Computing Information Flow Using Symbolic Model-Checking

Rohit Chadha ¹  Umang Mathur ²  Stefan Schwoon ³

¹University of Missouri
Columbia, Missouri, USA

²Indian Institute of Technology - Bombay
Mumbai

³LSV, ENS Cachan
France

December 17, 2014
Outline

Introduction

Preliminaries

Summary Calculation

Computing Information Leakage: Symbolic Algorithms

Moped-QLeak

Demo

Conclusions and Future Work

Thank You
Introduction

- Quantifying information leakage - Inferring information about inputs by observing public outputs
Introduction

- Quantifying information leakage - Inferring information about inputs by observing public outputs
- No leakage $\implies$ Outputs independent of inputs
Introduction

- Quantifying information leakage - Inferring information about inputs by observing public outputs
- No leakage $\implies$ Outputs independent of inputs
- Full leakage $\implies$ Unique input corresponding to given output
Introduction

- Quantifying information leakage - Inferring information about inputs by observing public outputs
  - No leakage $\implies$ Outputs independent of inputs
  - Full leakage $\implies$ Unique input corresponding to given output
  - Comparing leakage across programs - less leakage is desirable
Measuring Information Leakage

1. Min-entropy leakage measures vulnerability of the secret inputs to being guessed correctly in a single attempt of the adversary.

\[ \text{ME}(P) = \log \sum_{o \in O} \max_{s \in S} \mu(S = s | O = o). \]

2. Shannon entropy leakage measures expected number of guesses required to correctly guess the secret input.

\[ \text{SE}(P) = \log |S| - 1 \frac{|S|}{\sum_{o \in O} |P^{-1}(o)| \log |P^{-1}(o)|}. \]
Measuring Information Leakage

Several metrics - min-entropy, Shannon’s entropy, etc.,
Measuring Information Leakage

Several metrics - min-entropy, Shannon’s entropy, etc.,

1. Min-entropy leakage measures vulnerability of the secret inputs to being guessed correctly in a single attempt of the adversary

\[
\text{ME}_U(P) = \log \sum_{o \in O} \max_{s \in S} \mu(S = s \mid O = o).
\]
Measuring Information Leakage

Several metrics - min-entropy, Shannon’s entropy, etc.,

1. Min-entropy leakage measures vulnerability of the secret inputs to being guessed correctly in a single attempt of the adversary

\[
\text{ME}_U(P) = \log \sum_{o \in O} \max_{s \in S} \mu(S = s \mid O = o).
\]

2. Shannon entropy leakage measures expected number of guesses required to correctly guess the secret input

\[
\text{SE}_U(P) = \log |S| - \frac{1}{|S|} \sum_{o \in O} |P^{-1}(o)| \log |P^{-1}(o)|
\]
Consider the following example:
Example

Consider the following example:

```python
def example (input) :
    output = input % 8
    return output
```
Example

Consider the following example:

```python
def example (input) :
    output = input % 8
    return output
```

What would be the information leaked by the above program?
Consider the following example:

```python
def example(input):
    output = input % 8
    return output
```

What would be the information leaked by the above program

- using min-entropy?
Example

Consider the following example:

```python
def example (input) :
    output = input % 8
    return output
```

What would be the information leaked by the above program

- using min-entropy ?
- using Shannon entropy ?
Dining Cryptographers

Cryptographers A, B and C: Dine out

Payment done by One of A, B or C, or NSA

Determine if the NSA paid or not w/o revealing information about cryptographers
Dining Cryptographers

- Cryptographers A, B and C: Dine out
Dining Cryptographers

- Cryptographers A, B and C: Dine out
Dining Cryptographers

- Cryptographers A, B and C: Dine out

- Payment done by

One of A, B or C, or NSA

Determine if the NSA paid or not w/o revealing information about cryptographers
Dining Cryptographers

- Cryptographers A, B and C: Dine out

- Payment done by
  - One of A, B or C, or

Determine if the NSA paid or not w/o revealing information about cryptographers.
Cryptographers A, B and C: Dine out

Payment done by
- One of A, B or C, or
- NSA
Dining Cryptographers

- Cryptographers A, B and C: Dine out

- Payment done by
  - One of A, B or C, or
  - NSA

- Determine if the NSA paid or not w/o revealing information about cryptographers
Dining Cryptographers: Protocol

2 stage protocol:

1. Every two cryptographers establish a shared one-bit secret: Toss a coin

2. Each cryptographer publicly announces a bit, which is XOR of shared bits, if did not pay ¬(XOR of shared bits), otherwise.

\[
\begin{align*}
\text{XOR}(0, 1) &= 1 \\
\text{XOR}(1, 1) &= 0 \\
\end{align*}
\]

Stage-1 (left) and Stage-2 (right)

\[
\text{XOR}(\text{Announcement } A, \text{Announcement } B, \text{Announcement } C) = 0 \iff \text{NSA paid for the dinner}
\]


Dining Cryptographers: Protocol

2 stage protocol:

1. Every two cryptographers establish a shared one-bit secret: Toss a coin

Stage-1 (left) and Stage-2 (right)

\[ \text{XOR(Announcement } A, \text{ Announcement } B, \text{ Announcement } C) = 0 \iff \text{NSA paid for the dinner} \]
Dining Cryptographers: Protocol

2 stage protocol:

1. Every two cryptographers establish a shared one-bit secret: Toss a coin
2. Each cryptographer publicly announces a bit, which is

\[
\text{XOR}(0, 1) = 1 \\
\text{XOR}(1, 1) = 0 \quad \text{iff} \quad \text{NSA paid for the dinner}
\]
Dining Cryptographers: Protocol

2 stage protocol:

1. Every two cryptographers establish a shared one-bit secret: Toss a coin
2. Each cryptographer publicly announces a bit, which is
   ▶ XOR of shared bits, if did not pay
Dining Cryptographers: Protocol

2 stage protocol:

1. Every two cryptographers establish a shared one-bit secret: Toss a coin
2. Each cryptographer publicly announces a bit, which is
   - XOR of shared bits, if did not pay
   - \( \neg \) (XOR of shared bits), otherwise
Dining Cryptographers: Protocol

2 stage protocol:

1. Every two cryptographers establish a shared one-bit secret: Toss a coin
2. Each cryptographer publicly announces a bit, which is
   - XOR of shared bits, if did not pay
   - \( \neg (\text{XOR of shared bits}) \), otherwise

\[
\text{Stage-1 (left) and Stage-2 (right)}
\]

\[
\text{XOR(Announcement}_A, \text{Announcement}_B, \text{Announcement}_C) = 0 \iff \text{NSA paid for the dinner}
\]
Dining Cryptographers: Protocol

2 stage protocol:

1. Every two cryptographers establish a shared one-bit secret: Toss a coin
2. Each cryptographer publicly announces a bit, which is
   - XOR of shared bits, if did not pay
   - \( \neg \) (XOR of shared bits), otherwise

\[
\begin{align*}
\text{Stage-1 (left) and Stage-2 (right)}
\end{align*}
\]
Dining Cryptographers: Protocol

2 stage protocol:

1. Every two cryptographers establish a shared one-bit secret: Toss a coin
2. Each cryptographer publicly announces a bit, which is
   - XOR of shared bits, if did not pay
   - \(\neg (\text{XOR of shared bits})\), otherwise

\[
\begin{align*}
\text{XOR}(0, 1) &= 0 \\
\text{XOR}(1, 1) &= 0 \\
\text{XOR}(0, 1) &= 1
\end{align*}
\]

Stage-1 (left) and Stage-2 (right)

\[
\text{XOR(Announcement}_A, \text{ Announcement}_B, \text{ Announcement}_C) = 0
\]
iff

NSA paid for the dinner
Probabilistic Boolean Programs

- Global variables $G$: Input and output
Probabilistic Boolean Programs

- Global variables $\mathcal{G}$: Input and output
- Local variables: Internal calculations
Probabilistic Boolean Programs

- Global variables $G$: Input and output
- Local variables: Internal calculations
- Program statements: transform global and local variables
Probabilistic Boolean Programs

- Global variables $G$: Input and output
- Local variables: Internal calculations
- Program statements: Transform global and local variables
- For Program $P$, $F_P: 2^G \rightarrow 2^G \cup \{\bot\}$
Probabilistic Boolean Programs

- Global variables $\mathcal{G}$: Input and output
- Local variables: Internal calculations
- Program statements: transform global and local variables
- For Program $P$, $F_P: 2^\mathcal{G} \rightarrow 2^\mathcal{G} \cup \{\bot\}$
  - $F_P(\bar{g}_0) = \bot$ iff $P$ does not terminate
Global variables $G$: Input and output

Local variables: Internal calculations

Program statements: transform global and local variables

For Program $P$, $F_P: 2^G \rightarrow 2^G \cup \{\bot\}$

$F_P(\vec{g}_0) = \bot$ iff $P$ does not terminate

Summary - Joint probability distribution $\mu$
Probabilistic Boolean Programs

- Global variables $G$: Input and output
- Local variables: Internal calculations
- Program statements: transform global and local variables
- For Program $P$, $F_P : 2^G \rightarrow 2^G \cup \{\perp\}$
  
  $F_P(\vec{g}_0) = \perp$ iff $P$ does not terminate
- Summary - Joint probability distribution $\mu$
Algebraic Decision Diagrams

- Set of variables $\mathcal{V}$
Algebraic Decision Diagrams

- Set of variables $\mathcal{V}$
- Algebraic set $M$ ($M = [0, 1]$ for probabilistic statements, $M = \{0, 1\}$ implies BDDs)
Algebraic Decision Diagrams

- Set of variables $\mathcal{V}$
- Algebraic set $M$ ($M = [0, 1]$ for probabilistic statements, $M = \{0, 1\}$ implies BDDs)
- $\text{ADD} : 2^\mathcal{V} \rightarrow M$
Algebraic Decision Diagrams

- Set of variables $\mathcal{V}$
- Algebraic set $M$ ($M = [0, 1]$ for probabilistic statements, $M = \{0, 1\}$ implies BDDs)
- $\text{ADD} : 2^\mathcal{V} \rightarrow M$
- Efficient reduced representations, similar to BDDs
Algebraic Decision Diagrams

- Set of variables $\mathcal{V}$
- Algebraic set $M$ ($M = [0, 1]$ for probabilistic statements, $M = \{0, 1\}$ implies BDDs)
- $\text{ADD} : 2^\mathcal{V} \rightarrow M$
- Efficient reduced representations, similar to BDDs
Algebraic Decision Diagrams

- Set of variables $\mathcal{V}$
- Algebraic set $M$ ($M = [0, 1]$ for probabilistic statements, $M = \{0, 1\}$ implies BDDs)
- $\text{ADD} : 2^{\mathcal{V}} \rightarrow M$
- Efficient reduced representations, similar to BDDs

```
x
  y
  |  |
  z  z
  1  0
```

$\text{ADD (up)}$ and its reduced form (bottom)
Computing Summaries: Fixed Point Iteration

- Program statement $l \rightarrow \mu_l$
Computing Summaries: Fixed Point Iteration

- Program statement $l \rightarrow \mu_l$
- Can be represented efficiently as MTBBDS
Computing Summaries: Fixed Point Iteration

- Program statement $I \rightarrow \mu_I$
- Can be represented efficiently as MTBBDs

Stmt: $x = \neg x$

![Diagram of MTBBD representing the program statement](image)
Computing Summaries: Fixed Point Iteration

- Program statement $l \rightarrow \mu l$
- Can be represented efficiently as MTBBDs

- Compose statements
Computing Summaries: Fixed Point Iteration

- Program statement $l \rightarrow \mu_l$
- Can be represented efficiently as MTBBDS

- Compose statements
- Arrive at a fixed point (Summary $\mu$)
Min Entropy : Symbolic Algorithm

For a program \( P \), with

Algorithm 1: Symbolic computation of min-entropy leakage of a probabilistic program

Input:

- \( G \), \( G' \), and \( T \)

Output:

- \( \text{ME}_U(P) \)

1. \( T_{out}, P \leftarrow \text{abstract}(\max, G, T) \)
2. \( \text{sum}_{out} \leftarrow \text{val}(\text{abstract}(+, G', T_{out}, P)) \)
3. \( T_{term}, P \leftarrow \text{abstract}(+, G', T) \)
4. \( \text{sum}_{out} \leftarrow \text{sum}_{out} + (1 - \text{val}(\text{abstract}(\min, G, T_{term}, P))) \)
5. \( \text{return} \ \log \text{sum}_{out} \)
Min Entropy: Symbolic Algorithm

For a program $P$, with

- input set $S$ (uniform distribution),
Min Entropy : Symbolic Algorithm

For a program $P$, with

- input set $S$ (uniform distribution),
- output set $O$, and,

Algorithm 3: Symbolic computation of min-entropy leakage of a probabilistic program

Input: $G, G'$ and $TP$ the summary of $P$.

Output: $\text{ME}_U(P)$

1. $T_{out}, P \leftarrow \text{abstract}(\max, G, TP)$
2. $\text{sum}_{out} \leftarrow \text{val}(\text{abstract}(+, G', T_{out}, P))$
3. $T_{term}, P \leftarrow \text{abstract}(+, G', TP)$
4. $\text{sum}_{out} \leftarrow \text{sum}_{out} + (1 - \text{val}(\text{abstract}(\text{min}, G, T_{term}, P)))$
5. Return $\log \text{sum}_{out}$
Min Entropy : Symbolic Algorithm

For a program \( P \), with

- input set \( S \) (uniform distribution),
- output set \( O \), and,
- joint probability distribution \( \mu \),

\[
\text{Algorithm 4: Symbolic computation of min-entropy leakage of a probabilistic program}
\]
\[
\text{Input: } G, G' \text{ and } \mathcal{T}_P \text{ the summary of } P.
\]
\[
\text{Output: } \text{ME}_U(P)
\]
\[
1 \quad \text{begin}
2 \quad \mathcal{T}_{\text{out}}, P \leftarrow \text{abstract(max, } G, \mathcal{T}_P\text{)}
3 \quad \text{sum}_{\text{out}} \leftarrow \text{val(abstract(}+\text{, } G', \mathcal{T}_{\text{out}}, P\text{))}
4 \quad \mathcal{T}_{\text{term}}, P \leftarrow \text{abstract(}+\text{, } G', \mathcal{T}_P\text{)}
5 \quad \text{sum}_{\text{out}} \leftarrow \text{sum}_{\text{out}} + (1 - \text{val(abstract(min, } G, \mathcal{T}_{\text{term}}, P\text{)}))
6 \quad \text{return } \log \text{sum}_{\text{out}}
\]


Min Entropy : Symbolic Algorithm

For a program $P$, with
- input set $S$ (uniform distribution),
- output set $O$, and,
- joint probability distribution $\mu$,

the min-entropy leakage $\text{ME}_U(P)$ is

$$\text{ME}_U(P) = \log \sum_{o \in O} \max_{s \in S} \mu(S = s \mid O = o).$$
Min Entropy : Symbolic Algorithm

For a program $P$, with

- input set $S$ (uniform distribution),
- output set $O$, and,
- joint probability distribution $\mu$,

the min-entropy leakage $ME_U(P)$ is

$$ME_U(P) = \log \sum_{o \in O} \max_{s \in S} \mu(S = s \mid O = o).$$

Algorithm 6: Symbolic computation of min-entropy leakage of a probabilistic program

**Input:** $G, G'$ and $T_P$ the summary of $P$.  
**Output:** $ME_U(P)$

1. begin
2. $T_{out,P} \leftarrow \text{abstract}(\text{max}, G, T_P)$
3. $\text{sum}_{out} \leftarrow \text{val}(\text{abstract}(+, G', T_{out,P}))$
4. $T_{term,P} \leftarrow \text{abstract}(+, G', T_P)$
5. $\text{sum}_{out} \leftarrow \text{sum}_{out} + (1 - \text{val}(\text{abstract}(\text{min}, G, T_{term,P})))$;
6. **return** $\log \text{sum}_{out}$
Shannon Entropy: Symbolic Algorithm

\[
SE_u(P) = \log |S| - \frac{1}{|S|} \sum_{o \in O} |P^{-1}(o)| \log |P^{-1}(o)|
\]
Shannon Entropy : Symbolic Algorithm

\[
SE_U(P) = \log |S| - \frac{1}{|S|} \sum_{o \in O} |P^{-1}(o)| \log |P^{-1}(o)|
\]

Algorithm 8: Symbolic computation of Shannon entropy leakage of a probabilistic program

**Input:** \( G, G' \) and \( T_P \) the summary of \( P \).

**Output:** \( SE_U(P) \)

1. **Let** \( n \) be the number of variables in \( G \).

2. **begin**
   3. \( T_{\text{norm-eq-size},P} \leftarrow \text{divide}(\text{abstract}(+, G, T_P), 2^n) \)
   4. \( \text{val}_{\text{out}} \leftarrow (- \text{val}(\text{abstract}(\star, G', T_{\text{norm-eq-size},P}))) \)
   5. \( T_{\text{term},P} \leftarrow \text{abstract}(+, G', T_P) \)
   6. \( \text{prob}_{\text{out,non-term}} \leftarrow (1 - \frac{\text{val}(\text{abstract}(+, G, T_{\text{term},P}))}{2^n}) \)
   7. \( \text{val}_{\text{out,non-term}} \leftarrow (- \text{prob}_{\text{out,non-term}} \log \text{prob}_{\text{out,non-term}}) \)
   8. \( T_{\text{norm-\star out},P} \leftarrow \text{divide}(\text{abstract}(\star, G', T_P), 2^n) \)
   9. \( \text{val}_{\text{cond}} \leftarrow (-\text{val}(\text{abstract}(+, G, T_{\text{\star out},P}))) \)
   10. \( T_{\text{non-term},P} \leftarrow \text{subtract}(1, T_{\text{term},P}) \)
   11. \( \text{val}_{\text{cond,non-term}} \leftarrow (-\frac{\text{val}(\text{abstract}(\star, G, T_{\text{non-term-prob},P}))}{2^n}) \)
   12. **return** \( \text{val}_{\text{out}} + \text{val}_{\text{out,non-term}} - \text{val}_{\text{cond}} - \text{val}_{\text{cond,non-term}} \)
Tool Moped-QLeak: extends Moped
Tool Moped-QLeak: extends Moped

Source - C/C++
Tool Moped-QLeak: extends Moped

- Source - C/C++

- Input language *Remopla* - arrays, integers, struct’s, etc.,

```cpp
#define N 32
#define DEFAULT_INT_BITS N

unsigned int var1;
bool g;

module void f(unsigned int v, bool z) {
    bool k;
    pchoice :: 0.2 -> label2: k = g && z;
    :: 0.8 -> var1 = var1 + v;
}

module void main() {
    var1 = 53;
    pchoice :: 0.3 -> label1: g = true;
    :: 0.7 -> f(var1, !g);
}
```
Tool Moped-QLeak: extends Moped

Source - C/C++

Input language *Remopla* - arrays, integers, struct’s, etc.,

```c
#define N 32
#define DEFAULT_INT_BITS N

unsigned int var1;
bool g;

module void f(unsigned int v, bool z){
    bool k;
    pchoice
        :: 0.2 -> label2: k = g && z;
        :: 0.8 -> var1 = var1 + v;
    choicep
}

module void main(){
    var1 = 53;
    pchoice
        :: 0.3 -> label1: g = true;
        :: 0.7 -> f(var1, !g);
    choicep
}
```
Moped-QLeak

Modifications/Optimizations made:
Moped-QLeak

Modifications/Optimizations made:

- Algebraic operations
Moped-QLeak

Modifications/Optimizations made:
- Algebraic operations
- Variable orderings - manual
Moped-QLeak

Modifications/Optimizations made:

- Algebraic operations
- Variable orderings - manual

Salient features:
Moped-QLeak

Modifications/Optimizations made:
- Algebraic operations
- Variable orderings - manual

Salient features:
- Handles large number of bits (30 bits)
Moped-QLeak

Modifications/Optimizations made:
- Algebraic operations
- Variable orderings - manual

Salient features:
- Handles large number of bits (30 bits)
- Time taken in milliseconds

Consistently outperforms sqifc (Malacaria et al.)
Moped-QLeak

Modifications/Optimizations made:
- Algebraic operations
- Variable orderings - manual

Salient features:
- Handles large number of bits (30 bits)
- Time taken in milliseconds
- Consistently outperforms sqifc (Malacaria et. al)
<table>
<thead>
<tr>
<th>Example</th>
<th>Order</th>
<th>ME</th>
<th>SE</th>
<th>Time</th>
<th>Data types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Illustrative Example</td>
<td>I</td>
<td>3</td>
<td>2.03966e-05</td>
<td>0.215</td>
<td>bool</td>
</tr>
<tr>
<td>Electronic Purse</td>
<td>D</td>
<td>2</td>
<td>2</td>
<td>0.009</td>
<td>5 bit integers (Restricted)</td>
</tr>
<tr>
<td>Mix and Duplicate</td>
<td>S</td>
<td>16</td>
<td>16</td>
<td>0.041</td>
<td>bool</td>
</tr>
<tr>
<td>Binary Search</td>
<td>I</td>
<td>16</td>
<td>16</td>
<td>9.307</td>
<td>bool</td>
</tr>
<tr>
<td>Sanity Check</td>
<td>I</td>
<td>4</td>
<td>1.168e-7</td>
<td>0.060</td>
<td>bool</td>
</tr>
<tr>
<td>Implicit Flow</td>
<td>D</td>
<td>2.8074</td>
<td>1.757e-07</td>
<td>0.016</td>
<td>30 bit integers</td>
</tr>
<tr>
<td>Implicit Flow</td>
<td>D</td>
<td>2.8074</td>
<td>0.003</td>
<td>0.010</td>
<td>15 bit integers</td>
</tr>
<tr>
<td>Implicit Flow</td>
<td>D</td>
<td>2.8074</td>
<td>4.67189e-08</td>
<td>0.190</td>
<td>bool</td>
</tr>
<tr>
<td>Masked Copy</td>
<td>I</td>
<td>16</td>
<td>16</td>
<td>0.038</td>
<td>bool</td>
</tr>
<tr>
<td>Sum Query</td>
<td>D</td>
<td>4.80735</td>
<td>4.35132</td>
<td>0.034</td>
<td>5 bit integers (Restricted)</td>
</tr>
</tbody>
</table>
Related Work

- (Köpf et. al.,) : iteratively refine equivalence classes (deterministic only)

- (Klebanov et. al.,) : program to SMT formula, count outputs (deterministic, loop free only)

- (Biondi et. al.,) : forward symbolic execution - performance comparable to sqifc
Related Work

- (Köpf et. al.,) : iteratively refine equivalence classes (deterministic only)

- (Klebanov et. al.,) : program to SMT formula, count outputs (deterministic, loop free only)
Related Work

- (Köpf et. al.,) : iteratively refine equivalence classes (deterministic only)

- (Klebanov et. al.,) : program to SMT formula, count outputs (deterministic, loop free only)

- (Biondi et. al.,) : forward symbolic execution - performance comparable to sqifc
Related Work

- (Köpf et. al.,) : iteratively refine equivalence classes (deterministic only)
- (Klebanov et. al.,) : program to SMT formula, count outputs (deterministic, loop free only)
- (Biondi et. al.,) : forward symbolic execution - performance comparable to sqifc
Tool demonstration
Conclusions and Future Work

- Symbolic algorithms for measuring information leakage
Conclusions and Future Work

- Symbolic algorithms for measuring information leakage
- Integrable in any BDD based reachability analysis tool
Conclusions and Future Work

- Symbolic algorithms for measuring information leakage
- Interagble in any BDD based reachability analysis tool
- Summary calculation is the overhead - BDD size (algebraic operations) and variable orderings
Conclusions and Future Work

- Symbolic algorithms for measuring information leakage
- Interagble in any BDD based reachability analysis tool
- Summary calculation is the overhead - BDD size (algebraic operations) and variable orderings
- Future work:
Conclusions and Future Work

- Symbolic algorithms for measuring information leakage
- Interagble in any BDD based reachability analysis tool
- Summary calculation is the overhead - BDD size (algebraic operations) and variable orderings
- Future work:
  - Recursive algorithms
Conclusions and Future Work

- Symbolic algorithms for measuring information leakage
- Interagble in any BDD based reachability analysis tool
- Summary calculation is the overhead - BDD size (algebraic operations) and variable orderings
- Future work:
  - Recursive algorithms
  - Other symbolic model-checking frameworks - CEGAR
Conclusions and Future Work

- Symbolic algorithms for measuring information leakage
- Interagble in any BDD based reachability analysis tool
- Summary calculation is the overhead - BDD size (algebraic operations) and variable orderings
- Future work:
  - Recursive algorithms
  - Other symbolic model-checking frameworks - CEGAR
Thank You!