5-Gen: A Framework for Prototyping Applications Using Multilinear Maps and Branching Programs

Kevin Lewi Alex Mlozemoff Daniel Apon
Brent Carmer Adam Foltzer Daniel Wagner
Dave Archer Dan Boneh Jonathan Katz Mariana Raykova

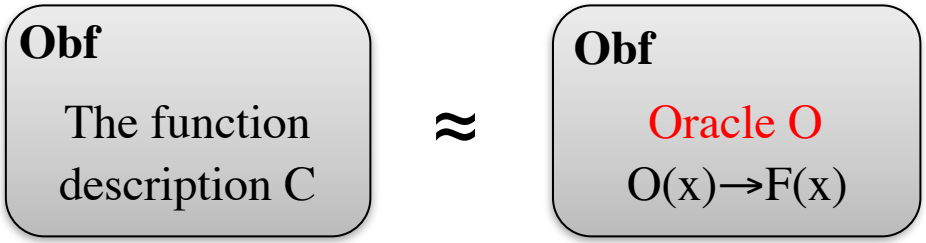
Obfuscation

- Make programs "unintelligible" while maintaining their functiona
- Hiding secrets in software?
  - Proprietary algorithms
  - Software patches
  - Malware detection

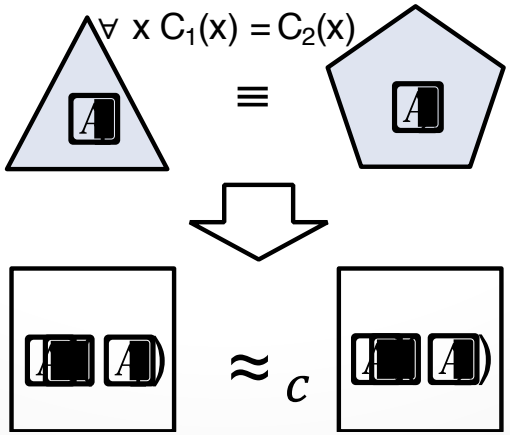
```
@P=split//,".URRUU
\c8R";@d=split//,"\nrekcah xinU / lreP
rehtona tsuJ";sub p{ @p{"r$p","u$p"}
=(P,P);pipe"r$p","u$p";++$p;($q*=2)+=$f=!
fork;map{$P=$P[$f^ord ($p{$_})&6];
$p{$_}=/ ^$P/ix?$P:close$_}keys%p}
p;p;p;p;p;map{$p{$_}=~/^[P.]/&& close$_}
%p;wait until$?;map{/^r/&&<$_>}%p;$_=
$d[$q];sleep rand(2)if/\S/;print
```

Obfuscation Security [BGIRSVY01]

- Virtual Black Box (VBB) security
  - Impossibility



- Indistinguishability Obfuscation



First candidate construction
[GGHRSW13]
and many others after

Functional Encryption (FE) [SW05]

- Decryption keys that reveal only functions of the encrypted message

Encryption

```
=(P,P);pipe"r$p","u$p";++$p;
($q*=2)+=$f=!fork;map{$P=
$P[$f^ord ($p{$_})&6];$p{$_}=/ ^
$P/ix?$P:close$_}keys%p}
p;p;p;p;p;map{$p{$_}=~/^[P.]/&&
close$_}%p;wait until$?;map{/^r/
&&<$_>}%p;$_=$d[$q];sleep
rand(2)if/\S/;print
```

Is the age over 21?

Driving License Category ?

"Flavors" of Functional Encryption

- Setup – generate parameters
  - Encryption key PK
  - Master secret key MSK

Private Key
$PK = MSK$

- KeyGen – generate function decryption keys
  - SK_F

Function Hiding
SK_F hides F

- Enc – generate ciphertext for a message
  - ct = Enc(PK, m)

Multi-Input FE (MIFE)
$ct_i = Enc(PK_i, x_i)$

- Dec – decrypt a ciphertext
  - $Dec(SK_F, ct) = F(m)$

Multi-Input FE (MIFE)
$Dec(SK_F, ct_1, \ldots, ct_n) = F(x_1, \ldots, x_n)$

Obfuscation and FE

- Function hiding public key FE ⇒ obfuscation

Obf(F)

SK_F

$Eval_{Obf(F)}(m)$

1. $ct = Enc(PK\ m)$
2. $Dec(SK_F, ct)$

- Multi-input Functional Encryption ⇒ obfuscation [AJ15]
  - n+1-slot MIFE ⇒ n-bit input obfuscation

Obf(F)

SK_U

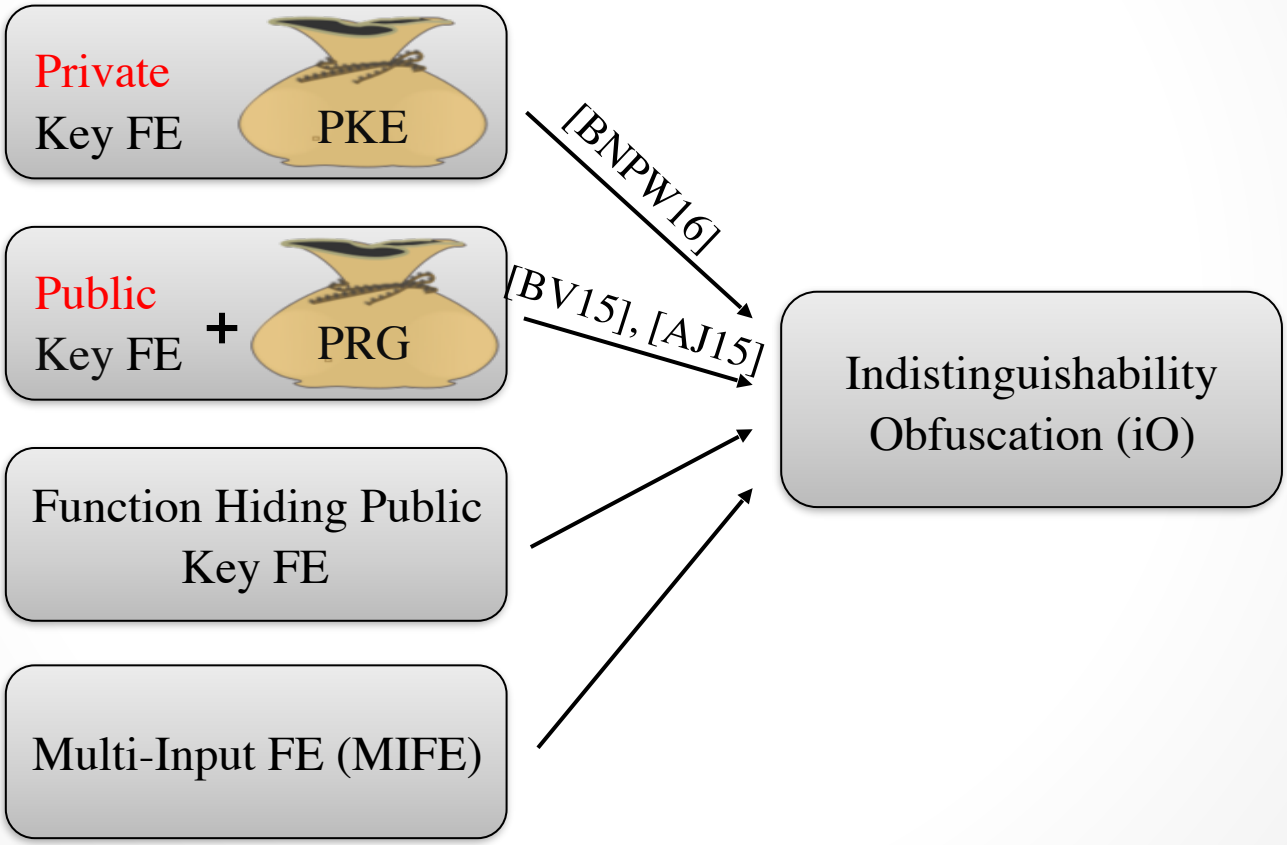
$Enc^{(0)}(F)$ $Enc^{(1)}(0)$ … $Enc^{(n)}(0)$

$Enc^{(1)}(1)$ … $Enc^{(n)}(1)$

$Eval_{Obf(F)}(m_1 \ldots m_n)$

$Dec\left(SK_u, Enc^{(0)}(F), Enc^{(1)}(m_1) \ldots Enc^{(n)}(m_n)\right)$

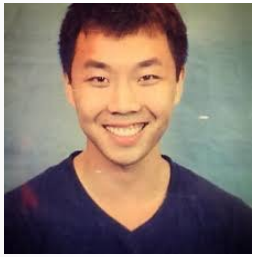
Obfuscation and FE

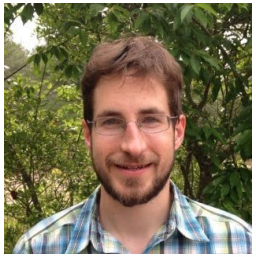


MMAPs Efficiency

Can we implement any of this?

FHE Efficiency

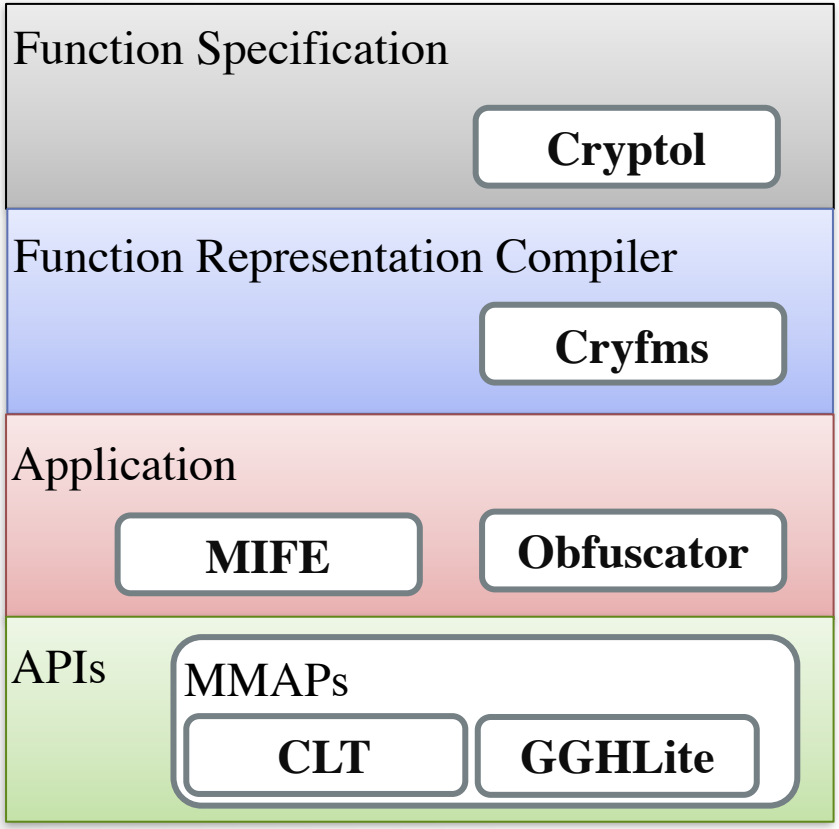


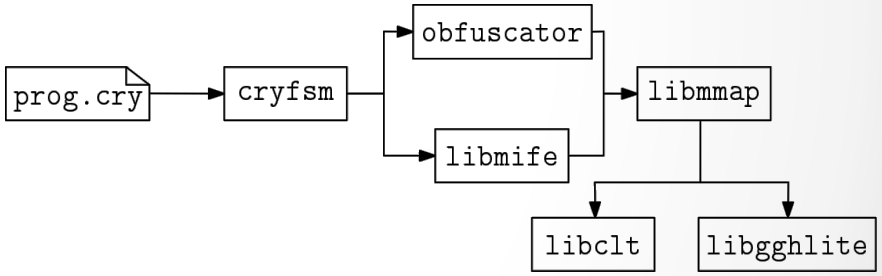


Summary of Results

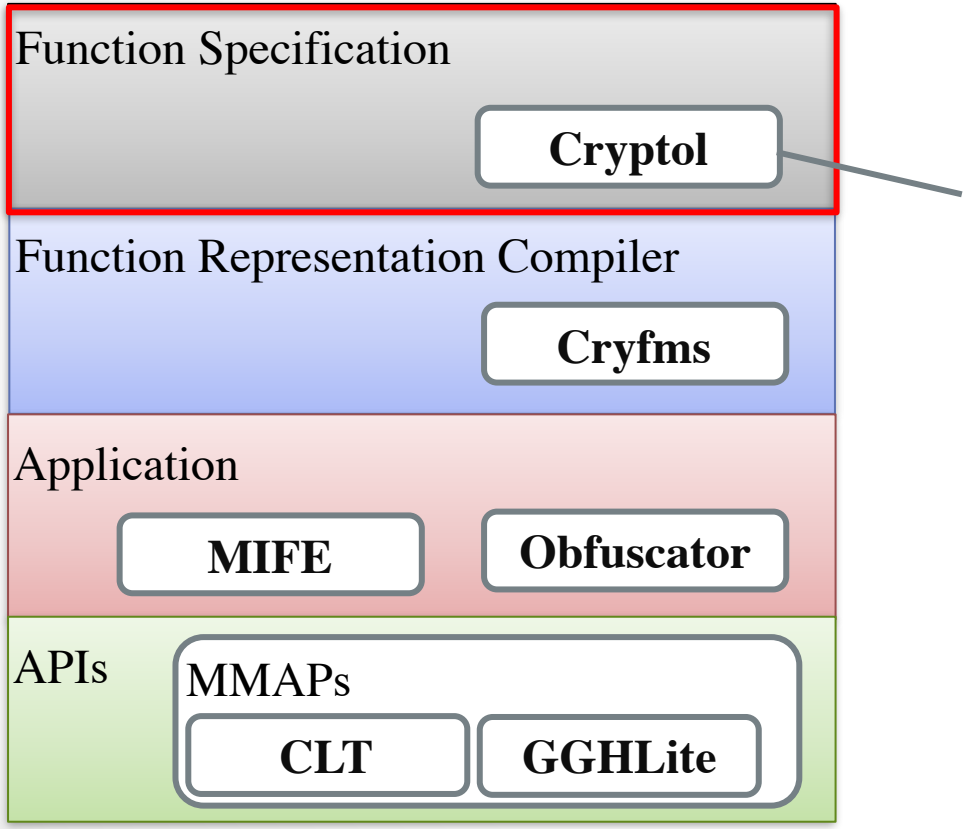
- **A framework for prototyping applications using multilinear maps and branching programs**
- Compiler from Cryptol to matrix branching programs (MBPs)
- API with multilinear map implementations
  - GGH Lite
  - CLT13 (no public encoding)
- Applications
  - Multi-input functional encryption (order revealing encryption (ORE))
  - Point function obfuscation
- Efficiency Evaluation

5Gen Architecture





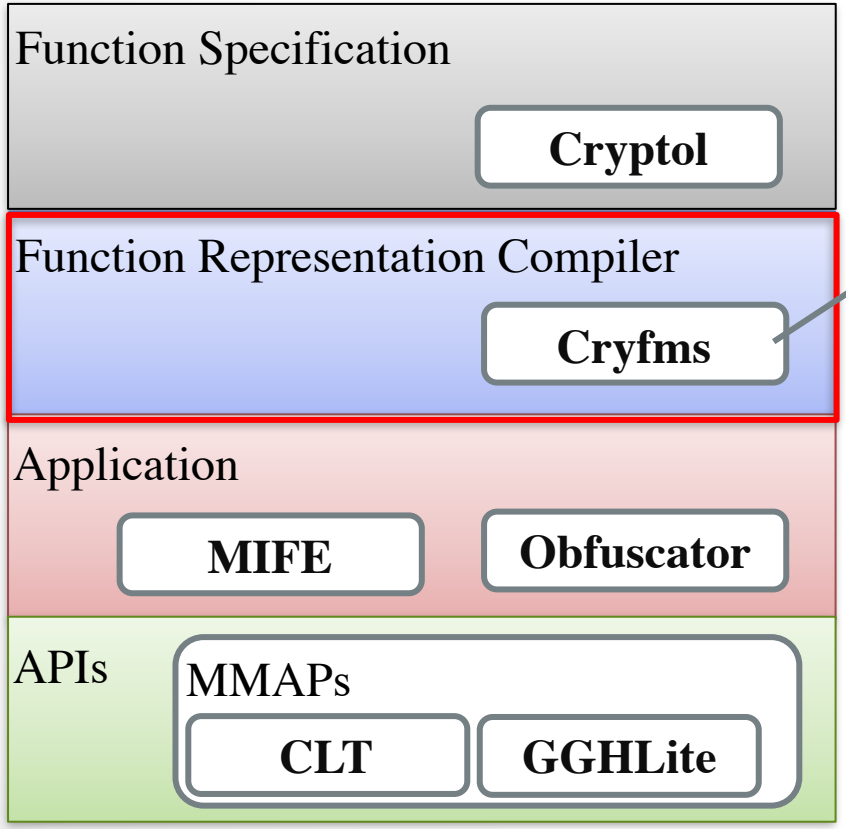
Function Representation



Language for describing cryptographic algorithms

- Executable format
- Formal property proofs for specification
- Prove equivalence with or generate implementations in other languages

Compiler



Compiler from Cryptol to minimal finite state machine and then MBP

- FSM optimizations – track reachable states
- MBP oprimizations

Branching Programs [B86]

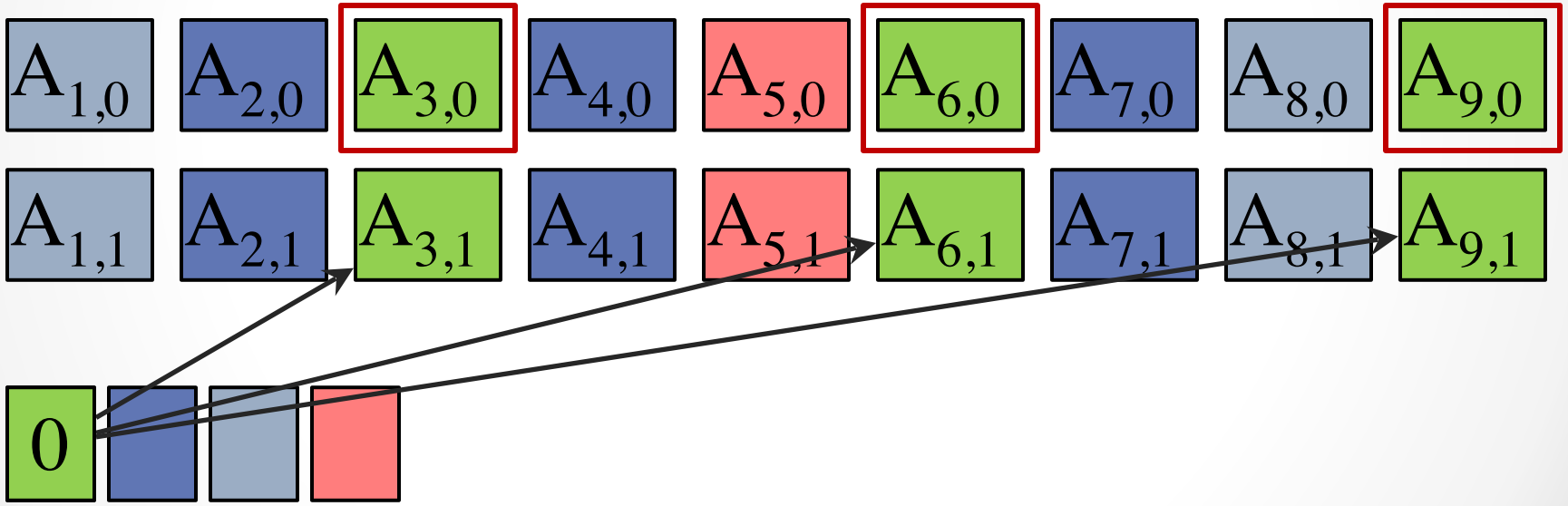
- BP of length *m* with *n* input bits is defined as
  $(inp(1), A_{1,0}, A_{1,1}), (inp(2), A_{2,0}, A_{2,1}), \ldots, (inp(m), A_{m,0}, A_{m,1})$
  - $A_{i,0}, A_{i,1} \in \{0,1\}^{k \times k}$
  - $inp(x) : [m] \rightarrow [n]$

- BP for *F* evaluates on input $x = (x_1, \ldots, x_n)$

$$F(x) = \begin{cases} 1, & \text{if } \prod_{i=1}^{n} A_{i,inp(i)} = I \\ 0, & \text{otherwise.} \end{cases}$$

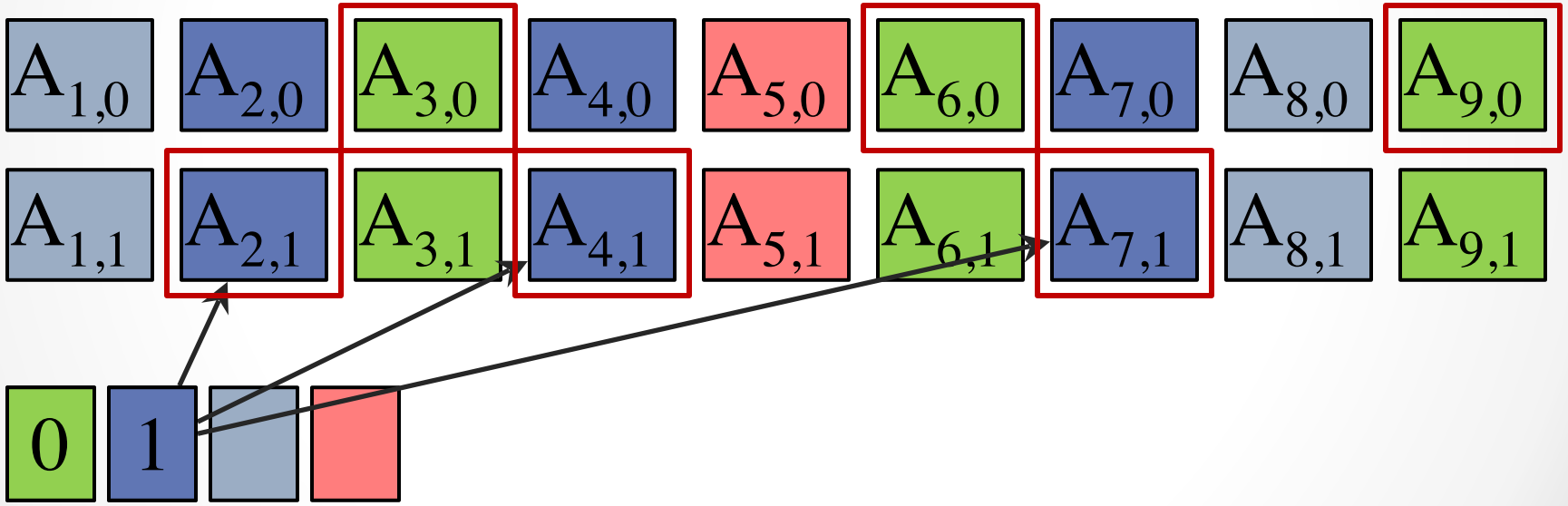
Branching Programs

- Example: BP of length 9 with 4-bit inputs



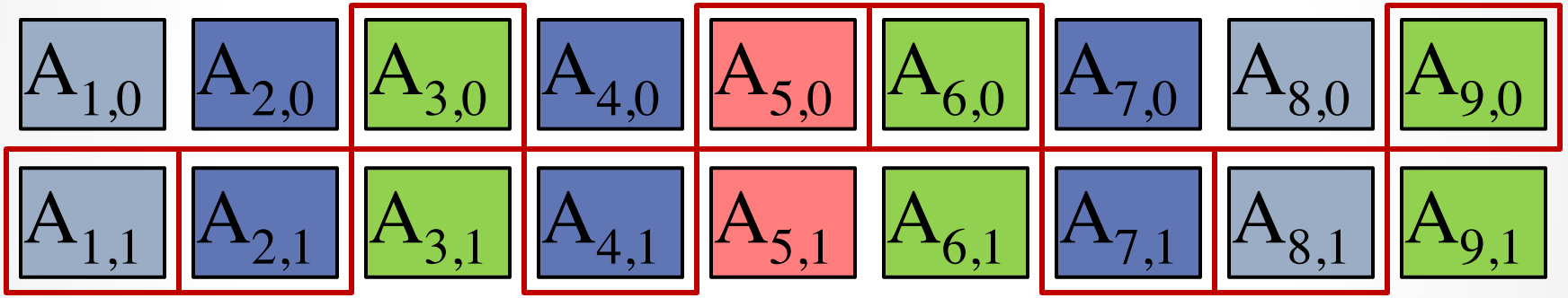
Branching Programs

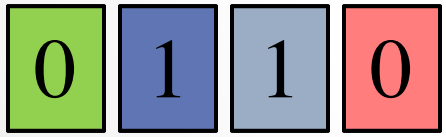
- Example: BP of length 9 with 4-bit inputs



Branching Programs

- Example: BP of length 9 with 4-bit inputs



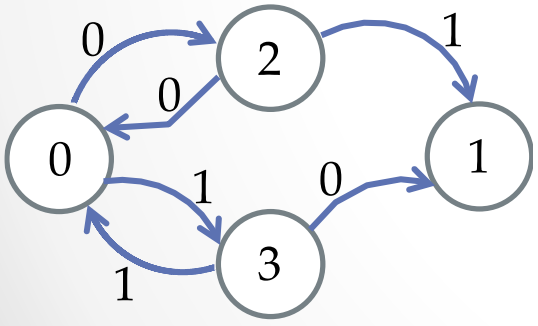


- Multiply the chosen 9 matrices.
  - If the product is I, output 1. Otherwise, output 0.

How to construct MBP?

- Barrington's construction: circuit to MBP
  - Size of the MBP is exponential in the circuit depth

- Finite State Automaton to MBP

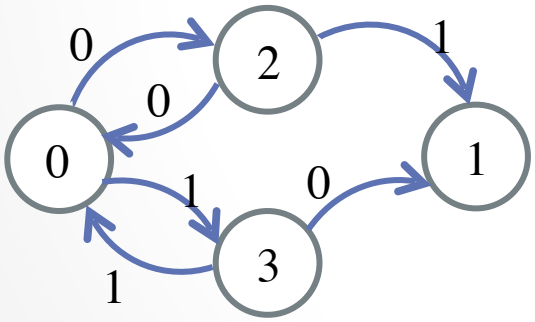


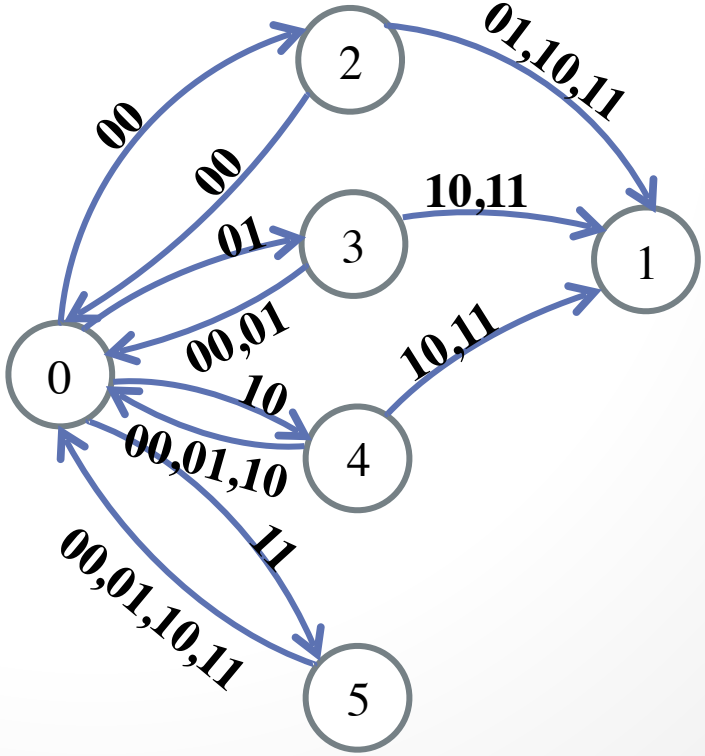
$$A_{1,0} = \begin{array}{c} \begin{matrix} 0 & 1 & 2 & 3 \end{matrix} \\ \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 \\ \mathbf{1} & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & \mathbf{1} \end{pmatrix} \end{array} \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \end{matrix} \qquad A_{2,0} = \begin{pmatrix} \mathbf{1} & 0 & \mathbf{1} & 0 \\ 0 & \mathbf{1} & 0 & \mathbf{1} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$A_{1,1} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 \\ \mathbf{1} & 0 & 0 & \mathbf{1} \end{pmatrix} \qquad A_{2,1} = \begin{pmatrix} \mathbf{1} & 0 & 0 & \mathbf{1} \\ 0 & \mathbf{1} & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Optimizations for MBP

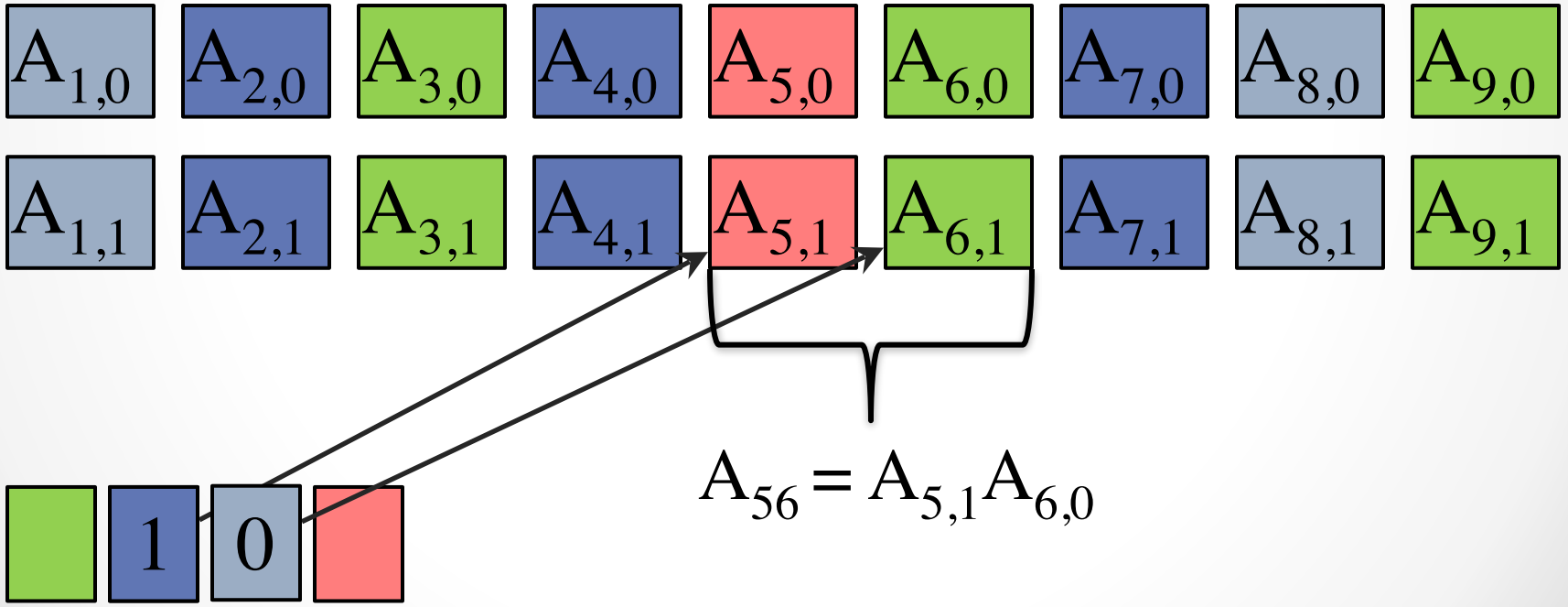
- Condensing the input representation
  - larger transition symbols alphabets (e.g. digits in larger base) – fewer transitions, more states





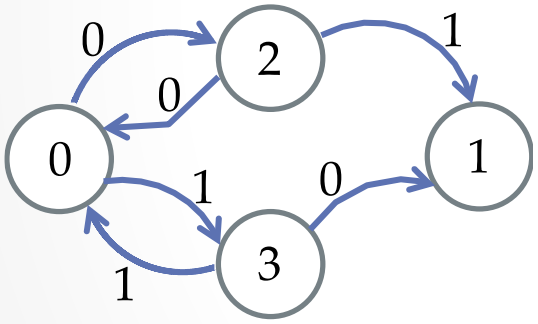
Optimizations for MBP

- Matrix premultiplication



Optimizations for MBP

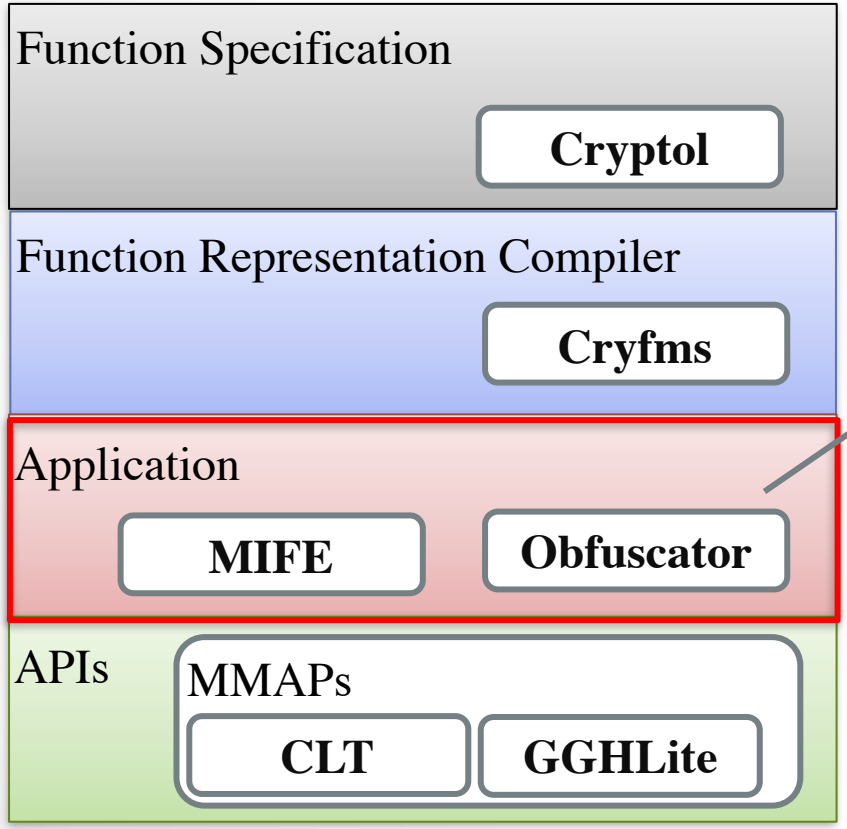
- Dimension reducing – for some bit positions the FSM transitions affect only some of the states



$$A_{1,0} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 \\ \mathbf{1} & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & \mathbf{1} \end{pmatrix} \qquad A_{2,0} = \begin{pmatrix} \mathbf{1} & 0 & \mathbf{1} & 0 \\ 0 & \mathbf{1} & 0 & \mathbf{1} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

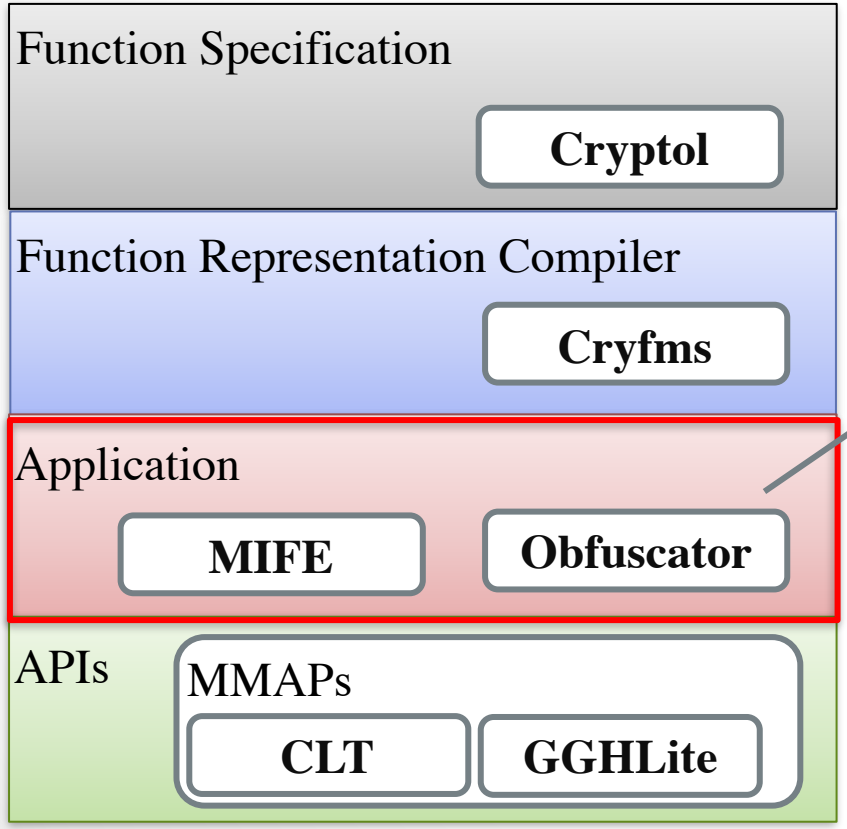
$$A_{1,1} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 \\ \mathbf{1} & 0 & 0 & \mathbf{1} \end{pmatrix} \qquad A_{2,1} = \begin{pmatrix} \mathbf{1} & 0 & 0 & \mathbf{1} \\ 0 & \mathbf{1} & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Applications



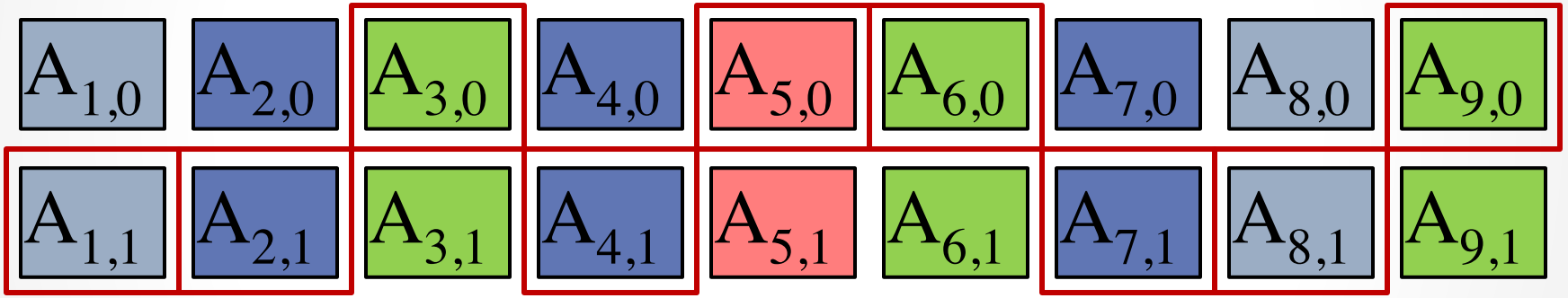
- MIFE
  - Order revealing encryption (ORE)
- Obfuscation
  - VBB point function obfuscation

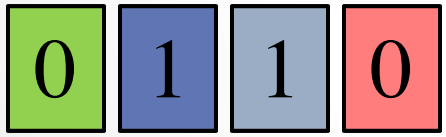
Applications



- MIFE
  - Order revealing encryption (ORE)
- **Obfuscation**
  - **VBB point function obfuscation**

Branching Programs

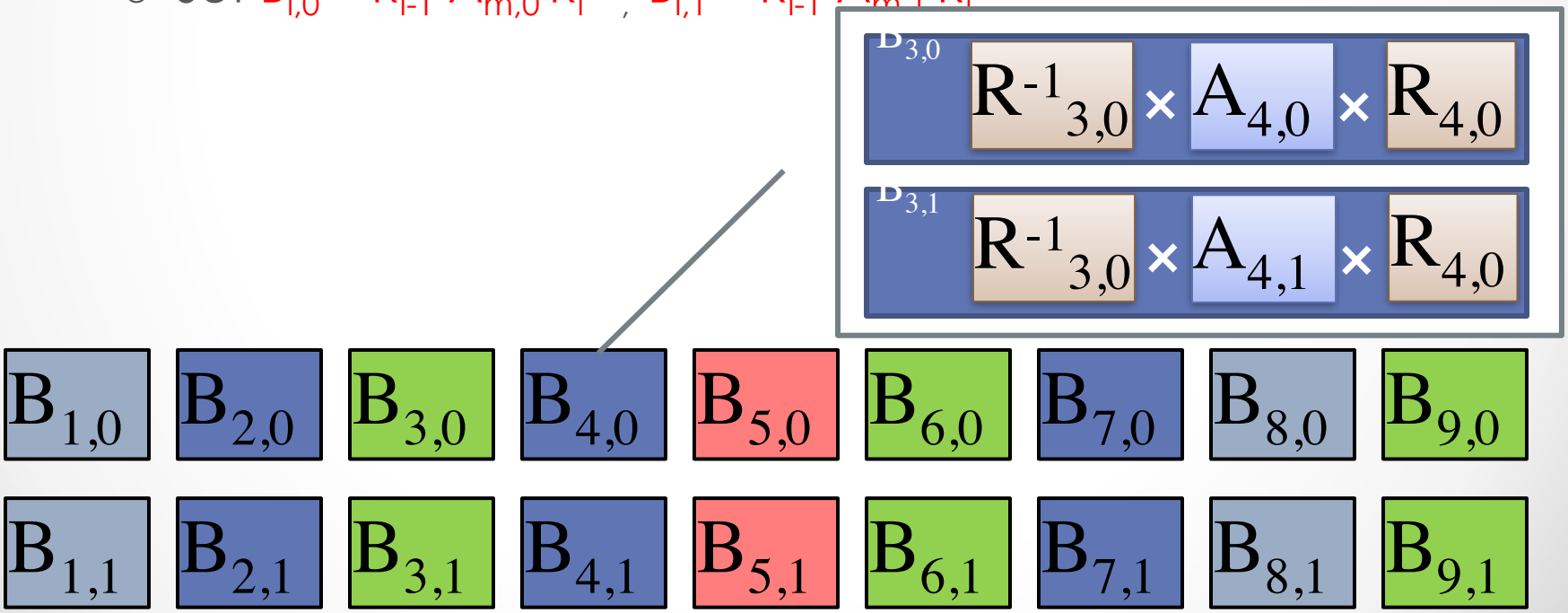




- Multiply the chosen 9 matrices.
  - If the product is I, output 1. Otherwise, output 0.

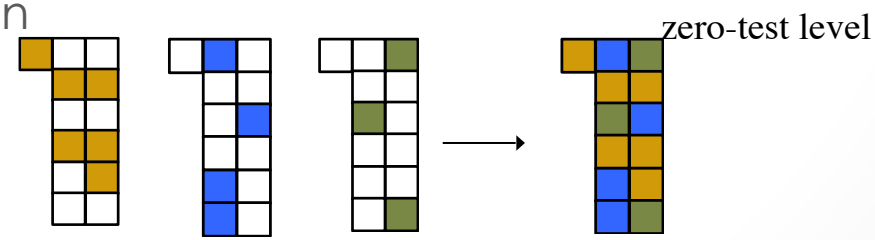
Obfuscation

- Randomized branching programs(RBP) [Kilian88]:
  - BP: $(inp(1), A_{1,0}, A_{1,1}), \ldots, (inp(m), A_{m,0}, A_{m,1})$
  - Sample invertible matrices $R_0, \ldots, R_m \in \{0,1\}^{k \times k}$
  - Set $B_{i,0} = R_{i-1} A_{m,0} R_i^{-1}$, $B_{i,1} = R_{i-1} A_{m,1} R_i^{-1}$



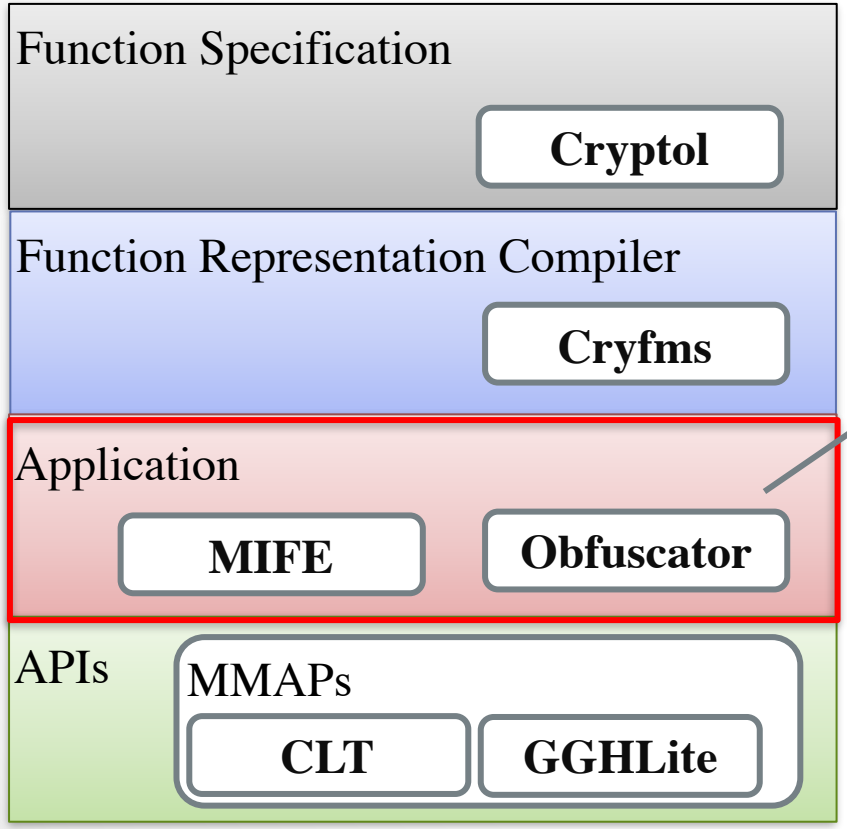
Obfuscation

- Multilinear maps (mmaps) [BS02]: $[a]_S$
  - $[a]_S + [b]_S = [a+b]_S$
  - $[a]_S + [b]_T = [ab]_{S \cup T}$
  - Zero test for level U – **is $[a]_U$ an encoding of 0?**
  - Candidate constructions and attacks
- Encode the randomized branching program matrices (each of their entries) with multilinear maps (mmaps)
  - Straddling encoding level sets – enforce input consistency, prevent partial evaluation



- Evaluate and zero-test

Applications

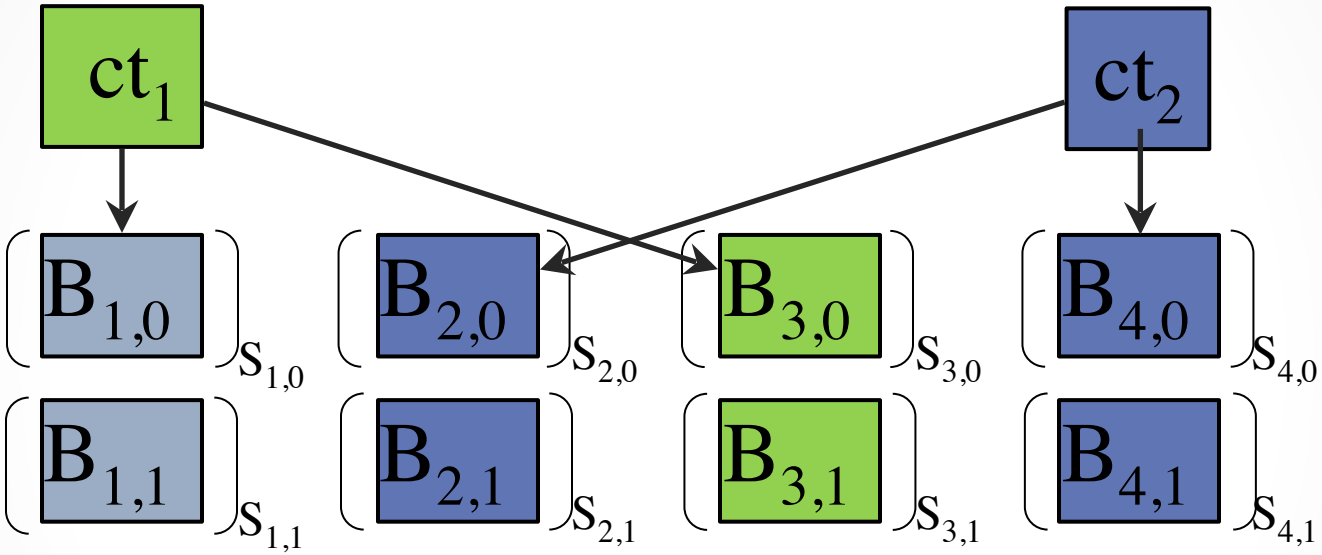


- **MIFE**
  - **Order revealing encryption (ORE)**
- Obfuscation
  - VBB point function obfuscation

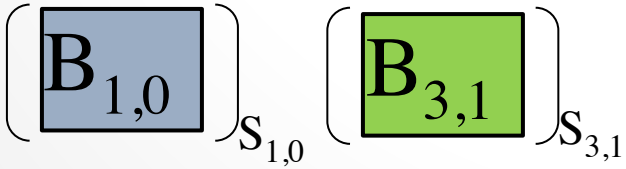
Private Single Key MIFE

- Setup – generate:
  - Master secret key MSK
  - Decryption key SK_F for multi-input function $F(x_1, \ldots, x_n)$
- Encryption
  - $ct^{(i)} = Enc^{(i)}(MSK, x_i)$
- Decryption
  - $Dec(SK_F, ct^{(1)}, \ldots, ct^{(n)}) = F(x_1, \ldots, x_n)$

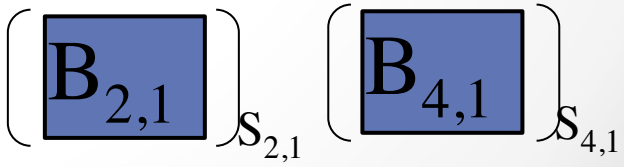
MIFE Construction [BLRSZZ15]



$\mathrm{Enc}^{(1)}(01)$:



$\mathrm{Enc}^{(2)}(11)$:



Mix-and-Match Attacks

$\mathrm{Enc}^{(1)}(01)$: $\left[B_{1,0} \right]_{S_{1,0}} \left[B_{3,1} \right]_{S_{3,1}}$

$\mathrm{Enc}^{(1)}(10)$: $\left[B_{1,1} \right]_{S_{1,1}} \left[B_{3,0} \right]_{S_{3,0}}$

$\mathrm{Enc}^{(1)}(11)$: $\left[B_{1,1} \right]_{S_{1,1}} \left[B_{3,1} \right]_{S_{3,1}}$

MIFE as a constrained formed of obfuscation

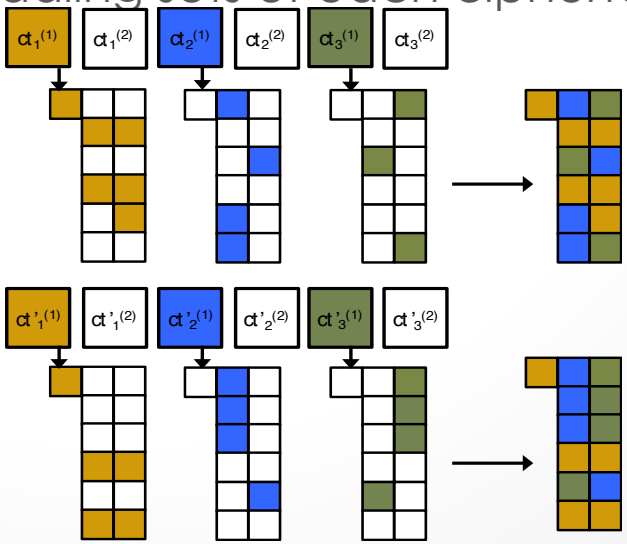
Exclusive Partition Families

- $(2^{\lambda},d)$- exclusive partition family;
  Let $\rho(i)$ be the binary representation of i
  - $U = (a_1, \ldots, a_d, b_{1,1}, \ldots, b_{2,\lambda}, \ldots, b_{d,1}, \ldots, b_{d,\lambda})$
  - $S_{i,1} = \{a_1\} \cup_{j \in [2,d]} \{b_{j,k} : \rho(i)[k] = 1\}$
  - $S_{i,j} = \{a_j\} \cup \{b_{j,k} : \rho(i)[k] = 0\}$

$i = 1001\ (9)$

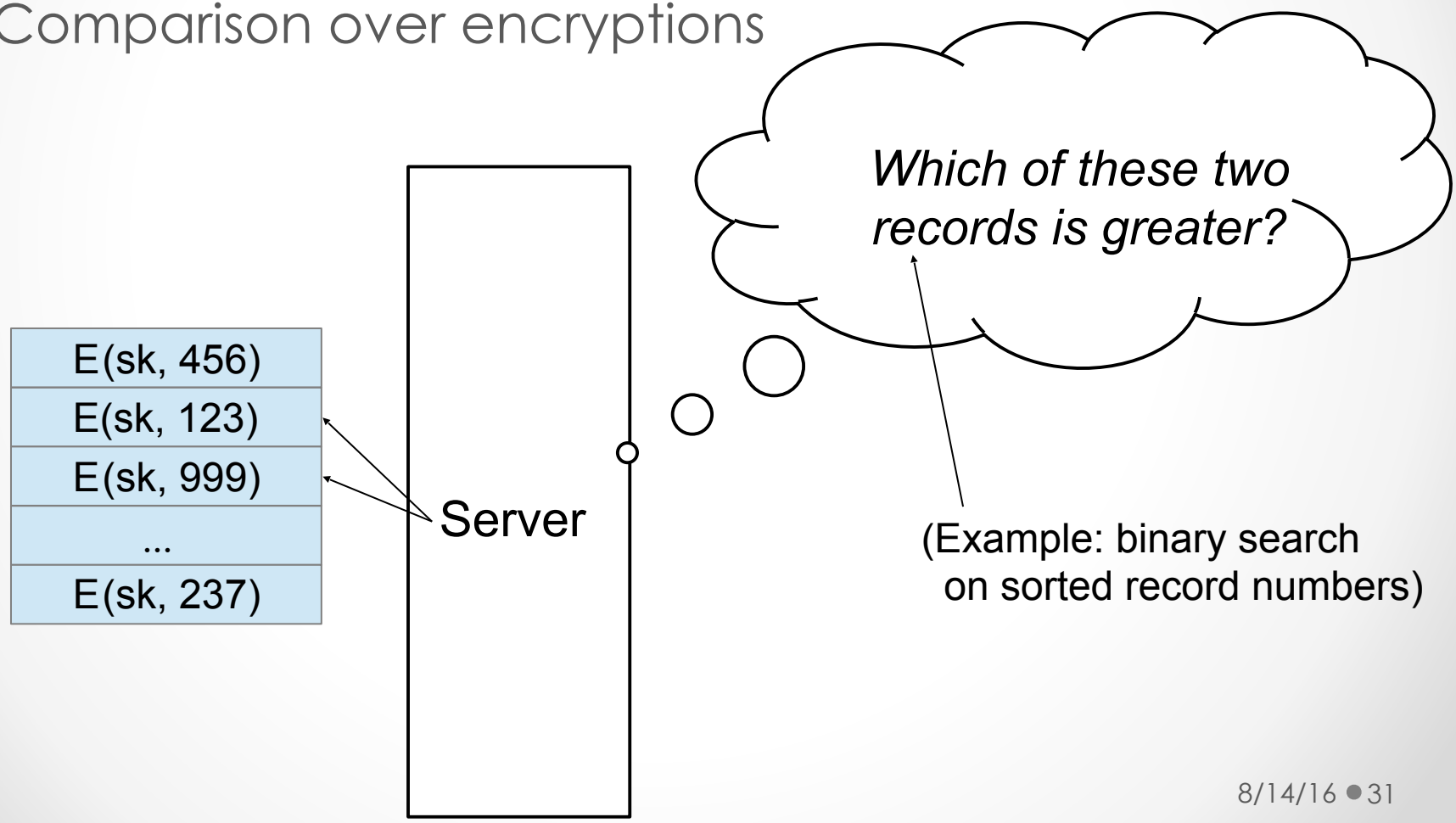
a_1	a_2	a_3	a_4
	$b_{2,1}$	$b_{3,1}$	$b_{4,1}$
	$b_{2,2}$	$b_{3,2}$	$b_{4,2}$
	$b_{2,3}$	$b_{3,3}$	$b_{4,3}$
	$b_{2,4}$	$b_{3,4}$	$b_{4,4}$

- Use a partition sampled at random from the exponential size set $\{ (S_{i,1}, \ldots, S_{i,d}) \}_{i \in [1,2^{\lambda}]}$ for the straddling sets of each ciphertext



Order-Revealing Encryption(ORE)

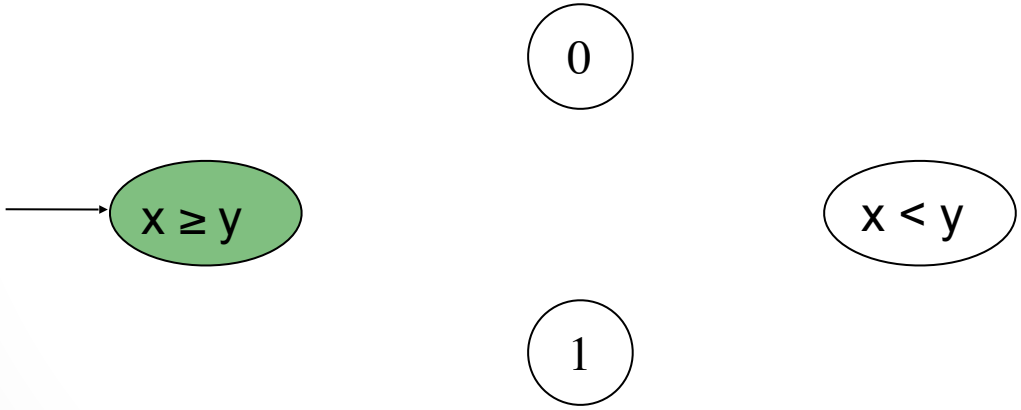
- Comparison over encryptions



Overview of Techniques

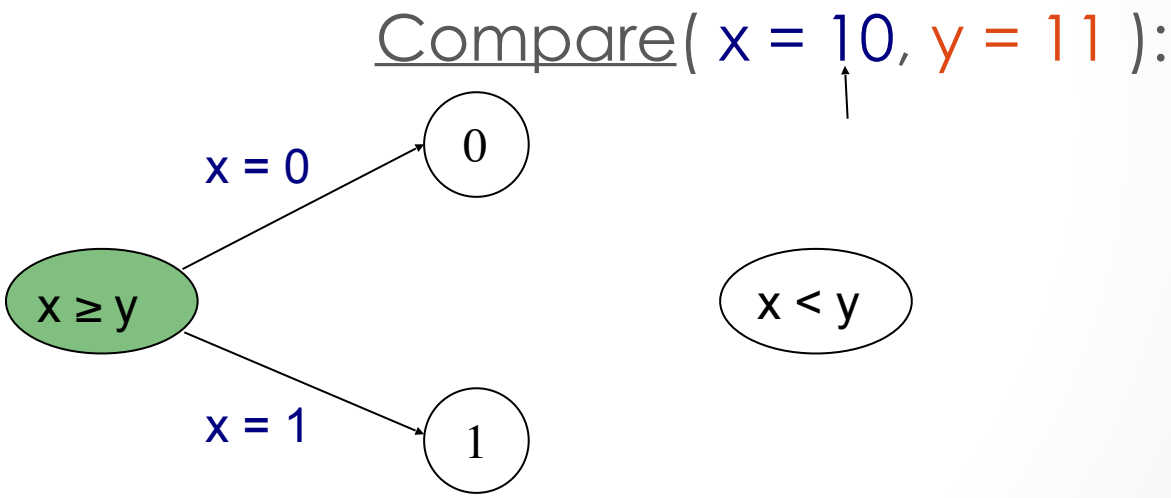
- View comparison function as (multi-input) finite automaton

Compare(x = 10, y = 11):



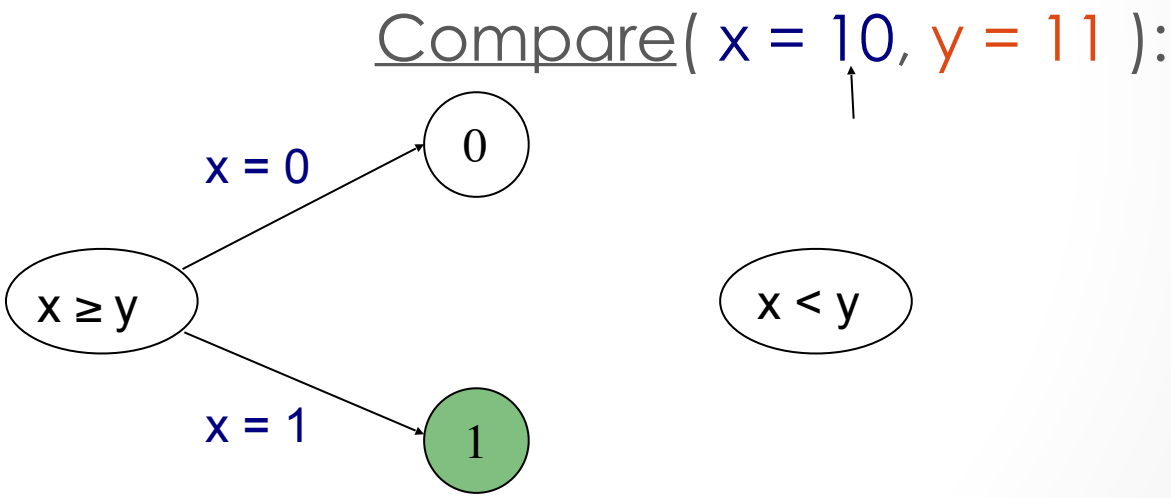
Overview of Techniques

- View comparison function as (multi-input) finite automaton



Overview of Techniques

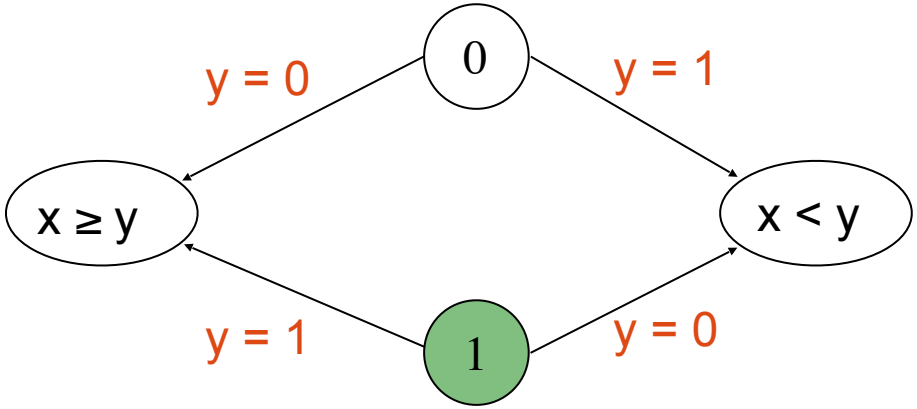
- View comparison function as (multi-input) finite automaton



Overview of Techniques

- View comparison function as (multi-input) finite automaton

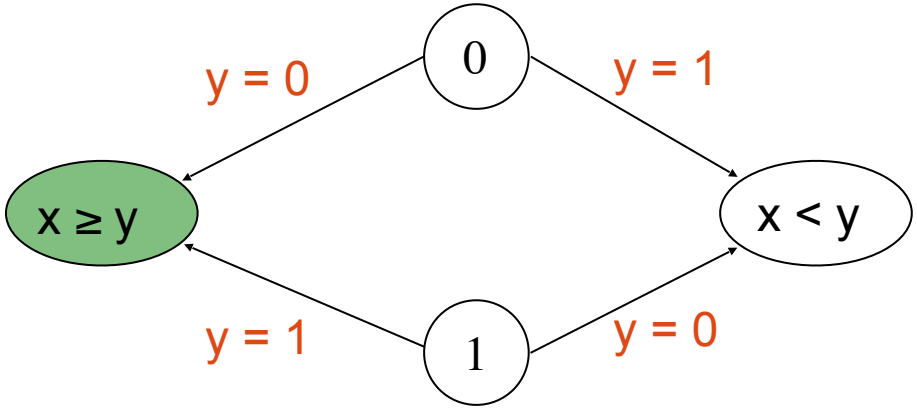
Compare(x = 10, y = 11):



Overview of Techniques

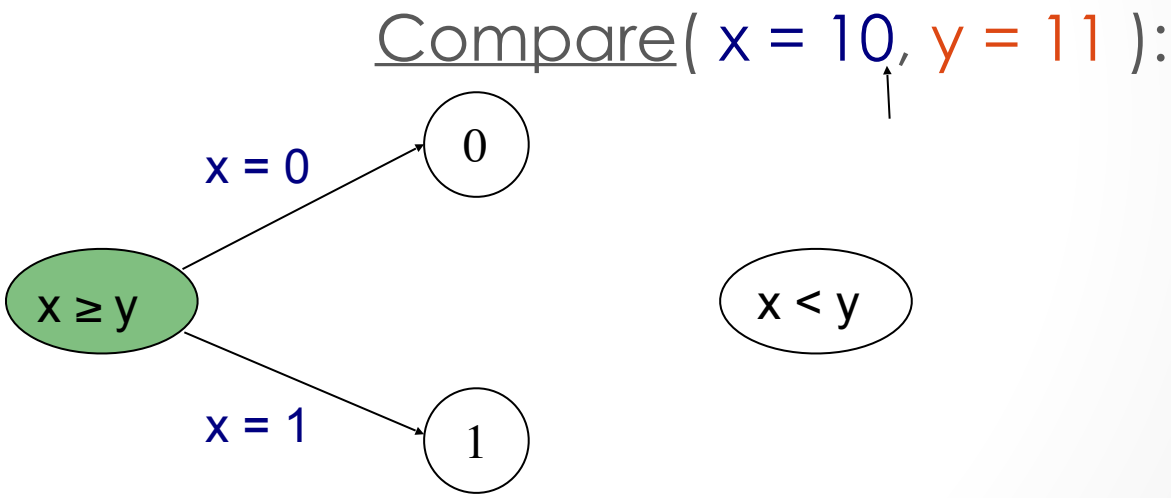
- View comparison function as (multi-input) finite automaton

Compare(x = 10, y = 11):



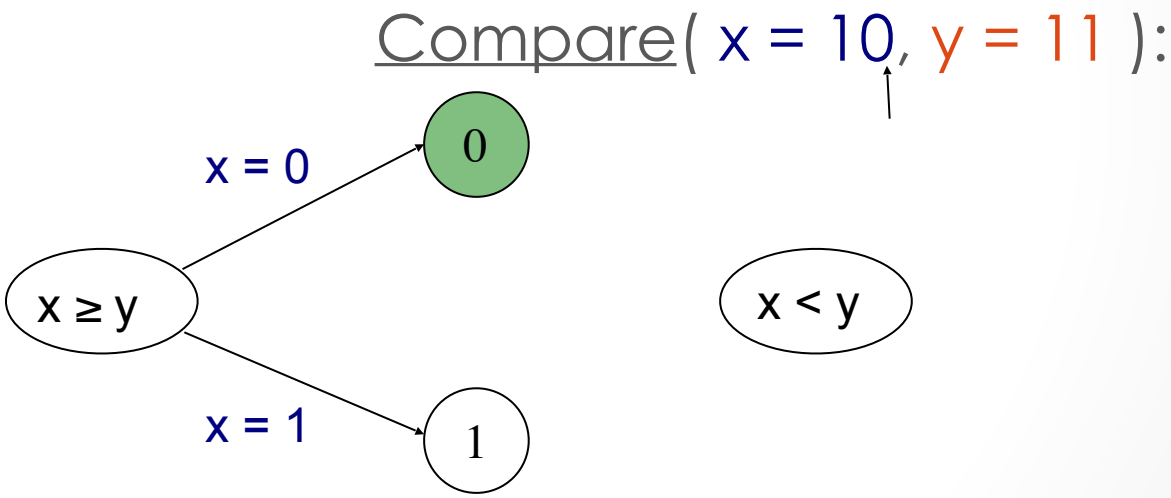
Overview of Techniques

- View comparison function as (multi-input) finite automaton



Overview of Techniques

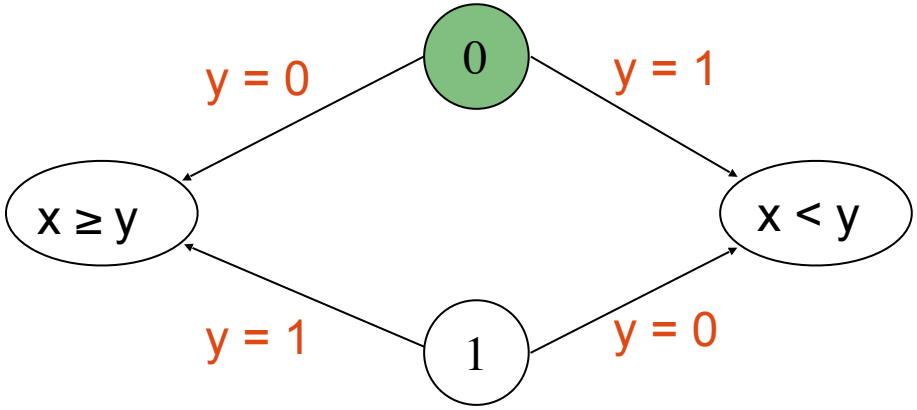
- View comparison function as (multi-input) finite automaton



Overview of Techniques

- View comparison function as (multi-input) finite automaton

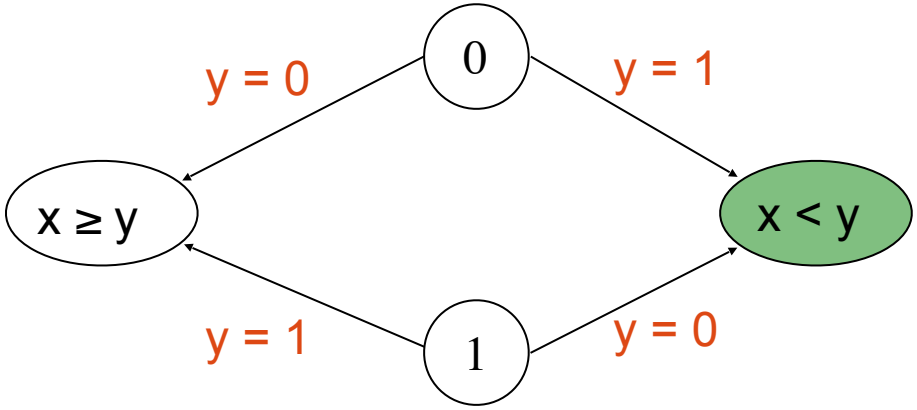
Compare(x = 10, y = 11):



Overview of Techniques

- View comparison function as (multi-input) finite automaton

Compare(x = 10, y = 11):



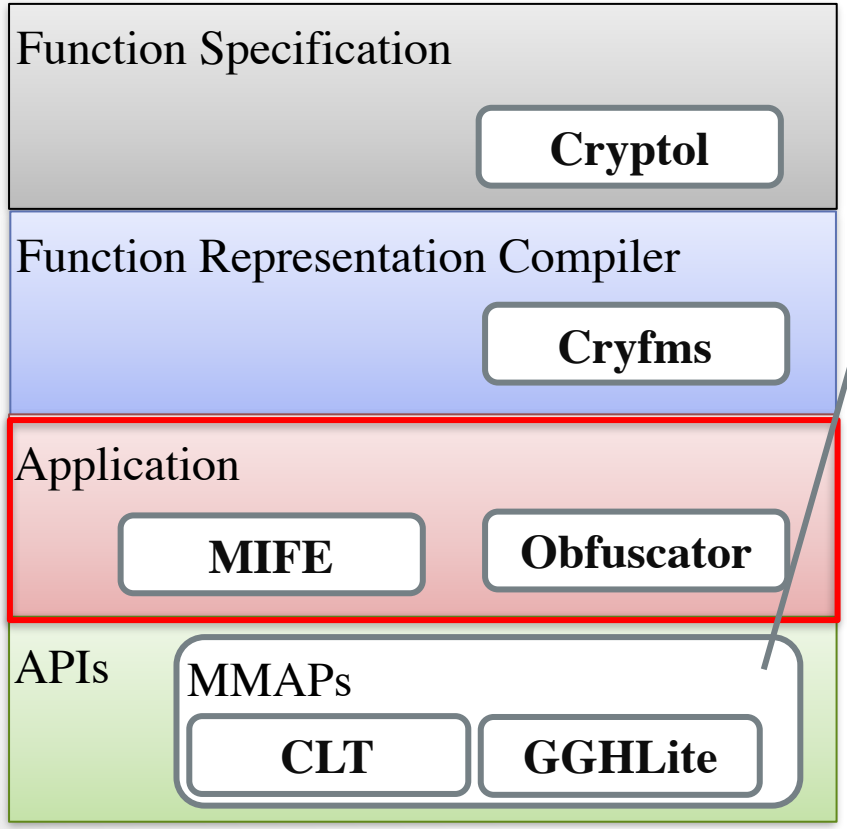
MIFE Instantiation

- 3-input DNF encryption

$$\mathsf{3DNF}(x, y, z) = (x_1 \wedge y_1 \wedge z_1) \vee \cdots \vee (x_n \wedge y_n \wedge z_n) \in \{0, 1\}$$

- Resemble “tribes” function
- Application: fraud detection

Applications



- MMAPs
  - GGHLite
  - CLT

vtable	function	comments
mmap_pp_vtable	fread/fwrite	read/write public parameters
	clear	clear public params
mmap_sk_vtable	init/clear	initialize/clear secret key
	fread/fwrite	read/write secret key
mmap_enc_vtable	init/clear	initialize/clear encoding
	fread/fwrite	read/write encoding
	set	copy encoding
	add	implements Add
	mul	implements Mult
	is_zero	implements ZeroTest
	encode	implements Encode

Multilinear Maps I

- **GGHLite**
  - Mmaps based on **ideal lattices**
  - A modification with parameter optimizations of [GGH13]
  - **Our implementation:**
    - Starting point: implementation of [ACLL15]
    - Extending capabilities of labelling encoding levels with respect to any subset of the zero-testing universe
    - Disk i/o for parameters
    - Parameter setting: follow [ACLL15]
  - *Attacks:* annihilation attacks [MSZ15]

Multilinear Maps II

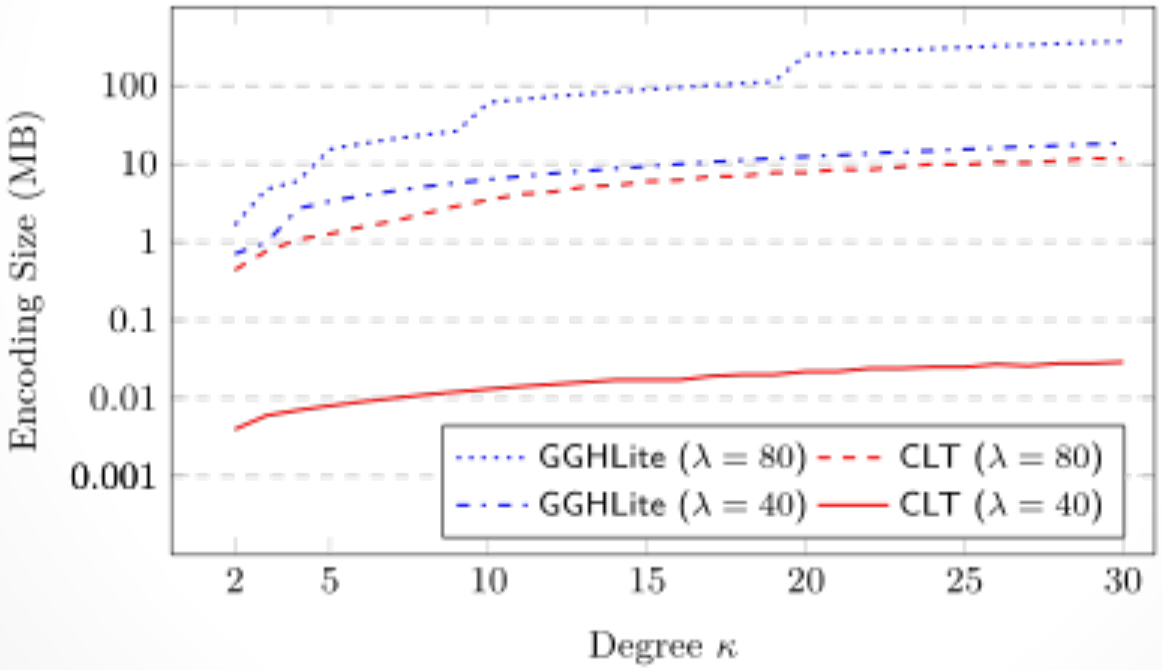
- **CLT**
  - Mmaps over the integers
  - [CLT13] without re-randomization public parameters
  - **Our implementation:**
    - Starting point implementation of [CLT13]
    - Encoding speedup - tree for computation of CRT
    - Disk i/o for parameters
    - Parameter setting: concrete estimates
  - *Attacks*: existing attacks with or without encodings of zero do not directly break our applications

Evaluation

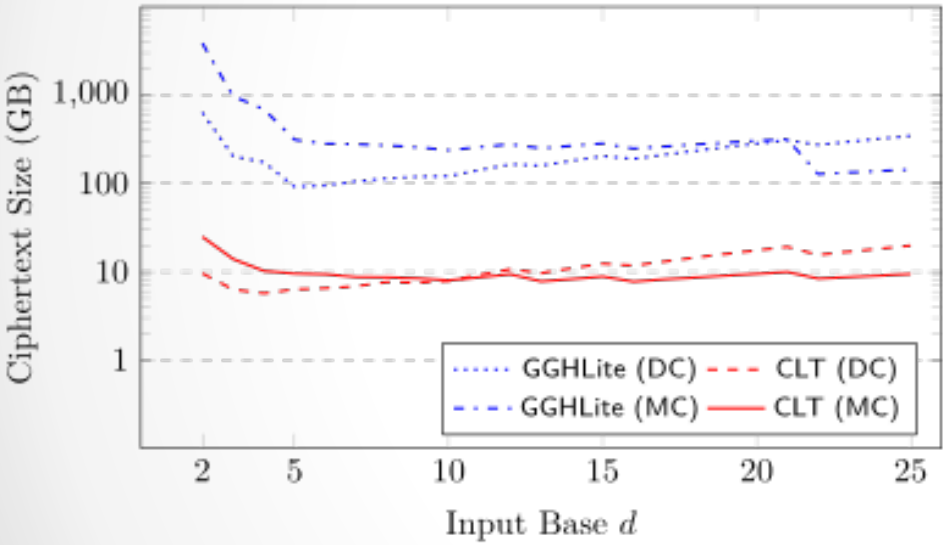
• • •

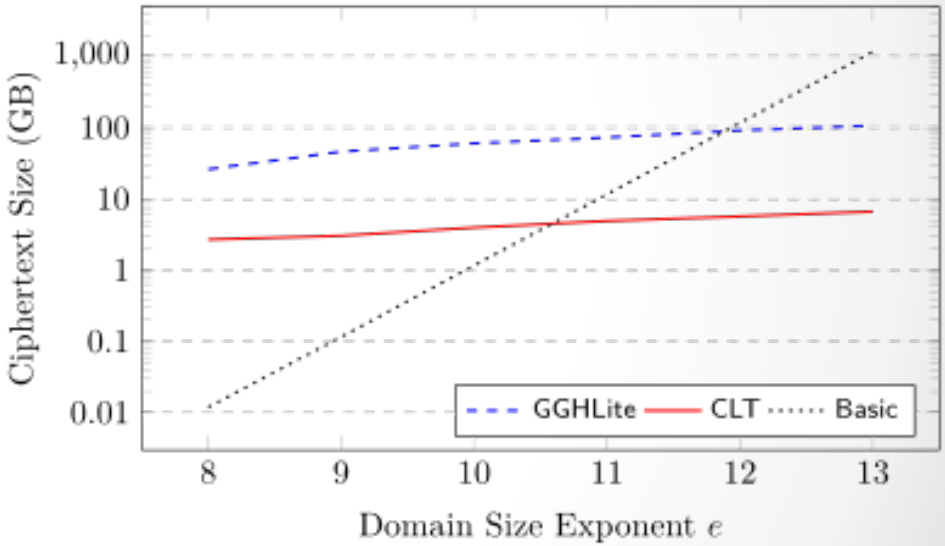
MMAP, ORE, 3DNF, Point Obfuscation

MMAP Encoding Sizes

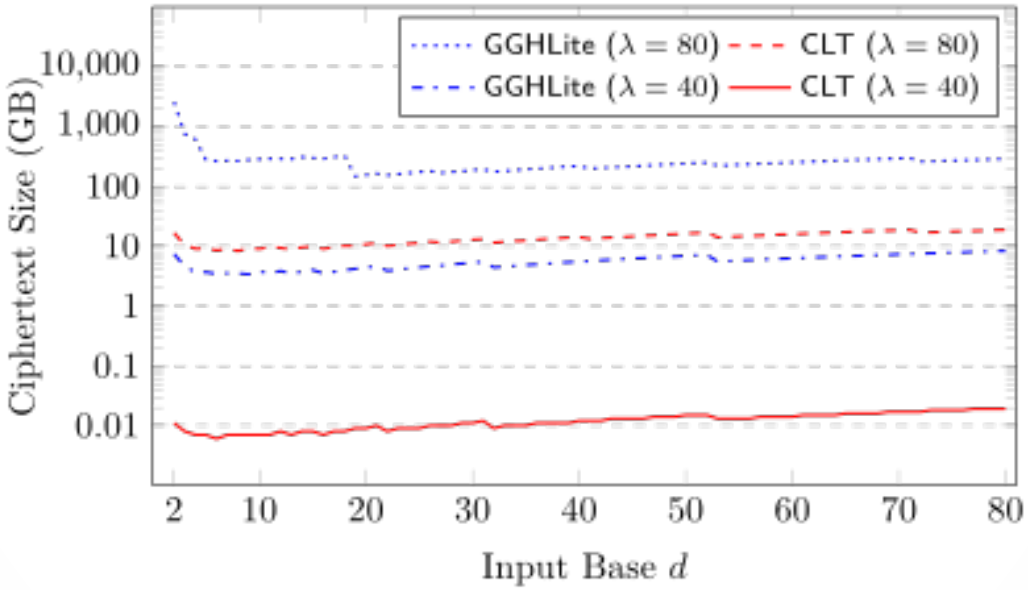


ORE Ciphertext Size





Obfuscation Size



Run-times

ORE

λ	experiment	encrypt	eval	|ct|	RAM
40	GGH (10^{12})	239 m	49 s	9.7 GB	8 GB
	GGH (10^{10})	198 m	43 s	6.4 GB	7 GB
	CLT (10^{12})	35 m	81 s	1.6 GB	20 GB
	CLT (10^{10})	24 m	54 s	1.0 GB	17 GB
80	GGH (10^{12})	151 h	8 m	92 GB	78 GB
	GGH (10^{10})	109 h	6 m	61 GB	72 GB
	CLT (10^{12})	489 m	8 m	7.3 GB	192 GB
	CLT (10^{10})	312 m	5 m	4.6 GB	166 GB

Obfuscation

λ	experiment	obf	eval	|obf|	RAM
40	GGH (40-bit)	1.8 h	6.8 s	3.6 GB	82 GB
	CLT (40-bit)	5.9 m	3.6 s	0.4 GB	2.7 GB
	GGH (80-bit)	3.0 h	29.7 s	13.7 GB	116 GB
	CLT (80-bit)	1.3 h	26.0 s	2.5 GB	20.5 GB
80	CLT (80-bit)	4.2 h	99.9 s	11.7 GB	227 GB

Conclusions

- **Framework for experimentation with mmaps and applications**
  - Modular – different parts can be used independently
- Evaluation of the efficiency of the state of the art

Thank you!