Synthesis of Adaptive Side-Channel Attacks

Quoc-Sang Phan\textsuperscript{1}, \textbf{Lucas Bang}\textsuperscript{2},
Corina S. Păsăreanu\textsuperscript{1,3}, Pasquale Malacaria\textsuperscript{4}, Tevfik Bultan\textsuperscript{2}

\textsuperscript{1}Carnegie Mellon University
Moffet Field, CA, USA

\textsuperscript{2}University of California, Santa Barbara
Santa Barbara, CA, USA

\textsuperscript{3}NASA Ames Research Center
Moffet Field, CA, USA

\textsuperscript{4}Queen Mary University of London
London E1 4NS, UK

Computer Security Foundations
Santa Barbara, CA, USA
24 August 2017
Overview

Figure: “RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis”
Motivating Example

High security input (secret): h
Low security input (public): l

int compare(h, l)
if (h <= l)
sleep(1);
else
sleep(2);
return 0;

Main channel:
Always 0. No information.

Side channel:
t = 1 ⇒ h ≤ l
t = 2 ⇒ h > l
Motivating Example

High security input (secret): $h$
Low security input (public): $l$

```c
int compare(h, l)
if (h <= l)
sleep(1);
else
sleep(2);
return 0;
```

Main channel: Always 0. No information.
Side channel: $t = 1 \Rightarrow h \leq l$
$t = 2 \Rightarrow h > l$
Motivating Example

High security input (secret): $h$
Low security input (public): $l$

```c
int compare(h, l)
    if(h <= l)
        sleep(1);
    else
        sleep(2);
    return 0;
```

Main channel: Always 0. No information.

Side channel:
\[ t = 1 \Rightarrow h \leq l \]
\[ t = 2 \Rightarrow h > l \]
Motivating Example

High security input (secret): \( h \)
Low security input (public): \( l \)

```c
int compare(h, l)
    if(h <= l)
        sleep(1);
    else
        sleep(2);
return 0;
```
Motivating Example

High security input (secret): $h$
Low security input (public): $l$

```c
int compare(h, l)
    if (h <= l)
        sleep(1);
    else
        sleep(2);
return 0;
```

Main channel:
Always 0. No information.

$t = 1 \Rightarrow h \leq l$
$t = 2 \Rightarrow h > l$
Motivating Example

High security input (secret): $h$
Low security input (public): $l$

```c
int compare(h, l)
    if (h <= l)
        sleep(1);
    else
        sleep(2);
return 0;
```

Main channel:
Always 0. No information.

Side channel:
$t = 1 \Rightarrow h \leq l$
Motivating Example

High security input (secret): $h$
Low security input (public): $l$

```c
int compare(h, l)
    if(h <= l)
        sleep(1);
    else
        sleep(2);
    return 0;
```

Main channel:
Always 0. No information.

Side channel:
$t = 1 \Rightarrow h \leq l$
$t = 2 \Rightarrow h > l$
\[ t = 1 \implies h \leq l \]
\[ t = 2 \implies h > l \]
\[ t = 1 \implies h \leq l \]
\[ t = 2 \implies h > l \]
\[ t = 1 \implies h \leq l \]
\[ t = 2 \implies h > l \]

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
</table>
\[ t = 1 \Rightarrow h \leq l \]
\[ t = 2 \Rightarrow h > l \]
\[ t = 1 \Rightarrow h \leq l \]
\[ t = 2 \Rightarrow h > l \]
\[ t = 1 \implies h \leq l \]
\[ t = 2 \implies h > l \]

\[
\begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\end{array}
\]

Too few divisions.
Unbalanced divisions.
Best tree induces maximum number of divisions and balanced divisions.
\( t = 1 \implies h \leq l \)
\( t = 2 \implies h > l \)
\[ t = 1 \Rightarrow h \leq l \]
\[ t = 2 \Rightarrow h > l \]

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
</table>

\( t = 1 \Rightarrow h \leq 6 \)

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
</table>

1 = 6
$$t = 1 \Rightarrow h \leq l$$
$$t = 2 \Rightarrow h > l$$

太少了，这会导致分枝不平衡。最佳树会产生最多的分枝数目并且是平衡的。
\[ t = 1 \quad \Rightarrow \quad h \leq l \]
\[ t = 2 \quad \Rightarrow \quad h > l \]
$t = 1 \Rightarrow h \leq l$
$t = 2 \Rightarrow h > l$
$t = 1 \Rightarrow h \leq l$
$t = 2 \Rightarrow h > l$

1 2 3 4 5 6 7 8

1 2 3 4 5 6 7 8

1 = 6

$1 = 6$

$t=1 \Rightarrow h \leq 6$
$t=2 \Rightarrow h > 6$
\[ t = 1 \implies h \leq l \]
\[ t = 2 \implies h > l \]
\[ t = 1 \implies h \leq l \]
\[ t = 2 \implies h > l \]
\[ t = 1 \Rightarrow h \leq l \]
\[ t = 2 \Rightarrow h > l \]
\[ t = 1 \Rightarrow h \leq l \]
\[ t = 2 \Rightarrow h > l \]
\[ t = 1 \Rightarrow h \leq l \]
\[ t = 2 \Rightarrow h > l \]
\[ t = 1 \Rightarrow h \leq l \]
\[ t = 2 \Rightarrow h > l \]
\[ t = 1 \Rightarrow h \leq l \]
\[ t = 2 \Rightarrow h > l \]
\[ t = 1 \implies h \leq l \]
\[ t = 2 \implies h > l \]
\[ t = 1 \Rightarrow h \leq l \]
\[ t = 2 \Rightarrow h > l \]

Too few divisions.
Unbalanced divisions.
Best tree induces **maximum # divisions**
\( t = 1 \Rightarrow h \leq l \)
\( t = 2 \Rightarrow h > l \)

Best tree induces \textbf{maximum \# divisions} and \textbf{balanced divisions}.
$t = 1 \Rightarrow h \leq l$
$t = 2 \Rightarrow h > l$

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

Best tree induces **maximum # divisions** and **balanced divisions**.
\[ t = 1 \implies h \leq l \]
\[ t = 2 \implies h > l \]

Best tree induces maximum # divisions and balanced divisions.
Best tree induces **maximum # divisions** and **balanced divisions**.
Best tree induces **maximum # divisions** and **balanced divisions**.
\[t = 1 \Rightarrow h \leq l\]
\[t = 2 \Rightarrow h > l\]

Best tree induces **maximum # divisions** and **balanced divisions**.
Best tree induces maximum # divisions and balanced divisions.
\[
t = 1 \implies h \leq l \\
t = 2 \implies h > l
\]

Best tree induces **maximum # divisions** and **balanced divisions**.
Best tree induces **maximum # divisions** and **balanced divisions**.
\[ t = 1 \implies h \leq l \]
\[ t = 2 \implies h > l \]

Best tree induces maximum # divisions and balanced divisions.
Best tree induces maximum # divisions and balanced divisions.

channel capacity
\[ t = 1 \implies h \leq l \]
\[ t = 2 \implies h > l \]

Best tree induces **maximum # divisions** and **balanced divisions**.

**channel capacity**  
**entropy**
Find the Best Tree...
Find the Best Tree...
Find the Best Attack!
Find the Best Tree...
Find the Best Attack!
How?
Our Approach

1. Symbolic execution of attacker + system model.
2. Generate attack tree, symbolic over $h$ and $\bar{L}$.
3. Optimize over all trees $\equiv$ maximization problem for $\bar{L}$. 
1. Symbolic execution of attacker + system model.
Our Approach

1. Symbolic execution of attacker + system model.
2. Generate attack tree, symbolic over $h$ and $\bar{L}$.
Our Approach

1. Symbolic execution of attacker + system model.
2. Generate attack tree, symbolic over $h$ and $\bar{L}$.
3. Optimize over all trees
Our Approach

1. Symbolic execution of attacker + system model.
2. Generate attack tree, symbolic over $h$ and $\bar{L}$.
3. Optimize over all trees $\equiv$ maximization problem for $\bar{L}$. 
Symbolic Execution

- Static program analysis technique.
- Execute program on symbolic rather than concrete inputs.
- Maintain path conditions, PC's, over symbolic inputs.
- When branch instruction encountered with condition $c$:
  - True branch: $\text{PC} \leftarrow \text{PC} \land c$
  - False branch: $\text{PC} \leftarrow \text{PC} \land \neg c$
- Check feasibility of PC using constraint solvers (Z3).
- Explore only feasible branches.
- During exploration, maintain side channel cost model.
- Results in symbolic tree (attack tree).
Symbolic Execution

- Static program analysis technique.
Symbolic Execution

- Static program analysis technique.
- Execute program on **symbolic** rather than concrete inputs.
Symbolic Execution

- Static program analysis technique.
- Execute program on **symbolic** rather than concrete inputs.
- Maintain **path conditions**, PCs, over symbolic inputs.

When branch instruction encountered with condition $c$:
- **True** branch: $\text{PC} \leftarrow \text{PC} \land c$
- **False** branch: $\text{PC} \leftarrow \text{PC} \land \neg c$

Check feasibility of PC using constraint solvers (Z3).

Explore only feasible branches.

During exploration, maintain side channel cost model.

Results in symbolic tree (attack tree).
Symbolic Execution

- Static program analysis technique.
- Execute program on **symbolic** rather than concrete inputs.
- Maintain **path conditions**, PCs, over symbolic inputs.
- When branch instruction encountered with condition $c$:
Symbolic Execution

- Static program analysis technique.
- Execute program on **symbolic** rather than concrete inputs.
- Maintain **path conditions**, PCs, over symbolic inputs.
- When branch instruction encountered with condition $c$:
  - True branch: $PC \leftarrow PC \land c$
Symbolic Execution

- Static program analysis technique.
- Execute program on **symbolic** rather than concrete inputs.
- Maintain **path conditions**, $PC$s, over symbolic inputs.
- When branch instruction encountered with condition $c$:
  - True branch: $PC \leftarrow PC \land c$
  - False branch: $PC \leftarrow PC \land \neg c$
Symbolic Execution

- Static program analysis technique.
- Execute program on **symbolic** rather than concrete inputs.
- Maintain **path conditions**, PCs, over symbolic inputs.
- When branch instruction encountered with condition $c$:
  - True branch: $PC \leftarrow PC \land c$
  - False branch: $PC \leftarrow PC \land \neg c$
- Check feasibility of $PC$ using constraint solvers (Z3).
Symbolic Execution

- Static program analysis technique.
- Execute program on **symbolic** rather than concrete inputs.
- Maintain **path conditions**, *PCs*, over symbolic inputs.
- When branch instruction encountered with condition *c*:
  - True branch: $PC \leftarrow PC \land c$
  - False branch: $PC \leftarrow PC \land \neg c$
- Check feasibility of *PC* using constraint solvers (Z3).
- Explore only feasible branches.
Symbolic Execution

- Static program analysis technique.
- Execute program on **symbolic** rather than concrete inputs.
- Maintain **path conditions**, PCs, over symbolic inputs.
- When branch instruction encountered with condition $c$:
  - True branch: $PC \leftarrow PC \land c$
  - False branch: $PC \leftarrow PC \land \neg c$
- Check feasibility of $PC$ using constraint solvers (Z3).
- Explore only feasible branches.
- During exploration, maintain side channel cost model.
Symbolic Execution

- Static program analysis technique.
- Execute program on **symbolic** rather than concrete inputs.
- Maintain **path conditions**, PCs, over symbolic inputs.
- When branch instruction encountered with condition $c$:
  - True branch: $PC \leftarrow PC \land c$
  - False branch: $PC \leftarrow PC \land \neg c$
- Check feasibility of $PC$ using constraint solvers (Z3).
- Explore only feasible branches.
- During exploration, maintain side channel cost model.
- Results in symbolic tree
Symbolic Execution

- Static program analysis technique.
- Execute program on **symbolic** rather than concrete inputs.
- Maintain **path conditions**, PCs, over symbolic inputs.
- When branch instruction encountered with condition $c$:
  - True branch: $PC \leftarrow PC \land c$
  - False branch: $PC \leftarrow PC \land \neg c$
- Check feasibility of $PC$ using constraint solvers (Z3).
- Explore only feasible branches.
- During exploration, maintain side channel cost model.
- Results in symbolic tree (attack tree).
Symbolic attack tree:

and all symbolic constraints between h and l

\[ L = l \]

\[ h \geq l \]

\[ L = l \]

\[ h \geq l \]

\[ L = l \]

\[ h \geq l \]

\[ L = l \]

\[ h \geq l \]

\[ L = l \]

\[ h \geq l \]

\[ L = l \]

\[ h \geq l \]

Each leaf: symbolic constraint on h given by \( \bar{L} \)

Find optimal \( \bar{L} = \langle l, l_1, l_2, l_{11}, l_{12}, l_{21}, l_{22} \rangle = \langle 4, 6, 2, 7, 5, 3, 1 \rangle \)
Symbolic attack tree:

$h$ and all $l$-choices symbolic constraints between $h$ and $l$ symbolic
Symbolic attack tree:

- $h$ and all $l$-choices symbolic
- constraints between $h$ and $l$ symbolic

Each leaf: symbolic constraint on $h$ given by

$\bar{L} = \langle l, l_1, l_2, l_{11}, l_{12}, l_{21}, l_{22} \rangle$

Find optimal $\bar{L} = \langle 4, 6, 2, 7, 5, 3, 1 \rangle$
Symbolic attack tree:

$h$ and all $l$-choices symbolic constraints between $h$ and $l$ symbolic

Each leaf: symbolic constraint on $h$ given by $\bar{L}$
Symbolic attack tree:

$h$ and all $l$-choices symbolic constraints between $h$ and $l$ symbolic

Each leaf: symbolic constraint on $h$ given by $\bar{L}$

Find optimal $\bar{L} = \langle l, l_1, l_2, l_{11}, l_{12}, l_{21}, l_{22} \rangle$
Symbolic attack tree:

$h$ and all $l$-choices symbolic constraints between $h$ and $l$ symbolic

Each leaf: symbolic constraint on $h$ given by $\bar{L}$

Find optimal $\bar{L} = \langle l, l_1, l_2, l_{11}, l_{12}, l_{21}, l_{22} \rangle = \langle 4, 6, 2, 7, 5, 3, 1 \rangle$
Finding Best Attack Tree
Method 1
Maximizing Number of Partition Divisions

foo(int l, int h)
    if (l<0)
        if (h<0) sleep(1)
        else if (h<5) sleep(2)
        else sleep(3)
    else
        if (h>1) sleep(4)
        else sleep(5)
Max-SMT: Maximum Satisfiability Modulo Theories

Find an assignment for $l$ and $h_i$ that maximizes the number of satisfiable constraints.

Optimal choice $l = -1$.

Max-SMT assignment $\equiv$ maximizing channel capacity.

MAX-SMT Problem: Find an assignment of values to variables that maximizes the number of simultaneously satisfied clauses.
Max-SMT: Maximum Satisfiability Modulo Theories

\[ C_1: \ l < 0 \land h_1 < 0 \]
\[ C_2: \ l < 0 \land h_2 \geq 0 \land h_2 < 5 \]
\[ C_3: \ l < 0 \land h_3 \geq 5 \]
\[ C_4: \ l \geq 0 \land h_4 > 1 \]
\[ C_5: \ l \geq 0 \land h_5 \leq 1 \]
Max-SMT: Maximum Satisfiability Modulo Theories

\[ C_1: \ l < 0 \land h_1 < 0 \]
\[ C_2: \ l < 0 \land h_2 \geq 0 \land h_2 < 5 \]
\[ C_3: \ l < 0 \land h_3 \geq 5 \]
\[ C_4: \ l \geq 0 \land h_4 > 1 \]
\[ C_5: \ l \geq 0 \land h_5 \leq 1 \]

- Find an assignment for \( l \) and \( h_i \) that maximizes the number of satisfiable constraints.
Max-SMT: Maximum Satisfiability Modulo Theories

\[ C_1: \quad l < 0 \land h_1 < 0 \]
\[ C_2: \quad l < 0 \land h_2 \geq 0 \land h_2 < 5 \]
\[ C_3: \quad l < 0 \land h_3 \geq 5 \]
\[ C_4: \quad l \geq 0 \land h_4 > 1 \]
\[ C_5: \quad l \geq 0 \land h_5 \leq 1 \]

- Find an assignment for \( l \) and \( h_i \) that maximizes the number of satisfiable constraints.
Max-SMT: Maximum Satisfiability Modulo Theories

\[C_1: \ l < 0 \land h_1 < 0\]
\[C_2: \ l < 0 \land h_2 \geq 0 \land h_2 < 5\]
\[C_3: \ l < 0 \land h_3 \geq 5\]
\[C_4: \ l \geq 0 \land h_4 > 1\]
\[C_5: \ l \geq 0 \land h_5 \leq 1\]

- Find an assignment for \(l\) and \(h_i\) that maximizes the number of satisfiable constraints.

Optimal choice \(l = -1\).

Max-SMT assignment \(\equiv\) maximizing channel capacity.
Max-SMT: Maximum Satisfiablity Modulo Theories

\[ C_1: \quad l < 0 \land h_1 < 0 \]
\[ C_2: \quad l < 0 \land h_2 \geq 0 \land h_2 < 5 \]
\[ C_3: \quad l < 0 \land h_3 \geq 5 \]
\[ C_4: \quad l \geq 0 \land h_4 > 1 \]
\[ C_5: \quad l \geq 0 \land h_5 \leq 1 \]

- Find an assignment for \( l \) and \( h_i \) that maximizes the number of satisfiable constraints.
- Optimal choice \( l = -1 \).
Max-SMT: Maximum Satisfiability Modulo Theories

\[ C_1: \quad l < 0 \land h_1 < 0 \]
\[ C_2: \quad l < 0 \land h_2 \geq 0 \land h_2 < 5 \]
\[ C_3: \quad l < 0 \land h_3 \geq 5 \]
\[ C_4: \quad l \geq 0 \land h_4 > 1 \]
\[ C_5: \quad l \geq 0 \land h_5 \leq 1 \]

- Find an assignment for \( l \) and \( h_i \) that maximizes the number of satisfiable constraints.
- Optimal choice \( l = -1 \).
- Max-SMT assignment \( \equiv \) maximizing channel capacity.
Max-SMT: Maximum Satisfiability Modulo Theories

\[ C_1: \ l < 0 \land h_1 < 0 \]
\[ C_2: \ l < 0 \land h_2 \geq 0 \land h_2 < 5 \]
\[ C_3: \ l < 0 \land h_3 \geq 5 \]
\[ C_4: \ l \geq 0 \land h_4 > 1 \]
\[ C_5: \ l \geq 0 \land h_5 \leq 1 \]

- Find an assignment for \( l \) and \( h_i \) that maximizes the number of satisfiable constraints.
- Optimal choice \( l = -1 \).
- Max-SMT assignment \( \equiv \) maximizing channel capacity.

MAX-SMT Problem: Find an assignment of values to variables that maximizes the number of simultaneously satisfied clauses.
Finding Best Attack Tree
Method 2
Finding Balanced Partitions

Find low inputs $L$ for an attack tree with optimally balanced divisions.

Maximizing Shannon entropy based on symbolic constraints.

Given probabilities, quantify information gain with Shannon entropy:

$$H = \sum_i p(C_i(h,l)) \log_2 \frac{1}{p(C_i(h,l))}$$

Compared with MAX-SMT:

Channel Capacity $= \log_2 \# \text{divisions}$

$H \leq \text{CC}$
Finding Balanced Partitions
Find low inputs $L$ for an attack tree with optimally balanced divisions

Compared with MAX-SMT:
Channel Capacity $= \log_2 \#\text{divisions}$

$H \leq CC$
Finding Balanced Partitions

Find low inputs $L$ for an attack tree with optimally balanced divisions

$\equiv$ Maximizing Shannon entropy based on symbolic constraints.
Finding Balanced Partitions

Find low inputs \( L \) for an attack tree with optimally balanced divisions

\[ \equiv \text{Maximizing Shannon entropy based on symbolic constraints.} \]

Given probabilities, quantify information gain with *Shannon entropy*:

\[ C_i(h, l) \]
Finding Balanced Partitions

Find low inputs $L$ for an attack tree with optimally balanced divisions

$\equiv$ Maximizing Shannon entropy based on symbolic constraints.

Given probabilities, quantify information gain with Shannon entropy:

$$p(C_i(h, l))$$
Finding Balanced Partitions

Find low inputs $L$ for an attack tree with optimally balanced divisions

≡ Maximizing Shannon entropy based on symbolic constraints.

Given probabilities, quantify information gain with *Shannon entropy*:

$$p(C_i(h, l)) \log_2 \frac{1}{p(C_i(h, l))}$$
Finding Balanced Partitions

Find low inputs $L$ for an attack tree with optimally balanced divisions

$\equiv$ Maximizing Shannon entropy based on symbolic constraints.

Given probabilities, quantify information gain with Shannon entropy:

$$\sum_i p(C_i(h, l)) \log_2 \frac{1}{p(C_i(h, l))}$$
Finding Balanced Partitions

Find low inputs $L$ for an attack tree with optimally balanced divisions

$\equiv$ Maximizing Shannon entropy based on symbolic constraints.

Given probabilities, quantify information gain with *Shannon entropy*:

$$\mathcal{H} = \sum_i p(C_i(h, l)) \log_2 \frac{1}{p(C_i(h, l))}$$
Finding Balanced Partitions

Find low inputs \( L \) for an attack tree with optimally balanced divisions

\[ \equiv \text{Maximizing Shannon entropy based on symbolic constraints.} \]

Given probabilities, quantify information gain with *Shannon entropy*:

\[
\mathcal{H} = \sum_i p(C_i(h, l)) \log_2 \frac{1}{p(C_i(h, l))}
\]

Compared with MAX-SMT:
Finding Balanced Partitions

Find low inputs $L$ for an attack tree with optimally balanced divisions

$\equiv$ Maximizing Shannon entropy based on symbolic constraints.

Given probabilities, quantify information gain with Shannon entropy:

$$H = \sum_i p(C_i(h, l)) \log_2 \frac{1}{p(C_i(h, l))}$$

Compared with MAX-SMT:

Channel Capacity $= \log_2 \#\text{divisions}$

$$H \leq CC$$
Maximizing Shannon Entropy Numerically

\[ L = l \]

\[ \text{cost}\langle 1 \rangle \quad h \geq l \]

\[ L = l_1 \]

\[ \text{cost}\langle 1 \rangle \quad h \geq l \quad h \geq l_1 \]

\[ \text{cost}\langle 2 \rangle \quad h < l \]

\[ L = l_2 \]

\[ \text{cost}\langle 2 \rangle \quad h < l \quad h < l_2 \]
Maximizing Shannon Entropy Numerically

\[ L = l \]

\[ \text{cost}\langle 1 \rangle \quad h \geq l \]

\[ L = l_1 \]

\[ \text{cost}\langle 1 \rangle \quad h \geq l \quad h \geq l_1 \]

\[ \text{cost}\langle 2 \rangle \quad h < l \]

\[ L = l_2 \]

\[ \text{cost}\langle 2 \rangle \quad h < l \quad h < l_2 \]

\[ C_1 = h < l \land h < l_1 \]

\[ C_2 = h < l \land h \geq l_1 \]

\[ C_3 = h \geq l \land h < l_2 \]

\[ C_4 = h \geq l \land h \geq l_2 \]
Maximizing Shannon Entropy Numerically

\[ L = I \]

\[ L = I_1 \]

\[ L = I_2 \]

\[ C_1 = h < I \land h < I_1 \]
\[ C_2 = h < I \land h \geq I_1 \]
\[ C_3 = h \geq I \land h < I_2 \]
\[ C_4 = h \geq I \land h \geq I_2 \]
Maximizing Shannon Entropy Numerically

\[ C_1 = h < l \land h < l_1 \]
Maximizing Shannon Entropy Numerically

\[ C_1 = h < l \wedge h < l_1 \]

Symbolic model counting functions computed with Barvinok.
Maximizing Shannon Entropy Numerically

\[ C_1 = h < l \land h < l_1 \]

Symbolic model counting functions computed with Barvinok.

Barvinok gives piecewise multi-variate polynomial.
Maximizing Shannon Entropy Numerically

\[ C_1 = h < l \land h < l_1 \]

Symbolic model counting functions computed with Barvinok.

Barvinok gives piecewise multi-variate polynomial.

\[ F_1(l, l_1, l_2) = \begin{cases} 
6 & : l > 6 \land l_1 > 6 \\
1 & : 1 \leq l \leq 6 \land l \leq l_1 \\
l_1 - 1 & : 1 \leq l_1 \leq 6 \land l_1 < l 
\end{cases} \]

\( F_1(\bar{L}) \) tells you the size of the partition cell for \( C_1 \), for given \( \bar{L} \).
Maximizing Shannon Entropy Numerically

<table>
<thead>
<tr>
<th>Condition</th>
<th>Function</th>
<th>Case 1</th>
<th>Case 2</th>
<th>Case 3</th>
<th>Case 4</th>
</tr>
</thead>
</table>
| $C_1 = h < l \land h < l_1$ | $F_1(\bar{i}) = \begin{cases} 
8 & : l > 8 \land l_1 > 8 \\
-1 & : 1 \leq l \leq 8 \land l \leq l_1 \\
l_1 - 1 & : 1 \leq l_1 \leq 8 \land l_1 < l 
\end{cases}$ | $8 : l_1 < 1 \land 8 < l$ | $l - l_1 : 1 \leq l_1 \leq l \leq 8$ | $l - 1 : l_1 < 1 \land l \leq 8$ | $9 - l : 1 \leq l_1 \leq 8 < l$ |
| $C_2 = h < l \land h \geq l_1$ | $F_2(\bar{i}) = \begin{cases} 
8 & : l_1 < 1 \land 8 < l \\
l - l_1 & : 1 \leq l_1 \leq l \leq 8 \\
l - 1 & : l_1 < 1 \land l \leq 8 \\
9 - l_1 & : 1 \leq l_1 \leq 8 < l 
\end{cases}$ | $8 : l_1 < 1 \land 8 < l_2$ | $l_2 - l : 1 \leq l \leq l_2 \leq 8$ | $l_2 - 1 : l < 1 \land l_2 \leq 8$ | $9 - l : 1 \leq l \leq 8 < l_2$ |
| $C_3 = h \geq l \land h < l_2$ | $F_3(\bar{i}) = \begin{cases} 
8 & : l < 1 \land 8 < l_2 \\
l_2 - l & : 1 \leq l \leq l_2 \leq 8 \\
l_2 - 1 & : l < 1 \land l_2 \leq 8 \\
9 - l & : 1 \leq l \leq 8 < l_2 
\end{cases}$ | $8 : l < 1 \land l_2 < 1$ | $9 - l : 1 \leq l \leq 8 \land l_2 < l$ | $9 - l_2 : 1 \leq l_2 \leq 8 \land l \leq l_2$ |
Maximizing Shannon Entropy Numerically

\[ C_1 = h < l \wedge h < l_1 \]

\[ F_1(\bar{l}) = \begin{cases} 
8 : l > 8 \wedge l_1 > 8 \\
9 : l > 8 \wedge l_1 < l \\
1 : 1 \leq l \leq 8 \wedge l_1 \leq l \\
4 : 1 \leq l \leq 8 \wedge l_1 < l \\
6 : 1 \leq l \leq 8 \wedge l_1 < l \\
7 : 1 \leq l \leq 8 \wedge l_1 < l \\
5 : 1 \leq l \leq 8 \wedge l_1 < l \\
3 : 1 \leq l \leq 8 \wedge l_1 < l \\
2 : 1 \leq l \leq 8 \wedge l_1 < l \\
1 : 1 \leq l \leq 8 \wedge l_1 < l \\
0 : 1 \leq l \leq 8 \wedge l_1 < l \\
\end{cases} \]

\[ C_2 = h < l \wedge h \geq l_1 \]

\[ F_2(\bar{l}) = \begin{cases} 
8 : l_1 < 1 \wedge 8 \geq l_1 \\
9 : l_1 < 1 \wedge 8 \geq l_1 \\
1 : 1 \leq l_1 \leq l \wedge 8 \geq l_1 \\
4 : 1 \leq l_1 \leq l \wedge 8 \geq l_1 \\
6 : 1 \leq l_1 \leq l \wedge 8 \geq l_1 \\
7 : 1 \leq l_1 \leq l \wedge 8 \geq l_1 \\
5 : 1 \leq l_1 \leq l \wedge 8 \geq l_1 \\
3 : 1 \leq l_1 \leq l \wedge 8 \geq l_1 \\
2 : 1 \leq l_1 \leq l \wedge 8 \geq l_1 \\
1 : 1 \leq l_1 \leq l \wedge 8 \geq l_1 \\
0 : 1 \leq l_1 \leq l \wedge 8 \geq l_1 \\
\end{cases} \]

\[ C_3 = h \geq l \wedge h < l_2 \]

\[ F_3(\bar{l}) = \begin{cases} 
8 : l < 1 \wedge 8 \leq l_2 \\
9 : l < 1 \wedge 8 \leq l_2 \\
1 : 1 \leq l \leq l_2 \wedge 8 \leq l_2 \\
4 : 1 \leq l \leq l_2 \wedge 8 \leq l_2 \\
6 : 1 \leq l \leq l_2 \wedge 8 \leq l_2 \\
7 : 1 \leq l \leq l_2 \wedge 8 \leq l_2 \\
5 : 1 \leq l \leq l_2 \wedge 8 \leq l_2 \\
3 : 1 \leq l \leq l_2 \wedge 8 \leq l_2 \\
2 : 1 \leq l \leq l_2 \wedge 8 \leq l_2 \\
1 : 1 \leq l \leq l_2 \wedge 8 \leq l_2 \\
0 : 1 \leq l \leq l_2 \wedge 8 \leq l_2 \\
\end{cases} \]

\[ C_4 = h \geq l \wedge h \geq l_2 \]

\[ F_4(\bar{l}) = \begin{cases} 
8 : l < 1 \wedge l_2 < 1 \\
9 : l < 1 \wedge l_2 < 1 \\
1 : 1 \leq l \leq l_2 \wedge l_2 < 1 \\
4 : 1 \leq l \leq l_2 \wedge l_2 < 1 \\
6 : 1 \leq l \leq l_2 \wedge l_2 < 1 \\
7 : 1 \leq l \leq l_2 \wedge l_2 < 1 \\
5 : 1 \leq l \leq l_2 \wedge l_2 < 1 \\
3 : 1 \leq l \leq l_2 \wedge l_2 < 1 \\
2 : 1 \leq l \leq l_2 \wedge l_2 < 1 \\
1 : 1 \leq l \leq l_2 \wedge l_2 < 1 \\
0 : 1 \leq l \leq l_2 \wedge l_2 < 1 \\
\end{cases} \]

\[ \frac{F_1(\bar{L})}{8} \]
Maximizing Shannon Entropy Numerically

<table>
<thead>
<tr>
<th>Condition</th>
<th>( F_i(\bar{L}) )</th>
</tr>
</thead>
</table>
| \( C_1 = h < l \land h < l_1 \) | \[
\begin{align*}
F_1(\bar{L}) = \begin{cases}
8 & : l > 8 \land l_1 > 8 \\
 l - 1 & : 1 \leq l \leq 8 \land l \leq l_1 \\
l_1 - 1 & : 1 \leq l_1 \leq 8 \land l_1 < l
\end{cases}
\end{align*}
\] |
| \( C_2 = h < l \land h \geq l_1 \) | \[
\begin{align*}
F_2(\bar{L}) = \begin{cases}
8 & : l_1 < 1 \land 8 < l \\
l - l_1 & : 1 \leq l_1 \leq l \leq 8 \\
l_1 - 1 & : l_1 < 1 \land l \leq 8 \\
 9 - l_1 & : 1 \leq l_1 \leq 8 < l
\end{cases}
\end{align*}
\] |
| \( C_3 = h \geq l \land h < l_2 \) | \[
\begin{align*}
F_3(\bar{L}) = \begin{cases}
8 & : l < 1 \land 8 < l_2 \\
l_2 - l & : 1 \leq l \leq l_2 \leq 8 \\
l_2 - 1 & : 1 \leq l \leq l_2 \leq 8 \\
 9 - l & : 1 \leq l \leq 8 < l_2
\end{cases}
\end{align*}
\] |
| \( C_4 = h \geq l \land h \geq l_2 \) | \[
\begin{align*}
F_4(\bar{L}) = \begin{cases}
8 & : l < 1 \land l_2 < 1 \\
 9 - l & : 1 \leq l \leq 8 \land l_2 < l \\
 9 - l_2 & : 1 \leq l_2 \leq 8 \land l \leq l_2
\end{cases}
\end{align*}
\] |

\[ \mathcal{H}(\bar{L}) = \frac{F_1(\bar{L})}{8} \]
Maximizing Shannon Entropy Numerically

<table>
<thead>
<tr>
<th>Condition</th>
<th>Expression $F_i(\bar{l})$</th>
</tr>
</thead>
</table>
| $C_1 = h < l \land h < l_1$ | $F_1(\bar{l}) = \begin{cases} 
0 : l > 8 \land l_1 > 8 \\
1 : 1 \leq l \leq 8 \land l \leq l_1 \\
1 : 1 \leq l_1 \leq 8 \land l_1 < l 
\end{cases}$ |
| $C_2 = h < l \land h \geq l_1$ | $F_2(\bar{l}) = \begin{cases} 
0 : l_1 < 1 \land 8 < l \\
l - l_1 : 1 \leq l_1 \leq l \leq 8 \\
l - 1 : l_1 < 1 \leq l \leq 8 \\
9 - l_1 : 1 \leq l_1 \leq 8 < l 
\end{cases}$ |
| $C_3 = h \geq l \land h < l_2$ | $F_3(\bar{l}) = \begin{cases} 
0 : l < 1 \land 8 < l_2 \\
l_2 - l : 1 \leq l \leq l_2 \leq 8 \\
l_2 - 1 : l < 1 \leq l_2 \leq 8 \\
9 - l : 1 \leq l \leq 8 < l_2 
\end{cases}$ |
| $C_4 = h \geq l \land h \geq l_2$ | $F_4(\bar{l}) = \begin{cases} 
0 : l < 1 \land l_2 < 1 \\
9 - l : 1 \leq l \leq 8 \land l_2 < l \\
9 - l_2 : 1 \leq l_2 \leq 8 \land l \leq l_2 
\end{cases}$ |

$H(\bar{L}) = \frac{F_1(\bar{L})}{8} \log_2 \frac{8}{F_1(\bar{L})} + \frac{F_2(\bar{L})}{8} \log_2 \frac{8}{F_2(\bar{L})} + \frac{F_3(\bar{L})}{8} \log_2 \frac{8}{F_3(\bar{L})} + \frac{F_4(\bar{L})}{8} \log_2 \frac{8}{F_4(\bar{L})}$
Maximizing Shannon Entropy Numerically

\[ H(\bar{L}) = \frac{F_1(\bar{L})}{8} \log_2 \frac{8}{F_1(\bar{L})} + \frac{F_2(\bar{L})}{8} \log_2 \frac{8}{F_2(\bar{L})} + \frac{F_3(\bar{L})}{8} \log_2 \frac{8}{F_3(\bar{L})} + \frac{F_4(\bar{L})}{8} \log_2 \frac{8}{F_4(\bar{L})} \]
Maximizing Shannon Entropy Numerically

\[ H(\bar{L}) = \frac{F_1(\bar{L})}{8} \log_2 \frac{8}{F_1(\bar{L})} + \frac{F_2(\bar{L})}{8} \log_2 \frac{8}{F_2(\bar{L})} + \frac{F_3(\bar{L})}{8} \log_2 \frac{8}{F_3(\bar{L})} + \frac{F_4(\bar{L})}{8} \log_2 \frac{8}{F_4(\bar{L})} \]

Numerically maximize \( H(\bar{L}) \)

\( \bar{L} = \langle 4, 2, 6 \rangle \)
Maximizing Shannon Entropy Numerically

\[
H(\bar{L}) = \frac{F_1(\bar{L})}{8} \log_2 \frac{8}{F_1(\bar{L})} + \frac{F_2(\bar{L})}{8} \log_2 \frac{8}{F_2(\bar{L})} + \frac{F_3(\bar{L})}{8} \log_2 \frac{8}{F_3(\bar{L})} + \frac{F_4(\bar{L})}{8} \log_2 \frac{8}{F_4(\bar{L})}
\]

Numerically maximize \( H(\bar{L}) \)

\[
\bar{L} = \langle 4, 2, 6 \rangle
\]

First two steps of optimal binary search attack on 8 secrets.
Finding Best Attack Tree
Method 3
Maximizing Shannon Entropy, Third Approach

Maximum Satisfiable Subsets (MSS).

Optimization version of SAT.

MaxH-MARCO algorithm:
1. Exhaustive enumeration of maximal partitions of the secret $h$.
2. Compute Shannon entropy for each maximal partition, select the one with largest Entropy.

MSS solution $\Rightarrow$ maximize Shannon entropy.
Maximizing Shannon Entropy, Third Approach

Maximum Satisfiable Subsets (MSS).

Optimization version of SAT.
Maximizing Shannon Entropy, Third Approach

Maximum Satisfiable Subsets (MSS).

Optimization version of SAT.

MaxH-MARCO algorithm:
Maximizing Shannon Entropy, Third Approach

Maximum Satisfiable Subsets (MSS).

Optimization version of SAT.

MaxH-MARCO algorithm:

1. Exhaustive enumeration of maximal partitions of the secret $h$. 
Maximizing Shannon Entropy, Third Approach

Maximum Satisfiable Subsets (MSS).

Optimization version of SAT.

MaxH-MARCO algorithm:

1. Exhaustive enumeration of maximal partitions of the secret $h$.
2. Compute Shannon entropy for each maximal partition,
Maximizing Shannon Entropy, Third Approach

Maximum Satisfiable Subsets (MSS).

Optimization version of SAT.

MaxH-MARCO algorithm:

1. Exhaustive enumeration of maximal partitions of the secret $h$.
2. Compute Shannon entropy for each maximal partition, select the one with largest Entropy.
Maximizing Shannon Entropy, Third Approach

Maximum Satisfiable Subsets (MSS).

Optimization version of SAT.

MaxH-MARCO algorithm:

1. Exhaustive enumeration of maximal partitions of the secret $h$.
2. Compute Shannon entropy for each maximal partition, select the one with largest Entropy.

MSS solution $\Rightarrow$ maximize Shannon entropy.
Finding Best Attack Tree
Finding Best Attack Tree
3 Methods
Finding Best Attack Tree
3 Methods
Do they work?
Finding Best Attack Tree
3 Methods

Do they work?

Yes
Implementation

- Java Symbolic Pathfinder (JPF / SPF) for symbolic execution.
- Specialized listeners for tracking observables (time, space).
- Latte and Barvinok for model counting path constraints.
- Max-SMT (Z3), MARCO (java + Z3) MSS.
- Mathematica’s NMAXIMIZE for numeric maximization.
- Heuristics: top-down greedy optimization.
## Case study: Law Enforcement Employment Database

From DARPA Space-Time Analysis for Cybersecurity (STAC)

### Server

- 41 classes, 2844 line of code.
- stores all employee records by ID in a database.
- Some employee IDs have restricted access.

### Client

Commands available for users: SEARCH, INSERT, GET, PUT, …

**SEARCH a b** has a timing channel: adaptive range query attack.
Case study: Law Enforcement Employment Database

Domain: 100 possible IDs in database (6.541 bits)
Case study: Law Enforcement Employment Database

Domain: 100 possible IDs in database (6.541 bits)

MAX-SMT

- Attack tree depth: 17 (complete attack)
- Running time: 21s
Case study: Law Enforcement Employment Database

Domain: 100 possible IDs in database (6.541 bits)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Attack tree depth</th>
<th>Running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX-SMT</td>
<td>17 (complete attack)</td>
<td>21s</td>
</tr>
<tr>
<td>Numeric Entropy Maximization</td>
<td>7 (complete attack)</td>
<td>57s</td>
</tr>
</tbody>
</table>
## Case study: Law Enforcement Employment Database

Domain: 100 possible IDs in database (6.541 bits)

<table>
<thead>
<tr>
<th>Method</th>
<th>Attack tree depth</th>
<th>Running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX-SMT</td>
<td>17 (complete attack)</td>
<td>21s</td>
</tr>
<tr>
<td>Numeric Entropy Maximization</td>
<td>7 (complete attack)</td>
<td>57s</td>
</tr>
<tr>
<td>Max SAT Subsets</td>
<td>7 (complete attack)</td>
<td>2m 36s</td>
</tr>
</tbody>
</table>
Case study: Law Enforcement Employment Database

Domain: 1,000,000 possible IDs in database (19.9 bits)
Case study: Law Enforcement Employment Database

Domain: 1,000,000 possible IDs in database (19.9 bits)

**MAX-SMT**

- Attack tree depth: 17
- Incomplete attack: leaks at most 12.5 out of 19.9 bits
- Running time: 18m 31s
Case study: Law Enforcement Employment Database

Domain: 1,000,000 possible IDs in database (19.9 bits)

<table>
<thead>
<tr>
<th>MAX-SMT</th>
</tr>
</thead>
<tbody>
<tr>
<td>➤ Attack tree depth: 17</td>
</tr>
<tr>
<td>➤ Incomplete attack: leaks at most 12.5 out of 19.9 bits</td>
</tr>
<tr>
<td>➤ Running time: 18m 31s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Numeric Entropy Maximization</th>
</tr>
</thead>
<tbody>
<tr>
<td>➤ Attack tree depth: 11</td>
</tr>
<tr>
<td>➤ Incomplete attack: leaks 10.0 out of 19.9 bits</td>
</tr>
<tr>
<td>➤ Running time: 15m 8s</td>
</tr>
<tr>
<td>Method</td>
</tr>
<tr>
<td>------------------------------</td>
</tr>
<tr>
<td>MAX-SMT</td>
</tr>
<tr>
<td>Numeric Entropy Maximization</td>
</tr>
<tr>
<td>Max SAT Subsets</td>
</tr>
</tbody>
</table>

Does not scale to this domain.
More Case Studies

We synthesized attacks for:

- ModPow used in RSA
- Compression Ratio Information Leak Made Easy (CRIME)
- `java.util.Arrays.equal()` (segment oracle attack)
Conclusions

- Symbolic execution of adversary model to get constraint tree.
- Solve optimization problem to get low inputs to maximize leakage: attack tree.
- MAX-SMT
  Symbolic Model Counting + Numeric Maximization
  Max-SAT-Subsets
- Experimentally validated our approach.
Questions?

Thank you.