_3, 0 \\
\quad \quad \quad \quad \hat{A}_3, 1 \quad \hat{A}_4
\end{align*}
\]
Mixed-input attack

Notations

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

\[
\begin{align*}
\text{Enc}(\sim A_{1,1}, 1) & \quad \text{Enc}(\sim A_{2,1}, 1) & \quad \text{Enc}(\sim A_{3,1}, 1) \\
\text{Enc}(\sim A_{0,1}) & \quad \text{Enc}(\sim A_{1,0}, 1) & \quad \text{Enc}(\sim A_{2,0}, 1) & \quad \text{Enc}(\sim A_{3,0}, 1) \\
\begin{array}{c}
\chi_1 \\
0
\end{array} & \begin{array}{c}
\chi_2 \\
0
\end{array} & \begin{array}{c}
\chi_1 \\
1
\end{array}
\end{align*}
\]
Preventing mixed-input attacks

- In the randomization phase ⇒ not in this talk
- Using the mmap ⇒ straddling set system
Preventing mixed-input attacks

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

Mmap degree: $\kappa = 5$

\[
\begin{align*}
\text{Enc}(\tilde{A}_0, 1) & \quad \text{Enc}(\tilde{A}_1, 1) & \quad \text{Enc}(\tilde{A}_2, 1) & \quad \text{Enc}(\tilde{A}_3, 1) \\
& \quad \text{Enc}(\tilde{A}_4, 1) \\
\end{align*}
\]

\[
\begin{align*}
\text{Enc}(\tilde{A}_1, 0) & \quad \text{Enc}(\tilde{A}_2, 0) & \quad \text{Enc}(\tilde{A}_3, 0) \\
& \quad x_1 & \quad x_2 & \quad x_1
\end{align*}
\]
Preventing mixed-input attacks

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

Mmap degree: $\kappa = 6$

\[
\begin{align*}
\text{Enc}(\overline{A_0}, 1) & \quad \text{Enc}(\overline{A_1}, 1, 2) \quad \text{Enc}(\overline{A_2}, 1, 1) \\
\text{Enc}(\overline{A_0}, 1) & \quad \text{Enc}(\overline{A_1}, 1) \quad \text{Enc}(\overline{A_2}, 1) \\
\text{Enc}(\overline{A_1}, 2) & \quad \text{Enc}(\overline{A_2}, 0, 1) \quad \text{Enc}(\overline{A_3}, 0, 1)
\end{align*}
\]
Preventing mixed-input attacks

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

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

\[
\begin{align*}
\text{Enc}(\widetilde{A}_1, 1) & \quad \text{Enc}(\widetilde{A}_2, 1) & \quad \text{Enc}(\widetilde{A}_3, 2) \\
\text{Enc}(\widetilde{A}_0, 1) & \quad \text{Enc}(\widetilde{A}_1, 2) & \quad \text{Enc}(\widetilde{A}_2, 1) & \quad \text{Enc}(\widetilde{A}_3, 1) \\
& \quad \text{Enc}(\widetilde{A}_4, 1) \\
\end{align*}
\]

\[
\begin{align*}
x_1 & \quad x_2 & \quad x_1 \\
0 & \quad 0 & \quad 1
\end{align*}
\]
Preventing mixed-input attacks

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

Mmap degree: $\kappa = 6$

\[
\begin{align*}
\text{Enc}(\widetilde{A}_0, 1) & \quad \text{Enc}(\widetilde{A}_{1,1}, 1) & \quad \text{Enc}(\widetilde{A}_{2,1}, 1) & \quad \text{Enc}(\widetilde{A}_{3,1}, 2) \\
\quad \text{Enc}(\widetilde{A}_{1,0}, 2) & \quad \text{Enc}(\widetilde{A}_{2,0}, 1) & \quad \text{Enc}(\widetilde{A}_{3,0}, 1) & \quad \text{Enc}(\widetilde{A}_4, 1)
\end{align*}
\]

\[
\begin{array}{ccc}
X_1 & X_2 & X_1 \\
0 & 0 & 1
\end{array}
\]

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?


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


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