Approximating Fully Homomorphic Encryption with Secure Hardware

Chris Fletcher
Marten van Dijk
Srini Devadas
Ling Ren
Xiangyao Yu

Massachusetts Institute of Technology
Department of EECS & CSAIL
Motivation

How do you trust the cloud to do computation on your private data … *while keeping your data private?*
Motivation

How do you trust the cloud to do computation on your private data … while keeping your data private?
Motivation

How do you trust the cloud to do computation on your private data … **while keeping your data private?**

Hidden requests:
1.) Do I have $E_k$(lyme) disease? I’ve been $E_k$(coughing)?

Untrusted servers, public programs, many resources

Hidden responses:
1.) $E_k$(Yes, you have lyme disease)

Users, limited resources

$E_k(A) = \text{encryption of } A$
Motivation

How do you trust the cloud to do computation on your private data … *while keeping your data private?*

**Hidden requests:**
1.) Do I have $E_k(\text{lyme})$ disease? I’ve been $E_k(\text{coughing})$?
2.) Compute taxes I owe given my $E_k(\text{W2})$ forms.
3.) Give me a route from $E_k(\text{home})$ to $E_k(\text{Stata})$

**Untrusted servers,**
**public programs,**
**many resources**

**Hidden responses:**
1.) $E_k(\text{Yes, you have lyme disease})$
2.) $E_k(\text{$984.00})$
3.) $E_k(\text{Take a right, left, right, right …})$

$E_k(A) =$ encryption of $A$
Motivation

How do you trust the cloud to do computation on your private data ... while keeping your data private?

Hidden requests:
1.) Do I have $E_k$(lyme) disease? I’ve been $E_k$(coughing)?
2.) Compute taxes I owe given my $E_k$(W2) forms.
3.) Give me a route from $E_k$(home) to $E_k$(Stata)

Hidden responses:
1.) $E_k$(Yes, you have lyme disease)
2.) $E_k($984.00)
3.) $E_k$(Take a right, left, right, right ...)

Server cannot learn anything about encrypted data
What’s on the menu

• Techniques for encrypted computation
• Setting
  – General purpose *batch* programs
  – Given a batch program + stock compiler (e.g., gcc)
  – … some simple program transformations allowed
What’s on the menu

• Techniques for encrypted computation
• Setting
  – General purpose batch programs
  – Given a batch program + stock compiler (e.g., gcc)
  – ... some simple program transformations allowed
What’s on the menu

- Techniques for encrypted computation
- Setting
  - General purpose *batch* programs
  - Given a batch program + stock compiler (e.g., gcc)
  - … some simple program transformations allowed

Part 1: FHE
Overhead from crypto/“program ambiguity”
No TCB
What’s on the menu

• Techniques for encrypted computation
• Setting
  – General purpose *batch* programs
  – Given a batch program + stock compiler (e.g., gcc)
  – … some simple program transformations allowed

---

**Part 1: FHE**

Overhead from crypto/“program ambiguity”
No TCB
What’s on the menu

• Techniques for encrypted computation
• Setting
  – General purpose *batch* programs
  – Given a batch program + stock compiler (e.g., gcc)
  – … some simple program transformations allowed

---

**Part 1: FHE**

Overhead from crypto/“program ambiguity”

No TCB

**Part 2: Secure processor**

Overhead from obfuscation

TCB = processor chip

“Approximation to FHE”
FHE background

- **FHE**: performing computation on ciphertexts ($E_k$(plain text)) without access to the decryption key

\[
\begin{align*}
\text{FHE.add}(E_k(2), E_k(3)) &= E_k(5) \\
\text{FHE.multiply}(E_k(2), E_k(3)) &= E_k(6)
\end{align*}
\]

With these, FHE can evaluate any circuit

- FHE satisfies the “Honest but Curious” model

- Ciphertexts have “noise”

*FHE contributions: C. Gentry, S. Halevi, Z. Brakerski, V. Vaikuntanathan, N.P. Smart …
FHE background

- **FHE**: performing computation on ciphertexts \( E_k(\text{plain text}) \) without access to the decryption key

\[
\begin{align*}
\text{FHE.add}(E_k(2), E_k(3)) &= E_k(5) \\
\text{FHE.multiply}(E_k(2), E_k(3)) &= E_k(6)
\end{align*}
\]

- The catch: they have about a factor of a billion overhead

- FHE satisfies the “Honest but Curious” model

- Ciphertexts have “noise”

*FHE contributions: C. Gentry, S. Halevi, Z. Brakerski, V. Vaikuntanathan, N.P. Smart …*
FHE background

- **FHE**: performing computation on ciphertexts \((E_k(\text{plain text}))\) *without access to the decryption key*

  \[
  \begin{align*}
  \text{FHE.add}(E_k(2), E_k(3)) &= E_k(5) \\
  \text{FHE.multiply}(E_k(2), E_k(3)) &= E_k(6)
  \end{align*}
  \]

  With these, FHE can evaluate any circuit

- The catch: they have about a factor of a billion overhead

- FHE satisfies the “Honest but Curious” model
  - Server is trusted to run the correct program and return correct results but may run other programs to try to learn about the data
  - *This means the program(s) that is (are) run are not trusted*

- Ciphertexts have “noise”

*FHE contributions: C. Gentry, S. Halevi, Z. Brakerski, V. Vaikuntanathan, N.P. Smart …*
FHE background

- **FHE:** performing computation on ciphertexts \((E_k(\text{plain text}))\) without access to the decryption key

\[
\begin{align*}
\text{FHE.add}(E_k(2), E_k(3)) &= E_k(5) \\
\text{FHE.multiply}(E_k(2), E_k(3)) &= E_k(6)
\end{align*}
\]

- The catch: they have about a factor of a billion overhead

- FHE satisfies the “Honest but Curious” model
  - Server is trusted to run the correct program and return correct results but may run other programs to try to learn about the data
  - *This means the program(s) that is (are) run are not trusted*

- Ciphertexts have “noise”
  - Ciphertext noise grows as x/+ ops are performed on the ciphertext
  - Noise is additive w/ + ops; **Grows exponentially** w/ multiply ops
  - Once noise exceeds threshold, ciphertext must be **refreshed**

*FHE contributions: C. Gentry, S. Halevi, Z. Brakerski, V. Vaikuntanathan, N.P. Smart …*
FHE background

• **FHE**: performing computation on ciphertexts ($E_k(\text{plain text})$) without access to the decryption key

  \[
  \text{FHE.add}(E_k(2), E_k(3)) = E_k(5)
  \]
  \[
  \text{FHE.multiply}(E_k(2), E_k(3)) = E_k(6)
  \]

  With these, FHE can evaluate any circuit

• The catch: they have about a factor of a billion overhead

• FHE satisfies the “Honest but Curious” model
  – Server is trusted to run the correct program and return correct results but may run other programs to try to learn about the data
  – *This means the program(s) that is (are) run are not trusted*

• Ciphertexts have “noise”  
  (R-LWE + modulus switching)
  – Ciphertext noise grows as x/+ ops are performed on the ciphertext
  – Noise is additive w/ + ops; **Grows exponentially** w/ multiply ops
  – Once noise exceeds threshold, ciphertext must be **refreshed**

*FHE contributions: C. Gentry, S. Halevi, Z. Brakerski, V. Vaikuntanathan, N.P. Smart …*
2-interactive protocols

• One interaction between user and server
2-interactive protocols

- One interaction between user and server

**Example: compute your taxes**

1.) User sends $E_k(x = W2 \text{ forms})$.  
   $T = \text{"run for one hour."}$

2.) Server runs $P = \text{TaxComp}(x, y)$.  
   $y = \text{public tax codes}$.

   * User has not seen/certified TaxComp().

3.) Server returns $E_k(\text{You owe govt. $900})$ or $E_k(\text{Need more time})$.
2-interactive protocols

- One interaction between user and server

1. \( E_k(x), \) Time budget \( T \)

User

Untrusted Server

Program \( P, \)
Public inputs \( y, \)
unlimited resources,
unlimited compute

2. Perform computation for \( O(T) \) time

3. \( E_k(P(x, y)) \) or \( E_k(\text{intermediate result}) \)

Example: compute your taxes

1.) User sends \( E_k(x = \text{W2 forms}) \).
   \( T = \text{“run for one hour.”} \)

2.) Server runs \( P = \text{TaxComp}(x, y) \).
   \( y = \text{public tax codes.} \)

   * User has not seen/certified TaxComp().

3.) Server returns \( E_k(\text{You owe govt. $900}) \) or \( E_k(\text{Need more time}) \)

- We only want to leak
  - Time budget \( T \) (an estimate of the running time for \( P(x, y) \))
  - The number of bits in \( E_k(x) \)

This can be shown to the be the least leakage possible.
Computation under encryption
Computation under encryption

- Encrypted
- Not encrypted
Computation under encryption

- **Encrypted**
  - Memory $M$ (init. $E_k(x)$)
  - $PC'$ (in $M$)

- **Not encrypted**

<table>
<thead>
<tr>
<th></th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$d$</th>
<th>$PC'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M$:</td>
<td>$E_k(...)$</td>
<td>$E_k(...)$</td>
<td>$E_k(...)$</td>
<td>$E_k(...)$</td>
<td>$E_k(...)$</td>
</tr>
</tbody>
</table>
Computation under encryption

- **Encrypted**
  - Memory $M$ (init. $E_k(x)$)
  - $PC'$ (in $M$)

- **Not encrypted**
  - Time budget $T$
  - FHE circuits for basic blocks $B_1 \ldots B_N$
  - Public address per basic block: $PC(B_{1..N})$

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$d$</th>
<th>$PC'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E_k(\ldots)$</td>
<td>$E_k(\ldots)$</td>
<td>$E_k(\ldots)$</td>
<td>$E_k(\ldots)$</td>
<td>$E_k(\ldots)$</td>
</tr>
</tbody>
</table>

For $B_i$:
- $M[a] = M[b] - M[c]$
- $M[d] = M[b] - M[a]$
- If $M[d] < 0$: $PC' = PC(BX)$
- else: $PC' = PC(BY)$

$M[a] \rightarrow M[b] \rightarrow M[c] \rightarrow PC'$
Computation under encryption

- **Encrypted**
  - Memory $M$ (init. $E_k(x)$)
  - $PC'$ (in $M$)

- **Not encrypted**
  - Time budget $T$
  - FHE circuits for basic blocks $B_1 \ldots B_N$
  - Public address per basic block: $PC(B_{1..N})$

M:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>PC'</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E_k(\cdots)$</td>
<td>$E_k(\cdots)$</td>
<td>$E_k(\cdots)$</td>
<td>$E_k(\cdots)$</td>
<td>$E_k(\cdots)$</td>
</tr>
</tbody>
</table>

$B_i$: $M[a] = M[b] - M[c]$  
$M[d] = M[b] - M[a]$  
If $M[d] < 0$: $PC' = PC(BX)$  
else: $PC' = PC(BY)$

Just arithmetic operations + branching for now
Computation under encryption

- Encrypted
  - Memory $M$ (init. $E_k(x)$)
  - $PC'$ (in $M$)

- Not encrypted
  - Time budget $T$
  - FHE circuits for basic blocks $B_1 \ldots B_N$
  - Public address per basic block: $PC(B_{1..N})$

\[
M_{t+1} = \sum_{n=1}^{N} \left( c_n \ast B_n(M_t) \right) \\
M[a] = M[b] - M[c] \\
M[d] = M[b] - M[a] \\
\text{If } M[d] < 0: \text{ } PC' = PC(BX) \\
\text{else: } \text{ } PC' = PC(BY)
\]


$E_k(\ldots)$ $E_k(\ldots)$ $E_k(\ldots)$ $E_k(\ldots)$ $E_k(\ldots)$

$\begin{array}{|c|c|c|c|c|c|}
\hline
\text{a} & \text{b} & \text{c} & \text{d} & \text{PC'} \\
\hline
\text{E_k(\ldots)} & \text{E_k(\ldots)} & \text{E_k(\ldots)} & \text{E_k(\ldots)} & \text{E_k(\ldots)} \\
\hline
\end{array}$

Naïve scheme to perform computation (for $t = 1 \ldots T$)

\[
c_n = \left( PC(B_n) \equiv PC' \right) \ast 1 : 0
\]
Computation under encryption

- Encrypted
  - Memory M (init. $E_k(x)$)
  - PC' (in M)

- Not encrypted
  - Time budget $T$
  - FHE circuits for basic blocks $B_1 \ldots B_N$
  - Public address per basic block: PC($B_{1..N}$)

\[
M_{t+1} = \sum_{n=1}^{N} \left( c_n \ast B_n(M_t) \right)
\]

$C_n = \begin{cases} 1 & \text{if } PC(B_n) \equiv PC' \\ 0 & \text{else} \end{cases}$

Conditional instruction execution

Just arithmetic operations + branching for now
Computation under encryption

- **Encrypted**
  - Memory $M$ (init. $E_k(x)$)
  - $PC'$ (in $M$)

- **Not encrypted**
  - Time budget $T$
  - FHE circuits for basic blocks $B_1 \ldots B_N$
  - Public address per basic block: $PC(B_{1..N})$

\[
\begin{array}{c|c|c|c|c}
\text{a} & \text{b} & \text{c} & \text{d} & \text{PC}' \\
\hline
E_k(\ldots) & E_k(\ldots) & E_k(\ldots) & E_k(\ldots) & E_k(\ldots) \\
\end{array}
\]

- \text{Naïve scheme to perform computation (for } t = 1 \ldots T) \\
\[ M_{t+1} = \sum_{n=1}^{N} \left( c_n \ast B_n(M_t) \right) \]
\[ c_n = \left( PC(B_n) \equiv PC' \right)?1:0 \]
- 1 basic block worth of forward progress (but its not efficient)
Computation under encryption

- **Encrypted**
  - Memory $M$ (init. $E_k(x)$)
  - $PC'$ (in $M$)

- **Not encrypted**
  - Time budget $T$
  - FHE circuits for basic blocks $B_1 \ldots B_N$
  - Public address per basic block: $PC(B_{1..N})$

\[
M: \begin{array}{cccccc}
  a & b & c & d & PC' \\
  E_k(\ldots) & E_k(\ldots) & E_k(\ldots) & E_k(\ldots) & E_k(\ldots)
\end{array}
\]

\[
B_i: \\
M[a] = M[b] - M[c] \\
M[d] = M[b] - M[a] \\
\text{If } M[d] < 0: PC' = PC(B_X) \\
\text{else: } PC' = PC(B_Y)
\]

Just arithmetic operations + branching for now

- *Naïve scheme to perform computation (for $t = 1 \ldots T$)*

\[
M_{t+1} = \sum_{n=1}^{N} \left( c_n \ast B_n(M_t) \right) \\
c_n = \left( PC(B_n) \equiv PC' \right) ? 1 : 0
\]

- 1 basic block worth of forward progress (but it's not efficient)
- Noise increases through ‘*’ ops
Control flow ambiguity

- Efficiency increases as $N$ decreases

$$M_{t+1} = \sum_{n=1}^{N} (c_n \ast B_n (M_t))$$
Control flow ambiguity

- Efficiency increases as $N$ decreases
  \[ M_{t+1} = \sum_{n=1}^{N} \left( c_n \cdot B_n(M_t) \right) \]
- Control flow ambiguity is set of possible values for PC’
Control flow ambiguity

- Efficiency increases as $N$ decreases
  \[ M_{t+1} = \sum_{n=1}^{N} (c_n \times B_n(M_t)) \]

- Control flow ambiguity is set of possible values for PC'}
Control flow ambiguity

- Efficiency increases as $N$ decreases

$$M_{t+1} = \sum_{n=1}^{N} \left( c_n \times B_n(M_t) \right)$$

- Control flow ambiguity is set of possible values for PC'

**Diagram:**

Left:
- Iteration 1
- $N=1$  A
- Diagram with nodes A, B, C, D, E, F, G, H

Right:
- Diagram with nodes A, B, C, D, E, F, H
Control flow ambiguity

• Efficiency increases as N decreases

\[ M_{t+1} = \sum_{n=1}^{N} (c_n \ast B_n(M_t)) \]

• Control flow ambiguity is set of possible values for PC’

**Iteration 1**

N=1   A
N=2   B, C
Control flow ambiguity

- Efficiency increases as N decreases
  \[ M_{t+1} = \sum_{n=1}^{N} (c_n \ast B_n(M_t)) \]
- Control flow ambiguity is set of possible values for PC’

**Iteration 1**

N=1  A
N=2  B, C
N=4  D, E, F, G
Control flow ambiguity

• Efficiency increases as N decreases

\[ M_{t+1} = \sum_{n=1}^{N} \left( c_n \times B_n(M_t) \right) \]

• Control flow ambiguity is set of possible values for PC’

**Iteration 1**

N=1  A
N=2  B, C
N=4  D, E, F, G
N=1  H
Control flow ambiguity

- Efficiency increases as N decreases

\[ M_{t+1} = \sum_{n=1}^{N} (c_n \ast B_n(M_t)) \]

- Control flow ambiguity is set of possible values for PC'

<table>
<thead>
<tr>
<th>Iteration 1</th>
<th>Iteration 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>N=1  A</td>
<td>A</td>
</tr>
<tr>
<td>N=2  B, C</td>
<td>B, C</td>
</tr>
<tr>
<td>N=4  D, E, F, G</td>
<td>D, E, F, G</td>
</tr>
<tr>
<td>N=1  H</td>
<td>H</td>
</tr>
</tbody>
</table>
Control flow ambiguity

- Efficiency increases as N decreases

\[
M_{t+1} = \sum_{n=1}^{N} (c_n * B_n(M_t))
\]

- Control flow ambiguity is set of possible values for PC’

<table>
<thead>
<tr>
<th>Iteration 1</th>
<th>Iteration 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>N=1</td>
<td>A</td>
</tr>
<tr>
<td>N=2</td>
<td>B, C</td>
</tr>
<tr>
<td>N=4</td>
<td>D, E, F, G</td>
</tr>
<tr>
<td>N=1</td>
<td>H</td>
</tr>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>B, C</td>
</tr>
<tr>
<td></td>
<td>D, E, F, G</td>
</tr>
<tr>
<td></td>
<td>H</td>
</tr>
</tbody>
</table>

Iteration 1

<table>
<thead>
<tr>
<th>Iteration 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
</tbody>
</table>

Iteration 2

<table>
<thead>
<tr>
<th>Iteration 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
</tbody>
</table>
Control flow ambiguity

- Efficiency increases as N decreases

\[ M_{t+1} = \sum_{n=1}^{N} (c_n \ast B_n(M_t)) \]

- Control flow ambiguity is set of possible values for PC’

<table>
<thead>
<tr>
<th>Iteration 1</th>
<th>Iteration 2</th>
<th>Iteration 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( N=1 ) A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>( N=2 ) B, C</td>
<td>B, C</td>
<td>B, C</td>
</tr>
<tr>
<td>( N=4 ) D, E, F, G</td>
<td>D, E, F, G</td>
<td>D, E, F, G</td>
</tr>
<tr>
<td>( N=1 ) H</td>
<td>H</td>
<td>H</td>
</tr>
</tbody>
</table>

\[ M_{t+1} = \sum_{n=1}^{N} (c_n \ast B_n(M_t)) \]
Control flow ambiguity

- Efficiency increases as N decreases
  \[ M_{t+1} = \sum_{n=1}^{N} (c_n \times B_n(M_t)) \]

- Control flow ambiguity is set of possible values for PC’

<table>
<thead>
<tr>
<th>Iteration 1</th>
<th>Iteration 2</th>
<th>Iteration 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>N=1 A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>N=2 B, C</td>
<td>B, C</td>
<td>B, C</td>
</tr>
<tr>
<td>N=4 D, E, F, G</td>
<td>D, E, F, G</td>
<td>D, E, F, H</td>
</tr>
<tr>
<td>N=1 H</td>
<td>H</td>
<td>H</td>
</tr>
</tbody>
</table>

- Graphs representing the iterations and control flow ambiguity.
Control flow ambiguity

- Efficiency increases as N decreases

\[ M_{t+1} = \sum_{n=1}^{N} (c_n \ast B_n(M_t)) \]

- Control flow ambiguity is set of possible values for PC’

### Iteration 1
- N=1: A
- N=2: B, C
- N=4: D, E, F, G
- N=1: H

### Iteration 2
- N=1: A
- N=2: B, C
- N=4: D, E, F, G
- N=1: H

**Problem:** PC’ is encrypted
Control flow ambiguity

- Efficiency increases as N decreases
  \[ M_{t+1} = \sum_{n=1}^{N} (c_n * B_n(M_t)) \]

- Control flow ambiguity is set of possible values for PC’

<table>
<thead>
<tr>
<th>Iteration 1</th>
<th>Iteration 2</th>
<th>Iteration 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>N=1 A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>N=2 B, C</td>
<td>B, C</td>
<td>B, C</td>
</tr>
<tr>
<td>N=4 D, E, F, G</td>
<td>D, E, F, G</td>
<td>D, E, F, H</td>
</tr>
<tr>
<td>N=1 H</td>
<td>H</td>
<td>H, A</td>
</tr>
</tbody>
</table>

Problem: PC’ is encrypted
Control flow ambiguity

- Efficiency increases as $N$ decreases

\[ M_{t+1} = \sum_{n=1}^{N} \left( c_n \times B_n(M_t) \right) \]

- Control flow ambiguity is set of possible values for PC’

<table>
<thead>
<tr>
<th>Iteration 1</th>
<th>Iteration 2</th>
<th>Iteration 1</th>
<th>Iteration 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N=1$ A</td>
<td>A</td>
<td>A</td>
<td>N=3 A, B, C</td>
</tr>
<tr>
<td>$N=2$ B, C</td>
<td>B, C</td>
<td>B, C</td>
<td></td>
</tr>
<tr>
<td>$N=4$ D, E, F, G</td>
<td>D, E, F, G</td>
<td>D, E, F, H</td>
<td></td>
</tr>
<tr>
<td>$N=1$ H</td>
<td>H</td>
<td>H, A</td>
<td></td>
</tr>
</tbody>
</table>
Control flow ambiguity

- Efficiency increases as N decreases
\[ M_{t+1} = \sum_{n=1}^{N} (c_n \ast B_n(M_t)) \]

- Control flow ambiguity is set of possible values for PC’

Iteration 1

<table>
<thead>
<tr>
<th>N</th>
<th>Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>B, C</td>
</tr>
<tr>
<td>4</td>
<td>D, E, F, G</td>
</tr>
<tr>
<td>1</td>
<td>H</td>
</tr>
</tbody>
</table>

Iteration 2

<table>
<thead>
<tr>
<th>N</th>
<th>Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>A, B, C</td>
</tr>
<tr>
<td>6</td>
<td>B, C, D, E, F, H</td>
</tr>
</tbody>
</table>

Problem:
PC’ is encrypted
Control flow ambiguity

- Efficiency increases as N decreases
  \[ M_{t+1} = \sum_{n=1}^{N} (c_n \ast B_n(M_t)) \]

- Control flow ambiguity is set of possible values for PC’

<table>
<thead>
<tr>
<th>Iteration 1</th>
<th>Iteration 2</th>
<th>Iteration 1</th>
<th>Iteration 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>N=1 A</td>
<td>A</td>
<td>A</td>
<td>N=3 A, B, C</td>
</tr>
<tr>
<td>N=2 B, C</td>
<td>B, C</td>
<td>B, C</td>
<td>N=6 B, C, D, E, F, H</td>
</tr>
<tr>
<td>N=4 D, E, F, G</td>
<td>D, E, F, G</td>
<td>D, E, F, H</td>
<td>...</td>
</tr>
<tr>
<td>N=1 H</td>
<td>H</td>
<td>H</td>
<td>H, A</td>
</tr>
</tbody>
</table>
Control flow ambiguity

- Efficiency increases as $N$ decreases
  \[ M_{t+1} = \sum_{n=1}^{N} \left( c_n \ast B_n(M_t) \right) \]

- Control flow ambiguity is set of possible values for $PC'$

<table>
<thead>
<tr>
<th>Iteration 1</th>
<th>Iteration 2</th>
<th>Iteration 1</th>
<th>Iteration 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N=1$ A</td>
<td>A</td>
<td>A</td>
<td>$N=3$ A, B, C</td>
</tr>
<tr>
<td>$N=2$ B, C</td>
<td>B, C</td>
<td>B, C</td>
<td>$N=6$ B, C, D, E, F, H</td>
</tr>
<tr>
<td>$N=4$ D, E, F, G</td>
<td>D, E, F, G</td>
<td>D, E, F, H</td>
<td>...</td>
</tr>
<tr>
<td>$N=1$ H</td>
<td>H</td>
<td>H</td>
<td>$N=7$ A, B, C, D, E, F, G, H</td>
</tr>
</tbody>
</table>

Problem: $PC'$ is encrypted
Control flow ambiguity

- Efficiency increases as $N$ decreases

$$M_{t+1} = \sum_{n=1}^{N} (c_n * B_n(M_t))$$

- Control flow ambiguity is set of possible values for $PC'$

**Problem:** $PC'$ is encrypted

<table>
<thead>
<tr>
<th>Iteration 1</th>
<th>Iteration 2</th>
<th>Iteration 1</th>
<th>Iteration 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N=1$</td>
<td>$A$</td>
<td>$N=3$</td>
<td>$A$, $B$, $C$</td>
</tr>
<tr>
<td>$N=2$</td>
<td>$B$, $C$</td>
<td>$N=6$</td>
<td>$B$, $C$, $D$, $E$, $F$, $H$</td>
</tr>
<tr>
<td>$N=4$</td>
<td>$D$, $E$, $F$, $G$</td>
<td>$N=7$</td>
<td>$A$, $B$, $C$, $D$, $E$, $F$, $G$, $H$</td>
</tr>
<tr>
<td>$N=1$</td>
<td>$H$</td>
<td>$H$, $A$</td>
<td></td>
</tr>
</tbody>
</table>
Path levelization

- **Goal:** reduce absolute computation/noise
Path levelization

- **Goal**: reduce absolute computation/noise

\[
M_{t+1} = \sum_{m=1}^{L_n} \left( c_{n,m} * L_{n,m}(M_t) \right) + \left( 1 - \sum_{m=1}^{L_n} c_{n,m} \right) * M_t
\]
Path levelization

- **Goal:** reduce absolute computation/noise

\[
M_{t+1} = \sum_{m=1}^{|L_n|} (c_{n,m} \ast L_{n,m}(M_t)) + \left(1 - \sum_{m=1}^{L_n} c_{n,m}\right) \ast M_t
\]

- \(L_n = \) the list of basic blocks whose max path length from the top of the loop is \(n\)

- \(L_{n,m} = \) the \(m^{th}\) basic block in level \(n\)
Path levelization

- **Goal:** reduce absolute computation/noise

\[ M_{t+1} = \sum_{m=1}^{\left| L_n \right|} \left(c_{n,m} \ast L_{n,m}(M_t)\right) + \left(1 - \sum_{m=1}^{\left| L_n \right|} c_{n,m}\right) \ast M_t \]

- \( L_n = \) the list of basic blocks whose max path length from the top of the loop is \( n \)
- \( L_{n,m} = \) the \( m^{th} \) basic block in level \( n \)
Path levelization

- **Goal:** reduce absolute computation/noise

\[
M_{t+1} = \sum_{m=1}^{|L_n|} (c_{n,m} \times L_{n,m}(M_t)) + \left( 1 - \sum_{m=1}^{|L_n|} c_{n,m} \right) \times M_t
\]

- \(L_n = \) the list of basic blocks whose max path length from the top of the loop is \(n\)
- \(L_{n,m} = \) the \(m^{th}\) basic block in level \(n\)
Path levelization

- **Goal**: reduce absolute computation/noise

\[
M_{t+1} = \sum_{m=1}^{L_n} (c_{n,m} \times L_{n,m}(M_t)) + \left(1 - \sum_{m=1}^{L_n} c_{n,m}\right) \times M_t
\]

- \(L_n\) = the list of basic blocks whose max path length from the top of the loop is \(n\)
- \(L_{n,m}\) = the \(m^{th}\) basic block in level \(n\)
Path levelization

- **Goal:** reduce absolute computation/noise

\[
M_{t+1} = \sum_{m=1}^{L_n} \left( c_{n,m} * L_{n,m}(M_t) \right) + \left( 1 - \sum_{m=1}^{L_n} c_{n,m} \right) \cdot NOP
\]

- \( L_n = \) the list of basic blocks whose max path length from the top of the loop is \( n \)
- \( L_{n,m} = \) the \( m^{th} \) basic block in level \( n \)
**Path levelization**

- **Goal:** reduce absolute computation/noise

\[
M_{t+1} = \sum_{m=1}^{\left|L_n\right|} \left(c_{n,m} \ast L_{n,m}(M_t)\right) + \left(1 - \sum_{m=1}^{\left|L_n\right|} c_{n,m}\right) \ast M_t
\]

- \(L_n = \) the list of basic blocks whose max path length from the top of the loop is \(n\)
- \(L_{n,m} = \) the \(m^{th}\) basic block in level \(n\)

Noise grows per level
Path levelization

- **Goal:** reduce absolute computation/noise

\[ M_{t+1} = \sum_{m=1}^{\left| L_n \right|} \left( c_{n,m} \times L_{n,m}(M_t) \right) + \left( 1 - \sum_{m=1}^{\left| L_n \right|} c_{n,m} \right) \times M_t \]

- \( L_n = \) the list of basic blocks whose max path length from the top of the loop is \( n \)
- \( L_{n,m} = \) the \( m^{th} \) basic block in level \( n \)

\[ M_{t+1} = c_n \times B_n(M_t) + (1 - c_n) \times M_t \]
Path levelization

- **Goal:** reduce absolute computation/noise

\[ M_{t+1} = \sum_{m=1}^{\left| L_n \right|} (c_{n,m} \ast L_{n,m}(M_t)) + \left(1 - \sum_{m=1}^{\left| L_n \right|} c_{n,m}\right) \ast M_t \]

- \( L_n = \) the list of basic blocks whose max path length from the top of the loop is \( n \)

- \( L_{n,m} = \) the \( m^{th} \) basic block in level \( n \)

Noise grows per basic block!

Noise grows per level
Program analysis

- Reset noise on PC’ at dominator basic blocks
  - PC’ is the noise limiting variable

Dominator = basic block that all paths must cross
Program analysis

- Reset noise on PC’ at dominator basic blocks
  - PC’ is the noise limiting variable

Basic block $B_0$: $M[a]$, $M[b]$, $M[c]$

- $M[d]$, PC’

If $PC(B_0) == PC'$:

\[
\begin{align*}
M[a] &= M[a] - M[b] \\
M[d] &= M[a] - M[c]
\end{align*}
\]

if $M[a] < 0$: $PC' = PC(B_x)$
else: $PC' = PC(B_y)$

Dominator = basic block that all paths must cross
Program analysis

• Reset noise on PC' at dominator basic blocks
  – PC' is the noise limiting variable

Basic block $B_0$: $M[a] \quad M[b] \quad M[c]$

  $M[d] \quad PC'$

If $PC(B_0) == PC'$:
  $M[a] = (PC(B_0) == PC') ? M[a] - M[b] : M[a]$
  $M[d] = (PC(B_0) == PC') ? M[a] - M[c] : M[d]$
  \(PC' = (PC(B_0) == PC') ? (M[a] < 0) ? B_X : B_Y\)

Dominator = basic block that all paths must cross
Program analysis

• Reset noise on PC’ at dominator basic blocks
  – PC’ is the noise limiting variable

Basic block $B_0$: $M[a]$, $M[b]$, $M[c]$

\[
M[d] = PC'
\]

If $PC(B_0) = PC'$:

\[
M[a] = M[a] - M[b] \\
M[d] = M[a] - M[c]
\]

If $M[a] < 0$: $PC' = PC(B_X)$

else: $PC' = PC(B_Y)$

Noise on PC’

\[
M[a] = (PC(B_0) == PC') ? M[a] - M[b] : M[a] \\
M[d] = (PC(B_0) == PC') ? M[a] - M[c] : M[d]
\]

PC’ = (PC(B_0) == PC') ? (M[a] < 0) ? $B_X$ : $B_Y$

Domino block $B_0$: Noise on $PC'$

Dominator = basic block that all paths must cross
Program analysis

- Reset noise on PC’ at dominator basic blocks
  - PC’ is the noise limiting variable

Basic block \( B_0 \):

\[
\begin{align*}
M[a] & \quad M[b] \quad M[c] \\
M[d] & \quad PC’
\end{align*}
\]

\[
\begin{align*}
M[a] &= M[a] - M[b] \\
M[d] &= M[a] - M[c]
\end{align*}
\]

If \( PC(B_0) = PC’ \):

- If \( M[a] < 0 \):
  - \( PC’ = PC(B_X) \)
- Else:
  - \( PC’ = PC(B_Y) \)

\[\text{Noise on PC’}\]

\[\begin{align*}
M[a] &= (PC(B_0) == PC’) \ ? M[a] - M[b] : M[a] \\
M[d] &= (PC(B_0) == PC’) \ ? M[a] - M[c] : M[d]
\end{align*}\]

\[PC’ = (PC(B_0) == PC’) \ ? (M[a] < 0) \ ? B_X : B_Y\]

Dominateor = basic block that all paths must cross
Program analysis

- Reset noise on PC’ at dominator basic blocks
  - PC’ is the noise limiting variable

Basic block $B_0$:

\[
\begin{align*}
M[a] &\quad M[b] &\quad M[c] \\
M[d] &\quad PC'
\end{align*}
\]

If $PC(B_0) == PC'$:

\[
\begin{align*}
M[a] &= M[a] - M[b] \\
M[d] &= M[a] - M[c]
\end{align*}
\]

- If $M[a] < 0$: $PC' = PC(B_X)$
- Else: $PC' = PC(B_Y)$

**Noise on PC’**

$M[a] = (PC(B_0) == PC') \ ? M[a] - M[b] : M[a]$

$M[d] = (PC(B_0) == PC') \ ? M[a] - M[c] : M[d]$

$PC' = (PC(B_0) == PC') \ ? (M[a] < 0) \ ? B_X : B_Y$

- At $L_1$ and $L_4$, the server knows $PC'$, even though it is encrypted.

- Dominator = basic block that all paths must cross

- Reset noise on PC’

- Reset noise on PC’
Data ambiguity

- So far, basic blocks look like:

\[
\begin{align*}
M[a] &= M[b] - M[c] \\
M[d] &= M[b] - M[a] \\
\text{If } M[d] < 0: & \quad PC' = PC(BX) \\
\text{else:} & \quad PC' = PC(BY)
\end{align*}
\]
Data ambiguity

- So far, basic blocks look like:

- What about indirect memory addressing?
  
  *E.g., “load word from memory”*

  \[ M[a] = M[M[b]], \quad PC' = PC' + 1 \]
Data ambiguity

- So far, basic blocks look like:

- What about indirect memory addressing?

  *E.g., “load word from memory”*

  \[ M[a] = M[M[b]], \quad PC' = PC' + 1 \]

- **Thought Q:** How many MIPS instructions to code this loop?

\[
\text{M.length} = 10^9 \text{ bytes} \\
\text{for} \ (i = 0, \ i < 100, \ i++) \ { \\
\quad \text{M[M[x] + i]}--
\]

\[
M[a] = M[b] - M[c] \\
M[d] = M[b] - M[a] \\
\text{If } M[d] < 0: \ PC' = PC(BX) \\
\text{else: } \ PC' = PC(BY)
\]
Data ambiguity

• So far, basic blocks look like:

• What about indirect memory addressing?

   \[ M[a] = M[M[b]], \quad PC' = PC' + 1 \]

• Thought Q: How many MIPS instructions to code this loop?

\[
M\.length = 10^9 \text{ bytes}
\]

\[
\text{for } (i = 0, \ i < 100, \ i++) \ { \ \\
\quad M[M[x] + i]--
\}
\]

\[
\text{Psuedo-MIPS}
\]

\[
\begin{align*}
R[5] &= 0 \\
\text{branch to END if } R[5] == 100 \\
R[7] &= M[R[6]] \\
M[R[6]] &= R[7] \\
\end{align*}
\]
Data ambiguity

- So far, basic blocks look like:

- What about indirect memory addressing?
  
  \[ M[a] = M[M[b]], \quad PC' = PC' + 1 \]

  \[ E.g., \text{“load word from memory”} \]

- Thought Q: How many MIPS instructions to code this loop?

  \[ M.length = 10^9 \text{ bytes} \]
  
  \[ \text{for } (i = 0, i < 100, i++) \{ \]
  
  \[ M[M[x] + i]-- \]
  
  \[ \} \]

- Arithmetic/branch instructions?

  \[ \text{Psuedo-MIPS} \]
  
  \[ R[5] = 0 \]
  
  \[ \text{branch to END if } R[5] == 100 \]
  
  
  \[ R[7] = M[R[6]] \]
  
  
  \[ M[R[6]] = R[7] \]
  
  
  \[ \text{jump} \]
Data ambiguity

- So far, basic blocks look like:

- What about indirect memory addressing?
  
  *E.g., “load word from memory”*
  
  \[
  M[a] = M[M[b]], \quad PC' = PC' + 1
  \]

- **Thought Q:** How many MIPS instructions to code this loop?

  \[
  M.length = 10^9 \text{ bytes}
  \]

  for (i = 0, i < 100, i++) {
    M[M[x] + i]--
  }

- Arithmetic/branch instructions?

  This would take \(O(10^9)\) unique arithmetic + branch instructions

\[
\begin{align*}
M[a] &= M[b] - M[c] \\
M[d] &= M[b] - M[a]
\end{align*}
\]

If \(M[d] < 0\):

\[
PC' = PC(BX)
\]

else:

\[
PC' = PC(BY)
\]

\[
\begin{align*}
R[5] &= 0 \\
\text{branch to END if } R[5] &= 100 \\
R[7] &= M[R[6]] \\
M[R[6]] &= R[7] \\
\text{jump}
\end{align*}
\]
Data ambiguity: a lower bound?

- **Program ambiguity** = $O(|M|)$ overhead
- ... the cost of expanding the entire program into a single FHE circuit in the worst case

- Data ambiguity is why data-dependent recursion is hard
  - Where is the stack pointer in the recursive call stack?
  - Levelization, dominators, etc. still apply
Data ambiguity: a lower bound?

- Program ambiguity = $O(|M|)$ overhead
- … the cost of expanding the entire program into a single FHE circuit in the worst case

- Data ambiguity is why data-dependent recursion is hard
  - Where is the stack pointer in the recursive call stack?
  - Levelization, dominators, etc. still apply

- Can we develop an impossibility result for “beating” data ambiguity that is based on crypto alone?
  - Want to show: “Any scheme that does better than a program’s data ambiguity leaks security”
  - Would give us a better sense of good/bad programs for FHE

- Program ambiguity is a hurdle given an efficient FHE scheme
Part 2: Ascend Secure Processor

(Architecture for Secure Computation on Encrypted Data)
Motivation

• FHE has large overheads from crypto and “program ambiguity”
Motivation

- FHE has large overheads from crypto and “program ambiguity”

- Big idea:
  
  Instruction obfuscation in hardware
Motivation

• FHE has large overheads from crypto and “program ambiguity”

• Big idea:
  
  Instruction obfuscation in hardware

• FHE evaluates *passive circuits* *(cannot make requests)*
  
  – $E_k(\text{Inputs}) \rightarrow E_k(\text{Outputs})$
  
  – PC’ is never decrypted $\rightarrow$ server cannot determine next instruction

• Ascend is a secure processor that makes *obfuscated* requests
  
  – *Ascend knows PC’ and makes obfuscated instruction/memory requests to hide what instruction is actually being evaluated*
  
  – *Internal work is obfuscated*
  
  – No more program ambiguity
• **What is a secure processor?**
  – Tamper-resistant chip: server cannot look inside
  – Server can monitor chip pins (I/O, power)
  – Server knows chip architecture
• What is a secure processor?
  – Tamper-resistant chip: server cannot look inside
  – Server can monitor chip pins (I/O, power)
  – Server knows chip architecture
Security model

• What is a secure processor?
  – Tamper-resistant chip: server cannot look inside
  – Server can monitor chip pins (I/O, power)
  – Server knows chip architecture

• Honest but curious server
  – Same as with FHE
  – Programs are untrusted! Server cannot be able learn about private data by running different programs
The problem with untrusted programs

- **Problem:** Data-dependent behavior allows server to learn by watching chip pins

![Diagram showing the problem with untrusted programs]

- $E_k(\text{Inputs})$, Curious()
- $E_k(\text{results})$
- (1) I/O pin traffic
- (2) power draw
- (3) How long until the program completes?
The problem with untrusted programs

- **Problem:** Data-dependent behavior allows server to learn by watching chip pins

```c
Curious():
while (Memory[0] & 0x1) {
    send requests to memory
}
return;
```

User's private memory

- $E_k(\text{Inputs})$, Curious()
- $E_k(\text{results})$
- (1) I/O pin traffic
- (2) Power draw
- (3) How long until the program completes?
The problem with untrusted programs

- **Problem:** Data-dependent behavior allows server to learn by watching chip pins

```
Curious():
while (Memory[0] & 0x1) {
    send requests to memory
}
return;
```

- **Tamper-resistant Processor**
  - $E_k(\text{Inputs}), \text{Curious}()$

- **Untrusted Server**
  - External RAM

- **User’s private memory**

- **Program enters loop:** causes pin traffic

- **(1) I/O pin traffic**
- **(2) Power draw**
- **(3) How long until the program completes?**
The problem with untrusted programs

- **Problem**: Data-dependent behavior allows server to learn by watching chip pins

```
Curious():
  while (Memory[0] & 0x1) {
    send requests to memory
  }
return;
```

Program enters loop: causes pin traffic

Program does not enter loop: \( E_k(\text{result}) \) appears –or– chip powers down
The Ascend secure processor

curious(M), secure processor

Tamper-resistant processor

Processor pins

Outside world

Off-chip request rate

0 time

curious(M) terminates

Power draw

0 time

curious(M) terminates
The Ascend secure processor

- Ascend’s security level
  - Given program P, time budget T, and any two inputs M and M’:
    
    **Make running** \( P(M) \) **for time** \( T \) **indistinguishable from running** \( P(M’) \) **for** \( T \) **from the perspective of the chip pins**
Ascend’s security level

- Given program P, time budget T, and any two inputs M and M’:

Make running $P(M)$ for time $T$ indistinguishable from running $P(M')$ for $T$ from the perspective of the chip pins
Strawman Ascend chip
Strawman Ascend chip

On each instruction fetch:
1.) Scan ENTIRE instruction memory to perform read
2.) Activate all circuits on chip
3.) Scan+re-encrypt ENTIRE data memory to perform read/write

Repeat this T times. (T is known to the server)
Strawman Ascend chip

On each instruction fetch:
1.) Scan ENTIRE instruction memory to perform read
2.) Activate all circuits on chip
3.) Scan+re-encrypt ENTIRE data memory to perform read/write

Repeat this T times.
(T is known to the server)

All circuits in Ascend must have data-independent power draws
Strawman Ascend chip

On each instruction fetch:

1.) Scan **ENTIRE** instruction memory to perform read
2.) Activate all circuits on chip
3.) Scan+re-encrypt **ENTIRE** data memory to perform read/write

Repeat this T times.
(T is known to the server)

All circuits in Ascend must have data-independent power draws

**Key idea to get security: instruction obfuscation in hardware**

To evaluate each instruction → perform work of all instructions in Ascend’s ISA
Power resistant gates, FFs, SRAM

- “All circuits in Ascend must have data-independent power draws”
Power resistant gates, FFs, SRAM

- “All circuits in Ascend must have data-independent power draws”

- **Normal circuits:** different bit inputs → different circuit transition behavior → **different power draws**

- Prior work in dual-rail/bit-masking circuits *de-correlate* circuit power draw with circuit inputs [TIRI02, TIRI04, SUZUKI04, MANGARD05, CHEN06, KONUR06]
Power resistant gates, FFs, SRAM

• “All circuits in Ascend must have data-independent power draws”

• **Normal circuits:** different bit inputs → different circuit transition behavior → **different power draws**

• Prior work in dual-rail/bit-masking circuits *de-correlate* circuit power draw with circuit inputs [TIRI02, TIRI04, SUZUKI04, MANGARD05, CHEN06, KONUR06]

• Primitives:
  1. Gates & flip-flops (→ instruction pipeline, register files)
     Looks like circuit is accessed every cycle with same inputs
Power resistant gates, FFs, SRAM

• “All circuits in Ascend must have data-independent power draws”

• **Normal circuits:** different bit inputs $\rightarrow$ different circuit transition behavior $\rightarrow$ **different power draws**

• Prior work in dual-rail/bit-masking circuits *de-correlate* circuit power draw with circuit inputs [TIRI02, TIRI04, SUZUKI04, MANGARD05, CHEN06, KONUR06]

• **Primitives:**

  1. **Gates & flip-flops** ($\rightarrow$ instruction pipeline, register files)
     Looks like circuit is accessed every cycle with same inputs

  2. **SRAMs** ($\rightarrow$ on-chip caches)
     Gives off same signature given any address + data
     Does leak *when* SRAM is accessed and whether op is read/write
2-interactive protocol revisited

1. Key sharing (several interactions)
2. $E_k(x)$, Time budget $T$ (1 interaction)
3. Initialize secure processor (allocate external RAM)
4. Perform computation $O(T)$ interactions
5. $E_k(P(x, y))$ or $E_k$(__intermediate result__) (1 interaction)
2-interactive protocol revisited

- Trusted computing base is the Ascend chip (no software)
  - Not P, nor the server OS, nor any communication channel (arrow)
  - Same leakage as before: $|E_k(x)|$ and “Time budget $T$”
2-interactive protocol revisited

- Trusted computing base is the Ascend chip (no software)
  - Not P, nor the server OS, nor any communication channel (arrow)
  - Same leakage as before: $|E_k(x)|$ and “Time budget $T$”

- During step 4: Curious() gives off the same signature regardless of whether it entered the loop
Strawman scheme revisited

Incredibly inefficient.

(Dummy work) / (Real work) = astronomical

1. Read/write all of memory (as bad as FHE’s data ambiguity)
2. Perform worst-case work for every instruction
Oblivious RAM (ORAM)

• The cost of scanning memory is expensive
  – Every access looks exactly the same to the server
  – Cost = linear with size of memory

• Probabilistic Oblivious RAM (ORAM)
  – Performs the same function as the scan, but is more efficient
  – Key idea:
    * Make the memory access pattern for \( P(M) \) indistinguishable from \( P(M') \)
    * Hides data, address, read/write for each access
  – Cost = logarithmic with size of memory

*ORAM was proposed by Goldreich et al.; the Path-ORAM, which we will use, was developed by Stefanov et al.
Ascend + ORAM

- Adding support for “Path-ORAM” in Ascend
Ascend + ORAM

- Adding support for “Path-ORAM” in Ascend

![Diagram of Ascend Chip and ORAM interface with trusted and untrusted components.]

- Trusted
- Untrusted (under server’s control)
Ascend + ORAM

- Adding support for “Path-ORAM” in Ascend

- Each (leaf L, addr, data) 3-tuple is encrypted; lives on path from root to X
• Adding support for “Path-ORAM” in Ascend

• Each (leaf L, addr, data) 3-tuple is *encrypted*; *lives on path from root to X*

• To access a block with program address A
  1. Lookup position map determine that A → leaf X
  2. Read path to leaf X into the ORAM interface’s local $
  3. Write back as many blocks in the local $ mapped to the path with leaf X as possible
Reducing on-chip ORAM storage

- **Problem:** 416 MB position map is too big to store in Ascend
- **Solution:** store the position map in another ORAM
Reducing on-chip ORAM storage

- **Problem:** 416 MB position map is too big to store in Ascend
- **Solution:** store the position map in another ORAM

To access addr A:
1. lookup position map(A) yields leaf $L_3$
2. read/write ORAM$_3$(L$_3$) yields leaf $L_2$
3. read/write ORAM$_2$(L$_2$) yields leaf $L_1$
4. read/write ORAM$_1$(L$_1$) yields (A, data)
Reducing on-chip ORAM storage

- **Problem:** 416 MB position map is too big to store in Ascend
- **Solution:** store the position map in another ORAM

- **At the end of the day…**

  Dedicated hardware storage: **209.4 KB**
  Data moved per access: ~86 KB → **5880 cycles / op (54X overhead)**

---

**To access addr A:**

1. lookup position map(A) yields leaf \( \text{L}_3 \)
2. read/write ORAM\(_3\)(\( \text{L}_3 \)) yields leaf \( \text{L}_2 \)
3. read/write ORAM\(_2\)(\( \text{L}_2 \)) yields leaf \( \text{L}_1 \)
4. read/write ORAM\(_1\)(\( \text{L}_1 \)) yields (A, data)
Reducing on-chip ORAM storage

• **Problem:** 416 MB position map is too big to store in Ascend
• **Solution:** store the position map in another ORAM

At the end of the day…

Dedicated hardware storage: 209.4 KB
Data moved per access: ~86 K B → 5880 cycles / op (54X overhead)

To access addr A:
1.) lookup position map(A) yields leaf L₃
2.) read/write ORAM₃(L₃) yields leaf L₂
3.) read/write ORAM₂(L₂) yields leaf L₁
4.) read/write ORAM₁(L₁) yields (A, data)

Original position map

175 KB
4054 cycles
Making Ascend more efficient

- Where we are now:

On each instruction fetch:

1. Read instruction ORAM
2. Activate all circuits on chip
3. Read/write data ORAM
Making Ascend more efficient

- Where we are now:

On each instruction fetch:

1.) Read instruction ORAM
2.) Activate all circuits on chip
3.) Read/write data ORAM

Problem: Performing worst-case work for every instruction is still very expensive

\[ \sim 4,054 \times 2 = \sim 8,108 \text{ cycles per instruction} \]
Making Ascend more efficient

- Where we are now:

On each instruction fetch:

1.) Read instruction ORAM
2.) Activate all circuits on chip
3.) Read/write data ORAM

Problem: Performing worst-case work for every instruction is still very expensive

~ 4,054 x 2 = ~ 8,108 cycles per instruction

Security level is independent of the rates at which Ascend activates internal circuits & accesses memory... as long as these rates are specified by the untrusted server
Amortizing expensive operations

• (Obvious) Idea: perform expensive ops less frequently
• Scheme: let the server specify frequencies for different ops
Amortizing expensive operations

- **(Obvious) Idea:** perform expensive ops less frequently
- **Scheme:** let the server specify frequencies for different ops
- **Example:** ~25% of instructions are memory instructions
Amortizing expensive operations

- **(Obvious) Idea:** perform expensive ops less frequently
- **Scheme:** let the server specify frequencies for different ops
- **Example:** ~25% of instructions are memory instructions
Amortizing expensive operations

- **(Obvious) Idea:** perform expensive ops less frequently
- **Scheme:** let the server specify frequencies for different ops
- **Example:** ~25% of instructions are memory instructions

---

**Diagram:**

- **Ascend Chip**
  - Register File
  - Decode
  - Exe 1, Exe 2, ..., Exe N
  - Load / Store
  - mux

- **Access 1 / instr**

- **ORAM Interface**

- **ORAM**
  - ORAM 1 (16 GB)
  - ORAM 2 (416 MB)
  - ORAM 3 (8.5 MB)

- **Memory**

- **Access 1 / 4 instrs**
Amortizing expensive operations

• **(Obvious) Idea:** perform expensive ops less frequently
• **Scheme:** let the server specify frequencies for different ops
• **Example:** ~25% of instructions are memory instructions

~ 5,067 cycles per instruction
Amortizing expensive operations

- **(Obvious) Idea:** perform expensive ops less frequently
- **Scheme:** let the server specify frequencies for different ops
- **Example:** ~25% of instructions are memory instructions

- Server has to guess frequencies
- **Question:** What happens when the server miss guesses?

→ Rest of the circuits in chip still activate at same frequency

**Access 1 / instr**

~ 5,067 cycles per instruction
Exploiting common case behavior

Ascend Chip

- Register File
- Decode
- PC
- Exe 1
- Exe 2
- Exe N
- Load / Store
- L1 ICache
- L1 DCache
- L2 Cache
- ORAM Interface
- ORAM 1 (16 GB)
- ORAM 2 (416 MB)
- ORAM 3 (8.5 MB)

Fire all execution units!
Exploiting common case behavior

Access 1 / instr

Ascend Chip

Memory

ORAM Interface

ORAM 1
(16 GB)

ORAM 2
(416 MB)

ORAM 3
(8.5 MB)
Exploiting common case behavior

• Say “average application” has:
  – 25% memory instructions
Exploiting common case behavior

• Say “average application” has:
  – 25% memory instructions
  – 1% L1 ICache miss rate, 10% L1 DCache miss rate
Exploiting common case behavior

- Say “average application” has:
  - 25% memory instructions
  - 1% L1 ICache miss rate, 10% L1 DCache miss rate
  - 5% L2 Cache miss rate
Power analysis: cache coherence

• Each cache read/write may hit, (miss and/or evict) blocks
• To hide which happens, Ascend performs union of all work
Power analysis: cache coherence

- Each cache read/write may hit, (miss and/or evict) blocks
- To hide which happens, Ascend performs union of all work
- Step 1: all accesses become read+writes
Power analysis: cache coherence

- Each cache read/write may hit, (miss and/or evict) blocks
- To hide which happens, Ascend performs union of all work
- Step 1: all accesses become read+writes
- Step 2: all accesses perform the union of all evictions/refills
Power analysis: cache coherence

- Each cache read/write may hit, (miss and/or evict) blocks
- To hide which happens, Ascend performs union of all work
- Step 1: all accesses become read+writes
- Step 2: all accesses perform the union of all evictions/refills
- Example (for an inclusive cache):

```
Instruction pipeline

L1 Instr cache          L1 Data cache

Shared L2 cache

ORAM Interface
```
Power analysis: cache coherence

- Each cache read/write may hit, (miss and/or evict) blocks
- To hide which happens, Ascend performs union of all work
- Step 1: all accesses become read+writes
- Step 2: all accesses perform the union of all evictions/refills
- Example (for an inclusive cache):

```
Instruction pipeline

L1 Instr cache ——— Access ——— L1 Data cache ——— Response ——— Eviction

Shared L2 cache

ORAM Interface
```
Power analysis: cache coherence

- Each cache read/write may hit, (miss and/or evict) blocks
- To hide which happens, Ascend performs union of all work
- Step 1: all accesses become read+writes
- Step 2: all accesses perform the union of all evictions/refills
- Example (for an inclusive cache):

![Diagram of cache hierarchy]

- Instruction pipeline
- L1 Instr cache
- L1 Data cache
- Shared L2 cache
- ORAM Interface
Power analysis: cache coherence

- Each cache read/write may hit, (miss and/or evict) blocks
- To hide which happens, Ascend performs union of all work
- Step 1: all accesses become read+writes
- Step 2: all accesses perform the union of all evictions/refills
- Example (for an inclusive cache):

```
Instruction pipeline

L1 Instr cache <--- Back invalidations ---^---
|                      |
| (arrows for clarity) |
|                      |
|                      |

L1 Data cache

Shared L2 cache

ORAM Interface
```

Eviction
Access
Response
Power analysis: cache coherence

- Each cache read/write may hit, (miss and/or evict) blocks
- To hide which happens, Ascend performs union of all work
- Step 1: all accesses become read+writes
- Step 2: all accesses perform the union of all evictions/refills
- Example (for an inclusive cache):

  ![Diagram of cache hierarchy]

  - Instruction pipeline
  - L1 Instr cache
  - L1 Data cache
  - Shared L2 cache
  - ORAM Interface
Power analysis: cache coherence

- Each cache *read/write* may hit, *(miss and/or evict)* blocks
- To hide which happens, Ascend performs union of all work
- **Step 1:** all accesses become read+writes
- **Step 2:** all accesses perform the union of all evictions/refills
- Example (for an inclusive cache):

![Diagram](image-url)

- Instruction pipeline
- L1 Instr cache
- L1 Data cache
- Shared L2 cache
- ORAM Interface
  - Access
  - Refill
Security analysis

curious(M) and curious(M') on Ascend

- Off-chip request rate
  - time = T

- Off-chip request address
  - time = T

- Power draw
  - time = T

Ascend

Processor pins

Outside world
Security analysis

- I/O pins
  - *How* the pins change is obfuscated by ORAM
  - *When* the pins change is set by server-specified intervals
Security analysis

• I/O pins
  – *How* the pins change is obfuscated by ORAM
  – *When* the pins change is set by server-specified intervals

• Power pins
  – Power draw of each circuit is obfuscated by PA-resistant logic
  – When each circuit is activated is set by server-specified intervals
  – Union of complex behavior (e.g., cache coherence) performed
Power/performance results

- Slowdown relative to running programs “in the clear”
  (in the clear = no encryption/access frequencies/ORAM)
- **uArch:** I/D L1 caches, shared/unified L2, simple pipeline

**Ascend:** ORAM@intervals, caches@intervals, power analysis-resistant logic

**Ascend power overhead:** ~6x for each benchmark
Multiple interactive protocols

- Generalization of the 2-interactive protocol
- Motivation:
  - 2-interactive protocol may return an intermediate result
Multiple interactive protocols

- Generalization of the 2-interactive protocol
- Motivation:
  - 2-interactive protocol may return an intermediate result

\[
\text{Repeat } i \text{ times, where the user controls } i, x_i, T_i
\]

1. \(E_k(x), \text{Time budget } T\)
2. Perform computation for \(O(T)\) time
3. \(E_k(P(x, y)) \text{ or } E_k(\text{intermediate result})\)

Untrusted Server

User

Program \(P\),
Public inputs \(y\),
unlimited resources,
unlimited compute

Program \(P\),
Public inputs \(y\),
unlimited resources,
unlimited compute

\(E_k(P(x_i, y)) \text{ or } E_k(\text{intermediate result})\)
Multiple interactive protocols

- Generalization of the 2-interactive protocol
- Motivation:
  - 2-interactive protocol may return an intermediate result
- Cons: additional information leakage ($i$, each $|E_k(x_i)|$, each $T_i$)

```
Repeat $i$ times, where the user controls $i, x_i, T_i$
```

Untrusted Server

1. $E_k(x), \text{ Time budget } T$
2. Perform computation for $O(T)$ time
3. $E_k(P(x, y))$ or $E_k(\text{intermediate result})$

User

Program $P$, Public inputs $y$, unlimited resources, unlimited compute

Untrusted Server

1. $E_k(x_i), \text{ Time budget } T_i$
2. Perform computation for $O(T_i)$ time
3. $E_k(P(x_i, y))$ or $E_k(\text{intermediate result})$

User

Program $P$, Public inputs $y$, unlimited resources, unlimited compute

Untrusted Server

1. $E_k(x_i), \text{ Time budget } T_i$
2. Perform computation for $O(T_i)$ time
3. $E_k(P(x_i, y))$ or $E_k(\text{intermediate result})$

User

Program $P$, Public inputs $y$, unlimited resources, unlimited compute
Multiple interactions + malicious servers

- **Problem:** If the server is malicious, it may cheat by:
  - Running a different $P$ on the user’s $x$
  - Running for less than $T$ time
  - Tampering with bits in ORAM

Repeat $i$ times, where the user controls $i, x_i, T_i$
Multiple interactions + malicious servers

- **Problem:** If the server is malicious, it may cheat by:
  - Running a different P on the user’s x
  - Running for less than T time
  - Tampering with bits in ORAM

- **Simple attack:**
  1. Server runs `Curious()` on user’s input
  2. Always returns to the user the result after T = 1 time has passed

```
Curious():
    while (M[0] & 0x1) {
        send requests to memory
    }
    return;
```

Untrusted Server

User

1. `Ek(x_i), Time budget T_i`

2. Perform computation for \(O(T_i)\) time

3. `Ek(P(x_i, y))` or `Ek(intermediate result)`
Multiple interactions + malicious servers

**Problem:** If the server is malicious, it may cheat by:
- Running a different $P$ on the user’s $x$
- Running for less than $T$ time
- Tampering with bits in ORAM

**Simple attack:**
1. Server runs `Curious()` on user’s input
2. Always returns to the user the result after $T = 1$ time has passed

**User stops interacting after several interactions, server learns:**
\[ M[0] \& 0x1 == 0 \]

MIT COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE LABORATORY

```cpp
Curious():
while (M[0] & 0x1) {
    send requests to memory
} return;
```
Certified execution

- **Goal:** allow the user to control leakage
- **Mechanism:** certified execution
Certified execution

- **Goal**: allow the user to control leakage
- **Mechanism**: certified execution
- The user gets a certificate that the server:
  - Runs the correct P
  - For the specified T amount of time
  - Does not tamper with any data stored in the external ORAM
Certified execution

- **Goal**: allow the user to control leakage
- **Mechanism**: certified execution
- The user gets a certificate that the server:
  - Runs the correct P
  - For the specified T amount of time
  - Does not tamper with any data stored in the external ORAM

- **What does the user need from Ascend?**
  1. Hash(P) || x || y
  2. Integrity verification on Ascend’s ORAM
Certified execution

- **Goal:** allow the user to control leakage
- **Mechanism:** certified execution
- The user gets a certificate that the server:
  - Runs the correct $P$
  - For the specified $T$ amount of time
  - Does not tamper with any data stored in the external ORAM

- **What does the user need from Ascend?**
  1. $\text{Hash}(P) \ || \ x \ || \ y$
  2. Integrity verification on Ascend’s ORAM

- #1: user needs either $P$ or $\text{Hash}(P)$ from a trusted third party
Certified execution

- **Goal**: allow the user to control leakage
- **Mechanism**: certified execution
- The user gets a certificate that the server:
  - Runs the correct $P$
  - For the specified $T$ amount of time
  - Does not tamper with any data stored in the external ORAM
- **What does the user need from Ascend?**
  1. $\text{Hash}(P) \ || \ x \ || \ y$
  2. Integrity verification on Ascend’s ORAM
- **#1**: user needs either $P$ or $\text{Hash}(P)$ from a trusted third party
- **#2**: Integrity verification: each block read from ORAM must be
  - **Authentic**: produced by Ascend
  - **Fresh**: correspond to the most recent version of the data
Integrity verification over ORAM

- **Goal:** implement integrity verification with *negligible* overhead

Each node is a 640 Byte/4 block bucket

$h_0 = h(B_0 || h_1 || h_2)$

Root hash
Integrity verification over ORAM

- **Goal:** implement integrity verification with *negligible* overhead
- **Scheme:** mirror Path-ORAM with an authentication tree

```
Level 0
h0 = \( H(B_0 \ || \ h_1 \ || \ h_2) \)

Level 1
h1 = \( H(B_1 \ || \ h_3 \ || \ h_4) \)

h2 = \( H(B_2 \ || \ h_5 \ || \ h_6) \)

Level 2
h3 = \( H(B_3) \)

h4 = \( H(B_4) \)

h5 = \( H(B_5) \)

h6 = \( H(B_6) \)
```

Each node is a 640 Byte/4 block bucket.

(h0 is stored internal to Ascend)
Integrity verification over ORAM

- **Goal:** implement integrity verification with *negligible* overhead
- **Scheme:** mirror Path-ORAM with an authentication tree

- **Example:**

  Access block in $B_2$

```
Level 0
  16 Bytes per hash
  (h₀ is stored internal to Ascend)

Level 1
  h₁ = H(B₁ || h₃ || h₄)
  h₂ = H(B₂ || h₅ || h₆)

Level 2
  h₃ = H(B₃)
  h₄ = H(B₄)
  h₅ = H(B₅)
  h₆ = H(B₆)
```

Each node is a 640 Byte/4 block bucket

ORAM Interface

- Ascend’s pins
- ORAM Local $\$
- ORAM Local $\$
- Reads
- Writes

```
h₀ = H(B₀ || h₁ || h₂)
Root hash
```

MIT COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE LABORATORY
Integrity verification over ORAM

• **Goal:** implement integrity verification with *negligible* overhead
• **Scheme:** mirror Path-ORAM with an authentication tree

**Example:**

Access block in $B_2$

1. Read ORAM path for leaf containing bucket $B_2$
Integrity verification over ORAM

- **Goal:** implement integrity verification with *negligible* overhead
- **Scheme:** mirror Path-ORAM with an authentication tree

- **Example:**
  Access block in $B_2$
  1. Read ORAM path for leaf containing bucket $B_2$
  2. Read sibling hashes from authentication tree
Integrity verification over ORAM

- **Goal:** implement integrity verification with *negligible* overhead
- **Scheme:** mirror Path-ORAM with an authentication tree

- **Example:**

  Access block in B$_2$
  1. Read ORAM path for leaf containing bucket B$_2$
  2. Read sibling hashes from authentication tree
  3. Compare hashes with root hash
  4. If the hashes matched, data is authentic/fresh!

1. ORAM Interface
2. Path ORAM
3. Authentication Tree
4. 16 Bytes per hash
5. (h$_0$ is stored internal to Ascend)
6. Each node is a 640 Byte/4 block bucket
7. 16 Bytes per hash
8. h$_0 = H$(B$_0$ || h$_1$ || h$_2$)
Integrity verification over ORAM

- **Goal:** implement integrity verification with *negligible* overhead
- **Scheme:** mirror Path-ORAM with an authentication tree

**Example:**

Access block in $B_2$

1. Read ORAM path for leaf containing bucket $B_2$
2. Read sibling hashes from authentication tree
3. Compare hashes with root hash
4. If the hashes matched, data is authentic/fresh!
5. Write back ORAM path, update hashes
Summary

Instruction obfuscation is viable;
Performance overheads are ~5x while running real applications*

– … without trusting any software
– … without building any part of the OS into hardware

*using ORAM for obfuscation
Ongoing and Future Work

• Reduce ORAM latency: Better ORAM schemes for our setting
  – We want small block sizes (ORAM was built for large blocks)
  – Most of our performance overhead comes from ORAM

• Refine Ascend microarchitecture
  – What does it take to obfuscate advanced pipelines?

• Larger/Interactive/multi-thread applications/virtualization
  – How do you run an OS on Ascend with multiple contexts?
  – … that belong to multiple users?
  – … that involve multiple Ascend chips?
  – Applications with multiple threads and memory sharing?
  – How do you use Ascend to query the internet, access file systems, stream/filter through large datasets?