Cryptanalysis of branching program obfuscators

Jung Hee Cheon[1], **Minki Hhan**[1], Jiseung Kim[1], Changmin Lee[1], **Alice Pellet-Mary**[2]

[1] Seoul National University

[2] ENS de Lyon

Crypto 2018

What is this talk about

Two partial attacks against some candidate obfuscators built upon the GGH13 multilinear map [GGH13a]

- an attack for specific choices of parameters
- a quantum attack

Main idea of the two attacks

Transform known weaknesses of the GGH13 map into concrete attacks against the candidate obfuscators

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

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

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])

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. $C_1(x) = C_2(x)\ \forall x$,

$$O(C_1) \simeq_c O(C_2)$$

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. $C_1(x) = C_2(x)\ \forall x$,

$$O(C_1) \simeq_c O(C_2)$$

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

Multilinear maps (mmaps) and iO

Observation

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

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 (mmap).

Three main candidate multilinear maps: GGH13, CLT13, GGH15

Caution

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

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

Caution

All these candidate multilinear maps suffer from weaknesses (e.g. encodings of zero, zeroizing attacks,...).
$\Rightarrow$ all current attacks against iO rely on the underlying mmap

In this talk: we exploit known weaknesses 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:

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)

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$^+$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

State of the art and contributions

iO (using GGH13) / Attacks	Branching program obfuscators				Circuit obfuscators
	[GGH+13b]	[BR14]	[AGIS14, MSW14] [PST14, BGK+14] [BMSZ16]	[GMM+16]	[Zim15, AB15] [DGG+16]
[MSZ16]		✓	✓		
[CGH17]*	✓				
This work 1† [CHKL18]	✓	✓	✓	✓	
This work 2‡ [Pel18]			✓	✓	✓

* for input-partitionable branching programs

† for specific choices of parameters

‡ in the quantum setting

Outline

1. Simple obfuscator
2. GGH13 multilinear map
3. Contributions

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

- 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 $\mathrm{inp} : \{1,\ldots,\ell\} \to \{1,\ldots,r\}$ (where r is the size of the input).

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

$x = 0 \quad 1 \quad 1$

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

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

- 2ℓ 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

$x = 0 \quad 1 \quad 1$

$$\begin{array}{cccccccc} & A_{1,1} & A_{2,1} & A_{3,1} & A_{4,1} & A_{5,1} & A_{6,1} & \\ {\color{red}A_0} & & & & & & & A_7 \\ & A_{1,0} & A_{2,0} & A_{3,0} & A_{4,0} & A_{5,0} & A_{6,0} & \end{array}$$

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

- 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 $\mathrm{inp} : \{1, \ldots, \ell\} \to \{1, \ldots, r\}$ (where r is the size of the input).

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

$$x = \begin{array}{ccc} 0 & 1 & 1 \\ \uparrow & & \end{array}$$

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

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

- 2ℓ 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

$$x = \begin{array}{ccc} 0 & 1 & 1 \\ \uparrow & & \end{array}$$

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

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

- 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 $\mathrm{inp} : \{1, \ldots, \ell\} \to \{1, \ldots, r\}$ (where r is the size of the input).

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

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

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

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

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

i	1	2	3	4	5	6
inp(i)	1	1	2	1	3	2

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

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

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

- 2ℓ 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

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

i	1	2	3	4	5	6
inp(i)	1	1	2	1	3	2

$x = 0 \quad 1 \quad 1$

$\uparrow$

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

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

- 2ℓ 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

$x = 0 \quad 1 \quad 1$

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

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

- 2ℓ 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

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

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

Cryptographic multilinear maps

Definition: κ-multilinear map

Different levels of encodings, from 1 to κ.
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: $\text{Zero-test}(\mathrm{Enc}(a, \kappa)) = \text{True iff } a = 0$.

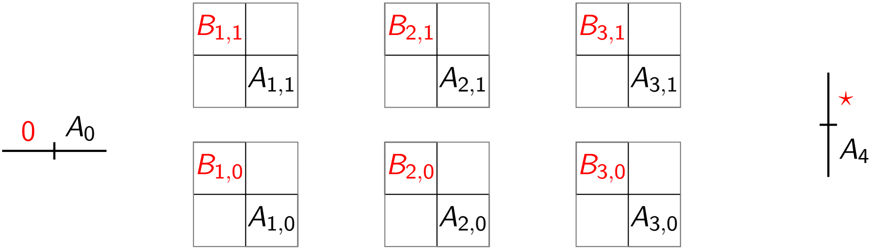
Simple obfuscator

- **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

$$\begin{array}{ccccc} & \boxed{A_{1,1}} & \boxed{A_{2,1}} & \boxed{A_{3,1}} & \\ \underline{A_0} & & & & \big| A_4 \\ & \boxed{A_{1,0}} & \boxed{A_{2,0}} & \boxed{A_{3,0}} & \end{array}$$

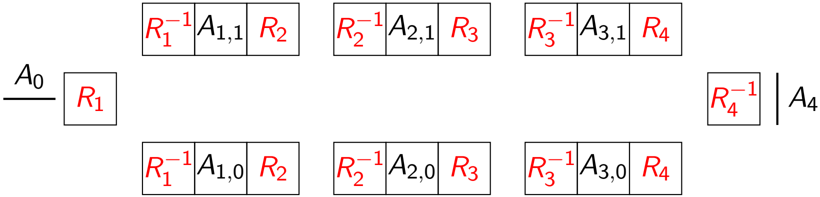
Simple obfuscator

- **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



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

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

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

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

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

$$\widetilde{A_0} \qquad \begin{array}{ccc} \boxed{\widetilde{A_{1,1}}} & \boxed{\widetilde{A_{2,0}}} & \boxed{\widetilde{A_{2,0}}} \\ \\ \boxed{\widetilde{A_{1,0}}} & \boxed{\widetilde{A_{2,0}}} & \boxed{\widetilde{A_{2,0}}} \end{array} \qquad \widetilde{A_4}$$

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

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

Outline

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 $n = 2^k$.
- The plaintext space is $\mathcal{P} = R/\langle g \rangle$ for a "small" element g in R.

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

Notation

We write $[x]_q$ the elements in R_q for $x \in R$.

The GGH13 multilinear map: encodings and zero-test

- Sample z uniformly in R_q and h in R of the order of $q^{1/2}$.
- **Encoding:** An encoding of a at level i is

$$u = [(a + rg)z^{-i}]_q$$

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

The GGH13 multilinear map: encodings and zero-test

- Sample z uniformly in R_q and h in R of the order of $q^{1/2}$.
- **Encoding:** An encoding of a at level i is

$$u = [(a + rg)z^{-i}]_q$$

  where $a + rg$ is a small element in $a + \langle g \rangle$.
- **Zero-testing:** A zero-testing parameter is defined by

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

The GGH13 multilinear map: encodings and zero-test

- Sample z uniformly in R_q and h in R of the order of $q^{1/2}$.
- **Encoding:** An encoding of a at level i is

$$u = [(a + rg)z^{-i}]_q$$

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

- **Zero-testing:** A zero-testing parameter is defined by

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

Zero-test

To test if $u = [cz^{-\kappa}]_q$ 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.

Outline

Global ideas of the two attacks

Main idea

Transform known weaknesses of the GGH13 map into concrete attacks against the candidate obfuscators.

Global ideas of the two attacks

Main idea

Transform known weaknesses of the GGH13 map into concrete attacks against the candidate obfuscators.

- Attack 1 [CHKL18]:
  - NTRU attack [ABD16, CJL16, KF17]
  - recover multiple of sensitive elements
  - classical polynomial time, for specific choices of parameters

Global ideas of the two attacks

Main idea

Transform known weaknesses of the GGH13 map into concrete attacks against the candidate obfuscators.

- Attack 1 [CHKL18]:
  - NTRU attack [ABD16, CJL16, KF17]
  - recover multiple of sensitive elements
  - classical polynomial time, for specific choices of parameters
- Attack 2 [Pel18]:
  - short principal ideal solver [CDPR16]
  - recover a sensitive element
  - quantum polynomial time [BS16] (or classical sub-exponential time [BEF^{+}17] for specific (unused) choices of parameters)

Attack 1: Starting point = NTRU

For two encodings $[a_1 \cdot z^{-1}]_q, [a_2 \cdot z^{-1}]_q$ for small a_1, a_2, we can compute

$$[a_1 \cdot z^{-1}]_q \cdot [a_2 \cdot z^{-1}]_q^{-1} = [a_1/a_2]_q$$

Attack 1: Starting point = NTRU

For two encodings $[a_1 \cdot z^{-1}]_q, [a_2 \cdot z^{-1}]_q$ for small a_1, a_2, we can compute

$$[a_1 \cdot z^{-1}]_q \cdot [a_2 \cdot z^{-1}]_q^{-1} = [a_1/a_2]_q$$

NTRU problem [ABD16, CJL16, KF17]

Let a_1, a_2 be *sufficiently small* elements of R. For a given NTRU instance $[a_1/a_2]_q$, we can efficiently recover

$$(c \cdot a_1, c \cdot a_2) \in R^2$$

for some small c for a given NTRU instance.

Attack 1: Starting point = NTRU

For two encodings $[a_1 \cdot z^{-1}]_q, [a_2 \cdot z^{-1}]_q$ for small a_1, a_2, we can compute

$$[a_1 \cdot z^{-1}]_q \cdot [a_2 \cdot z^{-1}]_q^{-1} = [a_1/a_2]_q$$

NTRU problem [ABD16, CJL16, KF17]

Let a_1, a_2 be *sufficiently small* elements of R. For a given NTRU instance $[a_1/a_2]_q$, we can efficiently recover

$$(c \cdot a_1, c \cdot a_2) \in R^2$$

for some small c for a given NTRU instance.

For another encoding $[a_3 \cdot z^{-1}]_q$, compute

$$[a_3 \cdot z^{-1}]_q / [a_1 \cdot z^{-1}]_q \cdot (c \cdot a_1) = c \cdot a_3 \in R.$$

Attack 1: Starting point = NTRU

For two encodings $[a_1 \cdot z^{-1}]_q, [a_2 \cdot z^{-1}]_q$ for small a_1, a_2, we can compute

$$[a_1 \cdot z^{-1}]_q \cdot [a_2 \cdot z^{-1}]_q^{-1} = [a_1/a_2]_q$$

NTRU problem [ABD16, CJL16, KF17]

Let a_1, a_2 be *sufficiently small* elements of R. For a given NTRU instance $[a_1/a_2]_q$, we can efficiently recover

$$(c \cdot a_1, c \cdot a_2) \in R^2$$

for some small c for a given NTRU instance.

For another encoding $[a_3 \cdot z^{-1}]_q$, compute

$$[a_3 \cdot z^{-1}]_q / [a_1 \cdot z^{-1}]_q \cdot (c \cdot a_1) = c \cdot a_3 \in R.$$

Thus we can compute $(ca_i \in R)_i$ using $([a_i \cdot z^{-1}]_q)_i$.

Attack 1: Starting point = NTRU

For two encodings $[a_1 \cdot z^{-1}]_q, [a_2 \cdot z^{-1}]_q$ for small a_1, a_2, we can compute

$$[a_1 \cdot z^{-1}]_q \cdot [a_2 \cdot z^{-1}]_q^{-1} = [a_1/a_2]_q$$

NTRU problem [ABD16, CJL16, KF17]

Let a_1, a_2 be *sufficiently small* elements of R. For a given NTRU instance $[a_1/a_2]_q$, we can efficiently recover

$$(c \cdot a_1, c \cdot a_2) \in R^2$$

for some small c for a given NTRU instance.

For another encoding $[a_3 \cdot z^{-1}]_q$, compute

$$[a_3 \cdot z^{-1}]_q / [a_1 \cdot z^{-1}]_q \cdot (c \cdot a_1) = c \cdot a_3 \in R.$$

Thus we can compute $(ca_i \in R)_i$ using $([a_i \cdot z^{-1}]_q)_i$.

Attack 1

- **Input:** An obfuscated program $\mathcal{O}(P)$ and plain program Q
- De-randomize the branching program
  - Solve NTRU simultaneously
  - Recover $\langle g \rangle$ using zero of program
  - Distinguish by Matrix Zeroizing Attack
- **Result:** Distinguishing Attack: $P = Q$?

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

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

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

Attack 1

- **Input:** An obfuscated program $\mathcal{O}(P)$ and plain program Q
- De-randomize the branching program
  - Solve NTRU simultaneously
  - Recover $\langle g \rangle$ using zero of program
  - Distinguish by Matrix Zeroizing Attack
- **Result:** Distinguishing Attack: $P = Q$?

	$c_{1,1}(\widetilde{A_{1,1}} + R_{1,1}g)$	$c_{2,1}(\widetilde{A_{2,1}} + R_{2,1}g)$	$c_{3,1}(\widetilde{A_{3,1}} + R_{3,1}g)$	
$c_0(\widetilde{A_0} + R_0 g)$				$c_4(\widetilde{A_4} + R_4 g)$
	$c_{1,0}(\widetilde{A_{1,0}} + R_{1,0}g)$	$c_{2,0}(\widetilde{A_{2,0}} + R_{2,0}g)$	$c_{3,0}(\widetilde{A_{3,0}} + R_{3,0}g)$	

Attack 1

- **Input:** An obfuscated program $\mathcal{O}(P)$ and plain program Q
- De-randomize the branching program
  - Solve NTRU simultaneously
  - Recover $\langle g \rangle$ using zero of program
  - Distinguish by Matrix Zeroizing Attack
- **Result:** Distinguishing Attack: $P = Q$?

	$c_{1,1}(\widetilde{A_{1,1}} + R_{1,1}g)$	$c_{2,1}(\widetilde{A_{2,1}} + R_{2,1}g)$	$c_{3,1}(\widetilde{A_{3,1}} + R_{3,1}g)$	
$c_0(\widetilde{A_0} + R_0 g)$				$c_4(\widetilde{A_4} + R_4 g)$
	$c_{1,0}(\widetilde{A_{1,0}} + R_{1,0}g)$	$c_{2,0}(\widetilde{A_{2,0}} + R_{2,0}g)$	$c_{3,0}(\widetilde{A_{3,0}} + R_{3,0}g)$	

These matrices $\in R$ rather that R_q

Attack 1

- **Input:** An obfuscated program $\mathcal{O}(P)$ and plain program Q
- De-randomize the branching program
  - Solve NTRU simultaneously
  - Recover $\langle g \rangle$ using zero of program
  - Distinguish by Matrix Zeroizing Attack
- **Result:** Distinguishing Attack: $P = Q$?

BP matrix	$\mathrm{Enc}(\widetilde{A})$
$\mathrm{Enc}(0)$	$[rg/z^{\kappa}]_q$

Attack 1

- **Input:** An obfuscated program $\mathcal{O}(P)$ and plain program Q
- De-randomize the branching program
  - Solve NTRU simultaneously
  - Recover $\langle g \rangle$ using zero of program
  - Distinguish by Matrix Zeroizing Attack
- **Result:** Distinguishing Attack: $P = Q$?

$$\begin{array}{ccccc} \text{BP matrix} & \text{Enc}(\widetilde{A}) & & c(\widetilde{A} + Rg) \\ & & \Rightarrow & \\ \text{Enc}(0) & [rg/z^{\kappa}]_q & & c'rg \end{array}$$

Attack 1

- **Input:** An obfuscated program $\mathcal{O}(P)$ and plain program Q
- De-randomize the branching program
  - Solve NTRU simultaneously
  - Recover $\langle g \rangle$ using zero of program
  - Distinguish by Matrix Zeroizing Attack
- **Result:** Distinguishing Attack: $P = Q$?

$$\begin{array}{ccccc} \text{BP matrix} & \text{Enc}(\widetilde{A}) & & c(\widetilde{A} + Rg) \\ & & \Rightarrow & \\ \text{Enc}(0) & [rg/z^{\kappa}]_q & & c' rg \end{array}$$

Collecting several top level zeros, recover $\langle g \rangle$

Attack 1

- **Input:** An obfuscated program $\mathcal{O}(P)$ and plain program Q
- De-randomize the branching program
  - Solve NTRU simultaneously
  - Recover $\langle g \rangle$ using zero of program
  - Distinguish by Matrix Zeroizing Attack
- **Result:** Distinguishing Attack: $P = Q$?

BP matrix	$\mathrm{Enc}(\widetilde{A})$	$\Rightarrow$	$c(\widetilde{A} + Rg)$	$\Rightarrow$	$c\widetilde{A} \bmod g$
$\mathrm{Enc}(0)$	$[rg/z^{\kappa}]_q$		$c'rg$		

Collecting several top level zeros, recover $\langle g \rangle$

Attack 1

- **Input:** An obfuscated program $\mathcal{O}(P)$ and plain program Q
- De-randomize the branching program
  - Solve NTRU simultaneously
  - Recover $\langle g \rangle$ using zero of program
  - Distinguish by Matrix Zeroizing Attack
- **Result:** Distinguishing Attack: $P = Q$?

$c\widetilde{A}$ mod g do not contain the randomness r and level parameter z

Attack 1: Mixed-input Attack

We remove the effects of scalar bundlings using algebraic ways

Attack 1: Mixed-input Attack

We remove the effects of scalar bundlings using algebraic ways (omitted)

Attack 1: Mixed-input Attack

We remove the effects of scalar bundlings using algebraic ways (omitted)

Mixed-input attack can be carried out!

Attack 1: Mixed-input Attack

We remove the effects of scalar bundlings using algebraic ways (omitted)

Mixed-input attack can be carried out!

- *Invalid* inputs can induce the different outputs of equivalent BPs

Attack 1: Mixed-input Attack

We remove the effects of scalar bundlings using algebraic ways (omitted)

Mixed-input attack can be carried out!

- *Invalid* inputs can induce the different outputs of equivalent BPs

i	1	2	3
inp(i)	1	1	2

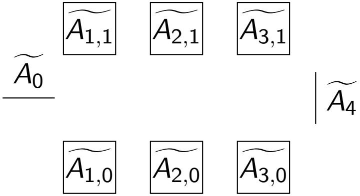
Attack 1: Mixed-input Attack

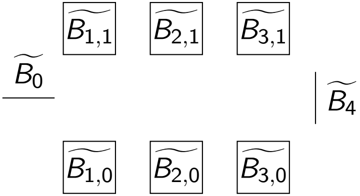
We remove the effects of scalar bundlings using algebraic ways (omitted)

Mixed-input attack can be carried out!

- *Invalid* inputs can induce the different outputs of equivalent BPs

i	1	2	3
inp(i)	1	1	2





Attack 1: Mixed-input Attack

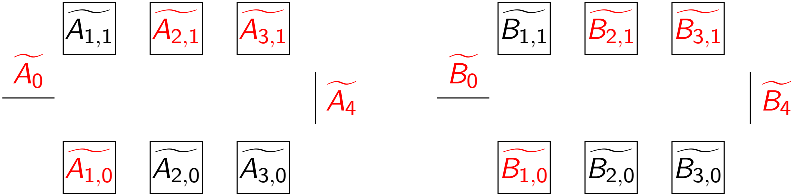
We remove the effects of scalar bundlings using algebraic ways (omitted)

Mixed-input attack can be carried out!

- *Invalid* inputs can induce the different outputs of equivalent BPs

i	1	2	3
inp(i)	1	1	2

invalid input
indices 0 1 1



Attack 1: Mixed-input Attack

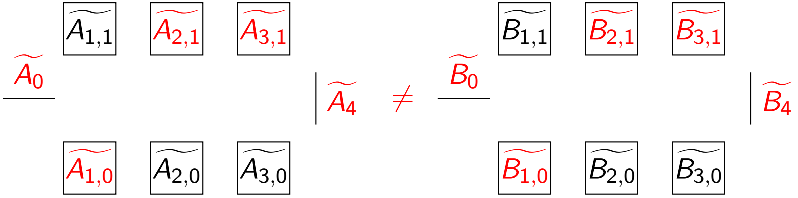
We remove the effects of scalar bundlings using algebraic ways (omitted)

Mixed-input attack can be carried out!

- *Invalid* inputs can induce the different outputs of equivalent BPs

i	1	2	3
inp(i)	1	1	2

invalid input
indices 0 1 1



outputs zero

outputs non-zero

Attack 1: Matrix Zeroizing Attack

We remove the effects of scalar bundlings using algebraic ways (omitted)

Matrix-zeroizing attack: extended mixed-input attack

- *Invalid* inputs can induce the different outputs of equivalent BPs

i	1	2	3
inp(i)	1	1	2

invalid input
indices 0 1 1

$$\frac{\widetilde{A_0}}{}\begin{array}{ccc} \boxed{\widetilde{A_{1,1}}} & \boxed{\widetilde{A_{2,1}}} & \boxed{\widetilde{A_{3,1}}} \\ & & \\ \boxed{\widetilde{A_{1,0}}} & \boxed{\widetilde{A_{2,0}}} & \boxed{\widetilde{A_{3,0}}} \end{array} \Big| \widetilde{A_4} \neq \frac{\widetilde{B_0}}{}\begin{array}{ccc} \boxed{\widetilde{B_{1,1}}} & \boxed{\widetilde{B_{2,1}}} & \boxed{\widetilde{B_{3,1}}} \\ & & \\ \boxed{\widetilde{B_{1,0}}} & \boxed{\widetilde{B_{2,0}}} & \boxed{\widetilde{B_{3,0}}} \end{array} \Big| \widetilde{B_4}$$

outputs zero

outputs non-zero

Attack 1: Matrix Zeroizing Attack

We remove the effects of scalar bundlings using algebraic ways (omitted)

Matrix-zeroizing attack: extended mixed-input attack

- *Invalid* inputs can induce the different outputs of equivalent BPs
- Summation of mixed-input can yield the different outputs of BPs

i	1	2	3
inp(i)	1	1	2

invalid input
010, 011, 100, 101, $\cdots$

$$\sum \; \widetilde{A_0} \; \begin{array}{ccc} \boxed{\widetilde{A_{1,1}}} & \boxed{\widetilde{A_{2,1}}} & \boxed{\widetilde{A_{3,1}}} \\ \boxed{\widetilde{A_{1,0}}} & \boxed{\widetilde{A_{2,0}}} & \boxed{\widetilde{A_{3,0}}} \end{array} \; \widetilde{A_4} \neq \widetilde{B_0} \; \begin{array}{ccc} \boxed{\widetilde{B_{1,1}}} & \boxed{\widetilde{B_{2,1}}} & \boxed{\widetilde{B_{3,1}}} \\ \boxed{\widetilde{B_{1,0}}} & \boxed{\widetilde{B_{2,0}}} & \boxed{\widetilde{B_{3,0}}} \end{array} \; \widetilde{B_4}$$

outputs zero

outputs non-zero

Attack 2: Starting point = Principal Ideal Problem

Given an obfuscated branching program, the evaluation of program is determined, for κ level encoding u, by the value

$$[u p_{zt}]_q.$$

Attack 2: Starting point = Principal Ideal Problem

Given an obfuscated branching program, the evaluation of program is determined, for κ level encoding u, by the value

$$[up_{zt}]_q.$$

When output of program is 1, $u = [rg/z^\kappa]_q$ holds and $[up_{zt}]_q = rh \in R$.

Short Principal Ideal Problem [BS16, CDPR16]

Given many multiples of h, we can recover

$$h \in R$$

in *quantum polynomial time*.

Attack 2: Starting point = Principal Ideal Problem

Given an obfuscated branching program, the evaluation of program is determined, for κ level encoding u, by the value

$$[up_{zt}]_q.$$

When output of program is 1, $u = [rg/z^\kappa]_q$ holds and $[up_{zt}]_q = rh \in R$.

Short Principal Ideal Problem [BS16, CDPR16]

Given many multiples of h, we can recover

$$h \in R$$

in *quantum polynomial time*.

We can compute the *double-zero* testing value at level 2κ as follows.

$$[(p_{zt}/h)^2]_q = [z^{2\kappa} \cdot g^{-2}]_q$$

Attack 2: Starting point = Principal Ideal Problem

Given an obfuscated branching program, the evaluation of program is determined, for κ level encoding u, by the value

$$[up_{zt}]_q.$$

When output of program is 1, $u = [rg/z^\kappa]_q$ holds and $[up_{zt}]_q = rh \in R$.

Short Principal Ideal Problem [BS16, CDPR16]

Given many multiples of h, we can recover

$$h \in R$$

in *quantum polynomial time*.

We can compute the *double-zero* testing value at level 2κ as follows.

$$[(p_{zt}/h)^2]_q = [z^{2\kappa} \cdot g^{-2}]_q$$

Remark: every computations works correctly.

Attack 2: Mixed-input Attack

- Run mixed-input attack on obfuscated program at level κ

i	1	2	3
inp(i)	1	1	2

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

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

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

Attack 2: Mixed-input Attack

- Run mixed-input attack on obfuscated program at level κ

i	1	2	3
inp(i)	1	1	2

invalid input
indices 0 1 1

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

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

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

Attack 2: Mixed-input Attack

- Run mixed-input attack on obfuscated program at level κ
  - We cannot evaluate it in obfuscated program due to constructions [1]

i	1	2	3
inp(i)	1	1	2

invalid input indices 0 1 1

$$\widetilde{A_0} \quad \boxed{\widetilde{A_{1,1}}} \quad \boxed{\widetilde{A_{2,1}}} \quad \boxed{\widetilde{A_{3,1}}} \quad \widetilde{A_4}$$

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

[1] level parameters, scalar bundlings

Attack 2: Mixed-input Attack

- ~~Run mixed-input attack on obfuscated program at level κ~~
  - We cannot evaluate it in obfuscated program due to constructions [1]

i	1	2	3
inp(i)	1	1	2

invalid input
indices 0 1 1

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

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

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

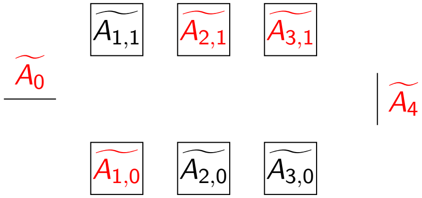
[1] level parameters, scalar bundlings

Attack 2: Mixed-input Attack

- ~~Run mixed-input attack on obfuscated program at level κ~~
  - We cannot evaluate it in obfuscated program due to constructions [1]
- Construct 2κ-level obfuscated program
- Run mixed-input attack on obfuscated program at level 2κ

i	1	2	3
inp(i)	1	1	2

invalid input
indices 0 1 1



[1] level parameters, scalar bundlings

Attack 2: Mixed-input Attack

- ~~Run mixed-input attack on obfuscated program at level κ~~
  - We cannot evaluate it in obfuscated program due to constructions [1]
- Construct 2κ-level obfuscated program
- Run mixed-input attack on obfuscated program at level 2κ

i	1	2	3
inp(i)	1	1	2

invalid input

indices 0 1 1 ? ? ?

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

$\widetilde{A_0}$ $\Big|\ \widetilde{A_4}$

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

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

$\widetilde{A_0}$ $\Big|\ \widetilde{A_4}$

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

[1] level parameters, scalar bundlings

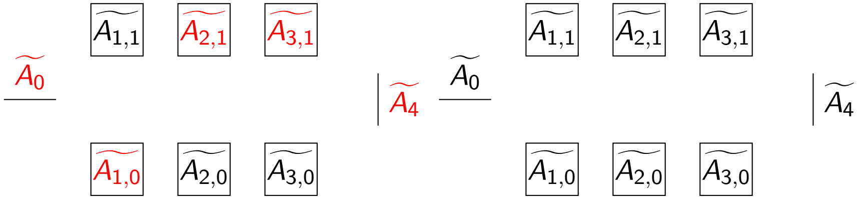
Attack 2: Mixed-input Attack

- ~~Run mixed-input attack on obfuscated program at level κ~~
  - We cannot evaluate it in obfuscated program due to constructions [1]
- Construct 2κ-level obfuscated program
- Run mixed-input attack on obfuscated program at level 2κ

i	1	2	3
inp(i)	1	1	2

invalid input

indices 0 1 1 1 0 1

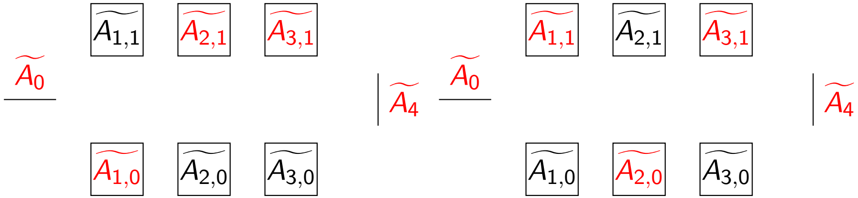


[1] level parameters, scalar bundlings

- ~~Run mixed-input attack on obfuscated program at level κ~~
  - We cannot evaluate it in obfuscated program due to constructions [1]
- Construct 2κ-level obfuscated program
- Run mixed-input attack on obfuscated program at level 2κ

i	1	2	3
inp(i)	1	1	2

invalid input
indices 0 1 1 1 0 1



[1] level parameters, scalar bundlings

Summary and work in progress

iO (using GGH13) / Attacks	Branching program obfuscators				Circuit obfuscators
	[GGH+13b]	[BR14]	[AGIS14, MSW14] [PST14, BGK+14] [BMSZ16]	[GMM+16]	[Zim15, AB15] [DGG+16]
[MSZ16]		✓	✓		
[CGH17]*	✓				
This work 1† [CHKL18]	✓	✓	✓	✓	
This work 2‡ [Pel18]			✓	✓	✓

* for input-partitionable branching programs

† for specific choices of parameters

‡ in the quantum setting

Summary and work in progress

iO (using GGH13) / Attacks	Branching program obfuscators				Circuit obfuscators
	[GGH+13b]	[BR14]	[AGIS14, MSW14] [PST14, BGK+14] [BMSZ16]	[GMM+16]	[Zim15, AB15] [DGG+16]
[MSZ16]		✓	✓		
[CGH17]⋆	✓				
This work 1† [CHKL18]	✓	✓	✓	✓	?
This work 2‡ [Pel18]	?		✓	✓	✓

⋆ for input-partitionable branching programs

† for specific choices of parameters

‡ in the quantum setting

Work in progress

- Quantum attack on [GGH$^+$13b]

Work in progress

- Quantum attack on [GGH$^+$13b]
  - Applying (modified) matrix-zeroizing attack!

Work in progress

- Quantum attack on [GGH$^+$13b]
  - Applying (modified) matrix-zeroizing attack!
  - We show that this combination of two work can obtain a quantum polynomial time distinguishing attack on [GGH$^+$13b]

Work in progress

- Quantum attack on [GGH$^+$13b]
  - Applying (modified) matrix-zeroizing attack!
  - We show that this combination of two work can obtain a quantum polynomial time distinguishing attack on [GGH$^+$13b]
- Classical attack for circuit obfuscations

Work in progress

- Quantum attack on [GGH$^+$13b]
  - Applying (modified) matrix-zeroizing attack!
  - We show that this combination of two work can obtain a quantum polynomial time distinguishing attack on [GGH$^+$13b]
- Classical attack for circuit obfuscations
  - Extending the NTRU attack!

Work in progress

- Quantum attack on [GGH$^+$13b]
  - Applying (modified) matrix-zeroizing attack!
  - We show that this combination of two work can obtain a quantum polynomial time distinguishing attack on [GGH$^+$13b]
- Classical attack for circuit obfuscations
  - Extending the NTRU attack!
  - We also try to find a countermeasure on the attack

Perspectives / Open problems

- Obfuscation for evasive functions
- Countermeasure on the attacks
- Parameter constraints to prevent our classical attack[2]: $n = \tilde{\Omega}(\kappa^2\lambda)$
  - This constraint agrees to the current best algorithms to solve the overstretched NTRU problem

[2] n: dimension of space, κ: multilinearity level, λ: security parameter
To prevent classical PIP attack and our attack: $n = \tilde{\Omega}(\max(\kappa^2\lambda, \lambda^2))$

Perspectives / Open problems

- Obfuscation for evasive functions
- Countermeasure on the attacks
- Parameter constraints to prevent our classical attack[2]: $n = \tilde{\Omega}(\kappa^2\lambda)$
  - This constraint agrees to the current best algorithms to solve the overstretched NTRU problem

Remark

- Proofs in idealized models VS Constructions with concrete schemes

[2] n: dimension of space, κ: multilinearity level, λ: security parameter
To prevent classical PIP attack and our attack: $n = \tilde{\Omega}(\max(\kappa^2\lambda, \lambda^2))$

Perspectives / Open problems

- Obfuscation for evasive functions
- Countermeasure on the attacks
- Parameter constraints to prevent our classical attack[2]: $n = \tilde{\Omega}(\kappa^2\lambda)$
  - This constraint agrees to the current best algorithms to solve the overstretched NTRU problem

Remark

- Proofs in idealized models VS Constructions with concrete schemes
- Concrete schemes do not fit in the idealized model

[2] n: dimension of space, κ: multilinearity level, λ: security parameter
To prevent classical PIP attack and our attack: $n = \tilde{\Omega}(\max(\kappa^2\lambda, \lambda^2))$

Perspectives / Open problems

- Obfuscation for evasive functions
- Countermeasure on the attacks
- Parameter constraints to prevent our classical attack[2]: $n = \tilde{\Omega}(\kappa^2 \lambda)$
  - This constraint agrees to the current best algorithms to solve the overstretched NTRU problem

Remark

- Proofs in idealized models VS Constructions with concrete schemes
- Concrete schemes do not fit in the idealized model

$\Rightarrow$ This gap can cause the significant weakness of concrete scheme!

[2] n: dimension of space, κ: multilinearity level, λ: security parameter
To prevent classical PIP attack and our attack: $n = \tilde{\Omega}(\max(\kappa^2 \lambda, \lambda^2))$

References I

Benny Applebaum and Zvika Brakerski.
Obfuscating circuits via composite-order graded encoding.
In TCC 2015, pages 528–556, 2015.

Martin R. Albrecht, Shi Bai, and Léo Ducas.
A subfield lattice attack on overstretched NTRU assumptions - cryptanalysis of some FHE and graded encoding schemes.
In Crypto 2016, pages 153–178, 2016.

Prabhanjan Ananth, Divya Gupta, Yuval Ishai, and Amit Sahai.
Optimizing obfuscation: Avoiding barrington's theorem.
In CCS 2014, pages 646–658. ACM, 2014.

Jean-François Biasse, Thomas Espitau, Pierre-Alain Fouque, Alexandre Gélin, and Paul Kirchner.
Computing generator in cyclotomic integer rings.
In Eurocrypt 2017, pages 60–88. Springer, 2017.

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.

Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai.
Protecting obfuscation against algebraic attacks.
In Eurocrypt 2014, pages 221–238, 2014.

Saikrishna Badrinarayanan, Eric Miles, Amit Sahai, and Mark Zhandry.
Post-zeroizing obfuscation: New mathematical tools, and the case of evasive circuits.
In Eurocrypt 2016, pages 764–791, 2016.

References II

Zvika Brakerski and Guy N Rothblum.
Obfuscating conjunctions.
Crypto 2014, 2014.

Jean-François Biasse and Fang Song.
Efficient quantum algorithms for computing class groups and solving the principal ideal problem in arbitrary degree number fields.
In SODA 2016, pages 893–902. Society for Industrial and Applied Mathematics, 2016.

Ronald Cramer, Léo Ducas, Chris Peikert, and Oded Regev.
Recovering short generators of principal ideals in cyclotomic rings.
In Eurocrypt 2016, pages 559–585, 2016.

Yilei Chen, Craig Gentry, and Shai Halevi.
Cryptanalyses of candidate branching program obfuscators.
In Eurocrypt 2017, pages 278–307. Springer, 2017.

Jung Hee Cheon, Jinhyuck Jeong, and Changmin Lee.
An algorithm for ntru problems and cryptanalysis of the ggh multilinear map without a low-level encoding of zero.
LMS Journal of Computation and Mathematics, 19(A):255–266, 2016.

Nico Döttling, Sanjam Garg, Divya Gupta, Peihan Miao, and Pratyay Mukherjee.
Obfuscation from low noise multilinear maps.
ePrint, Report 2016/599, 2016.

Rex Fernando, Peter Rasmussen, and Amit Sahai.
Preventing CLT attacks on obfuscation with linear overhead.
In Asiacrypt 2017, pages 242–271, 2017.

References III

Sanjam Garg, Craig Gentry, and Shai Halevi.
Candidate multilinear maps from ideal lattices.
In Eurocrypt 2017, pages 1–17. Springer, 2013.

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.
In TCC 2016, pages 241–268, 2016.

Paul Kirchner and Pierre-Alain Fouque.
Revisiting lattice attacks on overstretched ntru parameters.
In Eurocrypt 2017, pages 3–26. Springer, 2017.

Eric Miles, Amit Sahai, and Mor Weiss.
Protecting obfuscation against arithmetic attacks.
ePrint, Report 2014/878, 2014.

Eric Miles, Amit Sahai, and Mark Zhandry.
Annihilation attacks for multilinear maps: Cryptanalysis of indistinguishability obfuscation over GGH13.
In Crypto 2016, pages 629–658, 2016.

Rafael Pass, Karn Seth, and Sidharth Telang.
Indistinguishability obfuscation from semantically-secure multilinear encodings.
In Crypto 2014, pages 500–517, 2014.

References IV

Joe Zimmerman.
How to obfuscate programs directly.
In Eurocrypt 2015, pages 439–467, 2015.