Configuring timing parameters to ensure opacity

Étienne André

Université Sorbonne Paris Nord, LIPN, CNRS UMR 7030, F-93430 Villetaneuse, France
Based on joint works with Engel Lefaucheux, Didier Lime, Dylan Marinho and Sun Jun
Context: side-channel attacks

- Threats to a system using non-algorithmic weaknesses

Example

- Number of pizzas (and order time) ordered by the white house prior to major war announcements

\(^1\text{http://home.xnet.com/~warinner/pizzacites.html}\)
Context: side-channel attacks

- Threats to a system using non-algorithmic weaknesses
  - Cache attack
  - Electromagnetic attacks
  - Power attacks
  - Acoustic attacks
  - Timing attacks
  - etc.

- Example
  - Number of pizzas (and order time) ordered by the white house prior to major war announcements

\(^1\)http://home.xnet.com/~warinner/pizzacites.html
Context: side-channel attacks

- Threats to a system using non-algorithmic weaknesses
  - Cache attack
  - Electromagnetic attacks
  - Power attacks
  - Acoustic attacks
  - Timing attacks
  - etc.

- Example
  - Number of pizzas (and order time) ordered by the white house prior to major war announcements

\(^1\)http://home.xnet.com/~warinner/pizzacites.html
Context: timing attacks

- Principle: deduce private information from timing data (execution time)

Issues:

- May depend on the implementation (or, even worse, be introduced by the compiler)

- A potential solution: make the program last always its maximum execution time
  
  Drawback: loss of efficiency

~ Non-trivial problem
A simple example of timing attack

```python
# input pwd : Real password
# input attempt: Tentative password
for i = 0 to min(len(pwd, len(attempt)) - 1 do
    if pwd[i] =/= attempt[i] then
        return false
    done
return true
```
A simple example of timing attack

```plaintext
# input pwd : Real password
# input attempt: Tentative password

for i = 0 to min(len(pwd), len(attempt)) - 1 do
  if pwd[i] /= attempt[i] then
    return false
  end
done

return true
```

Execution time:

```
pwd        chou doufufu
attempt    cheese
```
A simple example of timing attack

```plaintext
# input pwd : Real password
# input attempt: Tentative password

for i = 0 to min(len(pwd), len(attempt)) - 1 do
    if pwd[i] /= attempt[i] then
        return false
    done

return true
```

pwd: chooudoufuu
attempt: cheesse

Execution time: $\epsilon$
A simple example of timing attack

```python
# input pwd : Real password
# input attempt: Tentative password

for i = 0 to min(len(pwd), len(attempt)) - 1 do
    if pwd[i] != attempt[i] then
        return false

return true
```

 Execution time: $\epsilon + \epsilon$
A simple example of timing attack

```plaintext
# input pwd : Real password
# input attempt: Tentative password

for i = 0 to min(len(pwd), len(attempt)) - 1 do
    if pwd[i] /= attempt[i] then
        return false
    done

return true
```

```
pwd
choou doufu

attempt
cheese
```

Execution time: $\epsilon + \epsilon + \epsilon$
A simple example of timing attack

```python
# input pwd : Real password
# input attempt: Tentative password
for i = 0 to min(len(pwd), len(attempt)) - 1 do
    if pwd[i] != attempt[i] then
        return false
    done
return true
```

Execution time: $\epsilon + \epsilon + \epsilon$

**Problem:** The execution time is proportional to the number of consecutive correct characters from the beginning of attempt
Outline

1 Problems

2 Timed automata

3 Execution-time opacity computation

4 Execution-time opacity synthesis

5 Experiments

6 Expiring opacity

7 Conclusion and perspectives
Our attacker model

Attacker capabilities

- Has access to the model (white box)
- Can only observe the execution time

Attacker goal

- Wants to deduce some private information based on these observations
Informal problems

Question: can we exhibit secure execution times?

Execution-time opacity computation

Compute execution times for which the attacker cannot deduce private information by observing the execution time
## Informal problems

**Question:** can we exhibit *secure execution times*?

<table>
<thead>
<tr>
<th>Execution-time opacity computation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Compute</strong> <em>execution times</em> for which the attacker cannot deduce private information by observing the execution time</td>
</tr>
</tbody>
</table>

**Question:** can we decide whether *all* execution times are secure?

<table>
<thead>
<tr>
<th>Full execution-time opacity</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Decide</strong> whether the attacker cannot deduce private information, for <em>all</em> execution times</td>
</tr>
</tbody>
</table>
Informal problems: configuration

Question: can we also configure internal timing constants to make the system resisting to timing attacks?

Execution-time opacity synthesis

Exhibit execution times and internal timing constants for which the attacker cannot deduce private information by observing the execution time
Outline

1 Problems

2 Timed automata

3 Execution-time opacity computation

4 Execution-time opacity synthesis

5 Experiments

6 Expiring opacity

7 Conclusion and perspectives
Timed automaton (TA)

- Finite-state automaton (sets of locations)

---

Timed automaton (TA)

- Finite-state automaton (sets of locations and actions)

---

warm-up

Timed automaton (TA)

- Finite-state automaton (sets of locations and actions) augmented with a set $X$ of clocks
- Real-valued variables evolving linearly at the same rate

---


---

![Diagram of a timed automaton with states and transitions labeled as idle, adding sugar, delivering coffee. Transitions are marked with actions like press? and coffee!.]
Timed automaton (TA)

- Finite-state automaton (sets of locations and actions) augmented with a set $X$ of clocks
  - Real-valued variables evolving linearly at the same rate
  - Can be compared to integer constants in invariants

Features

- Location invariant: property to be verified to stay at a location

---


---

Etienne André (Université Sorbonne Paris Nord)  Configuring timing parameters to ensure opacity

23rd April 2023
Timed automaton (TA)

- Finite-state automaton (sets of locations and actions) augmented with a set $X$ of clocks
  - Real-valued variables evolving linearly at the same rate
  - Can be compared to integer constants in invariants and guards

Features

- Location invariant: property to be verified to stay at a location
- Transition guard: property to be verified to enable a transition

![Diagram of a timed automaton]

---

Timed automaton (TA)

- Finite-state automaton (sets of locations and actions) augmented with a set $X$ of clocks
  - Real-valued variables evolving linearly at the same rate
  - Can be compared to integer constants in invariants and guards

Features

- Location invariant: property to be verified to stay at a location
- Transition guard: property to be verified to enable a transition
- Clock reset: some of the clocks can be set to 0 along transitions

Examples of executions

- \( y \leq 8 \)
  - Coffee!

- \( y = 5 \)
  - Cup!

- \( x \geq 1 \)
  - Press?
  - \( x \leftarrow 0 \)

- \( y \leq 5 \)
  - \( x \leftarrow 0 \)
  - Adding sugar

- \( x \leftarrow 0 \)
  - Idle

Example of concrete run for the coffee machine:

- Coffee with 2 doses of sugar
Examples of executions

- Example of concrete run for the coffee machine
  - Coffee with 2 doses of sugar
    - $x = 0$
    - $y = 0$
Examples of executions

- \( y \leq 5 \) to \( y \leq 8 \)
- \( x \leftarrow 0 \)
- \( y \leftarrow 0 \)
- \( x \geq 1 \)
- \( x \leftarrow 0 \)
- \( y = 5 \) coffee!
- \( y = 8 \) coffee!
- \( y = 5 \) cup!

Example of concrete run for the coffee machine

- Coffee with 2 doses of sugar
  - \( x = 0 \)
  - \( y = 0 \)
Examples of executions

Example of concrete run for the coffee machine

Coffee with 2 doses of sugar
Examples of executions

Example of concrete run for the coffee machine

- Coffee with 2 doses of sugar

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1.5</td>
<td>1.5</td>
</tr>
<tr>
<td>1.5</td>
<td>1.5</td>
</tr>
</tbody>
</table>
Examples of executions

- $y = 8$
  - coffee!

- $y \leq 5$
  - $y = 5$
  - cup!

- $x \geq 1$
  - press?
  - $x \leftarrow 0$

- $y = 8$
  - idle
  - adding sugar
  - delivering coffee

**Example of concrete run for the coffee machine**

**Coffee with 2 doses of sugar**

<table>
<thead>
<tr>
<th>Time</th>
<th>Action</th>
<th>$x$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>press?</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>1.5</td>
<td>press?</td>
<td>1.5</td>
<td>1.5</td>
</tr>
<tr>
<td>2.7</td>
<td></td>
<td>2.7</td>
<td>4.2</td>
</tr>
</tbody>
</table>
Examples of executions

- $y = 8$
  - coffee!
- $y \leq 5$
  - 
- $y = 5$
  - cup!
- $x \geq 1$
  - press?
  - $x \leftarrow 0$
  - $y \leftarrow 0$
- $x \leftarrow 0$

Example of concrete run for the coffee machine

- Coffee with 2 doses of sugar

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1.5</td>
<td>0</td>
<td>1.5</td>
</tr>
<tr>
<td>0</td>
<td>1.5</td>
<td>1.5</td>
</tr>
<tr>
<td>2.7</td>
<td>0</td>
<td>4.2</td>
</tr>
<tr>
<td>0</td>
<td>4.2</td>
<td>4.2</td>
</tr>
</tbody>
</table>
Examples of executions

Example of concrete run for the coffee machine

Coffee with 2 doses of sugar

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$x$</th>
<th>$y$</th>
<th>$x$</th>
<th>$y$</th>
<th>$x$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.5</td>
<td>0</td>
<td>1.5</td>
<td>0</td>
<td>0.8</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1.5</td>
<td>0</td>
<td>1.5</td>
<td>2.7</td>
<td>4.2</td>
<td>5</td>
</tr>
</tbody>
</table>
Examples of executions

y ≤ 5
y ≤ 8

Example of concrete run for the coffee machine

Coffee with 2 doses of sugar

x = 0 0 1.5 0 2.7 0 0.8 0.8 0.8
y = 0 0 1.5 1.5 4.2 4.2 5 5

Examples of executions

Example of concrete run for the coffee machine

Coffee with 2 doses of sugar
Examples of executions

Example of concrete run for the coffee machine

- Coffee with 2 doses of sugar

Consider the following states and actions:

Initial state:
- $x = 0$
- $y = 0$

Action: Press button
- $x \leftarrow 0$
- $y \leftarrow 0$

State: $y \leq 5$

Action: Press button
- $x \Rightarrow 1$

State: $y = 5$

Action: Add sugar
- $x \leftarrow 0$

State: $y \leq 8$

Action: Deliver coffee
- $x \Rightarrow 0$

Final state:
- $x = 0$
- $y = 8$

Note: The diagram illustrates the sequence of actions and states, showing the flow from idle to adding sugar, then delivering coffee, and finally delivering coffee again.
Outline

1 Problems
2 Timed automata
3 Execution-time opacity computation
4 Execution-time opacity synthesis
5 Experiments
6 Expiring opacity
7 Conclusion and perspectives
Formalization

Hypotheses:

- A start location $\ell_0$ and an end location $\ell_f$
- A special private location $\ell_{priv}$

Definition (execution-time opacity [And+22])

The system is ET-opaque if there exist two runs to $\ell_f$ of duration $d$

1. one visiting $\ell_{priv}$, and
2. one not visiting $\ell_{priv}$

Weak and full ET-opacity

**Definition (weak execution-time opacity)**

For each duration $d$,
There exists a run of duration $d$ visiting $l_{priv}$

$\Rightarrow$
There exists a run of duration $d$ not visiting $l_{priv}$

That is: private durations $\subseteq$ public durations

**Definition (full execution-time opacity)**

For each duration $d$,
There exists a run of duration $d$ visiting $l_{priv}$

$\Leftrightarrow$
There exists a run of duration $d$ not visiting $l_{priv}$

That is: private durations $=$ public durations
Illustrating weak and full execution-time opacity

There exist (at least) two runs of duration $d = 2$:

- Visiting $\ell_{\text{priv}} \ell_0 \ell_{\text{priv}} \ell_{\text{priv}} \ell_f$
- Not visiting $\ell_0 \ell_{\text{priv}} \ell_f$

Generally:

- Private execution times are $[1, 2.5]$.
- Public execution times are $[0, 3]$.

The system is weakly ET-opaque if $x \geq 1$ and $x \leq 2.5$.

The system is not fully ET-opaque if $x \leq 3$. 
Illustrating weak and full execution-time opacity

There exist (at least) two runs of duration $d = 2$:
Illustrating weak and full execution-time opacity

There exist (at least) two runs of duration \(d = 2\):

\[ x \geq 1, \quad x \leq 2.5, \quad x \leq 3\]
Illustrating weak and full execution-time opacity

\[ x \geq 1 \]
\[ b \]
\[ \ell_{priv} \]

\[ x \leq 2.5 \]
\[ c \]
\[ \ell_f \]

\[ x \leq 3 \]
\[ a \]
\[ \ell_0 \]

There exist (at least) two runs of duration \( d = 2 \):

- visiting \( \ell_{priv} \)

\[ \ell_0 \xrightarrow{1} \ell_0 \]
Illustrating weak and full execution-time opacity

There exist (at least) two runs of duration $d = 2$:

- Visiting $\ell_{priv}$
Illustrating weak and full execution-time opacity

- There exist (at least) two runs of duration $d = 2$:
  - visiting $\ell_{\text{priv}}$

  ![Diagram](image.png)
Illustrating weak and full execution-time opacity

There exist (at least) two runs of duration $d = 2$:

- visiting $\ell_{\text{priv}}$

```
\[
\begin{align*}
\ell_0 & \xrightarrow{1} \ell_0 & \ell_{\text{priv}} & \xrightarrow{1} \ell_{\text{priv}} & \ell_{\text{priv}} & \xrightarrow{c} \ell_f \\
\text{(visiting $\ell_{\text{priv}}$)}
\end{align*}
\]```
Illustrating weak and full execution-time opacity

There exist (at least) two runs of duration $d = 2$:

- Visiting $\ell_{\text{priv}}$
  
  - $\ell_0 \xrightarrow{1} \ell_0 \xrightarrow{b} \ell_{\text{priv}} \xrightarrow{1} \ell_{\text{priv}} \xrightarrow{c} \ell_f$
  
- Not visiting $\ell_{\text{priv}}$
  
  - $\ell_0 \xrightarrow{} \ell_0$
Illustrating weak and full execution-time opacity

There exist (at least) two runs of duration $d = 2$:

- visiting $\ell_{\text{priv}}$
  - $\ell_0 \xrightarrow{1} \ell_0 \xrightarrow{b} \ell_{\text{priv}} \xrightarrow{1} \ell_{\text{priv}} \xrightarrow{c} \ell_f$
- not visiting $\ell_{\text{priv}}$
  - $\ell_0 \xrightarrow{2} \ell_0$

$\ell_0 \xrightarrow{x \geq 1} \ell_{\text{priv}} \xrightarrow{x \leq 2.5} \ell_f$

Generally:
- private execution times are $[1, 2.5]$.
- public execution times are $[0, 3]$.
- private durations $\subseteq$ public durations.

The system is weakly ET-opaque.

The system is not fully ET-opaque.
Illustrating weak and full execution-time opacity

There exist (at least) two runs of duration $d = 2$: 

- **visiting $\ell_{priv}$**

- **not visiting $\ell_{priv}$**
Illustrating weak and full execution-time opacity

There exist (at least) two runs of duration \( d = 2 \):

The system is ET-opaque for a duration \( d = 2 \)
Illustrating weak and full execution-time opacity

There exist (at least) two runs of duration $d$ for all durations $d \in [1, 2.5]$:

- **visiting $l_{priv}$**
  - $l_0 \xrightarrow{1} l_0 \xrightarrow{b} l_{priv} \xrightarrow{d-1} l_{priv} \xrightarrow{c} l_f$

- **not visiting $l_{priv}$**
  - $l_0 \xrightarrow{d} l_0 \xrightarrow{a} l_f$

The system is **ET-opaque for all durations in $[1, 2.5]$**
Illustrating weak and full execution-time opacity

- There exist (at least) two runs of duration $d$ for all durations $d \in [1, 2.5]$
Illustrating weak and full execution-time opacity

- There exist (at least) two runs of duration $d$ for all durations $d \in [1, 2.5]$

- Generally:
  - **private** execution times are $[1, 2.5]$
  - **public** execution times are $[0, 3]$
Illustrating weak and full execution-time opacity

There exist \((\text{at least})\) two runs of duration \(d\) for all durations \(d \in [1, 2.5]\)

Generally:
- private execution times are \([1, 2.5]\)
- public execution times are \([0, 3]\)
- private durations \(\subseteq\) public durations
Illustrating weak and full execution-time opacity

There exist (at least) two runs of duration $d$ for all durations $d \in [1, 2.5]$.

Generally:
- Private execution times are $[1, 2.5]$.
- Public execution times are $[0, 3]$.
- Private durations $\subseteq$ public durations.

The system is weakly ET-opaque.
Illustrating weak and full execution-time opacity

There exist \((at least)\) two runs of duration \(d\) for all durations \(d \in [1, 2.5]\)

Generally:
- private execution times are \([1, 2.5]\)
- public execution times are \([0, 3]\)
- private durations \(\subseteq\) public durations

The system is weakly ET-opaque

- private durations \(\neq\) public durations
Illustrating weak and full execution-time opacity

- There exist (at least) two runs of duration $d$ for all durations $d \in [1, 2.5]$

- Generally:
  - private execution times are $[1, 2.5]$
  - public execution times are $[0, 3]$
  - private durations $\subseteq$ public durations

The system is weakly ET-opaque

- private durations $\neq$ public durations

The system is not fully ET-opaque
Execution-time opacity computation can be achieved

Theorem (Computability of execution-time opacity)

The answer to the execution-time opacity computation problem for timed automata can be effectively computed in the form of a finite union of intervals.

Proof: based on the region graph (see [And+22])

Exact complexity: unproved (EXPSPACE upper bound proved, but exponential hardness seems likely)

Remark: to be put in perspective with [Cas09]

- undecidability for a less expressive class, for a stronger notion of opacity

---


Full and weak execution-time opacity

Theorem (Full execution-time opacity [And+22])

Full execution-time opacity is **decidable** for timed automata

Theorem (Weak execution-time opacity [ALM23])

Weak execution-time opacity is **decidable** for timed automata


Outline

1. Problems
2. Timed automata
3. Execution-time opacity computation
4. Execution-time opacity synthesis
5. Experiments
6. Expiring opacity
7. Conclusion and perspectives
Towards configurable opaque systems...

Problems

- Can we configure some timing constants to guarantee opacity?

- Verification for one set of constants does not usually guarantee the correctness for other values

- Robustness [BMS13]: What happens if 50 is implemented with 49.99?

---

Towards configurable opaque systems...

Problems

- Can we configure some timing constants to guarantee opacity?

- Verification for one set of constants does not usually guarantee the correctness for other values

Robustness \[\text{BMS13}\]: What happens if 50 is implemented with 49.99?

A solution:

- Parameter synthesis
  - Consider that timing constants are unknown constants (parameters)

---

Parametric Timed Automaton (PTA)

- Timed automaton (sets of locations, actions and clocks)

---

Parametric Timed Automaton (PTA)

- Timed automaton (sets of locations, actions and clocks) augmented with a set $P$ of parameters
- Unknown constants compared to a clock in guards and invariants

$y \leq p_2$
$y = p_3$
coffee!

$y \leq \delta$

$y = p_2$
cup!

press?
$x \leftarrow 0$
y \leftarrow 0

$y = p_3$
coffee!

press?
$x \leftarrow 0$
y \leftarrow 0

$y \leq \delta$

$y = p_2$
cup!

configuring timing parameters to ensure opacity

## Two classes of parametric problems

<table>
<thead>
<tr>
<th>Emptiness problem</th>
<th>Synthesis problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is the set of <strong>parameter valuations</strong> ensuring the property <strong>empty</strong>?</td>
<td>Synthesize <em>all the</em>* parameter valuations** ensuring the property</td>
</tr>
</tbody>
</table>
Two classes of parametric problems

Emptiness problem
Is the set of parameter valuations ensuring the property empty?

Synthesis problem
Synthesize all the parameter valuations ensuring the property

4 concrete opacity problems:
- Decision problems: weak (resp. full) execution-time opacity emptiness
- Synthesis problems: weak (resp. full) execution-time opacity synthesis
Example

\[ \ell \leq 3 \quad \text{Private } [p_1, p_2] \]
\[ \ell \geq 2 \quad \text{Public } [0, 3] \]

<table>
<thead>
<tr>
<th>ET-opacity</th>
<th>Emptiness</th>
<th>Synthesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>weak</td>
<td></td>
<td></td>
</tr>
<tr>
<td>full</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

The diagram illustrates the states and transitions of a system with timing parameters. The states are labeled as $l_0$, $l_{priv}$, and $l_f$. The transitions are labeled with $a$, $b$, and $c$. The conditions $x \geq p_1$, $x \leq p_2$, and $x \leq 3$ are shown for the transitions.

The table summarizes the ET-opacity, Emptiness, and Synthesis:

<table>
<thead>
<tr>
<th>ET-opacity</th>
<th>Emptiness</th>
<th>Synthesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>weak</td>
<td>$\times (\exists v)$</td>
<td></td>
</tr>
<tr>
<td>full</td>
<td>$\times (\exists v)$</td>
<td></td>
</tr>
</tbody>
</table>

The private states are $[p_1, p_2]$ and the public states are $[0, 3]$.
Example

\[
x \geq p_1 \\
b \\
\ell_0 \xrightarrow{a} \ell_{\text{priv}} \xrightarrow{c} \ell_f \xleftarrow{x \leq p_2}
\]

Private \([p_1, p_2]\)
Public \([0, 3]\)

<table>
<thead>
<tr>
<th>ET-opacity</th>
<th>Emptiness</th>
<th>Synthesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>weak</td>
<td>(\times (\exists v))</td>
<td>(0 \leq p_1 \land p_2 \leq 3) (\land p_1 \leq p_2)</td>
</tr>
<tr>
<td>full</td>
<td>(\times (\exists v))</td>
<td></td>
</tr>
</tbody>
</table>
These valuations give a way to configure the system parameters to formally guarantee execution-time opacity.
Outline

1. Problems
2. Timed automata
3. Execution-time opacity computation
4. Execution-time opacity synthesis
   - Theory: undecidability
   - A practical approach
5. Experiments
6. Expiring opacity
7. Conclusion and perspectives
Execution-time opacity synthesis is (very) difficult

**Theorem (Undecidability of execution-time opacity-emptiness)**

*The mere existence of a parameter valuation such that there exists a duration for which execution-time opacity is achieved is undecidable.*

Proof idea: reduction from reachability-emptiness for PTAs [AHV93]

![Diagram](image)

**Remark:** decidable subclass

(see [And+22])


Undecidability

Theorem (Full execution-time opacity emptiness [And+22])

Full execution-time opacity emptiness is undecidable for parametric timed automata, and even for the subclass of L/U parametric timed automata.
Undecidability

Theorem (Full execution-time opacity emptiness [And+22]):

*Full execution-time opacity emptiness is undecidable for parametric timed automata, and even for the subclass of L/U parametric timed automata.*

In the following, we adopt a “best-effort” approach

- Approach not guaranteed to terminate in theory
Outline

1. Problems
2. Timed automata
3. Execution-time opacity computation
4. Execution-time opacity synthesis
   - Theory: undecidability
   - A practical approach
5. Experiments
6. Expiring opacity
7. Conclusion and perspectives
Computing execution-time opacity via reachability synthesis

Big picture:

- **Formalism:** parametric timed automata

- **Our approach:**
  1. Perform a (mild) transformation of the PTA
  2. Perform self-composition
  3. Apply parametric timed model checking (reachability-synthesis)

- **Tool support:** IMITATOR

Our transformation of the PTA in 4 overlays

1. Add a Boolean flag \( b \) to remember whether \( \ell_{\text{priv}} \) was visited.
2. Add a synchronization action \( \text{finish} \) on any transition to \( \ell_f \).
3. Measure the (parametric) duration to \( \ell_f \) thanks to a new clock \( x_{\text{abs}} \) and a new parameter \( d \).
4. Perform self-composition (i.e., a synchronization on shared actions of the PTA with a copy of itself).
Our transformation of the PTA in 4 overlays

1. Add a Boolean flag $b$ to remember whether $\ell_{priv}$ was visited
Our transformation of the PTA in 4 overlays

1. Add a Boolean flag \( b \) to remember whether \( \ell_{\text{priv}} \) was visited
2. Add a synchronization action \textit{finish} on any transition to \( \ell_f \)
Our transformation of the PTA in 4 overlays

1. Add a Boolean flag $b$ to remember whether $\ell_{\text{priv}}$ was visited
2. Add a synchronization action `finish` on any transition to $\ell_f$
3. Measure the (parametric) duration to $\ell_f$ thanks to a new clock $x_{\text{abs}}$ and a new parameter $d$
Our transformation of the PTA in 4 overlays

1. Add a Boolean flag $b$ to remember whether $\ell_{priv}$ was visited
2. Add a synchronization action `finish` on any transition to $\ell_f$
3. Measure the (parametric) duration to $\ell_f$ thanks to a new clock $x_{abs}$ and a new parameter $d$
4. Perform self-composition (i.e., a synchronization on shared actions of the PTA with a copy of itself)
Applying reachability-synthesis

We then synthesize all parameter valuations (including $d$) for which the following discrete state is reachable:

- the original automaton is in $\ell_f$ with $b = \text{true}$
- the copy automaton is in $\ell_f$ with $b' = \text{false}$
Applying reachability-synthesis

We then synthesize all parameter valuations (including $d$) for which the following discrete state is reachable:

- the original automaton is in $\ell_f$ with $b = \text{true}$
- the copy automaton is in $\ell_f$ with $b' = \text{false}$

Intuition:

- for the same duration (thanks to the synchronization on finish), we can reach $\ell_f$ “both” after visiting $\ell_{priv}$ (i.e., $b = \text{true}$) and not visiting $\ell_{priv}$ (i.e., $b = \text{false}$)
Outline

1. Problems
2. Timed automata
3. Execution-time opacity computation
4. Execution-time opacity synthesis
5. Experiments
6. Expiring opacity
7. Conclusion and perspectives
Experimental environment

Algorithms

1. Full execution-time opacity: “for a non-parametric TA, is the TA opaque for all execution times?”
2. Execution-time opacity synthesis: “for a PTA, synthesize some parameter valuations and execution times ensuring execution-time opacity”

Benchmarks

- Common PTA benchmarks
- Library of Java programs
  - Manually translated to PTAs
  - User-input variables translated to (non-timing) parameters (supported by IMITATOR)

See experiments at doi.org/10.5281/zenodo.3251141 and imitator.fr/static/ATVA19/

Outline

1 Problems

2 Timed automata

3 Execution-time opacity computation

4 Execution-time opacity synthesis

5 Experiments
   - IMITATOR in a nutshell
   - Non-parametric execution-time opacity computation
   - Parametric execution-time opacity synthesis

6 Expiring opacity

7 Conclusion and perspectives
Parameter synthesis using IMITATOR

IMITATOR: a parametric timed model checker

An extended PTA

A property

#synth AGnot(loc[researcher] = coffee)

The set of parameter valuations is symbolic

- Symbolic: finite set of linear constraints (polyhedra)
Parameter synthesis using IMITATOR

IMITATOR: a **parametric** timed model checker

### An extended PTA

### A property

\[
\text{#synth } \neg \text{not(loc[researcher] = coffee)}
\]

---

The set of **parameter valuations** is **symbolic**

- Symbolic: finite set of linear constraints (polyhedra)

- Two categories of properties
  - Synthesis: “(try to) synthesize all valuations for which the property holds”
  - Exhibition: “(try to) synthesize at least one valuation for which the property holds”
Distribution

Free and open source software: Available under the GNU-GPL license

Distribution:

- Binaries available for Linux platforms (no dependency, no install)
- Docker version
- Integrated as a virtual machine
- Comes with a user manual and an extensive benchmarks library [AMP21]

Try it!

www.imitator.fr

Outline

1 Problems

2 Timed automata

3 Execution-time opacity computation

4 Execution-time opacity synthesis

5 Experiments
   - IMITATOR in a nutshell
   - Non-parametric execution-time opacity computation
   - Parametric execution-time opacity synthesis

6 Expiring opacity

7 Conclusion and perspectives
### Experiments: (non-parametric) execution-time opacity

<table>
<thead>
<tr>
<th>Model</th>
<th>Name</th>
<th>Transf. PTA</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>(</td>
<td>A</td>
</tr>
<tr>
<td>[VNN18]</td>
<td>Fig. 5, [VNN18]</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>[GMR07]</td>
<td>Fig. 1b, [GMR07]</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>[GMR07]</td>
<td>Fig. 2a, [GMR07]</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>[GMR07]</td>
<td>Fig. 2b, [GMR07]</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Web privacy problem [Ben+15]</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Coffee</td>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Fischer-HSRV02</td>
<td>3</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>STAC:1:n</td>
<td>2</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>STAC:1:v</td>
<td>2</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>STAC:3:n</td>
<td>2</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>STAC:3:v</td>
<td>2</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>STAC:4:n</td>
<td>2</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>STAC:4:v</td>
<td>2</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>STAC:5:n</td>
<td>2</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>STAC:11A:v</td>
<td>2</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>STAC:11B:v</td>
<td>2</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>STAC:12c:v</td>
<td>2</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>STAC:12e:n</td>
<td>2</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>STAC:12e:v</td>
<td>2</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>STAC:14:n</td>
<td>2</td>
<td>3</td>
<td>8</td>
</tr>
</tbody>
</table>

Times are rounded.

\(\times\) = not vulnerable; 
\((\checkmark)\) = vulnerable, can be repaired; 
\(\sqrt{\checkmark}\) = vulnerable, cannot be repaired

---


Outline

1 Problems
2 Timed automata
3 Execution-time opacity computation
4 Execution-time opacity synthesis
5 Experiments
   • IMITATOR in a nutshell
   • Non-parametric execution-time opacity computation
   • Parametric execution-time opacity synthesis
6 Expiring opacity
7 Conclusion and perspectives
# Experiments: (parametric) execution-time opacity synthesis

<table>
<thead>
<tr>
<th>Model</th>
<th>Transf. PTA</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td>$</td>
<td>A</td>
</tr>
<tr>
<td>Fig. 5, [VNN18]</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Fig. 1b, [GMR07]</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Fig. 2, [GMR07]</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Web privacy problem [Ben+15]</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Coffee</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Fischer-HSRVO2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>STAC:3:v</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

$K$ = some valuations make the system non-vulnerable; $⊤$ = all valuations make the system non-vulnerable

---


Outline

1 Problems
2 Timed automata
3 Execution-time opacity computation
4 Execution-time opacity synthesis
5 Experiments
6 Expiring opacity
7 Conclusion and perspectives
What if the secret can expire?

Motivation: cache

- Deducing that some information was in the cache a long time ago might be useless
- Opacity with an expiration date [Amm+21]

### Expiring execution-time opacity

<table>
<thead>
<tr>
<th>Secret runs</th>
<th>Non-secret runs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runs visiting the private location (＝ private runs)</td>
<td>Runs not visiting the private location (＝ public runs)</td>
</tr>
<tr>
<td>expiring-ET-opacity</td>
<td></td>
</tr>
<tr>
<td>Private runs with ( \ell_{priv} ) entered ( \leq \Delta ) before the system completion</td>
<td>(i) Public runs and (ii) Private runs with ( \ell_{priv} ) entered ( &gt; \Delta ) before the system completion</td>
</tr>
</tbody>
</table>
Example

\[ x \geq 1 \quad \text{ET-opacity} \quad x \leq 2.5 \]

\[
\begin{array}{c}
\ell \rightarrow \ell_{0}
\end{array}
\]

\[
\begin{array}{c}
x \leq 3
\end{array}
\]

\[
\begin{array}{c}
\ell_{0} \rightarrow \ell_{priv}
\end{array}
\]

\[
\begin{array}{c}
c
\end{array}
\]

\[
\begin{array}{c}
\ell_{priv} \rightarrow \ell_f
\end{array}
\]

<table>
<thead>
<tr>
<th>ET-opacity</th>
<th>Secret</th>
<th>Non secret</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>weak</td>
<td>[1, 2.5]</td>
<td>[0, 3]</td>
<td>✓</td>
</tr>
<tr>
<td>full</td>
<td></td>
<td></td>
<td>×</td>
</tr>
<tr>
<td>weak-exp.</td>
<td>[1, 2.5]</td>
<td>(2, 2.5] ∪ [0, 3]</td>
<td>✓</td>
</tr>
<tr>
<td>full-exp.</td>
<td></td>
<td></td>
<td>×</td>
</tr>
</tbody>
</table>
Example

\[
x \geq 1 \quad \quad x \leq 2.5
\]

<table>
<thead>
<tr>
<th>ET-opacity</th>
<th>Secret</th>
<th>Non secret</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>weak</td>
<td>[1, 2.5]</td>
<td>[0, 3]</td>
<td>✓</td>
</tr>
<tr>
<td>full</td>
<td></td>
<td></td>
<td>✗</td>
</tr>
</tbody>
</table>

| \(\Delta = 1\) | weak-exp. | [1, 2.5] | (2, 2.5] \(\cup\) [0, 3] | ✓ |
| full-exp.     | [1, 2.5]   | (2.25, 2.5] \(\cup\) [0, 3] | ✗ |

| \(\Delta = 1.25\) | weak-exp. | [1, 2.5] | (2.25, 2.5] \(\cup\) [0, 3] | ✓ |
| full-exp.     | [1, 2.5]   | (2.25, 2.5] \(\cup\) [0, 3] | ✗ |
Some results [ALM23]

😊 Given $\Delta$, we can decide whether a TA is weakly (resp. fully) ET-opaque

😊 We can synthesize all $\Delta$ for which a TA is weakly ET-opaque

😊 The synthesis of all $\Delta$ for full ET-opacity remains open

😊 The emptiness problems over parametric timed automata are undecidable
  - Even for the L/U-PTA subclass

Outline

1 Problems
2 Timed automata
3 Execution-time opacity computation
4 Execution-time opacity synthesis
5 Experiments
6 Expiring opacity
7 Conclusion and perspectives
Conclusion

Context: vulnerability by timing-attacks

- Attacker model: observability of the global execution time
- Goal: avoid leaking information on whether some discrete state has been visited

Several decision and computation problems studied for timed automata

- Mostly decidable

Extension to parametric timed automata

- Quickly undecidable
- One procedure for one synthesis problem
  - Toolkit: IMITATOR
  - Benchmarks: concurrent systems and Java programs
Perspectives

- **Theoretical open problems**
  - Synthesis of expiring dates for weak expiring opacity
  - Execution-time opacity *emptiness* remains open for 1 clock
  - Case of U-PTAs or L-PTAs

- **Algorithmic open problems**
  - Weak (resp. full) execution-time opacity synthesis

- **Automated translation of Java programs**
  - Our translation required non-trivial creativity
  - How to automate it?
  - Finer grain needed for “untimed” instructions: probabilistic timings?

- **Reconfiguring a non-opaque system**
  - “From PTA parameter tuning back to the original system”
  - In programs: using *Wait* or *Sleep*?

Bibliography
References I


References II


Licensing
Source of the graphics used

**Title: Piratey, vector version**
*Author: Gustavb*
*Source: https://commons.wikimedia.org/wiki/File:Piratey,_vector_version.svg*
*License: CC by-sa*

**Title: Expired**
*Author: RRZEicons*
*Source: https://commons.wikimedia.org/wiki/File:Expired.svg*
*License: CC by-sa*
License of this document

This presentation can be published, reused and modified under the terms of the license Creative Commons Attribution-ShareAlike 4.0 Unported (CC BY-SA 4.0)

(TEX source available on demand)

Authors: Étienne André and Dylan Marinho

creativecommons.org/licenses/by-sa/4.0/