Hash functions, program secrets and lattices

Giorgos Zirdelis

University of Maryland

giorgos@umd.edu

Talk outline

Topic is lattice-based cryptography

- Hash Functions
- Program Obfuscation
- ...

Common theme: Quest for "*universal*" tools

Cryptographic Hash Functions

Hash functions

- Hash functions are used everywhere in cryptography
  - Both in theory and practice
  - Hash-and-Sign, Merkle tree, ₿, ...

    - SHA-2, SHA-3
    - Factoring
    - Discrete Log
    - Elliptic Curves
    - Isogeny-based
    - Lattice-based

Hash functions

- Hash functions are used everywhere in cryptography
  - Both in theory and practice
  - Hash-and-Sign, Merkle tree, ฿, ...

  - SHA-2, SHA-3
  - Factoring
  - Discrete Log
  - Elliptic Curves
  - Isogeny-based
  - Lattice-based

Goal: Given h, find $x \neq x'$ s.t. $h(x) = h(x')$

Security: such x, x' always exist but are hard to find

Hash functions

- Hash functions are used everywhere in cryptography
  - Both in theory and practice
  - Hash-and-Sign, Merkle tree, ₿, ...

  - SHA-2, SHA-3
  - Factoring
  - Discrete Log
  - Elliptic Curves
  - Isogeny-based
  - Lattice-based

Goal: Given h, find $x \neq x'$ s.t. $h(x) = h(x')$

Security: such x, x' always exist but are hard to find

- Which hash function is most secure?

  Provably answer this, at least in theory?

Most secure?

- What does most secure mean for some $\overset{\text{♛}}{\overline{h}}$?

Most secure?

- What does most secure mean for some $\overset{♛}{h}$?
- Collisions for $\overset{♛}{h}$ must be as hard, as in any other h

Most secure?

- What does most secure mean for some $\overset{♛}{h}$?
- Collisions for $\overset{♛}{h}$ must be as hard, as in any other h
- Implies a reduction: $\forall h, \;\; h \leq \overset{♛}{h}$
  - for fixed security parameter
  - $\overset{♛}{h}$ inherits hardness from all h

Most secure?

- What does most secure mean for some $\overset{♛}{h}$?

- Collisions for $\overset{♛}{h}$ must be as hard, as in any other h

- Implies a reduction: $\forall h, \;\; h \leq \overset{♛}{h}$
  - for fixed security parameter
  - $\overset{♛}{h}$ inherits hardness from all h

- What are the reduction steps?

Reduction steps $h \to C_h \to \overline{h}$

Hash function h

Reduction steps $h \to C_h \to \overline{h}$

Hash function h

1. Represent h in a universal way (e.g. use boolean circuits)
   - $h \to C_h$

Reduction steps $h \to C_h \to \overset{\text{♛}}{\overline{h}}$

Hash function h

1. Represent h in a universal way (e.g. use boolean circuits)
   - $h \to C_h$

2. Reduce C_h to a hash function $\overset{\text{♛}}{\overline{h}}$
   - $C_h \leq \overset{\text{♛}}{\overline{h}}$
   - Find collisions in $\overset{\text{♛}}{\overline{h}}$ $\Longrightarrow$ Find collisions in $C_h, \forall h$

Reduction steps $h \to C_h \to \overline{h}$

Hash function h

1. Represent h in a universal way (e.g. use boolean circuits)
   - $h \to C_h$
2. Reduce C_h to a hash function $\overline{h}$
   - $C_h \leq \overline{h}$
   - Find collisions in $\overline{h}$ $\implies$ Find collisions in $C_h, \forall h$
3. Declare $\overline{h}$ as the most secure (WC/AVG)

Reduction steps $h \to C_h \to \overset{♛}{h}$

Hash function h

1. Represent h in a universal way (e.g. use boolean circuits)
   - $h \to C_h$

2. Reduce C_h to a hash function $\overset{♛}{h}$
   - $C_h \leq \overset{♛}{h}$
   - Find collisions in $\overset{♛}{h} \implies$ Find collisions in $C_h, \forall h$

3. Declare $\overset{♛}{h}$ as the most secure (WC/AVG)

What should $\overset{♛}{h}$ be?

In the 90s...

- The question about $\overset{\text{♛}}{\overline{h}}$ was asked in [Papadimitriou '94]

  ...in the broader context of total problems (TFNP)

- It remained open, what $\overset{\text{♛}}{\overline{h}}$ to use...

  ...it all starts with the pigeonhole principle

In the 90s...

- The question about $\overset{\text{♛}}{\overline{h}}$ was asked in [Papadimitriou '94]

  ...in the broader context of total problems (TFNP)

- It remained open, what $\overset{\text{♛}}{\overline{h}}$ to use...

  ...it all starts with the pigeonhole principle

- We use it to define hash functions, prior to the reduction

The pigeonhole principle – a reminder

Any function $h : [n] \to [m]$ with $n > m$ must have collisions

i.e. when $|\text{domain}| > |\text{range}|$

$[n] = \{1, \ldots, n\}$

Define hash functions

- Define the set of **all** (poly-size) functions $h : [n] \to [m]$, with $n > m$

Define hash functions

- Define the set of **all** (poly-size) functions $h : [n] \to [m]$, with $n > m$
- i.e. all functions that compress their input
  - for convenience, we refer to these as hash functions

Define hash functions

- Define the set of **all** (poly-size) functions $h : [n] \rightarrow [m]$, with $n > m$

- i.e. all functions that compress their input
  - for convenience, we refer to these as hash functions

- compress input $\rightarrow$ collisions exist $\rightarrow$ **goal:** find collisions

Define hash functions

- Define the set of **all** (poly-size) functions $h : [n] \to [m]$, with $n > m$
- i.e. all functions that compress their input
  - for convenience, we refer to these as hash functions
- compress input $\to$ collisions exist $\to$ **goal:** find collisions
- This set is the union of:
  1. Cryptographic hash functions (e.g. SHA-3, SIS)
  2. Non-cryptographic hash functions (e.g. pairwise independence)

Represent h – (step 1) – why circuits?

Want a universal way to represent any (poly-size) hash function h

(i.e. convenient to work with)

Represent h – (step 1) – why circuits?

Want a universal way to represent any (poly-size) hash function h
(i.e. convenient to work with)

- Be agnostic of groups, rings, fields, distributions, keys*, ...
- Represent every hash function h, in the same way

Represent h – (step 1) – why circuits?

Want a universal way to represent any (poly-size) hash function h
(i.e. convenient to work with)

- Be agnostic of groups, rings, fields, distributions, keys*, ...
- Represent every hash function h, in the same way
- Use the (poly-size) boolean circuit C_h that implements h

$C_h : \{0,1\}^n \rightarrow \{0,1\}^m$ with $n > m$

n, m depend on the security parameter

* keys are hardcoded in C_h, i.e. C_{h_k} essentially

A subtle point

- By definition, $\{C_h\}$ includes **all** (poly-size) hash function circuits that map $\{0,1\}^n \to \{0,1\}^m$ with $n > m$
- Even for hash functions we might have not discovered yet!

A subtle point

- By definition, $\{C_h\}$ includes **all** (poly-size) hash function circuits that map $\{0,1\}^n \to \{0,1\}^m$ with $n > m$
- Even for hash functions we might have not discovered yet!

Note: we do not have to enumerate or explicitly know this set

The question in [Pap94] – (step 2)

- Question: Reduce C_h to a "natural" hash function $\overline{h}$?
  What is "natural" & why?

The question in [Pap94] – (step 2)

- Question: Reduce C_h to a "natural" hash function $\overset{♛}{\overline{h}}$?
  What is "natural" & why?

- Circuits vs "everyday" problems
  - NP-Hard: Circuit-SAT $\leq_p$ Subset-Sum, Clique, Vertex Cover, TSP
    i.e. "natural" problems

The question in [Pap94] – (step 2)

- Question: Reduce C_h to a "natural" hash function $\overline{h}$?
  What is "natural" & why?

- Circuits vs "everyday" problems
  - NP-Hard: Circuit-SAT $\leq_p$ Subset-Sum, Clique, Vertex Cover, TSP
    i.e. "natural" problems
- Under every C_h maybe a "natural" $\overline{h}$ like the Short Integer Solutions (SIS) is hidden...

The question in [Pap94] – (step 2)

- Question: Reduce C_h to a "natural" hash function h?
  What is "natural" & why?

- Circuits vs "everyday" problems
  - NP-Hard: Circuit-SAT $\leq_p$ Subset-Sum, Clique, Vertex Cover, TSP
    i.e. "natural" problems

- Under every C_h maybe a "natural" h like the Short Integer Solutions (SIS) is hidden...

- This would imply:

  Finding collisions in any C_h reduces to finding Short Integer Solutions!

Goal in summary

$$\underbrace{h \longrightarrow C_h}_{\text{step 1 – easy}} \longrightarrow \overset{\text{♛}}{h}$$

Goal in summary

$$h \longrightarrow \underbrace{C_h \longrightarrow \overset{♛}{h}}_{\text{step 2 – reduction}}$$

Our results (almost there)

We reduce any hash function to an *almost* lattice problem,

the constrained Short Integer Solutions problem (constrained-SIS)

- Sotiraki–Zampetakis–**Z** FOCS'18
- We believe the answer to be a lattice problem (ongoing work)
  $\overset{?}{\Rightarrow}$ lattice-based hash functions are the most secure

- We show the **first** $\overline{h}$
- Solves open problem from [Pap94]
- Our reduction is worst-case

next: SIS reminder

The SIS problem [Ajtai '96, Micciancio-Regev '04]

- Given $\mathbf{A} \leftarrow \mathbb{Z}_q^{m \times n}$ with $n > m \log q$ (s.t. collisions exist)
- Find distinct $\mathbf{x}_1, \mathbf{x}_2 \in \{0,1\}^n$ s.t.

The SIS problem [Ajtai '96, Micciancio-Regev '04]

- Given $\mathbf{A} \leftarrow \mathbb{Z}_q^{m \times n}$ with $n > m \log q$ (s.t. collisions exist)
- Find distinct $\mathbf{x}_1, \mathbf{x}_2 \in \{0,1\}^n$ s.t.

$$\boxed{\mathbf{A}} \cdot \boxed{\mathbf{x}_1} = \boxed{\mathbf{A}} \cdot \boxed{\mathbf{x}_2} \pmod q$$

The SIS problem [Ajtai '96, Micciancio-Regev '04]

- Given $\mathbf{A} \leftarrow \mathbb{Z}_q^{m \times n}$ with $n > m \log q$ (s.t. collisions exist)
- Find distinct $\mathbf{x}_1, \mathbf{x}_2 \in \{0,1\}^n$ s.t.

$$\mathbf{A} \cdot \mathbf{x}_1 = \mathbf{A} \cdot \mathbf{x}_2 \pmod q$$

...implies short $(\mathbf{x}_1 - \mathbf{x}_2) \in \{0, \pm 1\}^n$ s.t. $\mathbf{A}(\mathbf{x}_1 - \mathbf{x}_2) = \mathbf{0}$

The constrained-SIS problem

- Given $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ and semi-structured $\mathbf{G} \in \mathbb{Z}_q^{d \times n}$ with $n > (m+d)\log q$

The constrained-SIS problem

- Given $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ and semi-structured $\mathbf{G} \in \mathbb{Z}_q^{d \times n}$ with $n > (m+d)\log q$
- Find distinct $\mathbf{x}_1, \mathbf{x}_2 \in \{0,1\}^n$ s.t.

$$\boxed{\mathbf{A}} \cdot \boxed{\mathbf{x}_1} = \boxed{\mathbf{A}} \cdot \boxed{\mathbf{x}_2} \pmod{q}$$

The constrained-SIS problem

- Given $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ and semi-structured $\mathbf{G} \in \mathbb{Z}_q^{d \times n}$ with $n > (m+d)\log q$
- Find distinct $\mathbf{x}_1, \mathbf{x}_2 \in \{0,1\}^n$ s.t.

$$\mathbf{A} \cdot \mathbf{x}_1 = \mathbf{A} \cdot \mathbf{x}_2 \pmod q$$

$$\mathbf{G} \cdot \mathbf{x}_1 = \mathbf{G} \cdot \mathbf{x}_2 = \mathbf{0} \pmod q$$

constrained-SIS vs SIS

constrained-SIS (WC)	**SIS** (AVG)
$\mathbf{A}$ is arbitrary	$\mathbf{A}$ is uniformly random
$\mathbf{G}$ is semi-structured	–
$\mathbf{x}_1, \mathbf{x}_2 \in \{0,1\}^n$ s.t.	$\mathbf{x}_1, \mathbf{x}_2 \in \{0,1\}^n$ s.t.
$\mathbf{A}\mathbf{x}_1 = \mathbf{A}\mathbf{x}_2$ and $\mathbf{G}\mathbf{x}_1 = \mathbf{0} = \mathbf{G}\mathbf{x}_2$	$\mathbf{A}\mathbf{x}_1 = \mathbf{A}\mathbf{x}_2$

constrained-SIS vs SIS

constrained-SIS (WC)

$\mathbf{A}$ is arbitrary

$\mathbf{G}$ is semi-structured

$\mathbf{x}_1, \mathbf{x}_2 \in \{0,1\}^n$ s.t.

$\mathbf{A}\mathbf{x}_1 = \mathbf{A}\mathbf{x}_2$ and $\mathbf{G}\mathbf{x}_1 = \mathbf{0} = \mathbf{G}\mathbf{x}_2$

- unclear how to sample "most secure" keys

SIS (AVG)

$\mathbf{A}$ is uniformly random

–

$\mathbf{x}_1, \mathbf{x}_2 \in \{0,1\}^n$ s.t.

$\mathbf{A}\mathbf{x}_1 = \mathbf{A}\mathbf{x}_2$

- can sample keys
- unclear if SIS is the most secure h

constrained-SIS vs SIS

constrained-SIS (WC)	**SIS** (AVG)
$\mathbf{A}$ is arbitrary	$\mathbf{A}$ is uniformly random
$\mathbf{G}$ is semi-structured	–
$\mathbf{x}_1, \mathbf{x}_2 \in \{0,1\}^n$ s.t.	$\mathbf{x}_1, \mathbf{x}_2 \in \{0,1\}^n$ s.t.
$\mathbf{A}\mathbf{x}_1 = \mathbf{A}\mathbf{x}_2$ and $\mathbf{G}\mathbf{x}_1 = \mathbf{0} = \mathbf{G}\mathbf{x}_2$	$\mathbf{A}\mathbf{x}_1 = \mathbf{A}\mathbf{x}_2$
◦ unclear how to sample "most secure" keys	◦ can sample keys ◦ unclear if SIS is the most secure h

Goal: (aka reduction)

1. show that constrained-SIS is a hash function
2. reduce any hash function to constrained-SIS

Goal: constrained-SIS is a hash function – pt1

Goal:

1. show that constrained-SIS $= (\mathbf{A}, \mathbf{G})$ is a hash function

Goal: constrained-SIS is a hash function – pt1

Goal:

1. show that constrained-SIS $= (\mathbf{A}, \mathbf{G})$ is a hash function

- $\mathbf{A}$ and $\mathbf{G}$ must compress their common input

Goal: constrained-SIS is a hash function – pt1

Goal:

1. show that constrained-SIS $= (\mathbf{A}, \mathbf{G})$ is a hash function

- $\mathbf{A}$ and $\mathbf{G}$ must compress their common input
- $\mathbf{A}$ is compressing, choice of params ✓

Goal: constrained-SIS is a hash function – pt1

Goal:

1. show that constrained-SIS $= (\mathbf{A}, \mathbf{G})$ is a hash function

- $\mathbf{A}$ and $\mathbf{G}$ must compress their common input
- $\mathbf{A}$ is compressing, choice of params ✓
- $\mathbf{G}$ is compressing, choice of params...

Goal: constrained-SIS is a hash function – pt1

Goal:

1. show that constrained-SIS $= (\mathbf{A}, \mathbf{G})$ is a hash function

- $\mathbf{A}$ and $\mathbf{G}$ must compress their common input
- $\mathbf{A}$ is compressing, choice of params ✓
- $\mathbf{G}$ is compressing, choice of params...
- ...but why should $\mathbf{G}\mathbf{x} = \mathbf{0}$?

Goal: constrained-SIS is a hash function – pt1

Goal:

1. show that constrained-SIS $= (\mathbf{A}, \mathbf{G})$ is a hash function

- $\mathbf{A}$ and $\mathbf{G}$ must compress their common input
- $\mathbf{A}$ is compressing, choice of params ✓
- $\mathbf{G}$ is compressing, choice of params...
- ...but why should $\mathbf{Gx} = \mathbf{0}$?
    - $\mathbf{G}$ has structure ✓

The $\mathbf{G}$ in constrained-SIS

$$\mathbf{G} = \left(\begin{array}{cccc|c} \overbrace{1\;2\;4\;\cdots\;2^{\ell}}^{\log q} & \star\;\star\;\star\;\cdots\;\star & \ldots & \star\;\star\;\star\;\cdots\;\star & \star\;\star\;\star\;\cdots\;\star \\ 0 & 1\;2\;4\;\cdots\;2^{\ell} & \ldots & \star\;\star\;\star\;\cdots\;\star & \star\;\star\;\star\;\cdots\;\star \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \ldots & 1\;2\;4\;\cdots\;2^{\ell} & \underbrace{\boxed{\star\;\star\;\star\;\cdots\;\star}}_{n-d\log q} \end{array}\right)$$

- $\mathbf{G}$ similar to the gadget matrix from [Micciancio-Peikert '12]

The $\mathbf{G}$ in constrained-SIS

$$
\mathbf{G} = \left(\begin{array}{cccc|c}
\overbrace{1\ 2\ 4\ \cdots\ 2^{\ell}}^{\log q} & \star\ \star\ \star\ \cdots\ \star & \ldots & \star\ \star\ \star\ \cdots\ \star & \star\ \star\ \star\ \cdots\ \star \\
0 & 1\ 2\ 4\ \cdots\ 2^{\ell} & \ldots & \star\ \star\ \star\ \cdots\ \star & \star\ \star\ \star\ \cdots\ \star \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & \ldots & 1\ 2\ 4\ \cdots\ 2^{\ell} & \underbrace{\boxed{\star\ \star\ \star\ \cdots\ \star}}_{n-d\log q}
\end{array}\right)
$$

- $\mathbf{G}$ similar to the gadget matrix from [Micciancio-Peikert '12]
- $\mathbf{G}\mathbf{x} = \mathbf{0}$ can always be satisfied by some $\mathbf{x} \in \{0,1\}^n$

The $\mathbf{G}$ in constrained-SIS

$$\mathbf{G} = \left(\begin{array}{cccc|c} \overbrace{1\ 2\ 4\ \cdots\ 2^{\ell}}^{\log q} & \star\ \star\ \star\ \cdots\ \star & \ldots & \star\ \star\ \star\ \cdots\ \star & \star\ \star\ \star\ \cdots\ \star \\ 0 & 1\ 2\ 4\ \cdots\ 2^{\ell} & \ldots & \star\ \star\ \star\ \cdots\ \star & \star\ \star\ \star\ \cdots\ \star \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \ldots & 1\ 2\ 4\ \cdots\ 2^{\ell} & \underbrace{\boxed{\star\ \star\ \star\ \cdots\ \star}}_{n-d\log q} \end{array}\right) \cdot \left(\begin{array}{c} \cdot\cdot \\ \cdot\cdot \\ \cdot\cdot \\ \{0,1\} \\ \vdots \\ \{0,1\} \end{array}\right)$$

- $\mathbf{G}$ similar to the gadget matrix from [Micciancio-Peikert '12]
- $\mathbf{Gx} = \mathbf{0}$ can always be satisfied by some $\mathbf{x} \in \{0,1\}^n$
  - choose last $(n - d\log q)$ bits of $\mathbf{x}$ arbitrarily (last row)...

The $\mathbf{G}$ in constrained-SIS

$$\mathbf{G} = \left(\begin{array}{cccc|c} \overbrace{1\ 2\ 4\ \cdots\ 2^{\ell}}^{\log q} & \star\ \star\ \star\ \cdots\ \star & \ldots & \star\ \star\ \star\ \cdots\ \star & \star\ \star\ \star\ \cdots\ \star \\ 0 & 1\ 2\ 4\ \cdots\ 2^{\ell} & \ldots & \star\ \star\ \star\ \cdots\ \star & \star\ \star\ \star\ \cdots\ \star \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \ldots & 1\ 2\ 4\ \cdots\ 2^{\ell} & \underbrace{\boxed{\star\ \star\ \star\ \cdots\ \star}}_{n-d\log q} \end{array}\right) \cdot \left(\begin{array}{c} \vdots \\ \vdots \\ \{0,1\} \\ \vdots \\ \{0,1\} \end{array}\right)$$

- $\mathbf{G}$ similar to the gadget matrix from [Micciancio-Peikert '12]
- $\mathbf{Gx} = \mathbf{0}$ can always be satisfied by some $\mathbf{x} \in \{0,1\}^n$
  - choose last $(n - d\log q)$ bits of $\mathbf{x}$ arbitrarily (last row)...
  - ...$\mathbf{Gx} = \mathbf{0} \Leftrightarrow 1x_1 + 2x_2 + 4x_4 + \cdots + 2^{\ell}x_{2^{\ell}} = -\boxed{\star\ \star\ \star\ \cdots\ \star}$

The $\mathbf{G}$ in constrained-SIS

$$\mathbf{G} = \left(\begin{array}{cccc|c} \overbrace{1\ 2\ 4\ \cdots\ 2^{\ell}}^{\log q} & \star\ \star\ \star\ \cdots\ \star & \ldots & \star\ \star\ \star\ \cdots\ \star & \star\ \star\ \star\ \cdots\ \star \\ 0 & 1\ 2\ 4\ \cdots\ 2^{\ell} & \ldots & \star\ \star\ \star\ \cdots\ \star & \star\ \star\ \star\ \cdots\ \star \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \ldots & 1\ 2\ 4\ \cdots\ 2^{\ell} & \underbrace{\boxed{\star\ \star\ \star\ \cdots\ \star}}_{n-d\log q} \end{array}\right) \cdot \left(\begin{array}{c} \vdots \\ \{0,1\} \\ \vdots \\ \{0,1\} \end{array}\right)$$

- $\mathbf{G}$ similar to the gadget matrix from [Micciancio-Peikert '12]
- $\mathbf{Gx} = \mathbf{0}$ can always be satisfied by some $\mathbf{x} \in \{0,1\}^n$
  - choose last $(n - d\log q)$ bits of $\mathbf{x}$ arbitrarily (last row)...
  - ...$\mathbf{Gx} = \mathbf{0} \Leftrightarrow 1x_1 + 2x_2 + 4x_4 + \cdots + 2^{\ell}x_{2^{\ell}} = -\boxed{\star\ \star\ \star\ \cdots\ \star}$
  - rest of $\mathbf{x}$ is **uniquely** determined using backwards substitution & binary decomposition

an example with $d = 3, n = 10, q = 8$

$$\left(\begin{array}{ccc|ccc|ccc|c} 1 & 2 & 4 & 3 & 0 & 6 & 5 & 6 & 2 & 1 \\ 0 & 0 & 0 & 1 & 2 & 4 & 1 & 0 & 3 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 4 & \color{blue}{1} \end{array}\right) \cdot \begin{pmatrix} * \\ * \\ * \\ * \\ * \\ * \\ * \\ * \\ * \\ \color{blue}{1} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \pmod 8$$

an example with $d = 3, n = 10, q = 8$

$$\left(\begin{array}{ccc|ccc|ccc|c} 1 & 2 & 4 & 3 & 0 & 6 & 5 & 6 & 2 & 1 \\ 0 & 0 & 0 & 1 & 2 & 4 & 1 & 0 & 3 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 4 & 1 \end{array}\right) \cdot \begin{pmatrix} * \\ * \\ * \\ * \\ * \\ * \\ x_7 \\ x_8 \\ x_9 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \pmod 8$$

binary decomposition (last row)

$$1 \cdot x_7 + 2 \cdot x_8 + 4 \cdot x_9 + (1 \cdot 1) = 0 \pmod 8 \Rightarrow x_7 = x_8 = x_9 = 1$$

an example with $d = 3, n = 10, q = 8$

$$\left(\begin{array}{ccc|ccc|ccc|c} 1 & 2 & 4 & 3 & 0 & 6 & 5 & 6 & 2 & 1 \\ 0 & 0 & 0 & 1 & 2 & 4 & 1 & 0 & 3 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 4 & 1 \end{array}\right) \cdot \begin{pmatrix} * \\ * \\ * \\ x_4 \\ x_5 \\ x_6 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \pmod 8$$

binary decomposition (2nd row)

$$1 \cdot x_4 + 2 \cdot x_5 + 4 \cdot x_6 + \underbrace{(1 + 2 + 4 + 1)}_{\text{back substitution}} = 0 \pmod 8$$

an example with $d = 3, n = 10, q = 8$

$$\left(\begin{array}{ccc|ccc|ccc|c} 1 & 2 & 4 & 3 & 0 & 6 & 5 & 6 & 2 & 1 \\ 0 & 0 & 0 & 1 & 2 & 4 & 1 & 0 & 3 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 4 & 1 \end{array}\right) \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \pmod{8}$$

binary decomposition (1st row)

$$1 \cdot x_1 + 2 \cdot x_2 + 4 \cdot x_3 + \underbrace{17}_{\text{back substitution}} = 0 \pmod{8}$$

an example with $d = 3, n = 10, q = 8$

$$\left(\begin{array}{ccc|ccc|ccc|c} 1 & 2 & 4 & 3 & 0 & 6 & 5 & 6 & 2 & 1 \\ 0 & 0 & 0 & 1 & 2 & 4 & 1 & 0 & 3 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 4 & 1 \end{array}\right) \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \pmod 8$$

Goal: constrained-SIS is a hash function – pt2

- $2^{n-d\log q}$ different values of $\mathbf{x}$ can satisfy $\mathbf{G}\mathbf{x} = \mathbf{0}$
- same $\mathbf{x}$ are mapped as: $\mathbf{x} \mapsto \mathbf{A}\mathbf{x}$

Goal: constrained-SIS is a hash function – pt2

- $2^{n-d\log q}$ different values of $\mathbf{x}$ can satisfy $\mathbf{G}\mathbf{x} = \mathbf{0}$
- same $\mathbf{x}$ are mapped as: $\mathbf{x} \mapsto \mathbf{A}\mathbf{x}$
- range of $\mathbf{x} \mapsto \mathbf{A}\mathbf{x}$ is q^m

Goal: constrained-SIS is a hash function – pt2

- $2^{n-d\log q}$ different values of $\mathbf{x}$ can satisfy $\mathbf{Gx} = \mathbf{0}$
- same $\mathbf{x}$ are mapped as: $\mathbf{x} \mapsto \mathbf{Ax}$
- range of $\quad \mathbf{x} \mapsto \mathbf{Ax} \quad$ is q^m
- $2^{n-d\log q} > q^m$

Goal: constrained-SIS is a hash function – pt2

- $2^{n-d\log q}$ different values of $\mathbf{x}$ can satisfy $\mathbf{Gx} = \mathbf{0}$
- same $\mathbf{x}$ are mapped as: $\mathbf{x} \mapsto \mathbf{Ax}$
- range of $\quad \mathbf{x} \mapsto \mathbf{Ax} \quad$ is q^m
- $2^{n-d\log q} > q^m \quad \Rightarrow \quad$ enough $\mathbf{x}$ to have collisions in $\mathbf{A}$

  i.e. $|\text{domain}| > |\text{range}|$

Goal: constrained-SIS is a hash function – pt2

- $2^{n-d\log q}$ different values of $\mathbf{x}$ can satisfy $\mathbf{Gx} = \mathbf{0}$
- same $\mathbf{x}$ are mapped as: $\mathbf{x} \mapsto \mathbf{Ax}$
- range of $\quad \mathbf{x} \mapsto \mathbf{Ax} \quad$ is q^m
- $2^{n-d\log q} > q^m \quad \Rightarrow \quad$ enough $\mathbf{x}$ to have collisions in $\mathbf{A}$

  i.e. |domain| $>$ |range|

constrained-SIS is a hash function ✓

next: $C_h \leq$ constrained-SIS

Goal: $C_h \leq$ constrained-SIS

Reduce C_h to constrained-SIS

Goal: $C_h \leq$ constrained-SIS

Reduce C_h to constrained-SIS

Goal: Construct $\mathbf{A}, \mathbf{G}$ out of C_h

Goal: $C_h \leq$ constrained-SIS

Reduce C_h to constrained-SIS

Goal: Construct $\mathbf{A}, \mathbf{G}$ out of C_h

we start with $\mathbf{G}$

$C_h \leq$ constrained-SIS – the G pt

- Embed C_h in $\mathbf{G}$ using **OR** and **XOR** gates
    - Circuit gates in $C_h \rightarrow$ Linear equations in $\mathbf{G}$

$C_h \leq$ constrained-SIS – the $\mathbf{G}$ pt

- Embed C_h in $\mathbf{G}$ using **OR** and **XOR** gates
  - Circuit gates in $C_h \to$ Linear equations in $\mathbf{G}$

OR: $\boxed{x_1 \vee x_2} = y,\ z = x_1 \oplus x_2 \iff \boxed{\mathbf{1}y + \mathbf{2}z} - x_1 - x_2 = 0 \pmod{\mathbf{4}}$

XOR: $\boxed{x_1 \oplus x_2} = y,\ z = x_1 \wedge x_2 \iff \boxed{\mathbf{1}y + \mathbf{2}z} - x_1 - x_2 = 0 \pmod{\mathbf{4}}$

$C_h \leq$ constrained-SIS – the G pt

- Embed C_h in $\mathbf{G}$ using **OR** and **XOR** gates
  - Circuit gates in $C_h \to$ Linear equations in $\mathbf{G}$

OR: $\boxed{x_1 \vee x_2} = y,\ z = x_1 \oplus x_2 \Longleftrightarrow \boxed{\mathbf{1}y + \mathbf{2}z} - x_1 - x_2 = 0 \pmod{\mathbf{4}}$

XOR: $\boxed{x_1 \oplus x_2} = y,\ z = x_1 \wedge x_2 \Longleftrightarrow \boxed{\mathbf{1}y + \mathbf{2}z} - x_1 - x_2 = 0 \pmod{\mathbf{4}}$

$$\mathbf{G} = \begin{pmatrix} \text{output gate} & \text{output wires} & 0 & 0 & 0 \\ 0 & \text{intermediate gate} & \text{intermediate wires} & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & \text{input gate} & \text{input wires} \end{pmatrix}$$

$C_h \leq$ constrained-SIS – the $\mathbf{G}$ pt

- Embed C_h in $\mathbf{G}$ using **OR** and **XOR** gates
  - Circuit gates in $C_h \to$ Linear equations in $\mathbf{G}$

OR: $\boxed{x_1 \vee x_2} = y,\ z = x_1 \oplus x_2 \iff \boxed{\mathbf{1}y + \mathbf{2}z} - x_1 - x_2 = 0 \pmod{\mathbf{4}}$

XOR: $\boxed{x_1 \oplus x_2} = y,\ z = x_1 \wedge x_2 \iff \boxed{\mathbf{1}y + \mathbf{2}z} - x_1 - x_2 = 0 \pmod{\mathbf{4}}$

$$\mathbf{G} = \begin{pmatrix} \text{output gate} & \text{output wires} & 0 & 0 & 0 \\ 0 & \text{intermediate gate} & \text{intermediate wires} & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & \text{input gate} & \text{input wires} \end{pmatrix}$$

- $\mathbf{G}\mathbf{x}_1 = \mathbf{0} = \mathbf{G}\mathbf{x}_2$ represents evaluation of $C_h(x_1)$ <u>and</u> $C_h(x_2)$
  - $\mathbf{x}$ contains evaluation of $C_h(x)$ gate-by-gate
  - $\mathbf{x} = (\text{output}, \text{intermediate steps}, \text{input})^T$

$C_h \leq$ constrained-SIS – the $\mathbf{A}$ pt

$\mathbf{A}$ extracts the output of $C_h(x)$ from $\mathbf{x}$

$C_h \leq$ constrained-SIS – the $\mathbf{A}$ pt

$\mathbf{A}$ extracts the output of $C_h(x)$ from $\mathbf{x}$

- Set $\mathbf{A} = (1, 0, 0)$

$C_h \leq$ constrained-SIS – the $\mathbf{A}$ pt

$\mathbf{A}$ extracts the output of $C_h(x)$ from $\mathbf{x}$

- Set $\mathbf{A} = (1, 0, 0)$
- $\mathbf{x} = (\text{output}, \text{intermediate steps}, \text{input})^T$

$C_h \leq$ constrained-SIS – the A pt

$\mathbf{A}$ extracts the output of $C_h(x)$ from $\mathbf{x}$

- Set $\mathbf{A} = (1, 0, 0)$
- $\mathbf{x} = (\text{output}, \text{ intermediate steps}, \text{ input})^T$
- $\mathbf{Ax} = \text{output} \quad \Rightarrow \quad \mathbf{Ax} = C_h(x)$

$C_h \leq$ constrained-SIS – the $\mathbf{A}$ pt

$\mathbf{A}$ extracts the output of $C_h(x)$ from $\mathbf{x}$

- Set $\mathbf{A} = (1, 0, 0)$
- $\mathbf{x} = (\text{output}, \text{intermediate steps}, \text{input})^T$
- $\mathbf{Ax} = \text{output} \quad \Rightarrow \quad \mathbf{Ax} = C_h(x)$
- $\mathbf{Ax}_1 = \mathbf{Ax}_2$ implies $C_h(x_1) = C_h(x_2) \quad \Rightarrow$ a collision for h ✓

$C_h \leq$ constrained-SIS – the $\mathbf{A}$ pt

$\mathbf{A}$ extracts the output of $C_h(x)$ from $\mathbf{x}$

- Set $\mathbf{A} = (1, 0, 0)$
- $\mathbf{x} = (\text{output}, \text{intermediate steps}, \text{input})^T$
- $\mathbf{Ax} = \text{output} \quad \Rightarrow \quad \mathbf{Ax} = C_h(x)$
- $\mathbf{Ax}_1 = \mathbf{Ax}_2$ implies $C_h(x_1) = C_h(x_2) \quad \Rightarrow$ a collision for h ✓

$$
\begin{aligned}
\text{Find Short Integer Solutions for } \mathbf{A}, \mathbf{G} &\implies \text{Find collisions in constrained-SIS} \\
&\implies \text{Find collision in } C_h \text{ for any } h \\
&\implies \text{♛}
\end{aligned}
$$

Series of reductions

Our result shows that:

$$\text{SIS, LWE, SIVP, GapSVP, Minkowksi, n-SVP, SHA, DLog}, \ldots \leq \text{constrained–SIS}$$

These problems can be solved by finding collisions

Minkowski: $||v||_2 \leq \sqrt{n}\,\det(\mathcal{L})^{1/n}$

The post-quantum quest

A strong post-quantum guarantee for $\overset{\text{♛}}{\overline{h}}$?

The post-quantum quest

A strong post-quantum guarantee for $\overset{\text{♛}}{h}$?

 A quantum speedup for constrained-SIS

$\iff$ quantum speedup for finding collisions, in general

The post-quantum quest

A strong post-quantum guarantee for $\overset{\text{♛}}{h}$?

A quantum speedup for constrained-SIS

$\Longleftrightarrow$ quantum speedup for finding collisions, in general

"Morally", a quantum speedup for lattice problems(?)

$\Longleftrightarrow$ quantum speedup for finding collisions, in general

The post-quantum quest

A strong post-quantum guarantee for $\overset{\text{♛}}{\overline{h}}$?

A quantum speedup for constrained-SIS

$\Longleftrightarrow$ quantum speedup for finding collisions, in general

"Morally", a quantum speedup for lattice problems(?)

$\Longleftrightarrow$ quantum speedup for finding collisions, in general

This would be a strong implication. Should we expect this?

The post-quantum quest

A strong post-quantum guarantee for $\overset{\text{♛}}{h}$?

A quantum speedup for constrained-SIS

$\Longleftrightarrow$ quantum speedup for finding collisions, in general

"Morally", a quantum speedup for lattice problems(?)

$\Longleftrightarrow$ quantum speedup for finding collisions, in general

This would be a strong implication. Should we expect this?

This motivates the open problem section

Open problems

A worst-to-average case reduction from constrained-SIS to itself?

- Conjecture for $\overset{♛}{h}$: $\mathbf{A} \leftarrow \$, \quad \mathbf{G} \leftarrow \text{semi-random (random } \star)$ experiments?

Open problems

- A worst-to-average case reduction from constrained-SIS to itself?
  - Conjecture for h: $\mathbf{A} \leftarrow \$$, $\;\mathbf{G} \leftarrow$ semi-random (random $\star$) experiments?
- approx-SVP/CVP equivalent to constrained-SIS?
  - approx-SVP/CVP are fundamental lattice problems
  - Minkowski short vectors & pigeonhole principle

Open problems

- A worst-to-average case reduction from constrained-SIS to itself?
  - Conjecture for $\bar{h}$: $\mathbf{A} \leftarrow \$, \quad \mathbf{G} \leftarrow \text{semi-random (random } \star)$ experiments?
- approx-SVP/CVP equivalent to constrained-SIS?
  - approx-SVP/CVP are fundamental lattice problems
  - Minkowski short vectors & pigeonhole principle
- Direct reduction of specific hash functions to lattice problems?
  - SHA $\leq$ approx-SVP? $\quad \to \quad$ provable security level?

Open problems

- A worst-to-average case reduction from constrained-SIS to itself?
  - Conjecture for $\overline{h}$: $\mathbf{A} \leftarrow \$, \quad \mathbf{G} \leftarrow \text{semi-random (random } \star)$ experiments?
- approx-SVP/CVP equivalent to constrained-SIS?
  - approx-SVP/CVP are fundamental lattice problems
  - Minkowski short vectors & pigeonhole principle
- Direct reduction of specific hash functions to lattice problems?
  - SHA $\leq$ approx-SVP? $\rightarrow$ provable security level?
- ! Structured lattices in this framework? (e.g. ideal lattices)
  - understand potential & limitations of structured lattices
  - on structured lattices: more evidence for hardness, the better we sleep

Back to [Papadimitriou '94]

...mathematical principles that guarantee a solution...

Back to [Papadimitriou '94]

...mathematical principles that guarantee a solution...

- The handshake lemma. Given undirected graph $G(V, E)$:

$$\sum_{v \in V} \mathsf{deg}(v) = 2|E|$$

Back to [Papadimitriou '94]

...mathematical principles that guarantee a solution...

- The handshake lemma. Given undirected graph $G(V, E)$:

$$\sum_{v \in V} \mathsf{deg}(v) = 2|E|$$

- A vertex with odd degree, implies another vertex with odd degree

Back to [Papadimitriou '94]

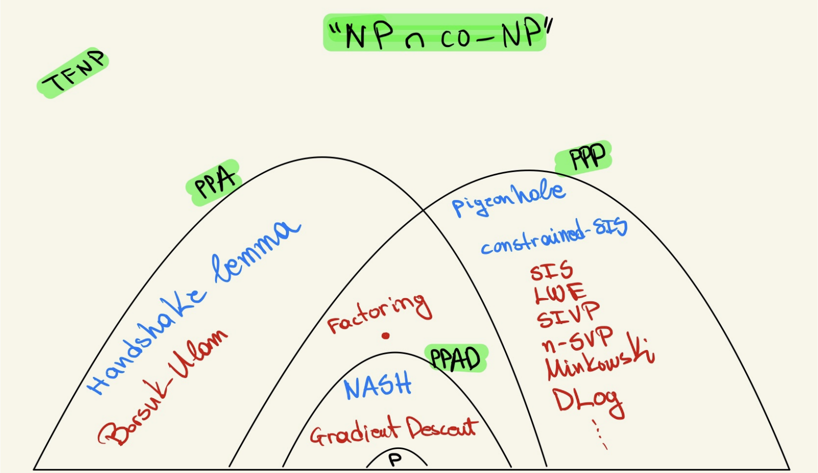
...mathematical principles that guarantee a solution...

- The handshake lemma. Given undirected graph $G(V, E)$:

$$\sum_{v \in V} \deg(v) = 2|E|$$

- A vertex with odd degree, implies another vertex with odd degree
- Hardness in finding this other odd degree vertex

Complexity of factoring integers? in PPA [Jerábek '16]



- How low can Factoring go?
- Factoring $\leq$ approx-SVP/CVP?

next: obfuscation!

modern crypto (we use it everyday)

modern crypto (more than this: foundations)

A non-exhaustive list:

- Public-key encryption – $(\mathsf{pk}, \mathsf{sk})$ e.g. RSA
- Zero Knowledge Proofs
- Multiparty Computation
- Attribute-Based Encryption
- Fully-Homomorphic Encryption

...

A non-exhaustive list:

- Public-key encryption – $(\textcolor{blue}{\mathsf{pk}}, \textcolor{blue}{\mathsf{sk}})$ e.g. RSA
- Zero Knowledge Proofs
- Multiparty Computation
- Attribute-Based Encryption
- Fully-Homomorphic Encryption

...

Super-tool to build crypto tools?

modern crypto (the answer: yes)

A non-exhaustive list:

- Public-key encryption – $(\mathsf{pk}, \mathsf{sk})$ e.g. RSA
- Zero Knowledge Proofs
- Multiparty Computation
- Attribute-Based Encryption
- Fully-Homomorphic Encryption

...

Super-tool to build crypto tools?

Program Obfuscation

Program Obfuscation

Main character: programs

Goal: hide program secrets

What is obfuscation? (main character)

- An **obfuscator** is a **program compiler**

$$x \longrightarrow \boxed{P} \longrightarrow P(x)$$

What is obfuscation? (main character $\rightarrow$ obf)

- An **obfuscator** is a **program compiler**

$$x \longrightarrow \boxed{P} \longrightarrow P(x)$$

$$\downarrow$$

obfuscator

What is obfuscation? (obf $\rightarrow$ code)

- An **obfuscator** is a **program compiler**

$$x \longrightarrow \boxed{P} \longrightarrow P(x)$$

$$\downarrow$$

$$\boxed{\textbf{obfuscator}}$$

$$\downarrow$$

$$x \longrightarrow \boxed{\widetilde{P}} \longrightarrow P(x)$$

What is obfuscation? (code hides secrets)

- An **obfuscator** is a **program compiler**

$$x \longrightarrow \boxed{P} \longrightarrow P(x)$$

$$\big\downarrow$$

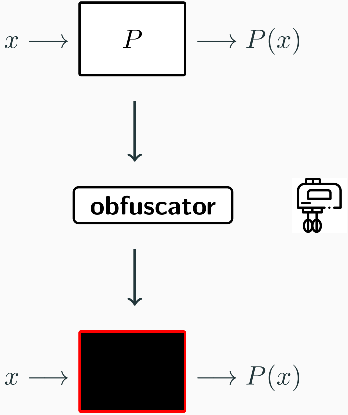
$$\boxed{\textbf{obfuscator}}$$

$$\big\downarrow$$

$$x \longrightarrow \boxed{\widetilde{P}} \longrightarrow P(x)$$

$\widetilde{P}$ hides implementation details of P
e.g. constants, variable values, procedures

- An **obfuscator** is a **program compiler**



VBB security: only learn $(x, P(x))$

Obfuscation in practice

- *Heuristic* solutions (obfuscation as a product)
- International C code obfuscation (since 1984)

Obfuscation in practice

- *Heuristic* solutions (obfuscation as a product)
- International C code obfuscation (since 1984)

- **Goal:** prove security based on a hard math problem
  - e.g. Lattice problems

Does VBB obfuscation exist?

$x \longrightarrow$  $\longrightarrow P(x)$

Too good to be true?

Does VBB obfuscation exist?

- VBB obfuscation is impossible in the general case ☹

Does VBB obfuscation exist?

- VBB obfuscation is impossible in the general case 😐

Does VBB obfuscation exist?

- VBB obfuscation is impossible in the general case
- There is a program P we cannot VBB obfuscate

Does VBB obfuscation exist?

- VBB obfuscation is impossible in the general case
- There is a program P we cannot VBB obfuscate
- ...and a few programs that **can** be VBB obfuscated

Does VBB obfuscation exist?

- VBB obfuscation is impossible in the general case
- There is a program P we cannot VBB obfuscate
- ...and a few programs that **can** be VBB obfuscated
  - simple programs that predate our work
    [Can97, Wee05, CD08, CRV10, BVWW16, Zha16]
    point functions, hyperplanes, conjunctions

Does VBB obfuscation exist?

- VBB obfuscation is impossible in the general case
- There is a program P we cannot VBB obfuscate
- ...and a few programs that **can** be VBB obfuscated
  - simple programs that predate our work
    [Can97, Wee05, CD08, CRV10, BVWW16, Zha16]
    point functions, hyperplanes, conjunctions

Can we obfuscate more programs

Our results

- Wichs-**Z** FOCS'17

  concurrent/independent GKW'17

  - Distribution-VBB obfuscate a large and expressive family of programs
  - Most general result so far, provably secure under the Learning-with-Errors assumption

Compute-and-Compare programs (definition)

$$f(\cdot) \stackrel{?}{=} y$$

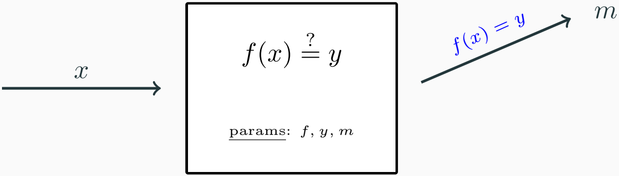
<u>params</u>: f, y, m

Compute-and-Compare programs (input)

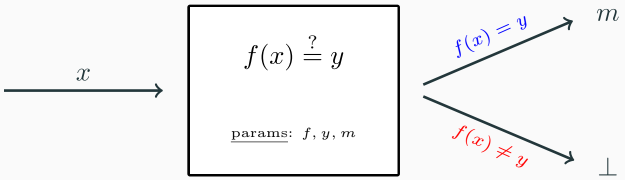
$$\xrightarrow{\quad x \quad} \boxed{f(x) \stackrel{?}{=} y}$$

params: f, y, m

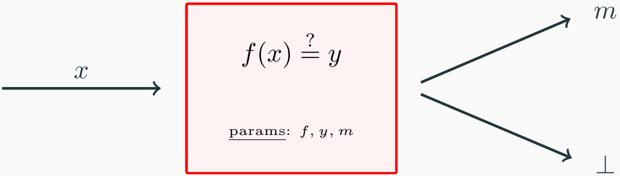
Compute-and-Compare programs (output)



Compute-and-Compare programs (output)



CC obfuscation & security



$\boxed{\text{Black-Box}}$ simulation security when y is **random** given f, m

Obfuscation **hides params:** f, y, m

Evasive programs

- if y is **random** given f, m...
- ...then for most $x \Rightarrow f(x) \neq y$
- why bother then?

Why obfuscate evasive programs?

Two groups of users

Can predict y	Cannot predict y
Correctness is meaningful	Correctness, not meaningful
Security, not meaningful	Security is meaningful

Applications

New applications ⊕

- Hide the access policy: upgrade Attribute-based Encryption to Predicate Encryption
  - re-use existing ABE keys (modular approach)
- Upgrade Witness Encryption to null iO
- Private authentication using biometric data
- Obfuscate conjunctions under LWE

Post-quantum applications

some recent work ➕

- Post-Quantum Multi-Party Computation
  [ABGKM, EUROCRYPT '21]
- Post-Quantum Zero-Knowledge in Constant Rounds
  [Bitansky-Shmueli, STOC '20]
- Weak Zero-Knowledge
  [Bitansky-Khurana-Paneth, STOC '19]
- Optimal Traitor-Tracing
  [CVWWW, TCC '18]

optimized construction [GVW'18]

perfect correctness [GKVW'20]

Encrypt your own secret key: Proofs and Heuristics

A fundamental question [GM'84]

- Is $\mathsf{Enc}(\mathsf{pk}, \mathsf{sk}_i)$ **always** secure?
  - **bit-by-bit** encryption of msg $=$ sk
- We give a **negative** answer ☹
  - public-key bit-by-bit **CPA** secure $\rightarrow$ circular insecure
    (strong/non-pq assumptions [Rot13, KRW15])
- We refute a Random-Oracle **heuristic** for security of $\mathsf{Enc}(\mathsf{pk}, \mathsf{sk}_i)$
  - the only heuristic transformation known
- Why investigate this type of security?

A fundamental question [GM'84]

- Is $\mathsf{Enc}(\mathsf{pk}, \mathsf{sk}_i)$ **always** secure?
    - **bit-by-bit** encryption of msg $= \mathsf{sk}$

- We give a **negative** answer ☹
    - public-key bit-by-bit **CPA** secure $\rightarrow$ circular insecure
      (strong/non-pq assumptions [Rot13, KRW15])

- We refute a Random-Oracle **heuristic** for security of $\mathsf{Enc}(\mathsf{pk}, \mathsf{sk}_i)$
    - the only heuristic transformation known

- Why investigate this type of security?
    - Fundamental question
    - Recently in the news! (iO candidates)
    - ➔ Fully-Homomorphic Encryption (bootstrapping)

Can Random Oracles help?

- Random Oracles (RO) are used both in theory and practice
  - Publicly accessible *gigantic* source of randomness
  - i.e. $\mathrm{RO}(x) = \text{random}$
- In practice, replacing RO = SHA-2/SHA-3
- In theory, replacing RO = it's complicated

Can Random Oracles really help?

Power of RO

- Transform **any** IND-CPA scheme to a circular secure one [BRS03]
- $\mathsf{Enc}_{\mathsf{RO}}(\mathsf{pk}, m) = \mathsf{Enc}(\mathsf{pk}, r),\ \mathsf{RO}(r) \oplus m$

Can Random Oracles really help?

Power of RO ⚖

- Transform **any** IND-CPA scheme to a circular secure one [BRS03]
- $\mathsf{Enc}_{\mathsf{RO}}(\mathsf{pk}, m) = \mathsf{Enc}(\mathsf{pk}, r),\ \mathsf{RO}(r) \oplus m$

Power of obfuscation ⚖

- We construct an IND-CPA scheme that **cannot** be upgraded as above...

  ...no matter which hash function is used to implement RO

Circular insecurity: sem $\to$ circ-insec

Assume bit encryption

secret key: $\mathsf{Dec}(\mathsf{sk}, \cdot)$

public key: $\mathsf{Enc}(\mathsf{pk}, \cdot)$

Circular insecurity: sem $\to$ circ-insec

Assume bit encryption

secret key: $\mathsf{Dec}(\mathsf{sk}, \cdot)$

$y \leftarrow \$$

- $\mathsf{sk}' \to (\mathsf{sk}, y)$

public key: $\mathsf{Enc}(\mathsf{pk}, \cdot)$

Circular insecurity: sem → circ-insec

Assume bit encryption

secret key: $\mathsf{Dec}(\mathsf{sk}, \cdot)$

$y \leftarrow \$$

- $\mathsf{sk}' \rightarrow (\mathsf{sk}, y)$

public key: $\mathsf{Enc}(\mathsf{pk}, \cdot)$

$\mathsf{Dec}_{\mathsf{sk}}(\cdot) \stackrel{?}{=} y$

params: sk, y, sk

- $\mathsf{pk}' \rightarrow (\mathsf{pk}, \mathsf{Obf})$

Assume bit encryption

secret key: $\mathsf{Dec}(\mathsf{sk}, \cdot)$

$y \leftarrow \$$

- $\mathsf{sk}' \rightarrow (\mathsf{sk}, y)$

public key: $\mathsf{Enc}(\mathsf{pk}, \cdot)$

$$\boxed{\mathsf{Dec}_{\mathsf{sk}}(\cdot) \stackrel{?}{=} y \quad \underline{\text{params}}: \mathsf{sk}, y, \mathsf{sk}}$$

- $\mathsf{pk}' \rightarrow (\mathsf{pk}, \mathsf{Obf})$
- y indep of sk $\Rightarrow$ semantic security

Assume bit encryption

secret key: $\mathsf{Dec}(\mathsf{sk}, \cdot)$

$y \leftarrow \$$

- $\mathsf{sk}' \to (\mathsf{sk}, y)$

public key: $\mathsf{Enc}(\mathsf{pk}, \cdot)$

$\mathsf{Enc}_{\mathsf{pk}}(y) \to \boxed{\mathsf{Dec}_{\mathsf{sk}}(\cdot) \stackrel{?}{=} y \quad \text{params: } \mathsf{sk}, y, \mathsf{sk}} \to \mathsf{sk}$

- $\mathsf{pk}' \to (\mathsf{pk}, \mathsf{Obf})$
- y indep of sk $\Rightarrow$ semantic security

Circular insecurity: sem → circ-insec

Assume bit encryption

secret key: $\mathsf{Dec}(\mathsf{sk}, \cdot)$

$y \leftarrow \$$

- $\mathsf{sk}' \rightarrow (\mathsf{sk}, y)$

public key: $\mathsf{Enc}(\mathsf{pk}, \cdot)$

$\mathsf{Enc}_{\mathsf{pk}}(y) \rightarrow \boxed{\mathsf{Dec}_{\mathsf{sk}}(\cdot) \stackrel{?}{=} y \quad \underline{\text{params}}\text{: } \mathsf{sk}, y, \mathsf{sk}} \rightarrow \mathsf{sk}$

- $\mathsf{pk}' \rightarrow (\mathsf{pk}, \mathsf{Obf})$
- y indep of sk $\Rightarrow$ semantic security
- ☠ recover sk $\Rightarrow$ break security!

Random Oracles: real vs ideal

- GKW'17 shows similar result for Fujisaki-Okamoto
- Caution: RO Model $\rightarrow$ Standard Model (SHA-3, ...)
- Ideally, we wouldn't need RO
  - comparable efficiency without RO?

is obfuscation a success story?

is obfuscation a success story?

- "relaxed" form of obfuscation ⇒ almost all crypto 👍
- **i**ndistinguishability **O**bfuscation
- **iO** most probably exists as of 2021 (non-pq)! [..., JLS'21, ...]
- other new constructions involve new **circular security** definitions

is obfuscation a success story?

- "relaxed" form of obfuscation ⇒ almost all crypto 👍
- **i**ndistinguishability **O**bfuscation
- **iO** most probably exists as of 2021 (non-pq)! [..., JLS'21, ...]
- other new constructions involve new **circular security** definitions

tales of **iO**

- Cryptographic hardness of NASH equilibria [AKV'05, BPR'15]
- 2-Round Multiparty Computation [GGHR'14, GP'15]
- Program Watermarking [CHNVW '16]
  - Quach-Wichs-**Z** TCC'18

...

is obfuscation a success story?

tales of **iO**

- "relaxed" form of obfuscation $\Rightarrow$ almost all crypto 👍
- **i**ndistinguishability **O**bfuscation
- **iO** most probably exists as of 2021 (non-pq)! [..., JLS'21, ...]
- other new constructions involve new **circular security** definitions

- Cryptographic hardness of NASH equilibria [AKV'05, BPR'15]
- 2-Round Multiparty Computation [GGHR'14, GP'15]
- Program Watermarking [CHNVW '16]
  - Quach-Wichs-**Z** TCC'18

...

$$\boxed{a(b+c)} \approx \boxed{ab+ac}$$

🏆

Recent work – future post-quantum directions

- Adaptive prefix encryption under LWE (**Z '21**)
    - prefix enc = original Hierarchical IBE
    - algebraic instead of bb IBE use – focus on params
    - ...but "really adaptive" post-quantum HIBE still open (EUROCRYPT '10 [ABB10, CHKP10])
    - pairings superior to lattices

Recent work – future post-quantum directions

- Adaptive prefix encryption under LWE (**Z '21**)
  - prefix enc = original Hierarchical IBE
  - algebraic instead of bb IBE use – focus on params
  - ...but "really adaptive" post-quantum HIBE still open (EUROCRYPT '10 [ABB10, CHKP10])
  - pairings superior to lattices
- (Zero-Knowledge) Proofs, Obfuscation (iO), FHE
  - very active for post-quantum – theory & practice

Recent work – future post-quantum directions

- Adaptive prefix encryption under LWE (**Z '21**)
  - prefix enc = original Hierarchical IBE
  - algebraic instead of bb IBE use – focus on params
  - ...but "really adaptive" post-quantum HIBE still open (EUROCRYPT '10 [ABB10, CHKP10])
  - pairings superior to lattices
- (Zero-Knowledge) Proofs, Obfuscation (iO), FHE
  - very active for post-quantum – theory & practice
- Structured lattices: new techniques/algorithms?
  - efficient ZK $\rightarrow$ new ideas

Recent work – future post-quantum directions

- Adaptive prefix encryption under LWE (**Z '21**)
  - prefix enc = original Hierarchical IBE
  - algebraic instead of bb IBE use – focus on params
  - ...but "really adaptive" post-quantum HIBE still open (EUROCRYPT '10 [ABB10, CHKP10])
  - pairings superior to lattices
- (Zero-Knowledge) Proofs, Obfuscation (iO), FHE
  - very active for post-quantum – theory & practice
- Structured lattices: new techniques/algorithms?
  - efficient ZK → new ideas
- IBE/ABE followed after PKE. Next primitive to _____ ?

Recent work – future post-quantum directions

- Adaptive prefix encryption under LWE (**Z '21**)
  - prefix enc = original Hierarchical IBE
  - algebraic instead of bb IBE use – focus on params
  - ...but "really adaptive" post-quantum HIBE still open (EUROCRYPT '10 [ABB10, CHKP10])
  - pairings superior to lattices
- (Zero-Knowledge) Proofs, Obfuscation (iO), FHE
  - very active for post-quantum – theory & practice
- Structured lattices: new techniques/algorithms?
  - efficient ZK $\rightarrow$ new ideas
- IBE/ABE followed after PKE. Next primitive to _____ ?
- LWR, LPN

Thank you!

```c
                                                                            char
                                                               _3141592654[3141
          ],__3141[3141];_314159[31415],_3141[31415];main(){register char*
      _3_141,*_3_1415, *_3__1415; register int _314,_31415,__31415,*_31,
    _3_14159,__3_1415;*_3141592654=__31415=2,_3141592654[0][_3141592654
   -1]=1[__3141]=5;__3_1415=1;do{_3_14159=_314=0,__31415++;for( _31415
  =0;_31415<(3,14-4)*__31415;_31415++)_31415[_3141]=_314159[_31415]= -
1;_3141[*_314159=_3_14159]=_314;_3_141=_3141592654+__3_1415;_3_1415=
__3_1415    +__3141;for                   (_31415 = 3141-
           __3_1415  ;                    _31415;_31415--
           ,_3_141 ++,                    _3_1415++){_314
           +=_314<<2 ;                    _314<<=1;_314+=
          *_3_1415;_31                     =_314159+_314;
          if(!(*_31+1)                     )* _31 =_314 /
          __31415,_314                     [_3141]=_314 %
          __31415 ;* (                     _3__1415=_3_141
         )+= *_3_1415                       = *_31;while(*
         _3__1415 >=                        31415/3141 ) *
         _3__1415+= -                       10,(*--_3__1415
        )++;_314=_314                       [_3141]; if ( !
        _3_14159 && *                       _3_1415)_3_14159
        =1,__3_1415 =                       3141-_31415;}if(
        _314+(__31415                        >>1)>=__31415 )
        while ( ++ *                         _3_141==3141/314
       )*_3_141--=0                          ;}while(_3_14159
       ) ; { char *                          __3_14= "3.1415";
       write((3,1),                          (--*__3_14,__3_14
       ),(_3_14159                            ++,++_3_14159))+
      3.1415926; }                            for ( _31415 = 1;
     _31415<3141-                             1;_31415++)write(
    31415% 314-(                              3,14),_3141592654[
  _31415    ] +                              "0123456789","314"
  [ 3]+1)-_314;                              puts((*_3141592654=0
,_3141592654))                                ;_314= *"3.141592";}
```

6th International Obfuscated C Code Contest (1989)