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, Bitcoin, ...

- 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, $\mathcal{B}$, ...

- 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, \( \mathcal{H} \), ...

- 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 $h$?
Most secure?

- What does most secure mean for some $h$?

- Collisions for $h$ must be as hard, as in any other $h$. 
Most secure?

- What does *most* secure mean for some $h$?

- Collisions for $h$ must be as hard, as in any other $h$.

- Implies a **reduction**: $\forall h, \ h \leq \overline{h}$
  - for fixed security parameter
  - $\overline{h}$ inherits hardness from all $h$
Most secure?

- What does *most* secure mean for some $h$?

- Collisions for $h$ must be as hard, as in any other $h$

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

- What are the reduction steps?
Reduction steps \( h \rightarrow C_h \rightarrow \hat{h} \)

Hash function \( h \)
Reduction steps \( h \to C_h \to h \)

Hash function \( h \)

1. **Represent** \( h \) in a **universal way** (e.g. use boolean circuits)
   - \( h \to C_h \)
Reduction steps \( h \rightarrow C_h \rightarrow \overline{h} \)

Hash function \( h \)

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

2. **Reduce** \( C_h \) to a hash function \( \overline{h} \)
   - \( C_h \leq \overline{h} \)
   - Find collisions in \( \overline{h} \) \( \iff \) Find collisions in \( C_h, \forall h \)
Reduction steps \( h \rightarrow C_h \rightarrow \hat{h} \)

Hash function \( h \)

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

2. **Reduce \( C_h \) to a hash function \( \hat{h} \)**
   - \( C_h \leq \hat{h} \)
   - Find collisions in \( \hat{h} \) \( \Longrightarrow \) Find collisions in \( C_h, \forall h \)

3. **Declare \( \hat{h} \) as the most secure** (WC/AVG)
Reduction steps \( h \rightarrow C_h \rightarrow \overline{h} \)

Hash function \( h \)

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

2. **Reduce \( C_h \) to a hash function** \( \overline{h} \)
   - \( C_h \leq \overline{h} \)
   - Find collisions in \( \overline{h} \) \( \Rightarrow \) Find collisions in \( C_h, \forall h \)

3. **Declare \( \overline{h} \) as the most secure** (WC/AVG)

What should \( \overline{h} \) be?
In the 90s...

- The question about $h$ was asked in [Papadimitriou ’94]
  ...in the broader context of total problems (TFNP)

- It remained open, what $h$ to use...
  ...it all starts with the pigeonhole principle
In the 90s...

- The question about $h$ was asked in [Papadimitriou ’94]
  ...in the broader context of total problems (TFNP)

- It remained open, what $h$ 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 $h : [n] \rightarrow [m]$, with $n > m$
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
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 \( \equiv \) \( \rightarrow \) goal: find collisions
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

- This set is the union of:
  1. Cryptographic hash functions (e.g. SHA-3, SIS)
  2. Non-cryptographic hash functions (e.g. pairwise independence)
Want a universal way to represent any (poly-size) hash function $h$
(i.e. convenient to work with)
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
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 $h$

$$C_h : \{0, 1\}^n \to \{0, 1\}^m \quad \text{with } n > m$$

$n, m$ 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 \( \{0, 1\}^n \rightarrow \{0, 1\}^m \) 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 \( \{0, 1\}^n \rightarrow \{0, 1\}^m \) 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 $h$?

What is "natural" & why?
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
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...**
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

\[ h \rightarrow C_h \rightarrow h \]

step 1 – easy
Goal in summary

\[ h \quad \rightarrow \quad C_h \quad \rightarrow \quad h \]

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)
  \( \Rightarrow \) lattice-based hash functions are the most secure

- We show the **first** \( h \)
- Solves open problem from [Pap94]
- Our reduction is worst-case

next: SIS reminder
The SIS problem [Ajtai ’96, Micciancio-Regev ’04]

- Given $A \leftarrow \mathbb{Z}_q^{m \times n}$ with $n > m \log q$ (s.t. collisions exist)
- Find distinct $x_1, x_2 \in \{0, 1\}^n$ s.t.
The SIS problem [Ajtai ’96, Micciancio-Regev ’04]

- Given $A \leftarrow \mathbb{Z}_q^{m \times n}$ with $n > m \log q$ (s.t. collisions exist)
- Find distinct $x_1, x_2 \in \{0, 1\}^n$ s.t.

$$A \cdot x_1 = A \cdot x_2 \pmod{q}$$
The SIS problem [Ajtai ’96, Micciancio-Regev ’04]

- Given $A \leftarrow \mathbb{Z}_q^{m \times n}$ with $n > m \log q$ (s.t. collisions exist)
- Find distinct $x_1, x_2 \in \{0, 1\}^n$ s.t.

\[ A \cdot x_1 = A \cdot x_2 \pmod{q} \]

...implies short $(x_1 - x_2) \in \{0, \pm 1\}^n$ s.t. $A(x_1 - x_2) = 0$
The constrained-SIS problem

- Given $A \in \mathbb{Z}_q^{m \times n}$ and semi-structured $G \in \mathbb{Z}_q^{d \times n}$ with
  
  \[ n > (m + d) \log q \]
The constrained-SIS problem

- Given $A \in \mathbb{Z}_q^{m \times n}$ and semi-structured $G \in \mathbb{Z}_q^{d \times n}$ with $n > (m + d) \log q$
- Find distinct $x_1, x_2 \in \{0, 1\}^n$ s.t.

\[
A \cdot x_1 = A \cdot x_2 \pmod{q}
\]

\[
G \cdot x_1 = G \cdot x_2 = 0 \pmod{q}
\]
The constrained-SIS problem

- Given $A \in \mathbb{Z}_q^{m \times n}$ and semi-structured $G \in \mathbb{Z}_q^{d \times n}$ with $n > (m + d) \log q$
- Find distinct $x_1, x_2 \in \{0, 1\}^n$ s.t.

\[
\begin{align*}
A \cdot x_1 &= A \cdot x_2 \pmod{q} \\
G \cdot x_1 &= G \cdot x_2 = 0 \pmod{q}
\end{align*}
\]
<table>
<thead>
<tr>
<th>constrained-SIS (WC)</th>
<th>SIS (AVG)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$ is arbitrary</td>
<td>$A$ is uniformly random</td>
</tr>
<tr>
<td>$G$ is semi-structured</td>
<td>–</td>
</tr>
<tr>
<td>$x_1, x_2 \in {0, 1}^n$ s.t. $Ax_1 = Ax_2$ and $Gx_1 = 0 = Gx_2$</td>
<td>$x_1, x_2 \in {0, 1}^n$ s.t. $Ax_1 = Ax_2$</td>
</tr>
</tbody>
</table>

**Goal:**

1. Show that constrained-SIS is a hash function.
2. Reduce any hash function to constrained-SIS.
**constrained-SIS vs SIS**

**constrained-SIS** *(WC)*

- $A$ is arbitrary
- $G$ is semi-structured
- $x_1, x_2 \in \{0, 1\}^n$ s.t. $Ax_1 = Ax_2$ and $Gx_1 = 0 = Gx_2$
- unclear how to sample "most secure" keys

**SIS** *(AVG)*

- $A$ is uniformly random
- $x_1, x_2 \in \{0, 1\}^n$ s.t. $Ax_1 = Ax_2$
- can sample keys
- unclear if SIS is the most secure $h$
### constrained-SIS vs SIS

#### constrained-SIS (WC)

- **A** is arbitrary
- **G** is semi-structured

\[
\begin{align*}
x_1, x_2 & \in \{0, 1\}^n \text{ s.t.} \\
Ax_1 &= Ax_2 \quad \text{and} \quad Gx_1 = 0 = Gx_2
\end{align*}
\]

- unclear how to sample "most secure" keys

#### SIS (AVG)

- **A** is uniformly random
- **G** is semi-structured

\[
\begin{align*}
x_1, x_2 & \in \{0, 1\}^n \text{ s.t.} \\
Ax_1 &= Ax_2
\end{align*}
\]

- can sample keys

- unclear if SIS is the most secure

**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 = (A, G) is a hash function
Goal: constrained-SIS is a hash function – pt1

Goal:

1. show that constrained-SIS = (A, G) is a hash function

- A and G must compress their common input
Goal: constrained-SIS is a hash function – pt1

Goal:

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

- A and G must compress their common input
- A is compressing, choice of params ✓
Goal: constrained-SIS is a hash function – pt1

Goal:

1. show that constrained-SIS = (A, G) is a hash function

- A and G must compress their common input
- A is compressing, choice of params ✓
- G is compressing, choice of params...
Goal: constrained-SIS is a hash function – pt1

Goal:

1. show that constrained-SIS = (A, G) is a hash function

- A and G must compress their common input
- A is compressing, choice of params ✓
- G is compressing, choice of params...
- ...but why should Gx = 0?
Goal: constrained-SIS is a hash function – pt1

Goal:

1. show that constrained-SIS = (A, G) is a hash function

- A and G must compress their common input
- A is compressing, choice of params ✓
- G is compressing, choice of params...
- ...but why should Gx = 0?
  - G has structure ✓
The $G$ in constrained-SIS

$G = \begin{pmatrix}
\begin{array}{cccccccc}
1 & 2 & 4 & \cdots & 2^\ell & \ast & \ast & \cdots & \ast \\
0 & 1 & 2 & 4 & \cdots & 2^\ell & \ast & \cdots & \ast \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 1 & 2 & 4 & \cdots & 2^\ell & \ast & \cdots & \ast
\end{array}
\end{pmatrix}
\begin{pmatrix}
\log q \\
{n - d \log q}
\end{pmatrix}$

- $G$ similar to the gadget matrix from [Micciancio-Peikert ’12]
The $G$ in constrained-SIS

\[
G = \begin{pmatrix}
\underbrace{1 2 4 \cdots 2^\ell}_{\log q} & * * * \cdots * & \cdots & * * * \cdots * & * * * \cdots * \\
0 & 1 2 4 \cdots 2^\ell & \cdots & * * * \cdots * & * * * \cdots * \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & \cdots & 1 2 4 \cdots 2^\ell & * * * \cdots * \\
\end{pmatrix}
\]

\[n - d \log q\]

- $G$ similar to the **gadget matrix** from [Micciancio-Peikert ’12]
- $Gx = 0$ can **always** be satisfied by some $x \in \{0, 1\}^n$
The $G$ in constrained-SIS

$G = \begin{pmatrix}
\log q \\
1 & 2 & 4 & \cdots & 2^\ell \\
0 & 1 & 2 & 4 & \cdots & 2^\ell \\
\vdots & \vdots & \ddots & \ddots & \vdots \\
0 & 0 & \cdots & 1 & 2 & 4 & \cdots & 2^\ell \\
\end{pmatrix}
\begin{pmatrix}
\star & \star & \star & \cdots & \star \\
\star & \star & \star & \cdots & \star \\
\vdots \\
\star & \star & \star & \cdots & \star \\
\end{pmatrix}
\begin{pmatrix}
\ldots \\
\ldots \\
\ldots \\
\ldots \\
\{0, 1\} \\
\{0, 1\} \\
\end{pmatrix}
$

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

$G = \begin{pmatrix}
\log q \\
\begin{array}{ccccccc}
1 & 2 & 4 & \cdots & 2^\ell & \star & \cdots & \star \\
0 & 1 & 2 & 4 & \cdots & 2^\ell & \star & \cdots & \star \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & \cdots & 1 & 2 & 4 & \cdots & 2^\ell & \star & \cdots & \star \\
\end{array}
\end{pmatrix}
\begin{pmatrix}
\star & \cdots & \star \\
\star & \cdots & \star \\
\vdots \\
\star & \cdots & \star \\
\end{pmatrix}
\begin{pmatrix}
\ldots \\
\ldots \\
\{0,1\} \\
\{0,1\}
\end{pmatrix}

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

$G = \begin{pmatrix}
\log q \\
1 & 2 & 4 & \cdots & 2^\ell & \cdots & \cdots & \cdots & \cdots & \cdots \\
0 & 1 & 2 & 4 & \cdots & 2^\ell & \cdots & \cdots & \cdots & \cdots \\
\vdots & \vdots & \vdots & \ddots & \vdots & \cdots & \cdots & \cdots & \cdots & \vdots \\
0 & 0 & \cdots & 1 & 2 & 4 & \cdots & 2^\ell \\
\end{pmatrix}
\begin{pmatrix}
\{0, 1\}
\end{pmatrix}
\Rightarrow
Gx = 0 \iff 1x_1 + 2x_2 + 4x_4 + \cdots + 2^\ell x_{2^\ell} = -\left[ \begin{array}{c} \star \star \star \cdots \star \\ n-d \log q \end{array} \right]

- $G$ similar to the gadget matrix from [Micciancio-Peikert '12]
- $Gx = 0$ can always be satisfied by some $x \in \{0, 1\}^n$
  - choose last $(n - d \log q)$ bits of $x$ arbitrarily (last row)...
  - $\ldots Gx = 0 \iff 1x_1 + 2x_2 + 4x_4 + \cdots + 2^\ell x_{2^\ell} = -\left[ \begin{array}{c} \star \star \star \cdots \star \\ n-d \log q \end{array} \right]
  - rest of $x$ is uniquely determined using backwards substitution & binary decomposition
an example with $d = 3$, $n = 10$, $q = 8$

\[
\begin{pmatrix}
1 & 2 & 4 & 3 & 0 & 6 & 5 & 6 & 2 & 1 \\
0 & 0 & 0 & 1 & 2 & 4 & 1 & 0 & 3 & 2 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 4 & 1
\end{pmatrix}
\cdot
\begin{pmatrix}
* \\
* \\
* \\
* \\
* \\
* \\
* \\
* \\
* \\
1
\end{pmatrix}
= \begin{pmatrix}
0 \\
0 \\
0
\end{pmatrix}
\pmod{8}
\]
an example with $d = 3, n = 10, q = 8$

\[
\begin{pmatrix}
1 & 2 & 4 & 3 & 0 & 6 & 5 & 6 & 2 & 1 \\
0 & 0 & 0 & 1 & 2 & 4 & 1 & 0 & 3 & 2 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 4 & 1
\end{pmatrix}
\cdot
\begin{pmatrix}
\star \\
\star \\
\star \\
\star \\
x_7 \\
x_8 \\
x_9 \\
1
\end{pmatrix}
= \begin{pmatrix}
0 \\
0 \\
0
\end{pmatrix} \pmod{8}
\]

**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 $d = 3, n = 10, q = 8$

$$
\begin{pmatrix}
1 & 2 & 4 & 3 & 0 & 6 & 5 & 6 & 2 & 1 \\
0 & 0 & 0 & 1 & 2 & 4 & 1 & 0 & 3 & 2 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 4 & 1
\end{pmatrix} \cdot \begin{pmatrix}
\ast \\
\ast \\
\ast \\
\ast \\
x_4 \\
x_5 \\
x_6 \\
1 \\
1 \\
1
\end{pmatrix} = \begin{pmatrix}
0 \\
0 \\
0
\end{pmatrix} \quad \text{(mod 8)}
$$

**binary decomposition (2nd row)**

$$1 \cdot x_4 + 2 \cdot x_5 + 4 \cdot x_6 + (1 + 2 + 4 + 1) = 0 \pmod{8}$$

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

\[
\begin{pmatrix}
1 & 2 & 4 & 3 & 0 & 6 & 5 & 6 & 2 & 1 \\
0 & 0 & 0 & 1 & 2 & 4 & 1 & 0 & 3 & 2 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 4 & 1
\end{pmatrix}
\cdot
\begin{pmatrix}
x_1 \\
x_2 \\
x_3
\end{pmatrix}
= \begin{pmatrix}
0 \\
0 \\
0
\end{pmatrix} \pmod{8}
\]

**binary decomposition (1st row)**

\[
1 \cdot x_1 + 2 \cdot x_2 + 4 \cdot x_3 + 17 = 0 \pmod{8}
\]

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

$$\begin{pmatrix} 1 & 2 & 4 & 3 & 0 & 6 & 5 & 6 & 2 & 1 \\ 0 & 0 & 0 & 1 & 2 & 4 & 1 & 0 & 3 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 4 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \pmod{8}$$
Goal: constrained-SIS is a hash function – pt2

- $2^n - d \log q$ different values of $x$ can satisfy $Gx = 0$
- same $x$ are mapped as: $x \rightarrow Ax$
Goal: constrained-SIS is a hash function – pt2

- \(2^{n-d \log q}\) different values of \(x\) can satisfy \(Gx = 0\)
- same \(x\) are mapped as: \(x \mapsto Ax\)
- range of \(x \mapsto Ax\) is \(q^m\)
Goal: constrained-SIS is a hash function – pt2

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

i.e. $|\text{domain}| > |\text{range}|$
Goal: constrained-SIS is a hash function – pt2

- $2^n - d \log q$ different values of $x$ can satisfy $Gx = 0$
- Same $x$ are mapped as: $x \mapsto Ax$
- Range of $x \mapsto Ax$ is $q^m$
- $2^n - d \log q > q^m \Rightarrow$ enough $x$ to have collisions in $A$
  i.e. $|\text{domain}| > |\text{range}|$

constrained-SIS is a hash function ✓

next: $C_h \leq$ constrained-SIS
Goal: $C_h \leq \text{constrained-SIS}$

Reduce $C_h$ to constrained-SIS
Goal: \( C_h \leq \text{constrained-SIS} \)

Reduce \( C_h \) to constrained-SIS

Goal: Construct \( A, G \) out of \( C_h \)
Goal: $C_h \leq \text{constrained-SIS}$

Reduce $C_h$ to constrained-SIS

Goal: Construct $A$, $G$ out of $C_h$

we start with $G$
Embed $C_h$ in $G$ using OR and XOR gates

- Circuit gates in $C_h$ $\rightarrow$ Linear equations in $G$
$C_h \leq$ constrained-SIS – the $G$ pt

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

**OR:**

\[
x_1 \lor x_2 = y, \quad z = x_1 \oplus x_2 \iff 1y + 2z - x_1 - x_2 = 0 \pmod{4}
\]

**XOR:**

\[
x_1 \oplus x_2 = y, \quad z = x_1 \land x_2 \iff 1y + 2z - x_1 - x_2 = 0 \pmod{4}
\]
Define $C_h \leq$ constrained-SIS – the G pt

- **Embed** $C_h$ in $G$ using OR and XOR gates

- Circuit gates in $C_h \rightarrow$ Linear equations in $G$

  - **OR:** $x_1 \lor x_2 = y, \ z = x_1 \oplus x_2 \iff 1y + 2z - x_1 - x_2 = 0 \pmod{4}$
  - **XOR:** $x_1 \oplus x_2 = y, \ z = x_1 \land x_2 \iff 1y + 2z - x_1 - x_2 = 0 \pmod{4}$

\[
G = \begin{pmatrix}
\text{output gate} & \text{output wires} & 0 & 0 & 0 \\
0 & \text{intermediate gate} & \text{intermediate wires} & 0 & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & \cdots & \text{input gate} & \text{input wires}
\end{pmatrix}
\]
\( C_h \preceq \text{constrained-SIS} – \text{the G pt} \)

- **Embed** \( C_h \) in \( G \) using **OR** and **XOR** gates
  - Circuit gates in \( C_h \) → Linear equations in \( G \)

  **OR:** \[
  \begin{align*}
  x_1 \lor x_2 = y, \quad z = x_1 \oplus x_2 \iff 1y + 2z - x_1 - x_2 &= 0 \pmod{4}
  \end{align*}
  \]

  **XOR:** \[
  \begin{align*}
  x_1 \oplus x_2 = y, \quad z = x_1 \land x_2 \iff 1y + 2z - x_1 - x_2 &= 0 \pmod{4}
  \end{align*}
  \]

\[
G = \begin{pmatrix}
\text{output gate} & \text{output wires} & 0 & 0 & 0 \\
0 & \text{intermediate gate} & \text{intermediate wires} & 0 & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & \ldots & \text{input gate} & \text{input wires}
\end{pmatrix}
\]

- \( Gx_1 = 0 = Gx_2 \) represents evaluation of \( C_h(x_1) \) and \( C_h(x_2) \)
  - \( x \) contains evaluation of \( C_h(x) \) gate-by-gate
  - \( x = (\text{output}, \text{intermediate steps}, \text{input})^T \)
$C_h \leq \text{constrained-SIS} – \text{the A pt}$

$\mathbf{A}$ extracts the output of $C_h(x)$ from $x$
$C_h \leq \text{constrained-SIS} - \text{the A pt}$

A extracts the output of $C_h(x)$ from $x$

- Set $A = (1, 0, 0)$
$C_h \leq \text{constrained-SIS} – \text{the A pt}$

A extracts the output of $C_h(x)$ from $x$

- Set $A = (1, 0, 0)$
- $x = (\text{output}, \text{intermediate steps}, \text{input})^T$
$C_h \leq$ constrained-SIS – the A pt

A extracts the output of $C_h(x)$ from x

- Set $A = (1, 0, 0)$
- $x = (\text{output, intermediate steps, input})^T$
- $Ax = \text{output} \implies Ax = C_h(x)$
$C_h \leq \text{constrained-SIS} – \text{the A pt}$

A extracts the output of $C_h(x)$ from $x$

- Set $A = (1, 0, 0)$
- $x = (\text{output, intermediate steps, input})^T$
- $Ax = \text{output} \implies Ax = C_h(x)$
- $Ax_1 = Ax_2$ implies $C_h(x_1) = C_h(x_2) \implies a \text{ collision for } h$
\( C_h \leq \text{constrained-SIS} – \text{the A pt} \)

A extracts the output of \( C_h(x) \) from x

- Set \( A = (1, 0, 0) \)
- \( x = (\text{output, intermediate steps, input})^T \)
- \( Ax = \text{output} \Rightarrow Ax = C_h(x) \)
- \( Ax_1 = Ax_2 \) implies \( C_h(x_1) = C_h(x_2) \Rightarrow \text{a collision for } h \checkmark \)

Find Short Integer Solutions for \( A, G \) \( \Rightarrow \) Find collisions in constrained-SIS
\( \Rightarrow \) Find collision in \( C_h \) for any \( h \)
\( \Rightarrow \) \( \text{\textdegree} \)

28
Series of reductions

Our result shows that:

SIS, LWE, SIVP, GapSVP, Minkowski, n-SVP, SHA, DLog, ... \leq \text{constrained–SIS}

These problems can be solved by finding collisions

Minkowski: \|v\|_2 \leq \sqrt{n} \det(L)^{1/n}
The post-quantum quest

A strong post-quantum guarantee for $h$?
A strong post-quantum guarantee for $\hat{h}$?

A quantum speedup for constrained-SIS

$\iff$ quantum speedup for finding collisions, in general
The post-quantum quest

A strong post-quantum guarantee for \( \hat{h} \)?

A quantum speedup for constrained-SIS
\( \iff \) quantum speedup for finding collisions, in general

"Morally", a quantum speedup for lattice problems(?)
\( \iff \) quantum speedup for finding collisions, in general
The post-quantum quest

A strong post-quantum guarantee for $h$?

A quantum speedup for constrained-SIS

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

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

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

This would be a strong implication. Should we expect this?
A strong post-quantum guarantee for $h$?

A quantum speedup for constrained-SIS

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

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

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

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

This motivates the open problem section
A worst-to-average case reduction from constrained-SIS to itself?

- Conjecture for $h$: $A \leftarrow \$, \ G \leftarrow$ semi-random (random $\star$) experiments?
A worst-to-average case reduction from constrained-SIS to itself?

- Conjecture for $h$: $A \leftarrow \$, \ G \leftarrow$ semi-random (random ⋆) 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 $h$: $A \leftarrow \$, \ G \leftarrow$ semi-random (random ⋆)
  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?
Open problems

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

- Conjecture for $h$: $A \leftarrow \$, \ G \leftarrow \text{semi-random (random \star)}$ 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
...mathematical principles that guarantee a solution...
...mathematical principles that guarantee a solution...

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

$$\sum_{v \in V} \deg(v) = 2|E|$$
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.
...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
How low can Factoring go?

Factoring \leq \text{approx-SVP/CVP?}
modern crypto (we use it everyday)
A non-exhaustive list:

- Public-key encryption – $(pk, sk)$ e.g. RSA
- Zero Knowledge Proofs
- Multiparty Computation
- Attribute-Based Encryption
- Fully-Homomorphic Encryption
...
A non-exhaustive list:

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

... 

*Super-tool to build crypto tools?*
A non-exhaustive list:

▶ Public-key encryption – \((p_k, s_k)\) e.g. RSA
▶ Zero Knowledge Proofs
▶ Multiparty Computation
▶ Attribute-Based Encryption
▶ Fully-Homomorphic Encryption

...  

Super-tool to build crypto tools?

.hammer Program Obfuscation
Program Obfuscation

Main character: programs

Goal: hide program secrets
What is obfuscation? (main character)

- An obfuscator is a program compiler

\[ x \rightarrow P \rightarrow P(x) \]
What is obfuscation? (main character $\rightarrow$ obf)

- An obfuscator is a program compiler.

$$x \rightarrow P \rightarrow P(x)$$

obfuscator
What is obfuscation? (obf → code)

▶ An obfuscator is a program compiler

\[ x \rightarrow P \rightarrow P(x) \]

\[ x \rightarrow \tilde{P} \rightarrow P(x) \]
What is obfuscation? (code hides secrets)

- An obfuscator is a **program compiler**

\[
x \rightarrow P \rightarrow P(x)
\]

\[
x \rightarrow \tilde{P} \rightarrow P(x)
\]

$\tilde{P}$ hides implementation details of $P$
e.g. constants, variable values, procedures
Virtual Black-Box (VBB) security [Had00, BGI+01]

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

\[ x \rightarrow P \rightarrow P(x) \]

**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?

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

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)

\[ f(\cdot) \overset{?}{=} y \]

params: \( f, y, m \)
Compute-and-Compare programs (input)

\[ f(x) = y \]

**params:** \( f, y, m \)
Compute-and-Compare programs

\[ f(x) = y \]

**params:** \( f, y, m \)

\[ f(x) \overset{?}{=} y \]

\[ f(x) = y \]

\[ m \]
Compute-and-Compare programs

\[ f(x) = y \]

\text{params: } f, y, m

\[ f(x) = y \]

\[ f(x) \neq y \]

\[ \perp \]
CC obfuscation & security

\[ f(x) \overset{?}{=} y \]

params: \( f, y, m \)

\[ \text{Black-Box simulation security when } y \text{ is random given } f, m \]

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 \( y \)
  - Correctness is meaningful
  - Security, not meaningful

- Cannot predict \( y \)
  - Correctness, not meaningful
  - Security is meaningful
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 $\text{Enc}(pk, sk_i)$ always secure?
  - bit-by-bit encryption of $\text{msg} = sk$

- 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 $\text{Enc}(pk, sk_i)$
  - the only heuristic transformation known

- Why investigate this type of security?
A fundamental question [GM’84]

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

- 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 $\text{Enc}(\text{pk}, \text{sk}_i)$
  - 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. $\text{RO}(x) = \text{random}$

- In practice, replacing $\text{RO} = \text{SHA-2/SHA-3}$
- In theory, replacing $\text{RO} = \text{it's complicated}$
Can Random Oracles really help?

Power of RO

- Transform any IND-CPA scheme to a circular secure one [BRS03]
- $\text{Enc}_{RO}(pk, m) = \text{Enc}(pk, r), \ RO(r) \oplus m$
Can Random Oracles really help?

Power of RO

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

Power of obfuscation

- We construct an IND-CPA scheme that \textbf{cannot} be upgraded as above...
  
  ...no matter which hash function is used to implement RO
Circular insecurity: \text{sem} \rightarrow \text{circ-insec}

Assume bit encryption

\textbf{secret key:} \ Dec(sk, \cdot) \hspace{2cm} \textbf{public key:} \ Enc(pk, \cdot)
Circular insecurity: $\text{sem} \rightarrow \text{circ-insec}$

Assume bit encryption

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

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

$y \leftarrow \$$

$\Rightarrow \text{sk' } \rightarrow (sk, y)$
Circular insecurity: $\text{sem} \rightarrow \text{circ-insec}$

Assume bit encryption

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

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

\[
y \leftarrow \$ \quad \text{Dec}_{sk}(\cdot) \overset{?}{=} y
\]

params: $sk, y, sk$

$\text{pk'} \rightarrow (pk, \text{Obf})$

$\text{sk'} \rightarrow (sk, y)$
Circular insecurity: $\text{sem} \rightarrow \text{circ-insec}$

Assume bit encryption

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

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

\[ y \leftarrow \$

$\text{Dec}_{sk}(\cdot) \overset{?}{=} y$

**params:** $sk, y, sk$

$\text{pk}' \rightarrow (pk, \text{Obf})$

$y$ indep of $sk \Rightarrow$ semantic security
Circular insecurity: $\text{sem} \rightarrow \text{circ-insec}$

Assume bit encryption

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

$y \leftarrow \$\n
$\blacktriangleright \; sk' \rightarrow (sk, y)$

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

$\text{Enc}_{pk}(y) \rightarrow \text{Dec}_{sk}(\cdot) \overset{?}{=} y$

- params: $sk, y, sk$

$\blacktriangleright \; pk' \rightarrow (pk, \text{Obf})$

$\blacktriangleright \; y \text{ indep of } sk \Rightarrow \text{semantic security}$
Circular insecurity: \( \text{sem} \rightarrow \text{circ-insec} \)

Assume bit encryption

\[
\begin{align*}
\text{secret key: } & \quad \text{Dec}(sk, \cdot) \\
\text{public key: } & \quad \text{Enc}(pk, \cdot) \\
\end{align*}
\]

\[
\begin{align*}
y & \leftarrow $ \\
\text{Enc}_{pk}(y) & \rightarrow \text{Dec}_sk(\cdot) \overset{?}{=} y \\
\text{params: } & \quad sk, y, sk \\
\text{pk}' & \rightarrow (pk, \text{Obf}) \\
\text{y indep of sk} & \Rightarrow \text{semantic security} \\
\text{recover sk} & \Rightarrow \text{break security!}
\end{align*}
\]
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?

Tales of obfuscation:
- "relaxed" form of obfuscation ⇒ almost all crypto thumbs-up
- iO→ ndistinguishability
- 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-ZTCC'18

\[
\frac{a(b+c)}{\iota} \approx ab + ac
\]
"relaxed" form of obfuscation ⇒ almost all crypto

indistinguishability Obfuscation

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 👍
- indistinguishability Obfuscation
- 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
  - ...
is obfuscation a success story?

- "relaxed" form of obfuscation ⇒ almost all crypto

- indistinguishability Obfuscation

- 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

\[ a(b + c) \approx ab + ac \]
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 → 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 → new ideas

- IBE/ABE followed after PKE. Next primitive to _____?
  - LWR, LPN
Thank you!