strategFTO: Untimed Control for Timed Opacity

Étienne André
Université Sorbonne Paris Nord, LIPN, CNRS UMR 7030
Villetaneuse, France

Shapagat Bolat
Engel Lefaucheux
Dylan Marinho
Université de Lorraine, CNRS, Inria, LORIA
Nancy, France

Abstract

We introduce a prototype tool strategFTO addressing the verification of a security property in critical software. We consider a recent definition of timed opacity where an attacker aims to deduce some secret while having access only to the total execution time. The system, here modelled by timed automata, is deemed opaque if for any execution time, there are either no corresponding runs, or both public and private corresponding runs. We focus on the untimed control problem: exhibiting a controller, i. e., a set of allowed actions, such that the system restricted to those actions is fully timed-opaque. We first show that this problem is not more complex than the full timed opacity problem, and then we propose an algorithm, implemented and evaluated in practice.

CCS Concepts: • **Security and privacy** → **Logic and verification**; • **Theory of computation** → *Quantitative automata*; *Verification by model checking*.

Keywords: opacity, timing leak, timed automata, security, control, IMITATOR.

ACM Reference Format:
Étienne André, Shapagat Bolat, Engel Lefaucheux, and Dylan Marinho. 2022. strategFTO: Untimed Control for Timed Opacity. In *Proceedings of the 8th ACM SIGPLAN International Workshop on Formal Techniques for Safety-Critical Systems (FTSCS '22), December 07, 2022, Auckland, New Zealand*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3563822.3568013

1 Introduction

We address here the control of timed systems to avoid timing leaks, i. e., the leakage of private information that can be deduced from time. We use as underlying model timed automata (TAs) [1], an extension of finite-state automata with real-valued clocks. Opacity is a key security property requiring that an external user should not be able to deduce whether the execution of a system contains a secret behaviour through its observation. This property was first formalized for labeled transition systems [15], by specifying a subset of secret paths and requiring that, for any secret path, there is a non-secret one with the same observation. Opacity raises challenging research issues such as 1) specifying formally opacity in various frameworks [15, 22], 2) verifying opacity properties [15, 26], and 3) developing mechanisms to design a system satisfying opacity while preserving functionality and performance [10, 16].

Franck Cassez proposed in [17] a first definition of *timed opacity* asking whether an attacker can deduce a secret by observing a set of observable actions together with their timestamp. He proved that opacity is undecidable for TAs, mainly from the undecidability of the language inclusion problem for TAs [1]. Based on this definition of opacity, some decidable subclasses were proposed, for real-time automata [29, 30] (a severely restricted subclass of TAs with a single clock), or over bounded-time [3].

In [7], we proposed a definition of opacity where the attacker only has access (in addition to the model knowledge) to the system *execution time*, i. e., the time from the initial location to a given location. The timed opacity problem therefore asks "for which execution times is the attacker unable to deduce whether a private location was visited?" The *full* timed opacity problem asks whether the system is timed-opaque for all execution times, i. e., the attacker is never able to deduce whether the private location was visited by an execution. We proved in [7] that this latter problem is decidable (in 3EXPTIME), and we proposed a practical algorithm using a parametric version of TAs [2], implemented in IMITATOR [4].

Contribution. If a system is not fully timed-opaque, there may be ways to tune it to enforce opacity. For instance, one could change internal delays, or add some `sleep()` or `Wait()` statements in the program (see e. g., [7]). In this paper, we consider a static (untimed) form of control of the system. This indicates whether there is a way of restricting the behavior of users to ensure full timed opacity. With that mindset, we assume the set of actions of the TA is partitioned into a set of *controllable* actions (that can be disabled) and a set of *uncontrollable* actions (that cannot be disabled). We address the following goal: exhibit a controller (i. e., a subset of the

FTSCS '22, December 07, 2022, Auckland, New Zealand

ACM ISBN 978-1-4503-9907-4/22/12…$15.00
https://doi.org/10.1145/3563822.3568013

system controllable actions to be kept in addition to the uncontrollable actions, while other controllable actions are disabled) guaranteeing the system to be fully timed-opaque. We propose an algorithm exhibiting a set of controllers ensuring opacity, implemented into a tool strategFTO, calling IMITATOR [4] for computing suitable opaque execution times, and PolyOp [9] for additional polyhedra operations.

Related Works. It is well known that observing the time taken by a system to finish some operation is a potential way to get information out of it (see e. g., [25]). As such, identifying which information is released by the timing of a system has been studied both from a security and a safety perspective.

From the security point of view, beyond the works related to timed opacity and TAs [3, 7, 17, 29, 30], the notion of non-interference has been widely studied. A first definition of timed non-interference was proposed for TAs in [11, 12]. This notion is extended to PTAs in [6], with a semi-algorithm implemented using IMITATOR [4]. In [20], another notion of timed interference called timed strong non-deterministic non-interference (SNNI) which was based on timed language equivalence between the automaton with hidden low-level actions and the automaton with removed low-level actions was developed. This notion is in some aspects stronger than the opacity notion we consider, and is undecidable. SNNI was adapted in [28] to allow some intentional information leakage and a form of control aimed at ensuring it was presented in [13]. Their framework gives to the attacker more information than the total execution time, and their control differs from ours to include that knowledge.

The *diagnosis* of TAs is one of the dominant research directions aimed at analysing information leakage from a safety perspective. Its goal is to detect, by observing the system, whether some faulty behaviour occured. As such, it is some form of dual to opacity. Diagnosis was first introduced for TAs in [27]. Diagnosability of a system is shown there to be decidable, though the actual diagnoser may be quite complex (see [14] for subclasses of TAs allowing simpler diagnoser, see also [19] for a summary of the main results on the diagnosis of TAs and [18] for a diagnosability focused control of TAs).

2 Preliminaries

We assume a set $\mathbb{X} = \{x_1, \ldots, x_H\}$ of *clocks*, i. e., real-valued variables that all evolve over time at the same rate. A clock valuation is a function $\mu : \mathbb{X} \to \mathbb{R}_{\geq 0}$. We write $\vec{0}$ for the clock valuation assigning 0 to all clocks. Given $d \in \mathbb{R}_{\geq 0}$, $\mu + d$ denotes the valuation s.t. $(\mu + d)(x) = \mu(x) + d$, for all $x \in \mathbb{X}$. Given $R \subseteq \mathbb{X}$, we define the *reset* of a valuation μ, denoted by $[\mu]_R$, as follows: $[\mu]_R(x) = 0$ if $x \in R$, and $[\mu]_R(x) = \mu(x)$ otherwise.

A clock guard g is a constraint over $\mathbb{X}$ defined by a conjunction of inequalities of the form $x \bowtie d$, with $d \in \mathbb{Z}$ and $\bowtie \in \{<, \leq, =, \geq, >\}$. Given g, we write $\mu \models g$ if the expression obtained by replacing each x with $\mu(x)$ in g evaluates to true.

Definition 2.1 (TA [1])**.** A TA $\mathcal{A}$ is a tuple $\mathcal{A} = (\Sigma, L, \ell_0, \ell_{priv}, \ell_f, \mathbb{X}, I, E)$, where: *i*) Σ is a finite set of actions, *ii*) L is a finite set of locations, *iii*) $\ell_0 \in L$ is the initial location, *iv*) $\ell_{priv} \in L$ is the private location, *v*) $\ell_f \in L$ is the final location, *vi*) $\mathbb{X}$ is a finite set of clocks, *vii*) I is the invariant, assigning to every $\ell \in L$ a clock guard $I(\ell)$, *viii*) E is a finite set of edges $e = (\ell, g, a, R, \ell')$ where $\ell, \ell' \in L$ are the source and target locations, $a \in \Sigma$, $R \subseteq \mathbb{X}$ is a set of clocks to be reset, and g is a clock guard.

Example 2.2. Consider the TA in Fig. 1a, using one clock x. ℓ_1 is the initial location, while we assume that ℓ_f is the *final* location, i. e., a location in which an attacker can measure the execution time from the initial location. ℓ_2 is the private location, i. e., a secret to be preserved: the attacker should not be able to deduce whether it was visited or not. ℓ_2 has an invariant $x \leq 3$ (boxed); other locations invariants are true.

Definition 2.3 (Semantics of a TA [1])**.** Given a TA $\mathcal{A} = (\Sigma, L, \ell_0, \ell_{priv}, \ell_f, \mathbb{X}, I, E)$, the semantics of $\mathcal{A}$ is given by the timed transition system (TTS) $T_{\mathcal{A}} = (S, s_0, \rightarrow)$, with

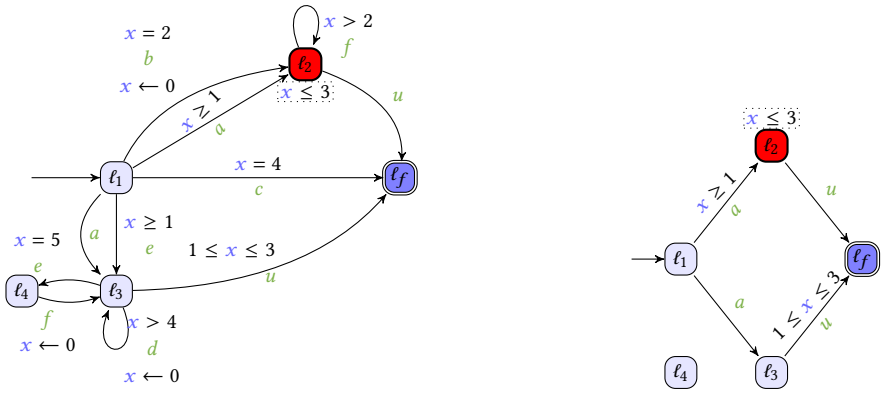
- $S = \{(\ell, \mu) \in L \times \mathbb{R}^H_{\geq 0} \mid \mu \models I(\ell)\}$, $s_0 = (\ell_0, \vec{0})$,
- $\rightarrow$ consists of the discrete and (continuous) delay transition relations: *i*) discrete transitions: $(\ell, \mu) \overset{e}{\mapsto} (\ell', \mu')$, if $(\ell, \mu), (\ell', \mu') \in S$, and there exists $e = (\ell, g, a, R, \ell') \in E$, such that $\mu' = [\mu]_R \models I(\ell')$, and $\mu \models g$. *ii*) delay transitions: $(\ell, \mu) \overset{d}{\mapsto} (\ell, \mu + d)$, with $d \in \mathbb{R}_{\geq 0}$, if $\forall d' \in [0, d], (\ell, \mu + d') \in S$.

Moreover we write $(\ell, \mu) \overset{(d,e)}{\longrightarrow} (\ell', \mu')$ for a combination of a delay and discrete transition if $\exists \mu'' : (\ell, \mu) \overset{d}{\mapsto} (\ell, \mu'') \overset{e}{\mapsto} (\ell', \mu')$. Given a TA $\mathcal{A}$ with semantics $(S, s_0, \rightarrow)$, a *run* of $\mathcal{A}$ is an alternating sequence of states of $T_{\mathcal{A}}$ and pairs of delays and edges starting from the initial state s_0 of the form $s_0, (d_0, e_0), s_1, \cdots$ where for all i, $e_i \in E$, $d_i \in \mathbb{R}_{\geq 0}$ and $s_i \overset{(d_i,e_i)}{\longrightarrow} s_{i+1}$. The *duration* of a finite run $\rho : s_0, (d_0, e_0), s_1, \cdots, (d_{i-1}, e_{i-1}), (\ell_i, \mu_i)$ is $dur(\rho) = \sum_{0 \leq j \leq i-1} d_j$.

Timed Opacity Definitions. We recall here the notion of timed opacity defined in [7].[1]

Given $\mathcal{A} = (\Sigma, L, \ell_0, \ell_{priv}, \ell_f, \mathbb{X}, I, E)$, and a run ρ, we say that ℓ_{priv} is reached on the way to ℓ_f in ρ if ρ is of the form $(\ell_0, \mu_0), (d_0, e_0), (\ell_1, \mu_1), \cdots, (\ell_m, \mu_m), (d_m, e_m), \cdots (\ell_n, \mu_n)$ for some $m, n \in \mathbb{N}$ such that $\ell_m = \ell_{priv}$, $\ell_n = \ell_f$ and $\forall 0 \leq i \leq m-1, \ell_i \neq \ell_f$. We denote by $Reach^{\mathcal{A}}_{\ell_{priv}}(\ell_f)$ the set of those runs, and refer to them as *private* runs. Conversely, we say that ℓ_{priv} is avoided on the way to ℓ_f in ρ if ρ is of

[1] We slightly modify the definitions from [7] by incorporating ℓ_{priv} within the definition of $\mathcal{A}$, and removing ℓ_{priv} and ℓ_f from the definition of $DReach^{priv}(\mathcal{A})$ and $DReach^{\neg priv}(\mathcal{A})$ to simplify the reading.



(a) A TA example

(b) $Control(\mathcal{A}, \{u, a\})$

Figure 1. Running example

the form $(\ell_0, \mu_0), (d_0, e_0), (\ell_1, \mu_1), \cdots, (\ell_n, \mu_n)$ with $\ell_n = \ell_f$ and $\forall 0 \leq i < n, \ell_i \notin \{\ell_{priv}, \ell_f\}$. We denote the set of those runs by $Reach^{\mathcal{A}}_{\neg \ell_{priv}}(\ell_f)$, and refer to them as *public* runs.

While we model the secret behaviour of the system using a private location ℓ_{priv} here, note that one could easily adapt these definitions if the secret is, for example, a set of locations, an action (this will be the case in our case study) or the value of a variable.

$DReach^{priv}(\mathcal{A})$ (resp. $DReach^{\neg priv}(\mathcal{A})$) is the set of all the durations of the runs for which ℓ_{priv} is reached (resp. avoided) on the way to ℓ_f. Formally: $DReach^{priv}(\mathcal{A}) = \{d \in \mathbb{R}_{\geq 0} \mid \exists \rho \in Reach^{\mathcal{A}}_{\ell_{priv}}(\ell_f)$ such that $d = dur(\rho)\}$ and $DReach^{\neg priv}(\mathcal{A}) = \{d \in \mathbb{R}_{\geq 0} \mid \exists \rho \in Reach^{\mathcal{A}}_{\neg \ell_{priv}}(\ell_f)$ such that $d = dur(\rho)\}$.

Definition 2.4 (full timed opacity)**.** Given a TA $\mathcal{A}$, we say that $\mathcal{A}$ is *fully timed-opaque* if $DReach^{priv}(\mathcal{A}) = DReach^{\neg priv}(\mathcal{A})$.

That is, a system is fully timed-opaque if, for any execution time d, there exists a run of duration d that reaches ℓ_f after going through ℓ_{priv} iff there exists another run of duration d that reaches ℓ_f without going through ℓ_{priv}. Hence, the attacker cannot deduce from the execution time whether ℓ_{priv} was visited or not.

Example 2.5. Consider again the TA in Fig. 1a. Recall that ℓ_2 is the private location. We have $DReach^{priv}(\mathcal{A}) = [1, 5]$ and $DReach^{\neg priv}(\mathcal{A}) = [1, 3] \cup [4, 4] \cup (5, +\infty)$. Since $DReach^{priv}(\mathcal{A}) \neq DReach^{\neg priv}(\mathcal{A})$, the system is *not* fully timed-opaque.

3 Untimed Control for Full Timed Opacity

In this section, we introduce an untimed control for controlling timed opacity. We assume $\Sigma = \Sigma_c \uplus \Sigma_u$ where Σ_c (resp. Σ_u) denote controllable (resp. uncontrollable) actions.

A (static, untimed) *strategy* of a TA $\mathcal{A}$ is a set of actions $\sigma \subseteq \Sigma$ that contains at least all uncontrollable actions (i. e., $\Sigma_u \subseteq \sigma \subseteq \Sigma$). A strategy induces a restriction of $\mathcal{A}$ where only the edges labeled by actions of σ are allowed:

Definition 3.1 (Controlled TA)**.** Given $\mathcal{A} = (\Sigma, L, \ell_0, \ell_{priv}, \ell_f, \mathbb{X}, I, E)$ with $\Sigma = \Sigma_u \uplus \Sigma_c$ and a strategy $\sigma \subseteq \Sigma$, the *control* of $\mathcal{A}$ using σ is the TA $\mathcal{A}' = Control(\mathcal{A}, \sigma) = (\sigma, L, \ell_0, \ell_{priv}, \ell_f, \mathbb{X}, I, E')$ where $E' = \{(\ell, g, a, R, \ell') \in E \mid a \in \sigma\}$.

Example 3.2. Consider again the TA $\mathcal{A}$ in Fig. 1a. Fix $\sigma = \{u, a\}$. Then $Control(\mathcal{A}, \sigma)$ is in Fig. 1b.

Strategies represent some modifications of the system that can be implemented to ensure full timed opacity.

Definition 3.3 (fully timed-opaque strategy)**.** A strategy σ is *fully timed-opaque* if $Control(\mathcal{A}, \sigma)$ is fully timed-opaque.

A strategy (even a maximal one) might achieve full timed opacity by blocking all runs (both private or public) from reaching the target. If reaching the target means completing a task, this might not be something one would desire. We call a strategy allowing to reach the target for at least some durations an *effective* strategy.

We define two slightly different problems: taking a TA $\mathcal{A}$ as input, the **full timed** (resp. **effective full time**) **opacity control emptiness problem** asks whether the set of fully (resp. effective fully) timed-opaque strategies for $\mathcal{A}$ is empty.

Note that, due to the presence of uncontrollable actions, the first problem (full timed opacity control emptiness) is not trivial. (If uncontrollable actions were not part of our definitions, choosing $\sigma = \emptyset$ would always yield an acceptable fully timed-opaque strategy.)

We will also refine those problems by considering a notion of *maximal* (i. e., most permissive) strategy w.r.t. full timed opacity based on the number of actions belonging to the strategy: given $\mathcal{A}$, a fully timed-opaque strategy σ is maximal if $\forall \sigma'$, if σ' is fully timed-opaque then $|\sigma'| \leq |\sigma|$. We define similarly *minimal* strategies (least permissive, i. e.,

disabling as many actions as possible) as well as maximal (resp. minimal) effective fully timed-opaque strategies, i. e., the set of largest (resp. smallest) effective fully timed-opaque strategies.

Example 3.4. Consider again the TA $\mathcal{A}$ in Fig. 1a. Assume $\Sigma_u = \{u\}$ and $\Sigma_c = \{a, b, c, d, e, f\}$. Fix $\sigma_1 = \{u, b, c\}$. We have $DReach^{priv}(Control(\mathcal{A}, \sigma_1)) = [2, 5]$ while $DReach^{\neg priv}(Control(\mathcal{A}, \sigma_1)) = [4, 4]$; therefore, σ_1 is not fully timed-opaque. Now fix $\sigma_2 = \{u, a, f\}$. We have $DReach^{priv}(Control(\mathcal{A}, \sigma_2)) = DReach^{\neg priv}(Control(\mathcal{A}, \sigma_2)) = [1, 3]$; therefore, σ_2 is fully timed-opaque.

In fact, it can be shown that the set of effective fully timed-opaque strategies for $\mathcal{A}$ is $\{\{u, a\}, \{u, a, e\}, \{u, a, f\}\}$; therefore, $\{u, a\}$ is the only minimal strategy, while $\{u, a, e\}, \{u, a, f\}$ are the two maximal strategies. In addition, $\{u, f\}$ is an example of a strategy that is not effective, as ℓ_f is always unreachable, whether ℓ_{priv} is visited or not.

Proposition 3.5 (complexity)**.** *One can compute the set of fully timed-opaque strategies over a TA $\mathcal{A}$ in* 3EXPTIME.

Proof. The full timed opacity decision problem (i. e., checking if a given TA is fully timed-opaque) is decidable for TAs in (at most) 3EXPTIME [7]. Moreover, reachability of the final state can be decided in PSPACE [1]. Thus, for any given strategy, one can check in triple exponential time whether it is (effective) fully timed-opaque.

Computing the list of (effective) fully timed-opaque strategies can be done naively by testing each possible strategy one by one and keeping the ones that satisfy the property. As there is an exponential number of possible strategies and repeating exponentially many times a 3EXPTIME algorithm remains in 3EXPTIME, this algorithm is in 3EXPTIME. □

As a corollary of the above, the (effective) full timed opacity control emptiness problem is in 3EXPTIME as well. More precisely, the above proof establishes that the complexity class of the (effective) full timed opacity control emptiness problem is the maximum between PSPACE and the complexity of the full timed opacity problem. As the latter is PSPACE-hard (being trivially harder than reachability), the two problems lie in the same complexity class. From a theoretical point of view, one thus cannot do better than the naive enumeration approach described here to solve the control problem.

Finding the maximal (resp. minimal) strategies can be done slightly more efficiently by starting from the set with every (resp. no) controllable action and enumerating the potential strategies by decreasing (resp. increasing) order as one could then potentially stop before full enumeration. In the worst case, this will however have the same complexity as the full enumeration.

4 Implementation and Experiments

We implemented our strategy generation in strategFTO, an entirely automated open-source tool written in Java.[2] Our tool iteratively constructs strategies, then checks full timed opacity by computing the private and public execution times and by checking their equality.

The exhibition of these execution times ($DReach^{\neg priv}(\mathcal{A})$ and $DReach^{priv}(\mathcal{A})$) is done in our implementation by an automated model modification (following the procedure described in [7], but which was not entirely automated in [7]) followed by a synthesis problem using a parametric extension of TAs [2]. The synthesis of the execution times itself is done by a call to an external tool—IMITATOR 3.3 *"Cheese Caramel au beurre salé"* [4]. strategFTO then checks whether both sets of execution times are equal; this is done by a call to another external tool—PolyOp 1.2[3], that performs polyhedral operations using PPL [9].

Algorithms. We implement not only the exhibition of all timed-opaque strategies (denoted by synthCtrl($\mathcal{A}$)), but also the following variants: *i*) synthMaxCtrl($\mathcal{A}$): synthesize all maximal strategies for $\mathcal{A}$; *ii*) synthMinCtrl($\mathcal{A}$): synthesize all minimal strategies; *iii*) witnessMaxCtrl($\mathcal{A}$): witness *one* maximal strategy; *iv*) witnessMinCtrl($\mathcal{A}$): witness *one* minimal strategy. We implemented these other algorithms by changing the exploration order of the strategies, and/or by triggering immediate termination upon the first exhibition of a strategy.

Input Model. The input TA model is given in the IMITATOR input syntax; while we presented a restricted setting in this paper for sake of clarity, our implementation in strategFTO is much more permissive, by allowing significant extensions of TAs with global (integer or Boolean) variables, multiple automata with synchronization, multi-rate clocks (including stopwatches), etc.

4.1 Proof of Concept Benchmark

As a proof of concept, we consider the TA model of an ATM (given in Fig. 2). The idea is that (as per our definition of timed opacity) the attacker only has access to the execution time, i. e., the time from the beginning of the program to reaching the end state. The secret is whether the ATM user has actually obtained cash (action *takeCash*).[4] The TA uses two clocks: x for "local" actions, and y for a global time measurement. First, the user starts the process (action *start*), then the ATM displays a welcome screen for 3 time units, followed by another screen requesting the password (action *askPwd*). Then, the user can submit a correct (action *correctPwd*) or incorrect (*incorrectPwd*) password; if no password is input

[2] Source code is at https://github.com/DylanMarinho/Controlling-TA. Models and experiment results are available at 10.5281/zenodo.7181848.
[3] https://github.com/etienneandre/PolyOp
[4] strategFTO allows not only private locations, but also actions.

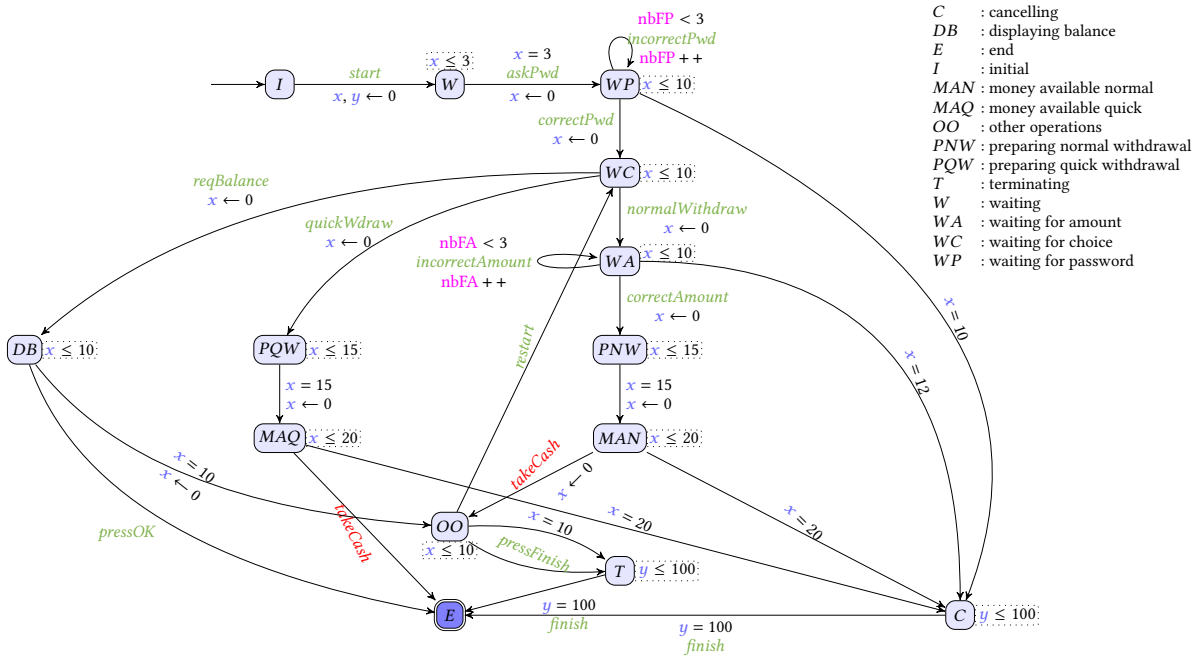


Figure 2. ATM benchmark

within 10 time units, the system moves to a cancelling phase. The same happens if 3 incorrect password have been input. After inputting the correct password, the user has the choice between a fixed-amount quick withdrawal (*quickWdraw*), a normal withdrawal (*normalWithdraw*) or a balance request (*reqBalance*).

The quick withdrawal triggers a 15-time unit preparation followed by the availability of the money, which the user can take immediately (action *takeCash*), thus terminating the procedure. If the user does not take the money, the system moves to the cancelling phase.

The normal withdrawal asks the user to input the desired amount; similar to the password, after 3 wrong amounts (action *incorrectAmount*), or upon timeout, the system moves to cancelling phase. After the user retrieves cash (action *takeCash*), they are asked whether they would like to perform another operation; if so (action *restart*), the system goes back to the choice location. Otherwise (action *pressFinish*), or unless a 10-time unit timeout is reached, the system moves to the terminating location. The balance request triggers the balance display, from which the user can immediately terminate the process (action *pressOK*), or go back to the choice menu.

The rationale is that, in the regular terminating and cancelling phases, the ATM terminates after constant time (invariant $y \leq 100$), avoiding leaking information. However, some actions may lead to quicker termination (quick withdrawal) or slower termination (multiple choices).

The uncontrollable actions are most of the user actions: *correctAmount*, *incorrectAmount*, *correctPwd*, *incorrectPwd*, *pressFinish*, *takeCash*. The controllable actions are the system actions (*askPwd*, *start*, *finish*) and some of the users actions that can be controlled by disabling the associated choice (*reqBalance*, *pressOK*, *quickWdraw*, *restart*).

4.2 Experiments

We exhibit in Table 1 controllers for our Fig. 2 benchmark computed by strategFTO, for all algorithms. For space concern, we tabulate the actions to *disable*; the strategy is therefore Σ minus these actions. Also note that, for witnessMaxCtrl and witnessMinCtrl, the *order* in which we compute the subsets of Σ has an impact on the result, as the algorithm stops as soon as *one* strategy is found. According to Table 1, the maximal strategies (i. e., the most permissive, disabling the least number of actions) are to disable either *restart* and *pressOK*, or *restart* and *reqBalance*. This is natural, as *restart* allows the user to restart a second operation, thus violating the constant-time nature of Fig. 2, while *pressOK* and *reqBalance*, if enabled together, allow a quick exit, shorter than a cash withdrawal operation—thus giving hint to the attacker that the *takeCash* secret did *not* occur.

Scalability. Then, we test the scalability of strategFTO w.r.t. the number of actions. We modify Fig. 2 by adding an increasingly large numbers of controllable actions; these actions do not play a role in the control (we basically add unguarded self-loops) but they will impact the computation

Table 1. Strategy synthesis for Fig. 2

Actions to disable	synthMinCtrl	witnessMinCtrl	synthMaxCtrl	witnessMaxCtrl	synthCtrl
restart, *pressOK*			√	√	√
restart, *reqBalance*			√		√
restart, *pressOK*, *quickWdraw*					√
restart, *pressOK*, *reqBalance*					√
restart, *quickWdraw*, *reqBalance*					√
restart, *pressOK*, *quickWdraw*, *reqBalance*	√	√			√

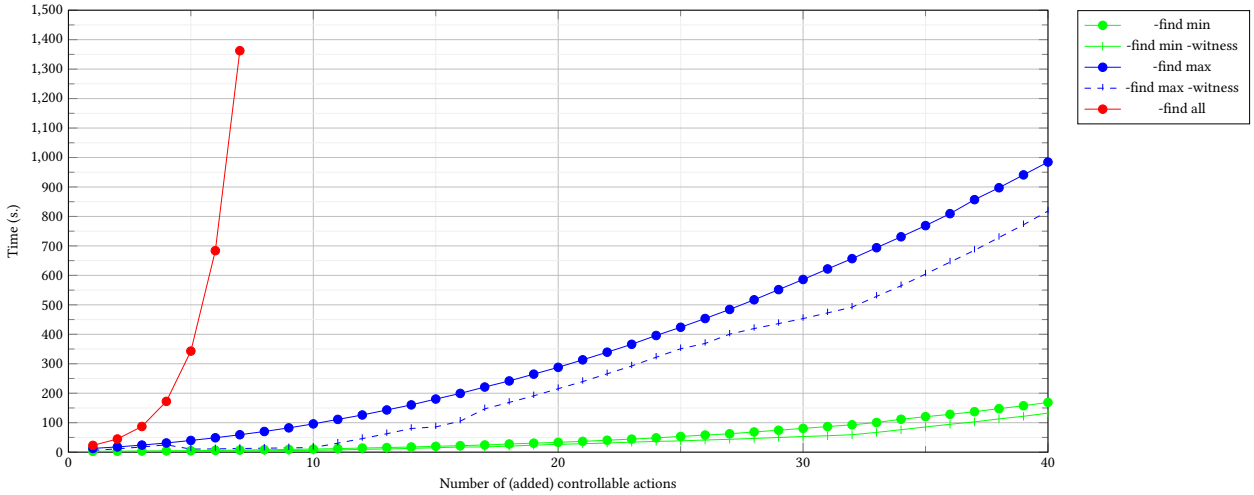


Figure 3. Execution times for scalability (in seconds; TO set at 1,800 s)

time, as we will need to consider an increasingly (and exponentially) larger number of subsets of actions. We add from 1 to 40 such actions, resulting (by adding the actions in Fig. 2) in a model with a number of controllable actions from 11 to 50. We plot these results in Fig. 3. From our results in Fig. 3, we see that, without surprise, the execution time for synthCtrl is exponential in the number of actions. However, synthMaxCtrl and synthMinCtrl behave much better, by remaining respectively below 15 minutes and three minutes, even for up to 50 controllable actions. In addition, it is important to notice that witnessMaxCtrl and witnessMinCtrl do not decrease the time very much compared to the full versions synthMaxCtrl and synthMinCtrl. This is because, at a given size, the number of strategies to be tested remains relatively small.

5 Conclusion

We introduced a prototype tool strategFTO implementing an algorithm to exhibit strategies to guarantee the full timed opacity of a system modeled by a timed automaton where the attacker only has access to the computation time. Even though relying on a simple enumeration of the subsets, our tool strategFTO shows good performance for synthesizing maximal or minimal strategies, with very reasonable times, even for several dozens of controllable actions.

Future Works. We plan to further optimize our implementation by maintaining a set of non-effective strategies, i. e., for which ℓ_f is unreachable: any strategy strictly included into a known non-effective strategy will necessarily be non-effective too, and therefore no full timed-opacity analysis is needed for this strategy. An option to efficiently represent this strategies set could be to store it using BDDs.

We also plan to strengthen strategies so that their choice may depend on how long has passed since the start of the execution. As these strategies still need a finite representation to be handled, this requires establishing exactly what strategies need to remember to chose optimally.

Our ultimate goal will be to extend timed automata to *parametric* timed automata, and use automated parameter synthesis techniques (e. g., [5, 8, 23]), with a parametric timed controller [21, 24].

Acknowledgments

This work is partially supported by the ANR-NRF French-Singaporean research program ProMiS (ANR-19-CE25-0015 / 2019 ANR NRF 0092) and the ANR research program BisoUS.

Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several universities as well as other organizations (see https://www.grid5000.fr).

References

[1] Rajeev Alur and David L. Dill. 1994. A theory of timed automata. *Theoretical Computer Science* 126, 2 (April 1994), 183–235. https://doi.org/10.1016/0304-3975(94)90010-8

[2] Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. 1993. Parametric real-time reasoning. In *STOC* (San Diego, California, United States), S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal (Eds.). ACM, New York, NY, USA, 592–601. https://doi.org/10.1145/167088.167242

[3] Ikhlass Ammar, Yamen El Touati, Moez Yeddes, and John Mullins. 2021. Bounded opacity for timed systems. *Journal of Information Security and Applications* 61 (Sept. 2021), 1–13. https://doi.org/10.1016/j.jisa.2021.102926

[4] Étienne André. 2021. IMITATOR 3: Synthesis of timing parameters beyond decidability. In *CAV* (virtual) *(Lecture Notes in Computer Science, Vol. 12759)*, Rustan Leino and Alexandra Silva (Eds.). Springer, 1–14. https://doi.org/10.1007/978-3-030-81685-8_26

[5] Étienne André, Jaime Arias, Laure Petrucci, and Jaco van de Pol. 2021. Iterative Bounded Synthesis for Efficient Cycle Detection in Parametric Timed Automata. In *TACAS* (Virtual) *(Lecture Notes in Computer Science, Vol. 12651)*, Jan Friso Groote and Kim G. Larsen (Eds.). Springer, 311–329. https://doi.org/10.1007/978-3-030-72016-2_17

[6] Étienne André and Aleksander Kryukov. 2020. Parametric non-interference in timed automata. In *ICECCS* (Singapore), Yi Li and Alan Liew (Eds.). 37–42. https://doi.org/10.1109/ICECCS51672.2020.00012

[7] Étienne André, Didier Lime, Dylan Marinho, and Jun Sun. 2022. Guaranteeing timed opacity using parametric timed model checking. *ACM Transactions on Software Engineering and Methodology* 31, 4 (Oct. 2022), 1–36. https://doi.org/10.1145/3502851

[8] Étienne André, Dylan Marinho, and Jaco van de Pol. 2021. A Benchmarks Library for Extended Timed Automata. In *TAP* (virtual) *(Lecture Notes in Computer Science, Vol. 12740)*, Frédéric Loulergue and Franz Wotawa (Eds.). Springer, 39–50. https://doi.org/10.1007/978-3-030-79379-1_3

[9] Roberto Bagnara, Patricia M. Hill, and Enea Zaffanella. 2008. The Parma Polyhedra Library: Toward a Complete Set of Numerical Abstractions for the Analysis and Verification of Hardware and Software Systems. *Science of Computer Programming* 72, 1–2 (2008), 3–21. https://doi.org/10.1016/j.scico.2007.08.001

[10] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. 2012. On the (im)possibility of obfuscating programs. *Journal of the ACM* 59, 2 (2012), 6:1–6:48. https://doi.org/10.1145/2160158.2160159

[11] Roberto Barbuti, Nicoletta De Francesco, Antonella Santone, and Luca Tesei. 2002. A Notion of Non-Interference for Timed Automata. *Fundamenta Informormaticae* 51, 1-2 (2002), 1–11.

[12] Roberto Barbuti and Luca Tesei. 2003. A Decidable Notion of Timed Non-Interference. *Fundamenta Informaticae* 54, 2-3 (2003), 137–150.

[13] Gilles Benattar, Franck Cassez, Didier Lime, and Olivier H. Roux. 2015. Control and synthesis of non-interferent timed systems. *Internat. J. Control* 88, 2 (2015), 217–236. https://doi.org/10.1080/00207179.2014.944356

[14] Patricia Bouyer, Fabrice Chevalier, and Deepak D'Souza. 2005. Fault Diagnosis Using Timed Automata. In *FoSSaCS* (Edinburgh, UK) *(Lecture Notes in Computer Science, Vol. 3441)*, Vladimiro Sassone (Ed.). Springer, 219–233. https://doi.org/10.1007/978-3-540-31982-5_14

[15] Jeremy W. Bryans, Maciej Koutny, Laurent Mazaré, and Peter Y. A. Ryan. 2008. Opacity generalised to transition systems. *International Journal of Information Security* 7, 6 (2008), 421–435. https://doi.org/10.1007/s10207-008-0058-x

[16] Béatrice Bérard, Serge Haddad, and Engel Lefaucheux. 2017. Probabilistic Disclosure: Maximisation vs. Minimisation. In *FSTTCS* (Kanpur, India) *(LIPIcs, Vol. 93)*, Satya V. Lokam and R. Ramanujam (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 13:1–13:14. https://doi.org/10.4230/LIPIcs.FSTTCS.2017.13

[17] Franck Cassez. 2009. The Dark Side of Timed Opacity. In *ISA* (Seoul, Korea) *(Lecture Notes in Computer Science, Vol. 5576)*, Jong Hyuk Park, Hsiao-Hwa Chen, Mohammed Atiquzzaman, Changhoon Lee, Tai-Hoon Kim, and Sang-Soo Yeo (Eds.). Springer, 21–30. https://doi.org/10.1007/978-3-642-02617-1_3

[18] Franck Cassez. 2010. Dynamic observers for fault diagnosis of timed systems. In *CDC* (Atlanta, Georgia, USA). IEEE, 4359–4364. https://doi.org/10.1109/CDC.2010.5717696

[19] Franck Cassez and Stavros Tripakis. 2013. *Fault Diagnosis of Timed Systems*. Wiley, 107–138. https://doi.org/10.1002/9781118558188.ch4

[20] Guillaume Gardey, John Mullins, and Olivier H. Roux. 2007. Non-Interference Control Synthesis for Security Timed Automata. *Electronic Notes in Theoretical Computer Science* 180, 1 (2007), 35–53. https://doi.org/10.1016/j.entcs.2005.05.046

[21] Ebru Aydin Gol. 2021. Control Synthesis for Parametric Timed Automata under Unavoidability Specifications. In *ECC* (Virtual Event / Delft, The Netherlands). IEEE, 740–745. https://doi.org/10.23919/ECC54610.2021.9655222

[22] Dominic J. D. Hughes and Vitaly Shmatikov. 2004. Information Hiding, Anonymity and Privacy: A Modular Approach. *Journal of Computer Security* 12, 1 (2004), 3–36. https://doi.org/10.3233/jcs-2004-12102

[23] Aleksandra Jovanović, Didier Lime, and Olivier H. Roux. 2015. Integer Parameter Synthesis for Real-Time Systems. *IEEE Transactions on Software Engineering* 41, 5 (2015), 445–461. https://doi.org/10.1109/TSE.2014.2357445

[24] Aleksandra Jovanović, Didier Lime, and Olivier H. Roux. 2019. A game approach to the parametric control of real-time systems. *Internat. J. Control* 92, 9 (2019), 2025–2036. https://doi.org/10.1080/00207179.2018.1426883

[25] Paul C. Kocher. 1996. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *CRYPTO* (Santa Barbara, California, USA) *(Lecture Notes in Computer Science, Vol. 1109)*, Neal Koblitz (Ed.). Springer, 104–113. https://doi.org/10.1007/3-540-68697-5_9

[26] Laurent Mazaré. 2004. Decidability of Opacity with Non-Atomic Keys. In *FAST* (Toulouse, France) *(IFIP, Vol. 173)*, Theodosis Dimitrakos and Fabio Martinelli (Eds.). Springer, 71–84. https://doi.org/10.1007/0-387-24098-5_6

[27] Stavros Tripakis. 2002. Fault Diagnosis for Timed Automata. In *FTRTFT* (Oldenburg, Germany) *(Lecture Notes in Computer Science, Vol. 2469)*, Werner Damm and Ernst-Rüdiger Olderog (Eds.). Springer, 205–224. https://doi.org/10.1007/3-540-45739-9_14

[28] Panagiotis Vasilikos, Flemming Nielson, and Hanne Riis Nielson. 2018. Secure Information Release in Timed Automata. In *POST* (Thessaloniki, Greece) *(Lecture Notes in Computer Science, Vol. 10804)*, Lujo Bauer and Ralf Küsters (Eds.). Springer, 28–52. https://doi.org/10.1007/978-3-319-89722-6_2

[29] Lingtai Wang and Naijun Zhan. 2018. Decidability of the Initial-State Opacity of Real-Time Automata. In *Symposium on Real-Time and Hybrid Systems - Essays Dedicated to Professor Chaochen Zhou on the Occasion of His 80th Birthday*, Cliff B. Jones, Ji Wang, and Naijun Zhan (Eds.). Lecture Notes in Computer Science, Vol. 11180. Springer, 44–60. https://doi.org/10.1007/978-3-030-01461-2_3

[30] Lingtai Wang, Naijun Zhan, and Jie An. 2018. The Opacity of Real-Time Automata. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 11 (2018), 2845–2856. https://doi.org/10.1109/TCAD.2018.2857363

Received 2022-09-08; accepted 2022-10-10