FTSCS

December 7th, 2022
Auckland, New Zealand

strategFTO: Untimed control for timed opacity

Étienne André¹,², Shapagat Bolat², Engel Lefaucheux², Dylan Marinho²

¹ Université Sorbonne Paris Nord, LIPN, CNRS UMR 7030, F-93430 Villetaneuse, France
² Université de Lorraine, CNRS, Inria, LORIA, Nancy, France

These works are partially supported by the ANR-NRF research program ProMiS (ANR-19-CE25-0015) and the ANR research program BisoUS.
Context: timing attacks

▶ Principle: deduce **private information** from timing data (execution time)
▶ Attacker: only knows the **execution time** (and the model)
  → no information about the actions that happen, etc.

Issues:

▶ May depend on the **implementation** (or, even worse, be introduced by the compiler)
▶ A relatively trivial solution: make the program last always its maximum execution time
  Drawback: **loss of efficiency**

⇝ Non-trivial problem
A simple example of timing attack

```c
# 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

```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
```

pwd  chicken
attempts cheese

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

```c
# 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: chicken

attempt: cheese

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

- **Problem**: The execution time is proportional to the number of consecutive correct characters from the beginning of attempt.
Informal problem

Question: can we exhibit secure execution times?

Timed-opacity computation
Exhibit execution times for which it is not possible to infer information on the internal behavior
Outline

Preliminaries: Timed Opacity: Formalism and Preliminary results

Contribution: (Untimed) Control for timed opacity

Perspectives
Timed automaton (TA)

- Finite state automaton (sets of locations)

---

Timed automaton (TA)

- Finite state automaton (sets of locations and actions)

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

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

---

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

---

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

---

The most critical system: The coffee machine

Example of concrete run for the coffee machine

Coffee with 2 doses of sugar

idle
adding sugar
delivering coffee
The most critical system: The coffee machine

Example of concrete run for the coffee machine
- Coffee with 2 doses of sugar

\[
\begin{align*}
x &:= 0 \\
y &:= 0 \\
x &:= 0 \\
y &:= 5 \\
x &\geq 1 \\
y &\leq 5 \\
y &\leq 8 \\
\end{align*}
\]
The most critical system: The coffee machine

Press?

\[ y = 8 \]
coffee!

\[ y \leq 5 \]

cup!

\[ y = 5 \]

Example of concrete run for the coffee machine

Coffee with 2 doses of sugar

x := 0
y := 0

x := 0

x \geq 1

idle

adding sugar

delivering coffee
The most critical system: The coffee machine

Example of concrete run for the coffee machine

Coffee with 2 doses of sugar
The most critical system: The coffee machine

Example of concrete run for the coffee machine

Coffee with 2 doses of sugar
The most critical system: The coffee machine

Example of concrete run for the coffee machine

Coffee with 2 doses of sugar
The most critical system: The coffee machine

Example of concrete run for the coffee machine

Coffee with 2 doses of sugar
The most critical system: The coffee machine

Example of concrete run for the coffee machine

Coffee with 2 doses of sugar

idle
adding sugar
delivering coffee
The most critical system: The coffee machine

Example of concrete run for the coffee machine

Coffee with 2 doses of sugar

idle
adding sugar
delivering coffee
The most critical system: The coffee machine

Example of concrete run for the coffee machine

Coffee with 2 doses of sugar
The most critical system: The coffee machine

Example of concrete run for the coffee machine

Coffee with 2 doses of sugar
Outline

Preliminaries: Timed Opacity: Formalism and Preliminary results
  Timed Opacity formalization
    Computation problem and results

Contribution: (Untimed) Control for timed opacity

Perspectives
Formalization

Hypotheses:

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

Definition (timed opacity)

The system is timed-opaque w.r.t. $\ell_{priv}$ on the way to $\ell_f$ for a duration $d$ if there exist at least two runs to $\ell_f$ of duration $d$

1. one passing by $\ell_{priv}$
2. one *not* passing by $\ell_{priv}$

---

Example

There exist (at least) two runs of duration $d = 2$: visiting $l_{\text{priv}}$ on the way to $l_f$.

But there exists a run of duration 1.5 reaching $l_f$ and visiting $l_{\text{priv}}$.

There exists no run of duration 1.5 reaching $l_f$ and not visiting $l_{\text{priv}}$.

We say that the system is timed-opaque w.r.t. $l_{\text{priv}}$ on the way to $l_f$.

We say that the system is not fully timed-opaque w.r.t. $l_{\text{priv}}$ on the way to $l_f$. 

$a \geq 2$

$x \leq 3$

$x \geq 1$

$x \leq 3$
Example

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

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

- Visiting \( l_{\text{priv}} \)
- Not visiting \( l_{\text{priv}} \)

We say that the system is timed-opaque w.r.t. \( l_{\text{priv}} \) on the way to \( l_f \).
Example

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

- Visiting \(l_{\text{priv}}\)

\[\xrightarrow{1} l_0 \xrightarrow{1} l_0\]
Example

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

- One run visiting $l_{priv}$.
- Another run not visiting $l_{priv}$.

We say that the system is timed-opaque w.r.t. $l_{priv}$ on the way to $l_f$. But there exists a run of duration 1.5 reaching $l_f$ and visiting $l_{priv}$.

There exists no run of duration 1.5 reaching $l_f$ and not visiting $l_{priv}$.

We say that the system is not fully timed-opaque w.r.t. $l_{priv}$ on the way to $l_f$. 
Example

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

- Visiting $\ell_{\text{priv}}$
  - $l_0 \xrightarrow{1} l_0 \xrightarrow{b} l_{\text{priv}} \xrightarrow{1} l_{\text{priv}}$

We say that the system is timed-opaque w.r.t. $\ell_{\text{priv}}$ on the way to $\ell_f$ for $a$.

But there exists a run of duration $1.5$ reaching $\ell_f$ and visiting $\ell_{\text{priv}}$.

There exists no run of duration $1.5$ reaching $\ell_f$ and not visiting $\ell_{\text{priv}}$.

We say that the system is not fully timed-opaque w.r.t. $\ell_{\text{priv}}$ on the way to $\ell_f$. 

Example

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

- Visiting $\ell_{\text{priv}}$
  - Run 1: $\ell_0 \xrightarrow{1} \ell_0 \xrightarrow{b} \ell_{\text{priv}} \xrightarrow{1} \ell_{\text{priv}}$}

We say that the system is timed-opaque w.r.t. $\ell_{\text{priv}}$ on the way to $\ell_f$.

But there exists a run of duration 1.5 reaching $\ell_f$ and visiting $\ell_{\text{priv}}$.

There exists no run of duration 1.5 reaching $\ell_f$ and not visiting $\ell_{\text{priv}}$.

We say that the system is not fully timed-opaque w.r.t. $\ell_{\text{priv}}$ on the way to $\ell_f$. 
Example

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

- Visiting $l_{\text{priv}}$:
  - $l_0 \to l_0 \to l_{\text{priv}} \to l_{\text{priv}} \to C$
  - $l_0 \to l_{\text{priv}} \to C$

- Not visiting $l_{\text{priv}}$:
  - $l_0 \to C$

We say that the system is timed-opaque w.r.t. $l_{\text{priv}}$ on the way to $l_f$. But there exists a run of duration 1.5 reaching $l_f$ and visiting $l_{\text{priv}}$.
There exist (at least) two runs of duration $d = 2$: 

- Visiting $\ell_{\text{priv}}$:
  - Run 1: $\ell_0 \xrightarrow{1} \ell_0 \xrightarrow{b} \ell_{\text{priv}} \xrightarrow{1} \ell_{\text{priv}} \xrightarrow{c} \ell_f$
  - Run 2: $\ell_0 \xrightarrow{2} \ell_0$

- Not visiting $\ell_{\text{priv}}$:
  - Run 1: $\ell_0 \xrightarrow{1} \ell_{\text{priv}} \xrightarrow{1} \ell_{\text{priv}} \xrightarrow{c} \ell_f$

We say that the system is timed-opaque w.r.t. $\ell_{\text{priv}}$ on the way to $\ell_f$.
Example

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

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

- Visiting $\ell_{\text{priv}}$
  - Run 1:
    - $\ell_0 \rightarrow \ell_0 \rightarrow \ell_{\text{priv}} \rightarrow \ell_{\text{priv}} \rightarrow \ell_f$
  - Run 2:
    - $\ell_0 \rightarrow \ell_0 \rightarrow \ell_{\text{priv}} \rightarrow \ell_{\text{priv}} \rightarrow \ell_f$

- Not visiting $\ell_{\text{priv}}$
  - Run 3:
    - $\ell_0 \rightarrow \ell_0 \rightarrow \ell_0 \rightarrow \ell_f$

We say that the system is timed-opaque w.r.t. $\ell_{\text{priv}}$ on the way to $\ell_f$ for a duration $d = 2$.
There exist (at least) two runs of duration \( d \) for all durations \( d \in [2,3] \):

- **visiting** \( \ell_{\text{priv}} \):
  - \( \ell_0 \rightarrow \ell_0 \rightarrow \ell_{\text{priv}} \rightarrow \ell_{\text{priv}} \rightarrow \ell_f \)
  - \( \ell_0 \rightarrow \ell_0 \rightarrow \ell_{\text{priv}} \rightarrow \ell_{\text{priv}} \rightarrow \ell_f \)

- **not visiting** \( \ell_{\text{priv}} \):
  - \( \ell_0 \rightarrow \ell_0 \rightarrow \ell_{\text{priv}} \rightarrow \ell_{\text{priv}} \rightarrow \ell_f \)

We say that the system is **timed-opaque** w.r.t. \( \ell_{\text{priv}} \) on the way to \( \ell_f \) for all durations in \([2,3]\)
Example

There exist (at least) two runs of duration $d$ for all durations $d \in [2, 3]$:

We say that the system is **timed-opaque w.r.t.** $\ell_{priv}$ on the way to $\ell_f$ for all durations in $[2,3]$
Example

There exist (at least) two runs of duration $d$ for all durations $d \in [2, 3]$:

We say that the system is timed-opaque w.r.t. $\ell_{\text{priv}}$ on the way to $\ell_f$ for all durations in $[2,3]$.

But
Example

There exist (at least) two runs of duration $d$ for all durations $d \in [2, 3]$:

We say that the system is timed-opaque w.r.t. $\ell_{\text{priv}}$ on the way to $\ell_f$ for all durations in $[2,3]$

But

There exists a run of duration 1.5 reaching $\ell_f$ and visiting $\ell_{\text{priv}}$
There exist (at least) two runs of duration \( d \) for all durations \( d \in [2, 3] \):

We say that the system is **timed-opaque** w.r.t. \( \ell_{priv} \) on the way to \( \ell_f \) for all durations in \([2, 3]\)

**But**

There exists a run of duration 1.5 reaching \( \ell_f \) and visiting \( \ell_{priv} \)

There exists no run of duration 1.5 reaching \( \ell_f \) and *not* visiting \( \ell_{priv} \)
Example

There exist (at least) two runs of duration \( d \) for all durations \( d \in [2, 3] \):

We say that the system is timed-opaque w.r.t. \( \ell_{\text{priv}} \) on the way to \( \ell_f \) for all durations in [2,3]

But

There exists a run of duration 1.5 reaching \( \ell_f \) and visiting \( \ell_{\text{priv}} \)

There exists no run of duration 1.5 reaching \( \ell_f \) and \textit{not} visiting \( \ell_{\text{priv}} \)

We say that the system is \textit{not fully} timed-opaque w.r.t. \( \ell_{\text{priv}} \) on the way to \( \ell_f \)
Outline

Preliminaries: Timed Opacity: Formalism and Preliminary results
  Timed Opacity formalization
  Computation problem and results

Contribution: (Untimed) Control for timed opacity

Perspectives
Problem: timed-opacity computation

Timed-opacity computation problem

Find durations $d$ ("execution times") of runs from $\ell_0$ to $\ell_f$ such that the system is timed-opaque w.r.t. $\ell_{priv}$ on the way to $\ell_f$

Theorem  
The durations $d$ such that the system is timed-opaque can be effectively computed and defined


Problem: timed-opacity computation

Timed-opacity computation problem

Find durations $d$ ("execution times") of runs from $\ell_0$ to $\ell_f$ such that the system is timed-opaque w.r.t. $\ell_{priv}$ on the way to $\ell_f$

**Theorem** The durations $d$ such that the system is timed-opaque can be effectively computed and defined

**Corollary** Asking if a TA is timed-opaque for all its execution times is decidable

---


Timed-opacity computation problem

Find durations $d$ ("execution times") of runs from $\ell_0$ to $\ell_f$ such that the system is timed-opaque w.r.t. $\ell_{\text{priv}}$ on the way to $\ell_f$

**Theorem**  The durations $d$ such that the system is timed-opaque can be effectively computed and defined

**Corollary**  Asking if a TA is timed-opaque for all its execution times is decidable

Proof: based on the region graph and RA-arithmetic [Wei99]

---


Outline

Preliminaries: Timed Opacity: Formalism and Preliminary results

Contribution: (Untimed) Control for timed opacity

Perspectives
Context & Informal problem

√ We can decide computation and decision problems for timed opacity

× What to do if the model is not (fully) timed-opaque?

We can decide computation and decision problems for timed opacity.

What to do if the model is not (fully) timed-opaque?

Full timed opacity control

Is it possible to disable some user actions to make the system fully timed-opaque?

Untimed control

Goal

Exhibit a controller guaranteeing the system to be fully timed-opaque

i.e., a subset of the actions to be kept, while other controllable actions are disabled
Untimed control

**Goal**

Exhibit a controller guaranteeing the system to be fully timed-opaque
i.e., a subset of the actions to be kept, while other controllable actions are disabled

We distinguish two kinds of actions:

- **uncontrollable:** required by the system or dependent on another agent
  - e.g., action dealing with a correct or incorrect password
- **controllable:** that can be disabled
Outline

Preliminaries: Timed Opacity: Formalism and Preliminary results

Contribution: (Untimed) Control for timed opacity
   A running example
   Our tool
   Proof of concept

Perspectives
A running example

Is the system fully timed-opaque?

▶ Passing by $\ell_2$: [1, 5]
▶ Not passing by $\ell_2$: [1, 3] $\cup$ [4, 4] $\cup$ [5, + inf)
$\Rightarrow$ Not fully timed-opaque
A running example

Uncontrollable $u$
Controllable $a, b, c, d, e, f$
Allowed $u + b, c$
Disabled $a, d, e, f$

Is the system fully timed-opaque?

- Passing by $\ell_2$: $[2, 5]$
- Not passing by $\ell_2$: $[4, 4]$
⇒ Not fully timed-opaque
A running example

Is the system fully timed-opaque?

▶ Passing by \( \ell_2 \): [1, 3]
▶ Not passing by \( \ell_2 \): [1, 3]
⇒ Fully timed-opaque
It can be shown that the set of sets of actions to allow is
\{u, a\} \quad \{u, a, e\} \quad \{u, a, f\}
A running example

It can be shown that the set of fully timed-opaque strategies is
\{u, a\} \quad \{u, a, e\} \quad \{u, a, f\}
A running example

It can be shown that the set of fully timed-opaque strategies is

\[
\begin{align*}
\{u, a\} & \quad \{u, a, e\} & \quad \{u, a, f\} \\
\text{minimal} & & \text{maximal}
\end{align*}
\]
Outline

Preliminaries: Timed Opacity: Formalism and Preliminary results

Contribution: (Untimed) Control for timed opacity
  A running example
  Our tool
  Proof of concept

Perspectives
strategFTO

- an automated open-source tool written in Java

- iteratively constructs strategies
  - computes the private and public execution times (using IMITATOR[And21])
  - checks full timed opacity by checking their equality (using POLYOp¹)
    - Method: by considering execution times as a timing parameter, and performing parameter synthesis

---


¹ https://github.com/etienneandre/PolyOp
Outline

Preliminaries: Timed Opacity: Formalism and Preliminary results

Contribution: (Untimed) Control for timed opacity
  A running example
  Our tool
  Proof of concept

Perspectives
A Proof of concept benchmark: an ATM

Uncontrollable actions
- correctAmount
- correctPwd
- incorrectAmount
- incorrectPwd
- pressFinish

Controllable system actions
- askPwd
- finish
- start

Controllable user actions
- reqBalance
- normalWithdraw
- pressOK
- quickWithdraw
- restart

Secret
- takeCash
### Actions to disable

<table>
<thead>
<tr>
<th>Option</th>
<th>synthMinControl</th>
<th>witnessMinControl</th>
<th>synthMaxControl</th>
<th>witnessMaxControl</th>
<th>synthControl</th>
</tr>
</thead>
<tbody>
<tr>
<td>restart, pressOK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>restart, reqBalance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>restart, pressOK, quickWithdraw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>restart, pressOK, reqBalance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>restart, quickWithdraw, reqBalance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- find min
- find min -witness
- find max
- find max -witness
- find all

- √

---

Methodology: add to the ATM model an increasing number of self-loop transitions
Outline

Preliminaries: Timed Opacity: Formalism and Preliminary results

Contribution: (Untimed) Control for timed opacity

Perspectives
Perspectives

Theory

- Use symbolic reasoning
  \[\rightarrow\] Instead of a simple enumeration

- Extend the method to **timed** control
Perspectives

Theory
- Use symbolic reasoning
  → Instead of a simple enumeration
- Extend the method to timed control

Algorithmic and implementation
- Automatic translation of programs to timed automata
- Repairing a non timed-opaque system

References II

