Program obfuscation

Alice Pellet-Mary

LIP, ENS de Lyon

PhD seminar, CWI
July 12, 2019

Obfuscation

An obfuscator should:

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

Obfuscation

An obfuscator should:

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

Two kind of obfuscators:

- practical obfuscators (white box)
- theoretical obfuscators (iO)

Obfuscation

An obfuscator should:

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

Two kind of obfuscators:

- practical obfuscators (white box)
- theoretical obfuscators (iO)

Overview of the talk

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

Outline of the talk

What is a program?

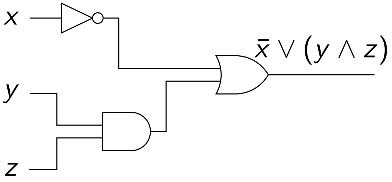
- C/C++/Python/$\cdots$ code;

What is a program?

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

What is a program?

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

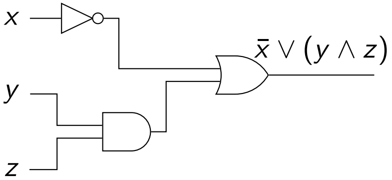


Notation

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

What is a program?

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



Notation

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

Virtual Black Box (VBB) obfuscation

Recall

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

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

Virtual Black Box (VBB) obfuscation

Recall

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

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

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

Virtual Black Box (VBB) obfuscation

Recall

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

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

- (Functionality) For all $C \in \mathcal{C}$, $\mathcal{O}(C) \equiv C$;
- (Efficiency) $\mathcal{O}$ is PPT $\Rightarrow$ for all $C \in \mathcal{C}$, $|\mathcal{O}(C)| \leq \text{poly}(|C|)$;

Virtual Black Box (VBB) obfuscation

Recall

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

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

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

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

Virtual Black Box (VBB) obfuscation

Recall

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

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

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

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

VBB obfuscation is impossible to achieve [BGI+01]

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

Indistinguishability Obfuscation (iO)

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

Indistinguishability Obfuscation (iO)

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

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

Indistinguishability Obfuscation (iO)

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

- (Functionality) For all $C \in \mathcal{C}$, $\mathcal{O}(C) \equiv C$;
- (Efficiency) $\mathcal{O}$ is PPT $\Rightarrow$ for all $C \in \mathcal{C}$, $|\mathcal{O}(C)| \leq \text{poly}(|C|)$;

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) $\mathcal{O}$ is PPT $\Rightarrow$ for all $C \in \mathcal{C}$, $|\mathcal{O}(C)| \leq \text{poly}(|C|)$;
- (indistinguishability) For all $C_1, C_2 \in \mathcal{C}$ with $C_1 \equiv C_2$,

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

If P = NP . . .

If P = NP, then iOs exist, e.g.:

$\Rightarrow$ There exist inefficient iOs (even if P $\neq$ NP)

If P = NP . . .

If P = NP, then iOs exist, e.g.:
$\Rightarrow$ There exist inefficient iOs (even if P $\neq$ NP)

$$\mathcal{O}(C) = \text{smallest circuit computing the same function as } C$$

Why is iO useful (1)

iO achieves “best possible” obfuscation

Why is iO useful (1)

iO achieves "best possible" obfuscation

Proof:

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

Why is iO useful (1)

iO achieves "best possible" obfuscation

Proof:

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

Why is iO useful (1)

iO achieves "best possible" obfuscation

Proof:

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

Why is iO useful (1)

iO achieves "best possible" obfuscation

Proof:

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

Why is iO useful (1)

iO achieves "best possible" obfuscation

Proof:

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

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

Why is iO useful (2)

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

Why is iO useful (2)

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

Example: black box decryption (symmetric setting)

Why is iO useful (2)

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

Example: black box decryption (symmetric setting)

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

Why is iO useful (2)

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

Example: black box decryption (symmetric setting)

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

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

Why is iO useful (2)

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

Example: black box decryption (symmetric setting)

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

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

Why is iO useful (2)

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

Example: black box decryption (symmetric setting)

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

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

Why is iO useful (2)

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

Example: black box decryption (symmetric setting)

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

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

Why is iO useful (2)

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

Example: black box decryption (symmetric setting)

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

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

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

Outline of the talk

Disclaimer

We only have candidate iO

(no construction based on standard cryptographic assumptions)

Three main categories

- Branching program obfuscators

Three main categories

- Branching program obfuscators
  - needs bootstrapping via fully homomorphic encryption

Three main categories

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

Three main categories

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

Three main categories

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

Three main categories

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

Three main categories

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

Security

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

Security

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

	number of candidates	still standing classically	still standing quantumly
Branching program iO	≈ 20	≈ 10	3
Circuit iO	≈ 8	≈ 8	0

All attacks rely on the underlying multilinear map

Restricted functionalities

VBB obfuscators based on RLWE for

Restricted functionalities

VBB obfuscators based on RLWE for

- point functions

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

Restricted functionalities

VBB obfuscators based on RLWE for

- 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

VBB obfuscators based on RLWE for

- 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

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

Practicability

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

Outline of the talk

Branching programs

A branching program represents a function (cf Turing machine, or circuit).

Branching programs

A branching program represents a function (cf Turing machine, or circuit).

A Branching Program (BP) is a collection of

- 2ℓ matrices $M_{i,b}$ (for $i \in \{1, \dots, \ell\}$ and $b \in \{0, 1\}$),
- two vectors M_0 and $M_{\ell+1}$,
- a vector inp $\in \{1, \dots, r\}^\ell$ (where r is the size of the input).

BP

	x_1	x_1	x_2	x_1	x_3	x_2	
M_0	$M_{1,1}$	$M_{2,1}$	$M_{3,1}$	$M_{4,1}$	$M_{5,1}$	$M_{6,1}$	M_7
	$M_{1,0}$	$M_{2,0}$	$M_{3,0}$	$M_{4,0}$	$M_{5,0}$	$M_{6,0}$	

Evaluation on $x = 0 \quad 1 \quad 1$

Branching programs

A branching program represents a function (cf Turing machine, or circuit).

A Branching Program (BP) is a collection of

- 2ℓ matrices $M_{i,b}$ (for $i \in \{1, \dots, \ell\}$ and $b \in \{0, 1\}$),
- two vectors M_0 and $M_{\ell+1}$,
- a vector inp $\in \{1, \dots, r\}^\ell$ (where r is the size of the input).

BP

	x_1	x_1	x_2	x_1	x_3	x_2	
	$M_{1,1}$	$M_{2,1}$	$M_{3,1}$	$M_{4,1}$	$M_{5,1}$	$M_{6,1}$	
M_0							M_7
	$M_{1,0}$	$M_{2,0}$	$M_{3,0}$	$M_{4,0}$	$M_{5,0}$	$M_{6,0}$	

Evaluation on $x = 0 \quad 1 \quad 1$

Branching programs

A branching program represents a function (cf Turing machine, or circuit).

A Branching Program (BP) is a collection of

- 2ℓ matrices $M_{i,b}$ (for $i \in \{1, \dots, \ell\}$ and $b \in \{0, 1\}$),
- two vectors M_0 and $M_{\ell+1}$,
- a vector inp $\in \{1, \dots, r\}^\ell$ (where r is the size of the input).

BP

$$\begin{array}{ccccccccc} & & x_1 & x_1 & x_2 & x_1 & x_3 & x_2 & \\ M_0 & \times & \begin{matrix} M_{1,1} \\ M_{1,0} \end{matrix} & \begin{matrix} M_{2,1} \\ M_{2,0} \end{matrix} & \begin{matrix} M_{3,1} \\ M_{3,0} \end{matrix} & \begin{matrix} M_{4,1} \\ M_{4,0} \end{matrix} & \begin{matrix} M_{5,1} \\ M_{5,0} \end{matrix} & \begin{matrix} M_{6,1} \\ M_{6,0} \end{matrix} & M_7 \end{array}$$

Evaluation on $x = 0 \quad 1 \quad 1$

$\uparrow$

Branching programs

A branching program represents a function (cf Turing machine, or circuit).

A Branching Program (BP) is a collection of

- 2ℓ matrices $M_{i,b}$ (for $i \in \{1, \dots, \ell\}$ and $b \in \{0,1\}$),
- two vectors M_0 and $M_{\ell+1}$,
- a vector inp $\in \{1, \dots, r\}^\ell$ (where r is the size of the input).

BP

$$\begin{array}{cccccccccc} & & x_1 & & x_1 & x_2 & x_1 & x_3 & x_2 & \\ M_0 & \times & \begin{array}{c} M_{1,1} \\ M_{1,0} \end{array} & \times & \begin{array}{c} M_{2,1} \\ \color{red}{M_{2,0}} \end{array} & \begin{array}{c} M_{3,1} \\ M_{3,0} \end{array} & \begin{array}{c} M_{4,1} \\ M_{4,0} \end{array} & \begin{array}{c} M_{5,1} \\ M_{5,0} \end{array} & \begin{array}{c} M_{6,1} \\ M_{6,0} \end{array} & M_7 \end{array}$$

Evaluation on $x = \color{red}{0} \;\; 1 \;\; 1$

$\qquad\qquad\qquad\quad\uparrow$

Branching programs

A branching program represents a function (cf Turing machine, or circuit).

A Branching Program (BP) is a collection of

- 2ℓ matrices $M_{i,b}$ (for $i \in \{1, \dots, \ell\}$ and $b \in \{0, 1\}$),
- two vectors M_0 and $M_{\ell+1}$,
- a vector inp $\in \{1, \dots, r\}^\ell$ (where r is the size of the input).

BP

$$
\begin{array}{ccccccccccccc}
 & & x_1 & & x_1 & & x_2 & & x_1 & & x_3 & & x_2 & & \\
M_0 & \times & \begin{array}{c} M_{1,1} \\ M_{1,0} \end{array} & \times & \begin{array}{c} M_{2,1} \\ M_{2,0} \end{array} & \times & \begin{array}{c} M_{3,1} \\ M_{3,0} \end{array} & & \begin{array}{c} M_{4,1} \\ M_{4,0} \end{array} & & \begin{array}{c} M_{5,1} \\ M_{5,0} \end{array} & & \begin{array}{c} M_{6,1} \\ M_{6,0} \end{array} & & M_7
\end{array}
$$

Evaluation on $x = 0 \quad 1 \quad 1$

$\uparrow$

Branching programs

A branching program represents a function (cf Turing machine, or circuit).

A Branching Program (BP) is a collection of

- 2ℓ matrices $M_{i,b}$ (for $i \in \{1, \dots, \ell\}$ and $b \in \{0, 1\}$),
- two vectors M_0 and $M_{\ell+1}$,
- a vector inp $\in \{1, \dots, r\}^\ell$ (where r is the size of the input).

BP

$$
\begin{array}{ccccccccccccc}
 & & x_1 & & x_1 & & x_2 & & x_1 & x_3 & x_2 & \\
M_0 & \times & \begin{matrix} M_{1,1} \\ M_{1,0} \end{matrix} & \times & \begin{matrix} M_{2,1} \\ M_{2,0} \end{matrix} & \times & \begin{matrix} M_{3,1} \\ M_{3,0} \end{matrix} & \times & \begin{matrix} M_{4,1} \\ M_{4,0} \end{matrix} & \begin{matrix} M_{5,1} \\ M_{5,0} \end{matrix} & \begin{matrix} M_{6,1} \\ M_{6,0} \end{matrix} & M_7
\end{array}
$$

Evaluation on $x = 0 \quad 1 \quad 1$

$\uparrow$

Branching programs

A branching program represents a function (cf Turing machine, or circuit).

A Branching Program (BP) is a collection of

- 2ℓ matrices $M_{i,b}$ (for $i \in \{1, \dots, \ell\}$ and $b \in \{0, 1\}$),
- two vectors M_0 and $M_{\ell+1}$,
- a vector inp $\in \{1, \dots, r\}^\ell$ (where r is the size of the input).

BP

$$\begin{array}{ccccccccccccc} & & x_1 & & x_1 & & x_2 & & x_1 & & x_3 & x_2 & \\ M_0 & \times & \begin{matrix} M_{1,1} \\ M_{1,0} \end{matrix} & \times & \begin{matrix} M_{2,1} \\ M_{2,0} \end{matrix} & \times & \begin{matrix} M_{3,1} \\ M_{3,0} \end{matrix} & \times & \begin{matrix} M_{4,1} \\ M_{4,0} \end{matrix} & \times & \begin{matrix} M_{5,1} \\ M_{5,0} \end{matrix} & \begin{matrix} M_{6,1} \\ M_{6,0} \end{matrix} & M_7 \end{array}$$

Evaluation on $x = 0 \quad 1 \quad 1$

$\uparrow$

Branching programs

A branching program represents a function (cf Turing machine, or circuit).

A Branching Program (BP) is a collection of

- 2ℓ matrices $M_{i,b}$ (for $i \in \{1, \ldots, \ell\}$ and $b \in \{0, 1\}$),
- two vectors M_0 and $M_{\ell+1}$,
- a vector inp $\in \{1, \ldots, r\}^\ell$ (where r is the size of the input).

BP

$$\begin{array}{ccccccccccccc} & & x_1 & & x_1 & & x_2 & & x_1 & & x_3 & & x_2 & & \\ M_0 & \times & \begin{matrix} M_{1,1} \\ M_{1,0} \end{matrix} & \times & \begin{matrix} M_{2,1} \\ M_{2,0} \end{matrix} & \times & \begin{matrix} M_{3,1} \\ M_{3,0} \end{matrix} & \times & \begin{matrix} M_{4,1} \\ M_{4,0} \end{matrix} & \times & \begin{matrix} M_{5,1} \\ M_{5,0} \end{matrix} & \times & \begin{matrix} M_{6,1} \\ M_{6,0} \end{matrix} & & M_7 \end{array}$$

Evaluation on $x = 0 \quad 1 \quad 1$

$\uparrow$

Branching programs

A branching program represents a function (cf Turing machine, or circuit).

A Branching Program (BP) is a collection of

- 2ℓ matrices $M_{i,b}$ (for $i \in \{1, \dots, \ell\}$ and $b \in \{0,1\}$),
- two vectors M_0 and $M_{\ell+1}$,
- a vector inp $\in \{1, \dots, r\}^\ell$ (where r is the size of the input).

BP

$$M_0 \times \begin{array}{c} x_1 \\ M_{1,1} \\ M_{1,0} \end{array} \times \begin{array}{c} x_1 \\ M_{2,1} \\ M_{2,0} \end{array} \times \begin{array}{c} x_2 \\ M_{3,1} \\ M_{3,0} \end{array} \times \begin{array}{c} x_1 \\ M_{4,1} \\ M_{4,0} \end{array} \times \begin{array}{c} x_3 \\ M_{5,1} \\ M_{5,0} \end{array} \times \begin{array}{c} x_2 \\ M_{6,1} \\ M_{6,0} \end{array} \times M_7$$

Evaluation on $x = 0 \quad 1 \quad 1$

Branching programs

A branching program represents a function (cf Turing machine, or circuit).

A Branching Program (BP) is a collection of

- 2ℓ matrices $M_{i,b}$ (for $i \in \{1,\ldots,\ell\}$ and $b \in \{0,1\}$),
- two vectors M_0 and $M_{\ell+1}$,
- a vector inp $\in \{1,\ldots,r\}^\ell$ (where r is the size of the input).

BP

$$
\begin{array}{ccccccccccccccc}
 & & x_1 & & x_1 & & x_2 & & x_1 & & x_3 & & x_2 & & \\
M_0 & \times & \begin{array}{c} M_{1,1} \\ M_{1,0} \end{array} & \times & \begin{array}{c} M_{2,1} \\ M_{2,0} \end{array} & \times & \begin{array}{c} M_{3,1} \\ M_{3,0} \end{array} & \times & \begin{array}{c} M_{4,1} \\ M_{4,0} \end{array} & \times & \begin{array}{c} M_{5,1} \\ M_{5,0} \end{array} & \times & \begin{array}{c} M_{6,1} \\ M_{6,0} \end{array} & \times & M_7 \begin{array}{l} = 0 \to 0 \\ \neq 0 \to 1 \end{array}
\end{array}
$$

Evaluation on $x = 0 \quad 1 \quad 1$

Cryptographic multilinear maps

Definition: κ-multilinear map

Different levels of encodings, from 1 to κ.
Write $\text{Enc}(a, i)$ 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

[GGH$^+$13, BR14, BGK$^+$14, PST14, AGIS14, MSW14, GMM$^+$16]

- **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 a multilinear map
- **Output:** The encoded matrices and vectors

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

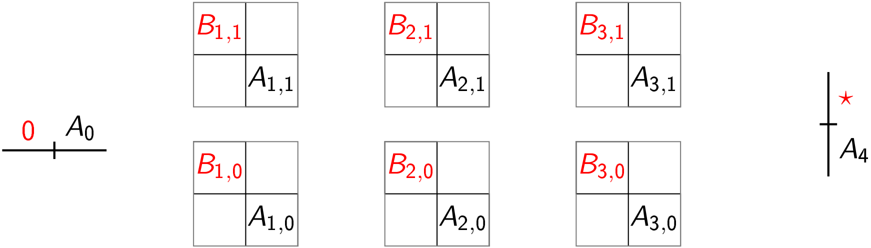
$$\underline{A_0} \qquad\qquad\qquad\qquad\qquad\qquad \Big| A_4$$

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

Simple obfuscator

[GGH+13, BR14, BGK+14, PST14, AGIS14, MSW14, GMM+16]

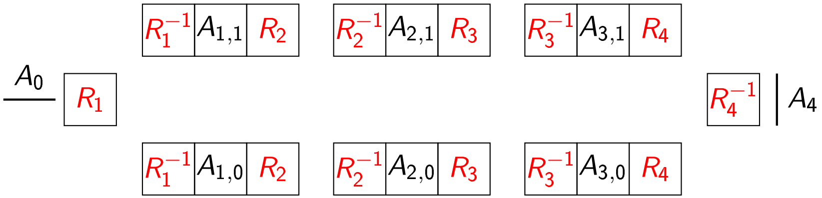
- **Input:** A branching program
- Randomize the branching program
  - Add random diagonal blocks
  - Killian's randomization
  - Multiply by random (non zero) bundling scalars
- Encode the matrices using a multilinear map
- **Output:** The encoded matrices and vectors



Simple obfuscator

[GGH+13, BR14, BGK+14, PST14, AGIS14, MSW14, GMM+16]

- **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 a multilinear map
- **Output:** The encoded matrices and vectors



Simple obfuscator

[GGH+13, BR14, BGK+14, PST14, AGIS14, MSW14, GMM+16]

- **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 a multilinear map
- **Output:** The encoded matrices and vectors

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

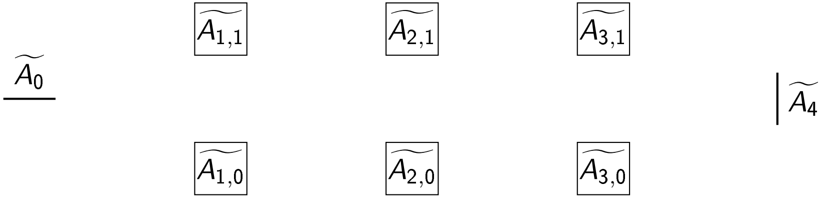
$$\underline{A_0} \qquad\qquad\qquad\qquad\qquad\qquad \Big| A_4$$

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

Simple obfuscator

[GGH+13, BR14, BGK+14, PST14, AGIS14, MSW14, GMM+16]

- **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 a multilinear map
- **Output:** The encoded matrices and vectors



Simple obfuscator

[GGH+13, BR14, BGK+14, PST14, AGIS14, MSW14, GMM+16]

- **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 a multilinear map
- **Output:** The encoded matrices and vectors

$$\mathrm{Enc}(\widetilde{A_0}) \quad \begin{array}{ccc} \mathrm{Enc}\left(\boxed{\widetilde{A_{1,1}}}\right) & \mathrm{Enc}\left(\boxed{\widetilde{A_{2,1}}}\right) & \mathrm{Enc}\left(\boxed{\widetilde{A_{3,1}}}\right) \\ \\ \mathrm{Enc}\left(\boxed{\widetilde{A_{1,0}}}\right) & \mathrm{Enc}\left(\boxed{\widetilde{A_{2,0}}}\right) & \mathrm{Enc}\left(\boxed{\widetilde{A_{3,0}}}\right) \end{array} \quad \mathrm{Enc}(\widetilde{A_4})$$

Mixed-input attack

Notations

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

$$
\begin{array}{ccccc}
 & \boxed{\widehat{A_{1,1}}} & \boxed{\widehat{A_{2,1}}} & \boxed{\widehat{A_{3,1}}} & \\
\underline{\widehat{A_0}} & & & & \Big|\,\widehat{A_4} \\
 & \boxed{\widehat{A_{1,0}}} & \boxed{\widehat{A_{2,0}}} & \boxed{\widehat{A_{3,0}}} & \\
 & x_1 & x_2 & x_1 &
\end{array}
$$

Mixed-input attack

Notations

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

$$\begin{array}{ccccc} & \boxed{\widehat{A_{1,1}}} & \boxed{\widehat{A_{2,1}}} & \boxed{\widehat{A_{3,1}}} & \\ \widehat{A_0} & & & & \widehat{A_4} \\ & \boxed{\widehat{A_{1,0}}} & \boxed{\widehat{A_{2,0}}} & \boxed{\widehat{A_{3,0}}} & \\ & x_1 & x_2 & x_1 & \\ & 1 & 1 & 1 & \end{array}$$

Mixed-input attack

Notations

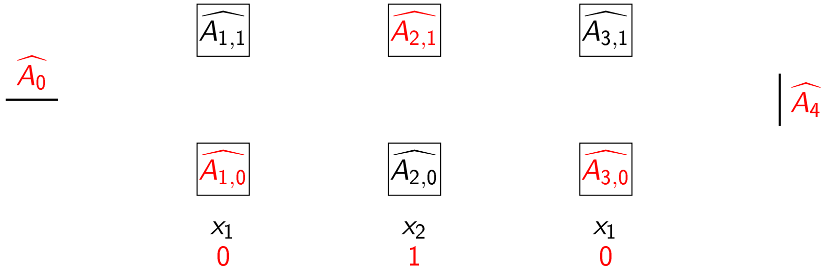
- $A_{i,b}$ input branching program
- $\widetilde{A_{i,b}}$ after randomisation
- $\widehat{A_{i,b}}$ after encoding with a multilinear map (output of the iO)

$$\begin{array}{ccccc} & \boxed{\widehat{A_{1,1}}} & \boxed{\widehat{A_{2,1}}} & \boxed{\widehat{A_{3,1}}} & \\ \underline{\widehat{A_0}} & & & & \Big|\,\widehat{A_4} \\ & \boxed{\widehat{A_{1,0}}} & \boxed{\widehat{A_{2,0}}} & \boxed{\widehat{A_{3,0}}} & \\ & x_1 & x_2 & x_1 & \\ & 1 & 0 & 1 & \end{array}$$

Mixed-input attack

Notations

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



Mixed-input attack

Notations

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

$$\underline{\widehat{A_0}} \qquad \begin{matrix} \boxed{\widehat{A_{1,1}}} & \boxed{\widehat{A_{2,1}}} & \boxed{\widehat{A_{3,1}}} \\ \boxed{\widehat{A_{1,0}}} & \boxed{\widehat{A_{2,0}}} & \boxed{\widehat{A_{3,0}}} \\ x_1 & x_2 & x_1 \\ 0 & 0 & 0 \end{matrix} \qquad \Big| \, \widehat{A_4}$$

Mixed-input attack

Notations

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

	$\boxed{\widehat{A_{1,1}}}$	$\boxed{\widehat{A_{2,1}}}$	$\boxed{\widehat{A_{3,1}}}$	
$\widehat{A_0}$				$\widehat{A_4}$
	$\boxed{\widehat{A_{1,0}}}$	$\boxed{\widehat{A_{2,0}}}$	$\boxed{\widehat{A_{3,0}}}$	
	x_1	x_2	x_1	
	0	0	1	

Mixed-input attack

Notations

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

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

Preventing mixed-input attacks

- In the randomization phase $\Rightarrow$ not in this talk [GGH+13, BR14]
- Using the mmap $\Rightarrow$ straddling set system
  [BGK+14, PST14, AGIS14, MSW14, GMM+16]

Preventing mixed-input attacks

- In the randomization phase $\Rightarrow$ not in this talk [GGH$^+$13, BR14]
- Using the mmap $\Rightarrow$ straddling set system
  [BGK$^+$14, PST14, AGIS14, MSW14, GMM$^+$16]

Mmap degree: $\kappa = 5$

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

Preventing mixed-input attacks

- In the randomization phase $\Rightarrow$ not in this talk [GGH+13, BR14]
- Using the mmap $\Rightarrow$ straddling set system
  [BGK+14, PST14, AGIS14, MSW14, GMM+16]

Mmap degree: $\kappa = 6$

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

Preventing mixed-input attacks

- In the randomization phase $\Rightarrow$ not in this talk [GGH$^+$13, BR14]
- Using the mmap $\Rightarrow$ straddling set system [BGK$^+$14, PST14, AGIS14, MSW14, GMM$^+$16]

Mmap degree: $\kappa = 6$

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

Preventing mixed-input attacks

- In the randomization phase $\Rightarrow$ not in this talk [GGH+13, BR14]
- Using the mmap $\Rightarrow$ straddling set system
  [BGK+14, PST14, AGIS14, MSW14, GMM+16]

Mmap degree: $\kappa = 6$

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

Total level: 7 $\Rightarrow$ cannot zero-test

What to remember

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

What to remember

+ iO would be very useful (at least for theory) . . .
− . . . but no constructions from standard assumptions yet

What to remember

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

What to remember

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

What to remember

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

Questions?

References I

Benny Applebaum and Zvika Brakerski.
Obfuscating circuits via composite-order graded encoding.
In Theory of Cryptography Conference, pages 528–556. Springer, 2015.

Daniel Apon, Nico Döttling, Sanjam Garg, and Pratyay Mukherjee.
Cryptanalysis of indistinguishability obfuscations of circuits over ggh13.
In 44th International Colloquium on Automata, Languages, and Programming (ICALP 2017). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

Prabhanjan Ananth, Divya Gupta, Yuval Ishai, and Amit Sahai.
Optimizing obfuscation: Avoiding barrington's theorem.
In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pages 646–658. ACM, 2014.

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

Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai.
Protecting obfuscation against algebraic attacks.
In Annual International Conference on the Theory and Applications of Cryptographic Techniques, pages 221–238. Springer, 2014.

Zvika Brakerski and Guy N. Rothblum.
Virtual black-box obfuscation for all circuits via generic graded encoding.
In Theory of Cryptography Conference, pages 1–25. Springer, 2014.

References II

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

Yilei Chen, Craig Gentry, and Shai Halevi.
Cryptanalyses of candidate branching program obfuscators.
In Annual International Conference on the Theory and Applications of Cryptographic Techniques, pages 278–307. Springer, 2017.

Jung Hee Cheon, Minki Hhan, Jiseung Kim, and Changmin Lee.
Cryptanalyses of branching program obfuscations over ggh13 multilinear map from the ntru problem.
In Annual International Cryptology Conference, pages 184–210. Springer, 2018.

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

Nico Döttling, Sanjam Garg, Divya Gupta, Peihan Miao, and Pratyay Mukherjee.
Obfuscation from low noise multilinear maps.
In International Conference in Cryptology in India, pages 329–352. Springer, 2018.

Rex Fernando, Peter MR Rasmussen, and Amit Sahai.
Preventing clt attacks on obfuscation with linear overhead.
In International Conference on the Theory and Application of Cryptology and Information Security, pages 242–271. Springer, 2017.

Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters.
Candidate indistinguishability obfuscation and functional encryption for all circuits.
pages 40–49, 2013.

References III

Sanjam Garg, Eric Miles, Pratyay Mukherjee, Amit Sahai, Akshayaram Srinivasan, and Mark Zhandry.
Secure obfuscation in a weak multilinear map model.
In Theory of Cryptography Conference, pages 241–268. Springer, 2016.

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

Eric Miles, Amit Sahai, and Mor Weiss.
Protecting obfuscation against arithmetic attacks.
IACR Cryptology ePrint Archive, 2014:878, 2014.

Eric Miles, Amit Sahai, and Mark Zhandry.
Annihilation attacks for multilinear maps: Cryptanalysis of indistinguishability obfuscation over ggh13.
In Annual International Cryptology Conference, pages 629–658. Springer, 2016.

Alice Pellet-Mary.
Quantum attacks against indistinguishablility obfuscators proved secure in the weak multilinear map model.
In Annual International Cryptology Conference, pages 153–183. Springer, 2018.

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

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

References IV

Joe Zimmerman.
How to obfuscate programs directly.
In Annual International Conference on the Theory and Applications of Cryptographic Techniques, pages 439–467. Springer, 2015.

History (GGH13-based branching program obfuscation)

	2013	2014	2015	2016	2017	2018
Constructions	[GGH$^+$13]	[BR14] [BGK$^+$14, PST14] [AGIS14, MSW14]		[GMM$^+$16]	[FRS17]	
Attacks				[MSZ16]	[CGH17] [ADGM17]	[CHKL18] [Pel18]

History (GGH13-based branching program obfuscation)

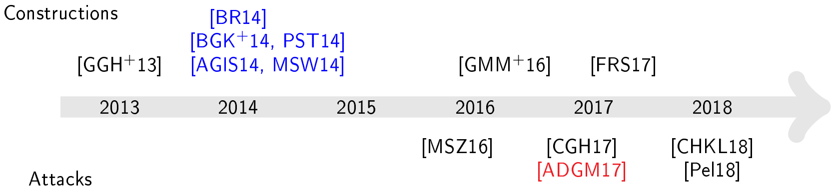
Constructions

	2013	2014	2015	2016	2017	2018
Constructions	[GGH+13]	[BR14] [BGK+14, PST14] [AGIS14, MSW14]		[GMM+16]	[FRS17]	
Attacks				[MSZ16]	[CGH17] [ADGM17]	[CHKL18] [Pel18]

Attacks

[MSZ16]: all constructions without diagonal blocks

History (GGH13-based branching program obfuscation)



[MSZ16]: all constructions without diagonal blocks

[ADGM17]: idem MSZ but from circuits

History (GGH13-based branching program obfuscation)

Constructions

[GGH$^+$13]

[BR14]
[BGK$^+$14, PST14]
[AGIS14, MSW14]

[GMM$^+$16]

[FRS17]

2013 2014 2015 2016 2017 2018

[MSZ16]

[CGH17]
[ADGM17]

[CHKL18]
[Pel18]

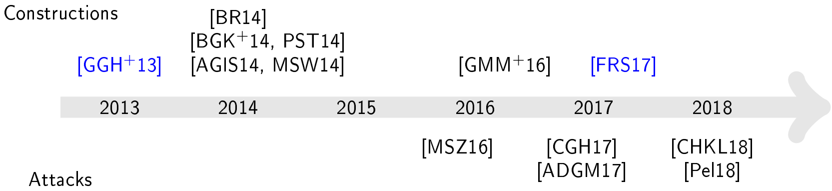
Attacks

[MSZ16]: all constructions without diagonal blocks

[ADGM17]: idem MSZ but from circuits

[CGH17]: use input-partitionability (cf CLT13)

History (GGH13-based branching program obfuscation)

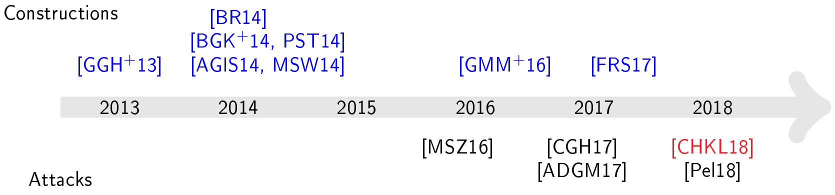


[MSZ16]: all constructions without diagonal blocks

[ADGM17]: idem MSZ but from circuits

[CGH17]: use input-partitionability (cf CLT13) $\Rightarrow$ prevented by [FRS17]

History (GGH13-based branching program obfuscation)



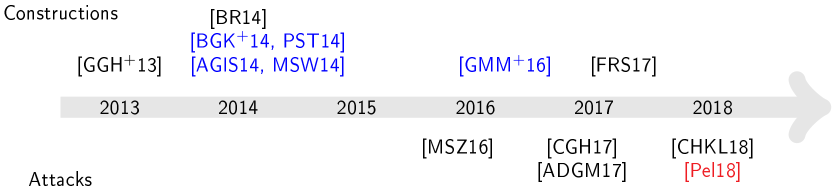
[MSZ16]: all constructions without diagonal blocks

[ADGM17]: idem MSZ but from circuits

[CGH17]: use input-partitionability (cf CLT13) $\Rightarrow$ prevented by [FRS17]

[CHKL18]: NTRU attack for specific choices of parameters

History (GGH13-based branching program obfuscation)



[MSZ16]: all constructions without diagonal blocks

[ADGM17]: idem MSZ but from circuits

[CGH17]: use input-partitionability (cf CLT13) $\Rightarrow$ prevented by [FRS17]

[CHKL18]: NTRU attack for specific choices of parameters

[Pel18]: quantum attack

Current status

iOs / Attacks	[GGH$^+$13]	[BR14, BGK$^+$14, PST14, AGIS14, MSW14]	[GMM$^+$16]	circuit obfuscators [Zim15, AB15, DGG$^+$18]
[MSZ16]		fully broken		
[CGH17]	input-partitionable			
[CHKL18]	some parameters		some parameters	
[Pel18]			quantum	quantum

Still standing classically:

- [GGH$^+$13]+[FRS17]
- [GMM$^+$16]
- all circuit obfuscators

Still standing quantumly:

- [GGH$^+$13]+[FRS17]