Quantum attack against some candidate obfuscators based on GGH13

Alice Pellet-Mary

LIP, ENS de Lyon

Séminaire C2
November 16, 2018
What is this talk about

Quantum attack against some candidate obfuscators built upon the GGH13 multilinear map [GGH13a]

What is this talk about

Quantum attack against some candidate obfuscators built upon the GGH13 multilinear map [GGH13a]

► GGH13 is known to be weak in quantum world

What is this talk about

Quantum attack against some candidate obfuscators built upon the GGH13 multilinear map [GGH13a]

- GGH13 is known to be weak in quantum world
- Transform this weakness into concrete attack on obfuscators

What is this talk about

Quantum attack against some candidate obfuscators built upon the GGH13 multilinear map [GGH13a]

- GGH13 is known to be weak in quantum world
- Transform this weakness into concrete attack on obfuscators
- Nothing quantum in this talk

Obfuscation

**Obfuscator**

An obfuscator $O$ for a class of circuits $C$ is an efficiently computable function over $C$ such that

$$\forall C \in C, \forall x, C(x) = O(C)(x)$$

In this talk, $C =$ polynomial size circuits
Obfuscation

Obfuscator

An obfuscator $O$ for a class of circuits $C$ is an efficiently computable function over $C$ such that

$$\forall C \in C, \forall x, C(x) = O(C)(x)$$

In this talk, $C = \text{polynomial size circuits}$

Security.

- VBB: $O(C)$ acts as a black box computing $C$
Obfuscation

Obfuscator

An obfuscator $O$ for a class of circuits $C$ is an efficiently computable function over $C$ such that

$$\forall C \in C, \forall x, C(x) = O(C)(x)$$

In this talk, $C = \text{polynomial size circuits}$

Security.

- $\forall \text{BB}: O(C) \text{ acts as a black box computing } C \text{ (impossible, [BGI+01])}$

---

Obfuscation

Obfuscator

An obfuscator $O$ for a class of circuits $\mathcal{C}$ is an efficiently computable function over $\mathcal{C}$ such that

$$\forall C \in \mathcal{C}, \forall x, C(x) = O(C)(x)$$

In this talk, $\mathcal{C} =$ polynomial size circuits

Security.

- VBB: $O(C)$ acts as a black box computing $C$ (impossible, [BGI+01])
- iO: $\forall C_1 \equiv C_2$, i.e. $C_1(x) = C_2(x) \ \forall x$, 
  $$O(C_1) \simeq_c O(C_2)$$

---

Why is iO interesting?

1. iO achieves “best possible” obfuscation
Why is iO interesting?

1. iO achieves “best possible” obfuscation

   **Proof:**
   - let $O$ be an iO obfuscator and $O'$ be another obfuscator
Why is iO interesting?

1. iO achieves “best possible” obfuscation

Proof:

- let \( O \) be an iO obfuscator and \( O' \) be another obfuscator
- for any \( C \in \mathcal{C} \), \( O(C) \approx_{c} O(O'(C)) \)
Why is iO interesting?

1. iO achieves “best possible” obfuscation

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

1. iO achieves “best possible” obfuscation

Proof:
- let \( O \) be an iO obfuscator and \( O' \) be another obfuscator
- for any \( C \in \mathcal{C} \), \( O(C) \cong_c O(O'(C)) \)
- \( O(O'(C)) \) reveals less info than \( O'(C) \)
- \( O(C) \) reveals less info than \( O'(C) \)
Why is iO interesting?

1. iO achieves “best possible” obfuscation

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

2. Many cryptographic constructions from iO: functional encryption, deniable encryption, NIKZs, oblivious transfer, ...
Multilinear maps (mmaps) and iO

Observation
Almost all iO constructions for all circuits rely on multilinear maps (mmaps)

Three main candidate multilinear maps: GGH13, CLT13, GGH15
Multilinear maps (mmaps) and iO

Observation
Almost all iO constructions for all circuits rely on multilinear maps (mmaps)

Three main candidate multilinear maps: GGH13, CLT13, GGH15

Caution
All these candidate multilinear maps suffer from weaknesses (e.g. encodings of zero, zeroizing attacks, ...).
⇒ all current attacks against iO rely on the underlying mmap
Multilinear maps (mmaps) and iO

**Observation**

Almost all iO constructions for all circuits rely on multilinear maps (mmaps)

Three main candidate multilinear maps: **GGH13, CLT13, GGH15**

**Caution**

All these candidate multilinear maps suffer from weaknesses (e.g. encodings of zero, zeroizing attacks,...).

⇒ all current attacks against iO rely on the underlying mmap

**In this talk:** we exploit known weakness of GGH13 to mount concrete attacks against some iO using it.
History (branching program obfuscators based on GGH13)

Some candidate iO for all circuits and attacks:
History (branching program obfuscators based on GGH13)

Some candidate iO for all circuits and attacks:

2013: [GGH$^{+}13b$], first candidate

2014-2016: [AGIS14, BGK$^{+}14$, BR14, MSW14, PST14, BMSZ16], with proofs in idealized models (the mmap is supposed to be somehow ideal)

2016: [MSZ16], attack against all candidates above except [GGH$^{+}13b$]

2016: [GMM$^{+}16$], proof in a weaker idealized model (captures [MSZ16])

2017: [CGH17], attack against [GGH$^{+}13b$], in input-partitionable case

2017: [FRS17], prevent [CGH17] attack

2018: [CHKL18], attack against all obfuscators, for specific choices of parameters

A. Pellet-Mary
Quantum attack against some iO
Séminaire C2 6/20
History (branching program obfuscators based on GGH13)

Some candidate iO for all circuits and attacks:

2013: [GGH$^{+}13b$], first candidate

2014-2016: [AGIS14, BGK$^{+}14$, BR14, MSW14, PST14, BMSZ16], with proofs in idealized models (the mmap is supposed to be somehow ideal)

2016: [MSZ16], attack against all candidates above except [GGH$^{+}13b$]

2016: [GMM$^{+}16$], proof in a weaker idealized model (captures [MSZ16])
History (branching program obfuscators based on GGH13)

Some candidate iO for all circuits and attacks:

2013: [GGH\textsuperscript{+}13b], first candidate

2014-2016: [AG+S14, BGK\textsuperscript{+}14, BR14, MSW14, PST14, BMSZ16], with proofs in idealized models (the mmap is supposed to be somehow ideal)

2016: [MSZ16], attack against all candidates above except [GGH\textsuperscript{+}13b]

2016: [GMM\textsuperscript{+}16], proof in a weaker idealized model (captures [MSZ16])

2017: [CGH17], attack against [GGH\textsuperscript{+}13b], in input-partitionable case

2017: [FRS17], prevent [CGH17] attack
History (branching program obfuscators based on GGH13)

Some candidate iO for all circuits and attacks:

2013: [GGH$^+13b$], first candidate

2014-2016: [AGIS14, BGK$^+14$, BR14, MSW14, PST14, BMSZ16], with proofs in idealized models (the mmap is supposed to be somehow ideal)

2016: [MSZ16], attack against all candidates above except [GGH$^+13b$]

2016: [GMM$^+16$], proof in a weaker idealized model (captures [MSZ16])

2017: [CGH17], attack against [GGH$^+13b$], in input-partitionable case

2017: [FRS17], prevent [CGH17] attack

2018: [CHKL18], attack against all obfuscators, for specific choices of parameters
## State of the art and contribution

<table>
<thead>
<tr>
<th>Attacks</th>
<th>Branching program obfuscators</th>
<th>Circuit obfuscators</th>
</tr>
</thead>
<tbody>
<tr>
<td>iO (using GGH13)</td>
<td>[GGH⁺13b] [BR14] [AGIS14, MSW14] [PST14, BGK⁺14] [GMM⁺16]</td>
<td>[Zim15, AB15] [DGG⁺16]</td>
</tr>
<tr>
<td>[MSZ16]</td>
<td>✓ ✓ ✓</td>
<td></td>
</tr>
<tr>
<td>[CGH17]*</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>[CHKL18]†</td>
<td>✓ ✓ ✓ ✓</td>
<td></td>
</tr>
<tr>
<td>This talk‡</td>
<td>✓ ✓ ✓ ✓</td>
<td></td>
</tr>
</tbody>
</table>

* for input-partitionable branching programs  ‡ in the quantum setting  † for specific choices of parameters
# State of the art and contribution

<table>
<thead>
<tr>
<th>iO (using GGH13) Attacks</th>
<th>Branching program obfuscators</th>
<th>Circuit obfuscators</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[GGH$^{+13b}$]</td>
<td>[AGIS14, MSW14]</td>
</tr>
<tr>
<td></td>
<td>[BR14]</td>
<td>[PST14, BGK$^{+14}$]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[BMSZ16]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[GMM$^{+16}$]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[Zim15, AB15]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[DGG$^{+16}$]</td>
</tr>
</tbody>
</table>

- **[MSZ16]**
  - ✓
  - ✓

- **[CGH17]**
  - ✓

- **[CHKL18]**
  - ✓
  - ✓
  - ✓

- **This talk**
  - ✓
  - ✓
  - ✓

- * for input-partitionable branching programs
- † for specific choices of parameters
- ‡ in the quantum setting
Outline of the talk

1. Simple obfuscator

2. The attack
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 $\text{inp}: \{1, \ldots, \ell\} \to \{1, \ldots, r\}$ (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>$\text{inp}(i)$</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

$x = 0 \ 1 \ 1$

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

$A_{1,0} \ A_{2,0} \ A_{3,0} \ A_{4,0} \ A_{5,0} \ A_{6,0}$
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 $i \in \{1, \ldots, \ell\}$ 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>

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

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

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

<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>

$x = 0 \quad 1 \quad 1$

$A_0 \times A_{1,1} \quad A_2,1 \quad A_{3,1} \quad A_{4,1} \quad A_{5,1} \quad A_{6,1} \quad A_7$

$A_1,0 \quad A_{2,0} \quad A_{3,0} \quad A_{4,0} \quad A_{5,0} \quad A_{6,0}$
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

- $2\ell$ 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).

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

\[
A_0 \times A_{1,1} \times A_{2,1} \times A_{3,1} \times A_{4,1} \times A_{5,1} \times A_{6,1} \times A_7
\]

\[
x = \begin{pmatrix} 0 & 1 & 1 \end{pmatrix}
\]
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 $\text{inp} : \{1, \ldots, \ell\} \rightarrow \{1, \ldots, r\}$ (where $r$ is the size of the input).

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

$$
\begin{array}{cccccccc}
  A_0 & \times & A_{1,1} & \times & A_{2,1} & \times & A_{3,1} & A_{4,1} & A_{5,1} & A_{6,1} & A_7 \\
& & A_{1,0} & & A_{2,0} & & A_{3,0} & A_{4,0} & A_{5,0} & A_{6,0} & \\
x & = & 0 & 1 & 1 &
\end{array}
$$
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>

$x = 0 \ 1 \ 1$

\[
A_0 \times A_{1,1} \times A_{2,1} \times A_{3,1} \times A_{4,1} \quad A_{5,1} \quad A_{6,1} \quad A_7
\]

\[
A_{1,0} \times A_{2,0} \times A_{3,0} \times A_{4,0} \quad A_{5,0} \quad A_{6,0}
\]
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

- $2\ell$ 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).

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

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

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

\[
x = 0 \quad 1 \quad 1 \\
\uparrow
\]

\[
A_0 \times A_{1,1} \times A_{2,1} \times A_{3,1} \times A_{4,1} \times A_{5,1} \times A_{6,1} \times A_7
\]
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 $\text{inp} : \{1, \ldots, \ell\} \to \{1, \ldots, r\}$ (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>$\text{inp}(i)$</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

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

$x = 0 \ 1 \ 1$
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 $\text{inp} : \{1, \ldots, \ell\} \rightarrow \{1, \ldots, r\}$ (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>

$x = 0 \ 1 \ 1$

$$A_0 \times A_{1,1} \times A_{2,1} \times A_{3,1} \times A_{4,1} \times A_{5,1} \times A_{6,1} \times A_7 = 0 \rightarrow 0 \neq 0 \rightarrow 1$$
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

\[ A_0 \]

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

\[ A_4 \]
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

\[
\begin{align*}
B_{1,1} & \quad | & \quad B_{2,1} & \quad | & \quad B_{3,1} \\
A_{1,1} & \quad | & \quad A_{2,1} & \quad | & \quad A_{3,1} \\
B_{1,0} & \quad | & \quad B_{2,0} & \quad | & \quad B_{3,0} \\
A_{1,0} & \quad | & \quad A_{2,0} & \quad | & \quad A_{3,0} \\
\end{align*}
\]

\[
\begin{align*}
0 & \quad | & \quad A_0 & \quad | & \quad * & \quad | & \quad A_4 \\
\end{align*}
\]
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

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

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

\[ \text{Enc}(\tilde{A}_0), \text{Enc}(\tilde{A}_{1,0}), \text{Enc}(\tilde{A}_1,1), \text{Enc}(\tilde{A}_2,1), \text{Enc}(\tilde{A}_3,1), \text{Enc}(\tilde{A}_4) \]
Outline of the talk

1. Simple obfuscator

2. The attack
Reminder: $\kappa$-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 $a = 0$. 
The GGH13 map

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

**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 $a = 0 \mod p$. 
GGH13 in a quantum world

The GGH13 map

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

**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 $a = 0 \mod p$.

With a quantum computer

$\text{Double-zero-test}(\text{Enc}(a, 2\kappa)) = \text{True}$ iff $a = 0 \mod p^2$.
Mixed-input attack

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

$\hat{A}_0$

$\hat{A}_{1,1}$  $\hat{A}_{2,1}$  $\hat{A}_{3,1}$

$\hat{A}_{1,0}$  $\hat{A}_{2,0}$  $\hat{A}_{3,0}$

$x_1$  $x_2$  $x_1$
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)
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)
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)
**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)

\[
\begin{align*}
\hat{A}_0 & \quad \hat{A}_{1,1} & \quad \hat{A}_2,1 & \quad \hat{A}_3,1 \\
\hat{A}_{1,0} & \quad \hat{A}_{2,0} & \quad \hat{A}_3,0 & \quad \hat{A}_4 \\
#1 & \quad #2 & \quad #3 & \quad #4
\end{align*}
\]

\[
\begin{align*}
x_1 & \quad 0 \\
x_2 & \quad 0 \\
x_1 & \quad 0
\end{align*}
\]
Mixed-input attack

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

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

---

$$
\begin{align*}
\tilde{A}_0 & \quad \tilde{A}_{1,1} & \quad \tilde{A}_{2,1} & \quad \tilde{A}_{3,1} & \quad \tilde{A}_4 \\
A_{1,0} & \quad A_{2,0} & \quad A_{3,0} \\
\chi_1 & \quad \chi_2 & \quad \chi_1 \\
0 & \quad 0 & \quad 1
\end{align*}
$$
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)

\[ \begin{array}{ccc}
\begin{array}{c}
B_{1,1} \\
A_{1,1}
\end{array} & \begin{array}{c}
B_{2,1} \\
A_{2,1}
\end{array} & \begin{array}{c}
B_{3,1} \\
A_{3,1}
\end{array} \\
\begin{array}{c}
B_{1,0} \\
A_{1,0}
\end{array} & \begin{array}{c}
B_{2,0} \\
A_{2,0}
\end{array} & \begin{array}{c}
B_{3,0} \\
A_{3,0}
\end{array} \\
\begin{array}{c}
x_1 \\
0
\end{array} & \begin{array}{c}
x_2 \\
0
\end{array} & \begin{array}{c}
x_1 \\
1
\end{array}
\end{array} \]
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)

\[ A_0 \]

\[
\begin{array}{c|c|c}
R_1^{-1} & A_{1,1} & R_2 \\
R_1^{-1} & A_{1,0} & R_2 \\
\end{array}
\quad
\begin{array}{c|c|c}
R_2^{-1} & A_{2,1} & R_3 \\
R_2^{-1} & A_{2,0} & R_3 \\
\end{array}
\quad
\begin{array}{c|c|c}
R_3^{-1} & A_{3,1} & R_4 \\
R_3^{-1} & A_{3,0} & R_4 \\
\end{array}
\quad
\begin{array}{c|c}
R_4^{-1} & A_4 \\
\end{array}
\]

\[ x_1 \quad 0 \quad x_2 \quad 0 \quad x_1 \quad 1 \]
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)

\[
\begin{align*}
A_0 & \quad \alpha_{1,1} \times A_{1,1} & \quad \alpha_{2,1} \times A_{2,1} & \quad \alpha_{3,1} \times A_{3,1} \\
\alpha_{1,0} \times A_{1,0} & \quad \alpha_{2,0} \times A_{2,0} & \quad \alpha_{3,0} \times A_{3,0} \\
\chi_1 & \quad 0 & \quad x_2 & \quad 0 & \quad x_1 & \quad 1
\end{align*}
\]
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)
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)

\[
\begin{align*}
\text{Enc}(\tilde{A}_0, 1) & \quad \text{Enc}(\tilde{A}_1, 1) & \quad \text{Enc}(\tilde{A}_2, 1) & \quad \text{Enc}(\tilde{A}_3, 1) & \quad \text{Enc}(\tilde{A}_4, 1) \\
\text{Enc}(\hat{A}_0, 1) & \quad \text{Enc}(\hat{A}_1, 1) & \quad \text{Enc}(\hat{A}_2, 1) & \quad \text{Enc}(\hat{A}_3, 1) & \quad \text{Enc}(\hat{A}_4, 1) \\
\chi_1 & \quad \chi_2 & \quad \chi_1 & \quad 0 & \quad 0 & \quad 1
\end{align*}
\]
Preventing mixed-input attacks

- In the randomization phase ⇒ not in this talk
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:** $\kappa = 5$

\[
\begin{align*}
\text{Enc}(\overline{A_0}, 1) & \quad \text{Enc}(\overline{A_1,1}, 1) & \quad \text{Enc}(\overline{A_2,1}, 1) & \quad \text{Enc}(\overline{A_3,1}, 1) \\
\text{Enc}(\overline{A_1,0}, 1) & \quad \text{Enc}(\overline{A_2,0}, 1) & \quad \text{Enc}(\overline{A_3,0}, 1) \\
& \quad x_1 & \quad x_2 & \quad x_1
\end{align*}
\]
Preventing mixed-input attacks

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

Mmap degree: $\kappa = 6$

\[
\begin{align*}
Enc(\widetilde{A}_0, 1) & \quad Enc(\widetilde{A}_1, 1) & \quad Enc(\widetilde{A}_2, 1) & \quad Enc(\widetilde{A}_3, 1) & \quad Enc(\widetilde{A}_4, 1) \\
Enc(\widetilde{A}_1, 0) & \quad Enc(\widetilde{A}_2, 0) & \quad Enc(\widetilde{A}_3, 0) & \quad Enc(\widetilde{A}_4, 1) \\
& \quad x_1 & \quad x_2 & \quad x_1
\end{align*}
\]
Preventing mixed-input attacks

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

Mmap degree: \( \kappa = 6 \)

\[
\begin{array}{ccc}
\text{Enc}(\overline{A_0},1) & \text{Enc}(\overline{A_1,1},1) & \text{Enc}(\overline{A_2,1},1) & \text{Enc}(\overline{A_3,1},2) \\
\text{Enc}(\overline{A_1,0},2) & \text{Enc}(\overline{A_2,0},1) & \text{Enc}(\overline{A_3,0},1) & \text{Enc}(\overline{A_4},1) \\
\end{array}
\]

\[
\begin{array}{c}
x_1 \\
0 \\
\end{array} \quad \begin{array}{c}
x_2 \\
0 \\
\end{array} \quad \begin{array}{c}
x_1 \\
1 \\
\end{array}
\]
Preventing mixed-input attacks

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

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

\[
\begin{align*}
&\text{Enc}(\overline{A_1}, 1) \quad \text{Enc}(\overline{A_2}, 1) \quad \text{Enc}(\overline{A_3}, 2) \\
&\text{Enc}(\overline{A_0}, 1) \\
&\text{Enc}(\overline{A_1}, 2) \quad \text{Enc}(\overline{A_2}, 1) \quad \text{Enc}(\overline{A_3}, 1) \\
&x_1 \quad x_2 \quad x_1 \\
&0 \quad 0 \quad 1
\end{align*}
\]

Total level: 7 ⇒ cannot zero-test
Attack idea: double mixed input

Reminder

In quantum world, we have

\[
\text{Double-zero-test(Enc}(a, 2\kappa)) = \text{True iff } a = 0 \mod p^2
\]
Attack idea: double mixed input

Reminder

In quantum world, we have

$$\text{Double-zero-test}(\text{Enc}(a, 2\kappa)) = \text{True} \iff a = 0 \mod p^2$$

\[
\begin{align*}
\text{Enc}(\overline{A_0}, 1) & \quad \text{Enc}(\overline{A_1,1}, 1) & \quad \text{Enc}(\overline{A_2,1}, 1) & \quad \text{Enc}(\overline{A_3,1}, 2) \\
\text{Enc}(\overline{A_1,0}, 2) & \quad \text{Enc}(\overline{A_2,0}, 1) & \quad \text{Enc}(\overline{A_3,0}, 1) & \quad \text{Enc}(\overline{A_4}, 1) \Rightarrow \text{Level 7}
\end{align*}
\]
Attack idea: double mixed input

Reminder

In quantum world, we have

\[
\text{Double-zero-test}(\text{Enc}(a, 2\kappa)) = \text{True iff } a = 0 \mod p^2
\]
Attack idea: double mixed input

Reminder

In quantum world, we have

\[
\text{Double-zero-test}(\text{Enc}(a, 2\kappa)) = \text{True} \iff a = 0 \mod p^2
\]
Attack idea: double mixed input

Reminder

In quantum world, we have

$$\text{Double-zero-test}(\text{Enc}(a, 2\kappa)) = \text{True} \iff a = 0 \mod p^2$$
Objective: Find $C_1 \equiv C_2$ s.t. double mixed input product is 0 on $C_1$ and $\neq 0$ on $C_2$, e.g. the two mixed-input are 0 mod 2 for $C_1 \Rightarrow$ product is 0 mod 2 the two mixed-input are $\neq 0$ mod 2 for $C_2 \Rightarrow$ product is $\neq 0$ mod 2
Reminder: iO

\[ \forall C_1 \equiv C_2, \quad O(C_1) \simeq_c O(C_2) \]

Objective: Find \( C_1 \equiv C_2 \) s.t. double mixed input product is 0 on \( C_1 \) and \( \neq 0 \) on \( C_2 \), e.g.

- the two mixed-input are 0 \( \text{ mod } p \) for \( C_1 \)
  \[ \Rightarrow \text{ product is } 0 \text{ \( \text{ mod } p^2 \)} \]

- the two mixed-input are \( \neq 0 \) \( \text{ mod } p \) for \( C_2 \)
  \[ \Rightarrow \text{ product is } \neq 0 \text{ \( \text{ mod } p^2 \)} \]
One example of $C_1$ and $C_2$

$C_1$: $(1 \ 0) \quad (1 \ 0) \quad (1 \ 0)
\quad (1 \ 0) \quad (1 \ 0) \quad (1 \ 0)
\quad (1 \ 0) \quad (1 \ 0) \quad (1 \ 0)
\quad (0 \ 1) \quad (0 \ 1) \quad (0 \ 1)
\quad x_1 \quad x_2 \quad x_1

$\Rightarrow \forall x, \ C_1(x) = 0$
One example of $C_1$ and $C_2$

$C_1$: \[
\begin{pmatrix}
1 & 0 \\
0 & 1
\end{pmatrix}
\begin{pmatrix}
1 & 0 \\
0 & 1
\end{pmatrix}
\begin{pmatrix}
1 & 0 \\
0 & 1
\end{pmatrix}
\begin{pmatrix}
0 \\
1
\end{pmatrix} \Rightarrow \forall x, C_1(x) = 0
\]

$C_2$: \[
\begin{pmatrix}
1 & 0 \\
0 & 1
\end{pmatrix}
\begin{pmatrix}
1 & 0 \\
0 & 1
\end{pmatrix}
\begin{pmatrix}
0 & 1 \\
1 & 0
\end{pmatrix}
\begin{pmatrix}
0 \\
1
\end{pmatrix} \Rightarrow \forall x, C_2(x) = 0
\]
One example of $C_1$ and $C_2$

$C_1$: $(1 \ 0) \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \Rightarrow \forall x, \ C_1(x) = 0$

$C_2$: $(1 \ 0) \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \Rightarrow \forall x, \ C_2(x) = 0$

• $C_1 \equiv C_2$
One example of $C_1$ and $C_2$

$C_1$: \[
\begin{pmatrix}
1 & 0 \\
0 & 1
\end{pmatrix}
\begin{pmatrix}
1 & 0 \\
0 & 1
\end{pmatrix}
\begin{pmatrix}
1 & 0 \\
0 & 1
\end{pmatrix}
\]
\[x_1 \ x_2 \ x_1\]

$C_2$: \[
\begin{pmatrix}
1 & 0 \\
0 & 1
\end{pmatrix}
\begin{pmatrix}
1 & 0 \\
0 & 1
\end{pmatrix}
\begin{pmatrix}
1 & 0 \\
0 & 1
\end{pmatrix}
\]
\[x_1 \ x_2 \ x_1\]

\[
\begin{pmatrix}
0 \\
1
\end{pmatrix} \Rightarrow \forall x, \ C_1(x) = 0
\]

\[
\begin{pmatrix}
0 \\
1
\end{pmatrix} \Rightarrow \forall x, \ C_2(x) = 0
\]

- $C_1 \equiv C_2$
- the two mixed-input products are 0 for $C_1$
One example of $C_1$ and $C_2$

$$
C_1: \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \Rightarrow \forall x, C_1(x) = 0
$$

$$
C_2: \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \Rightarrow \forall x, C_2(x) = 0
$$

- $C_1 \equiv C_2$
- The two mixed-input products are 0 for $C_1$
- The two mixed-input products are $\neq 0$ for $C_2$
One example of $C_1$ and $C_2$

$C_1$: \[
\begin{pmatrix}
1 & 0 \\
0 & 1
\end{pmatrix},
\begin{pmatrix}
1 & 0 \\
0 & 1
\end{pmatrix},
\begin{pmatrix}
0 & 1 \\
0 & 1
\end{pmatrix}
\]
\[
\Rightarrow \forall x, \ C_1(x) = 0
\]

$C_2$: \[
\begin{pmatrix}
1 & 0 \\
0 & 1
\end{pmatrix},
\begin{pmatrix}
1 & 0 \\
0 & 1
\end{pmatrix},
\begin{pmatrix}
0 & 1 \\
0 & 1
\end{pmatrix}
\]
\[
\Rightarrow \forall x, \ C_2(x) = 0
\]

- $C_1 \equiv C_2$
- the two mixed-input products are 0 for $C_1$
- the two mixed-input products are $\neq 0$ for $C_2$

We can distinguish $O(C_1)$ from $O(C_2)$
Conclusion (1/2)

Counter-intuitive remark

This attack works only against the recent schemes (with stronger security proofs)
Conclusion (1/2)

Counter-intuitive remark

This attack works only against the recent schemes (with stronger security proofs)

Why?

- Previous schemes prevent mixed-input attack using the randomization phase
  - difficult to get a security proof
Conclusion (1/2)

Counter-intuitive remark

This attack works only against the recent schemes (with stronger security proofs)

Why?

- Previous schemes prevent mixed-input attack using the randomization phase
  - difficult to get a security proof
- New schemes use the mmap
  - easy to get a proof (in idealized model)
Conclusion (1/2)

Counter-intuitive remark
This attack works only against the recent schemes (with stronger security proofs)

Why?

- Previous schemes prevent mixed-input attack using the randomization phase
  - difficult to get a security proof

- New schemes use the mmap
  - easy to get a proof (in idealized model)

- GGH13 mmap is not ideal
  - easier for an attacker to exploit its weakness
Conclusion (2/2)

Remarks

- Quantum poly time or classical $2^{O(\sqrt{n})}$ time
Conclusion (2/2)

Remarks

- Quantum poly time or classical $2^{O(\sqrt{n})}$ time
- Double mixed input attacks can be extended to circuit obfuscators
## Conclusion (2/2)

<table>
<thead>
<tr>
<th>Attacks</th>
<th>iO (using GGH13)</th>
<th>Branching program obfuscators</th>
<th>Circuit obfuscators</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[GGH(^{+}13)]</td>
<td>[BR14]</td>
<td>[GMM(^{+}16)]</td>
</tr>
<tr>
<td>[MSZ16]</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[CGH17]*</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[CHKL18](^{†})</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>This talk(^{‡})</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

* for input-partitionable branching programs  
† for specific choices of parameters  
‡ in the quantum setting

---

‡ This talk

‡ in the quantum setting

† for specific choices of parameters

* for input-partitionable branching programs

‡ in the quantum setting

† for specific choices of parameters

* for input-partitionable branching programs

‡ in the quantum setting

† for specific choices of parameters

* for input-partitionable branching programs

‡ in the quantum setting

† for specific choices of parameters

* for input-partitionable branching programs

‡ in the quantum setting

† for specific choices of parameters

* for input-partitionable branching programs

‡ in the quantum setting

† for specific choices of parameters

* for input-partitionable branching programs

‡ in the quantum setting

† for specific choices of parameters

* for input-partitionable branching programs

‡ in the quantum setting

† for specific choices of parameters

* for input-partitionable branching programs

‡ in the quantum setting

† for specific choices of parameters

* for input-partitionable branching programs

‡ in the quantum setting

† for specific choices of parameters

* for input-partitionable branching programs

‡ in the quantum setting

† for specific choices of parameters

* for input-partitionable branching programs

‡ in the quantum setting

† for specific choices of parameters

* for input-partitionable branching programs

‡ in the quantum setting

† for specific choices of parameters

* for input-partitionable branching programs

‡ in the quantum setting

† for specific choices of parameters

* for input-partitionable branching programs

‡ in the quantum setting

† for specific choices of parameters

* for input-partitionable branching programs

‡ in the quantum setting

† for specific choices of parameters

* for input-partitionable branching programs

‡ in the quantum setting

† for specific choices of parameters

* for input-partitionable branching programs

‡ in the quantum setting

† for specific choices of parameters

* for input-partitionable branching programs

‡ in the quantum setting

† for specific choices of parameters

* for input-partitionable branching programs

‡ in the quantum setting

† for specific choices of parameters

* for input-partitionable branching programs

‡ in the quantum setting

† for specific choices of parameters

* for input-partitionable branching programs

‡ in the quantum setting

† for specific choices of parameters

* for input-partitionable branching programs

‡ in the quantum setting

† for specific choices of parameters

* for input-partitionable branching programs

‡ in the quantum setting

† for specific choices of parameters

* for input-partitionable branching programs

‡ in the quantum setting

† for specific choices of parameters

* for input-partitionable branching programs

‡ in the quantum setting

† for specific choices of parameters

* for input-partitionable branching programs

‡ in the quantum setting

† for specific choices of parameters

* for input-partitionable branching programs

‡ in the quantum setting

† for specific choices of parameters

* for input-partitionable branching programs

‡ in the quantum setting

† for specific choices of parameters

* for input-partitionable branching programs

‡ in the quantum setting

† for specific choices of parameters

* for input-partitionable branching programs

‡ in the quantum setting

† for specific choices of parameters

* for input-partitionable branching programs

‡ in the quantum setting

† for specific choices of parameters

* for input-partitionable branching programs

‡ in the quantum setting

† for specific choices of parameters

* for input-partitionable branching programs

‡ in the quantum setting

† for specific choices of parameters

* for input-partitionable branching programs

‡ in the quantum setting

† for specific choices of parameters

* for input-partitionable branching programs

‡ in the quantum setting

† for specific choices of parameters

* for input-partitionable branching programs

‡ in the quantum setting

† for specific choices of parameters

* for input-partitionable branching programs

‡ in the quantum setting

† for specific choices of parameters

* for input-partitionable branching programs

‡ in the quantum setting

† for specific choices of parameters

* for input-partitionable branching programs
Remarks

- Quantum poly time or classical $2^{O(\sqrt{n})}$ time
- Double mixed input attacks can be extended to circuit obfuscators
- [GGH\+13b]: only BP/circuit obfuscator currently standing in quantum

---

[GGH\+13b] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits, FOCS.
Conclusion (2/2)

Remarks

- Quantum poly time or classical $2^{O(\sqrt{n})}$ time
- Double mixed input attacks can be extended to circuit obfuscators
- [GGH+13b]: only BP/circuit obfuscator currently standing in quantum

Open problems

- Quantum attack against [GGH+13b]

Conclusion (2/2)

Remarks
- Quantum poly time or classical $2^{O(\sqrt{n})}$ time
- Double mixed input attacks can be extended to circuit obfuscators
- [GGH$^{+}$13b]: only BP/circuit obfuscator currently standing in quantum

Open problems
- Quantum attack against [GGH$^{+}$13b]
- Obfuscation for evasive functions

Conclusion (2/2)

Remarks

- Quantum poly time or classical $2^{O(\sqrt{n})}$ time
- Double mixed input attacks can be extended to circuit obfuscators
- $[\text{GGH}^{+}13\text{b}]:$ only BP/circuit obfuscator currently standing in quantum

Open problems

- Quantum attack against $[\text{GGH}^{+}13\text{b}]$
- Obfuscation for evasive functions

Questions?

---

[GGH$^{+}13\text{b}$] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits, FOCS.
References

Benny Applebaum and Zvika Brakerski.
Obfuscating circuits via composite-order graded encoding.

Prabhanjan Ananth, Divya Gupta, Yuval Ishai, and Amit Sahai.
Optimizing obfuscation: Avoiding barrington's theorem.

On the (im) possibility of obfuscating programs.

Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai.
Protecting obfuscation against algebraic attacks.

Saikrishna Badrinarayanan, Eric Miles, Amit Sahai, and Mark Zhandry.

Zvika Brakerski and Guy N Rothblum.
Obfuscating conjunctions.

Jean-François Biasse and Fang Song.
Efficient quantum algorithms for computing class groups and solving the principal ideal problem in arbitrary degree number fields.
Ronald Cramer, Léo Ducas, Chris Peikert, and Oded Regev.  
Recovering short generators of principal ideals in cyclotomic rings.  

Yilei Chen, Craig Gentry, and Shai Halevi.  
Cryptanalyses of candidate branching program obfuscators.  

Jung Hee Cheon, Minki Hhan, Jiseung Kim, and Changmin Lee.  
Cryptanalyses of branching program obfuscations over ggh13 multilinear map from the ntru problem.  

Nico Döttling, Sanjam Garg, Divya Gupta, Peihan Miao, and Pratyay Mukherjee.  
Obfuscation from low noise multilinear maps.  

Rex Fernando, Peter Rasmussen, and Amit Sahai.  
Preventing CLT attacks on obfuscation with linear overhead.  

Sanjam Garg, Craig Gentry, and Shai Halevi.  
Candidate multilinear maps from ideal lattices.  

Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters.  
Candidate indistinguishability obfuscation and functional encryption for all circuits.  
FOCS 2013, 2013.
Sanjam Garg, Eric Miles, Pratyay Mukherjee, Amit Sahai, Akshayaram Srinivasan, and Mark Zhandry.
Secure obfuscation in a weak multilinear map model.

Eric Miles, Amit Sahai, and Mor Weiss.
Protecting obfuscation against arithmetic attacks.

Eric Miles, Amit Sahai, and Mark Zhandry.
Annihilation attacks for multilinear maps: Cryptanalysis of indistinguishability obfuscation over GGH13.

Rafael Pass, Karn Seth, and Sidharth Telang.
Indistinguishability obfuscation from semantically-secure multilinear encodings.

Joe Zimmerman.
How to obfuscate programs directly.
The GGH13 multilinear map

Define $R = \mathbb{Z}[X]/(X^n + 1)$ with $n = 2^k$. 
The GGH13 multilinear map

- Define $R = \mathbb{Z}[X]/(X^n + 1)$ with $n = 2^k$.
- Sample $g$ a “small” element in $R$.
  $\Rightarrow$ the plaintext space is $\mathcal{P} = R/\langle g \rangle$. 

Sample $q$ a large integer.
$\Rightarrow$ the encoding space is $R_q = R/(qR) = \mathbb{Z}_{q}[X]/(X^{n+1})$. 

Notation
We write $[r]$ or $\{r\}$ the elements in $R_q$. 

A. Pellet-Mary
Quantum attack against some iO
Séminaire C2 24/20
The GGH13 multilinear map

- Define $R = \mathbb{Z}[X]/(X^n + 1)$ with $n = 2^k$.
- Sample $g$ a “small” element in $R$.
  $\Rightarrow$ the plaintext space is $\mathcal{P} = R/\langle g \rangle$.
- Sample $q$ a “large” integer.
  $\Rightarrow$ the encoding space is $R_q = R/(qR) = \mathbb{Z}_q[X]/(X^n + 1)$.

Notation

We write $[r]_q$ or $[r]$ the elements in $R_q$. 
The GGH13 multilinear map: encodings

- Sample $z$ uniformly in $R_q$.
- **Encoding**: An encoding of $a$ at level $i$ is

$$u = \left[ \frac{a + rg}{z^i} \right]_q$$

where $a + rg$ is a small element in $a + \langle g \rangle$. 

A. Pellet-Mary

Quantum attack against some iO

Séminaire C2
The GGH13 multilinear map: encodings

- Sample $z$ uniformly in $R_q$.
- **Encoding**: An encoding of $a$ at level $i$ is

\[ u = \left[ \frac{a + rg}{z^i} \right]_q \]

where $a + rg$ is a small element in $a + \langle g \rangle$.

### Addition and multiplication

**Addition**:

\[
\left[ \frac{a_1 + r_1g}{z^i} \right]_q + \left[ \frac{a_2 + r_2g}{z^i} \right]_q = \left[ \frac{a_1 + a_2 + r'g}{z^i} \right]_q .
\]

**Multiplication**:

\[
\left[ \frac{a_1 + r_1g}{z^i} \right]_q \cdot \left[ \frac{a_2 + r_2g}{z^j} \right]_q = \left[ \frac{a_1 \cdot a_2 + r'g}{z^{i+j}} \right]_q .
\]
The GGH13 multilinear map: zero-test

- Sample $h$ in $R$ of the order of $q^{1/2}$.
- Define

\[ p_{zt} = [z^\kappa h g^{-1}]_q. \]
The GGH13 multilinear map: zero-test

- Sample $h$ in $R$ of the order of $q^{1/2}$.
- Define

$$p_{zt} = [z^{\kappa}hg^{-1}]_q.$$ 

**Zero-test**

To test if $u = [c/z^{\kappa}]$ is an encoding of zero (i.e. $c = 0 \mod g$), compute

$$[u \cdot p_{zt}]_q = [chg^{-1}]_q.$$

This is small iff $c$ is a small multiple of $g$. 
Quantum double-zero-test

Reminder

Zero-test:  \( p_{zt} = [z^\kappa h g^{-1}]_q \).
Quantum double-zero-test

Reminder

Zero-test: \( p_{zt} = [z^\kappa h g^{-1}]_q \).

- Get multiple top-level encoding of zero \( u_i = [c_i g / z^\kappa]_q \)
Quantum double-zero-test

Reminder

Zero-test: \( p_{zt} = [z^\kappa h g^{-1}]_q \).

- Get multiple top-level encoding of zero \( u_i = [c_i g / z^\kappa]_q \)
- Zero-test them \( \Rightarrow [u_i p_{zt}]_q = c_i h \)
Quantum double-zero-test

Reminder

Zero-test: \( p_{zt} = [z^\kappa h g^{-1}]_q \).

- Get multiple top-level encoding of zero \( u_i = [c_i g / z^\kappa]_q \)
- Zero-test them \( \Rightarrow [u_i p_{zt}]_q = c_i h \)
- Recover ideal \( \langle h \rangle \) from the \( c_i h \)
Quantum double-zero-test

Reminder

Zero-test: \( p_{zt} = [z^\kappa h g^{-1}]_q \).

- Get multiple top-level encoding of zero \( u_i = [c_i g / z^\kappa]_q \)
- Zero-test them \( \Rightarrow [u_i p_{zt}]_q = c_i h \)
- Recover ideal \( \langle h \rangle \) from the \( c_i h \)
- Recover \( h \) from \( \langle h \rangle \) (quantum poly time [BS16, CDPR16])

---

[BS16] J.-F. Biasse and F. Song. Efficient quantum algorithms for computing class groups and solving the principal ideal problem in arbitrary degree number fields, SODA.

Quantum double-zero-test

Reminder

Zero-test: \( p_{zt} = [z^\kappa hg^{-1}]_q \).

- Get multiple top-level encoding of zero \( u_i = [c_i g / z^\kappa]_q \)
- Zero-test them \( \Rightarrow [u_i p_{zt}]_q = c_i h \)
- Recover ideal \( \langle h \rangle \) from the \( c_i h \)
- Recover \( h \) from \( \langle h \rangle \) (quantum poly time [BS16, CDPR16])
- Create \( p'_{zt} = [p_{zt}^2 / h^2]_q = [z^{2\kappa} g^{-2}]_q \)

---

[BS16] J.-F. Biasse and F. Song. Efficient quantum algorithms for computing class groups and solving the principal ideal problem in arbitrary degree number fields, SODA.

Quantum double-zero-test

Reminder

Zero-test: \( p_{zt} = [z^\kappa hg^{-1}]_q \).

- Get multiple top-level encoding of zero \( u_i = [c_ig/z^\kappa]_q \)
- Zero-test them \( \Rightarrow [u_i p_{zt}]_q = c_i h \)
- Recover ideal \( \langle h \rangle \) from the \( c_i h \)
- Recover \( h \) from \( \langle h \rangle \) (quantum poly time [BS16, CDPR16])
- Create \( p'_{zt} = [p_{zt}^2/h^2]_q = [z^{2\kappa}g^{-2}]_q \)

\[ [up'_{zt}]_q \text{ small } \iff u = [cg^2/z^{2\kappa}]_q \text{ for some small } c \]
\[ \iff u \text{ is a double zero at level } 2\kappa \]

[BS16] J.-F. Biasse and F. Song. Efficient quantum algorithms for computing class groups and solving the principal ideal problem in arbitrary degree number fields, SODA.