Timed Automata as a Formalism for Expressing Security: A Survey on Theory and Practice

JOHAN ARCILE and ÉTIENNE ANDRÉ, Université de Lorraine, CNRS, Inria, LORIA, Nancy, France

Timed automata are a common formalism for the verification of concurrent systems subject to timing constraints. They extend finite-state automata with clocks, that constrain the system behavior in locations, and to take transitions. While timed automata were originally designed for safety (in the wide sense of correctness w.r.t. a formal property), they were progressively used in a number of works to guarantee security properties. In this work, we review works studying security properties for timed automata over the past two decades. We notably review theoretical works, with a particular focus on opacity, as well as more practical works, with a particular focus on attack trees and their extensions. We derive main conclusions concerning open perspectives, as well as tool support.

CCS Concepts: • Theory of computation → Timed and hybrid models; Verification by model checking; • Security and privacy → Formal security models; Logic and verification;

Additional Key Words and Phrases: Timed automata, cybersecurity, opacity, attack trees, survey

ACM Reference format:
Johan Arcile and Étienne André. 2022. Timed Automata as a Formalism for Expressing Security: A Survey on Theory and Practice. ACM Comput. Surv. 55, 6, Article 127 (December 2022), 36 pages.
https://doi.org/10.1145/3534967

1 INTRODUCTION

Numerous critical information systems rely on communication via a shared network, such as the Internet. Data passing through such networks is often sensitive and requires secrecy. If not handled carefully, then information such as private data, authentication code, timing information or localization can be accessible to anyone on the network. This can result in security attacks to retrieve or alter sensitive data [87, 96, 133]. To prevent such intrusions, various security methods and protocols have been developed. Yet, these security decisions do not always avoid intrusion.

To analyze the security of information systems and highlight their weaknesses, technical standards such as Failure Mode Effects and Criticality Analysis [49, 189] have been used since the early days of critical information systems. More structured, model-based approaches have since been explored, such as the ADVISE method [123], which allows for automatic generation of quantitative metrics, or formalisms such as team automata [32] and attack trees [105], respectively aiming at a formal representation of the behavior of the system and the attacker. Those approaches allow us to check some security properties, such as authentication (not being able to lie about oneself), secrecy.
(private information is not visible to public parties), integrity (information cannot be altered), and non-interference (not being able to deduce another user’s actions).

**Time and security.** Some security protocols are time sensitive by combining concurrency and real-time constraints. It has been noted in, e.g., References [35, 106] that time is a potential attack vector against otherwise secure systems. That is, it is possible that a secure system (“non-interferent”) can become insecure (“interferent”) when timing constraints are added [74]. In addition, it is possible to correlate the execution time and the value of an encryption key via so-called timing attacks [35, 42, 68, 101, 106]. In 2018, the Spectre vulnerability [100] exploited speculative execution to bring secret information into the cache. Subsequently, cache-timing attacks were launched to exfiltrate these secrets.

**Timed automata (TAs)** [12] are a formalism extending finite-state automata with real-time variables called clocks. Timed automata were well studied over the past three decades, and the formalism is supported by a wide range of model checking tools and techniques (e.g., References [33, 58, 67, 84, 122, 126, 132, 138, 190]). Among them, UPPAAL [122] is certainly one of the most efficient tools for reachability-based verification, with numerous other extensions.

Over the past two decades, a growing number of works considered timed automata in a security context. Some of these works studied the theory of some security properties (such as non-interference) in the context of TAs. Some other works used timed automata as a convenient formalism to formally verify some concrete security properties on a given case study, using some model checker (typically UPPAAL). Some other works used timed automata as a target formalism from security formalisms, typically attacking trees and their extensions, thus allowing us to use existing model checkers to guarantee these properties.

**Related surveys.** To the best of our knowledge, no previous work specifically surveys timed automata in the context of security problems.

Concerning timed automata, Alur and Madhusudan survey in Reference [15] decision problems related to the formalism. That survey focuses on theoretical aspects of TAs and is not related to specific applications. Bin Waez, Dingel, and Rudie [175] survey tools and methods using TAs in general; again, no focus is made on security applications. Also, Fontana and Cleaveland [71, 97] survey various extensions of TAs and provide conversion algorithms from these extensions to the original “vanilla” TA formalism. Concerning extensions of timed automata, André surveyed in Reference [17] decision problems related to various parametric extensions of timed automata [14].

Regarding applications of formal methods to security issues, Dalal et al. benchmark [57] the performance of state-of-the-art tools for the verification of security protocols, while Avalle et al. survey [28] formal verification techniques for security protocol implementations. The real-time aspect of systems is not considered. Souri and Norouzi [159] survey formal methods in the context of Internet of Things (IoT) applications, including security aspects. Although some of the presented papers use TAs as a modeling formalism, the focus of the survey is on the applications rather than the methods. Finally, in Reference [181] (which extends Reference [105]), Widel et al. survey methods using attack trees, a formalism used in the context of security modeling. Among those methods, TAs are present as a way to model such trees, but the focus is not specifically made on TAs.

**Contribution.** We provide here a survey of the use of timed automata and their extensions in the context of security analyses. We believe surveying works relating timed automata and security can help readers to understand the state of the art of both the theory (“what security problems are decidable for timed automata and their subclasses”) and the practice (“what can be practically achieved, using what software”). We emphasize on open problems and on practical TA model checkers that can be used for solving security problems.
Somehow surprisingly, while timed automata seem to be an interesting formalism to model interactions between time and concurrency in the framework of security property, the number of such works is "reasonable" (very roughly a hundred or so). Therefore, we aim at a sense of near-completeness in this survey so as to provide readers with a complete overview of the works using timed automata as an approach toward verifying security properties. In addition, we use a limited set of TA examples on which we illustrate various (theoretical) notions from the literature so as to exemplify the (sometimes subtle) differences between these notions in a unified manner.

When surveying these works, two large research directions emerged:

1. the works studying theoretical aspects, notably linked to the decidability of various opacity and non-interference properties, on various subclasses of TAs;
2. the works translating increasingly larger extensions of attack trees into (extensions of) TAs.

These works are therefore surveyed in two dedicated sections. Most remaining works concern applications to various areas: verification of protocols, of controllers, formalization of role-based access control (RBAC) models, and so on.

Methodology. We collected works using different methods: (i) performing search engine queries, typically DBLP and Google Scholar, (ii) collecting relevant citations to our own works, and (iii) collecting relevant citations from and to all the aforementioned works. We aim at (near-)completeness, with the goal to be as complete as possible regarding works in the domain of timed automata and security.

Outline of the article. Section 2 gives a formal definition of timed automata syntax and semantics. Some extensions of TAs and related tools mentioned in the survey are also presented. We survey specifically works studying timed automata and non-interference in Section 3. We then survey more practical methods concerning translations of attack trees and their extensions into timed automata in Section 4. We then review in Section 5 remaining works that all address various application domains in the area of cyber-physical systems, notably approaches related to formalizing RBAC models, verifying controllers and detecting attacks on communication protocols. We give some general perspectives in Section 6.

2 TIMED AUTOMATA

Timed automata [12] are a common formalism to model concurrent timed systems, and formally verify them against temporal ("order of the events") and timed ("relative or absolute time at which events occur") properties. Among other advantages, TAs can express both a continuous representation of time, and concurrency. Notably, checking TAs against timed properties expressed in timed computation tree logic (TCTL) [11], a well-known extension of CTL [52], is decidable [12]—which includes reachability properties.

Syntactically, a TA is a directed and connected graph, featuring an initial vertex, and extended with clocks, i.e., real-valued variables that evolve at the same rate. Vertices are called locations and are associated with invariants. Invariants are constraints on clocks, authorizing entry into a location or forcing its exit. The edges are annotated with guards, along which clocks can be reset to 0.

We recall the formal definition of both the syntax and semantics of TAs; we also review some extensions of TAs mentioned in this article.

2.1 Clock Guards

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

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

### 2.2 Timed Automata

**Definition 2.1 (Timed Automaton).** A TA $\mathcal{A}$ is a tuple $\mathcal{A} = (\Sigma, \mathcal{L}, \ell_0, \mathbb{X}, I, E)$, where:

1. $\Sigma$ is a finite set of actions,
2. $\mathcal{L}$ is a finite set of locations,
3. $\ell_0 \in \mathcal{L}$ is the initial location,
4. $\mathbb{X}$ is a finite set of clocks,
5. $I$ is the invariant, assigning to every $\ell \in \mathcal{L}$ a clock guard $I(\ell)$,
6. $E$ is a finite set of edges $e = (\ell, g, a, R, \ell')$ where $\ell, \ell' \in \mathcal{L}$ 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 Figure 1(a), containing two clocks $x$ and $y$. This TA features no invariant. For each transition, we denote vertically (from top to bottom) first the guard (if any), then the action, and finally the clocks resets (if any). In Figure 1, we also give the edge name together with the action (of the form “$e : a” where $e$ is the edge name and $a$ is the action). The edge “names” do not strictly speaking belong to the TA, but they will ease describing our subsequent examples using Figure 1.

For example, in Figure 1(a), the self-loop from $\ell_0$ to $\ell_0$ via action $h$ is guarded by clock guard “$x = 2$” and resets clock $x$. The name of this edge is $e_1$.

From the initial location $\ell_0$ in Figure 1(a), it is possible to reach either $\ell_1$ or $\ell_2$. Reaching $\ell_2$ via $l_1$ can be done by simply waiting 5 time units (guard “$y = 5$”), while reaching $\ell_1$ via $l_1$ or $\ell_2$ via $l_2$ is more involved, as (assuming both clocks are initially 0) the TA will need to take several times the self-loop on $\ell_0$ resetting $x$ when some guard (“$x = 2$ or $x = 3$) is satisfied, while keeping $y$ unchanged, until eventually the guard “$y = 5 \land x = 0$” is satisfied.

#### 2.2.1 Syntactical Transformations.

Sometimes, and notably in works related to non-interference or opacity, we will assume that actions are partitioned into a set of low-level actions $\Sigma_L$ and a set $\Sigma_U$ of the form $x - x' \Rightarrow c$. Several syntax variations exist in the literature, notably diagonal constraints of the form $x - x' \Rightarrow c$. 

---

1. Several syntax variations exist in the literature, notably diagonal constraints of the form $x - x' \Rightarrow c$. 

ACM Computing Surveys, Vol. 55, No. 6, Article 127. Publication date: December 2022.
of high-level actions $\Sigma_H$. The restriction of a TA to low-level actions discards all edges labeled with high-level actions.

**Definition 2.3 (Restriction).** Let $\mathcal{A} = (\Sigma, \mathcal{L}, \ell_0, \mathcal{X}, I, E)$ be a TA with $\Sigma = \Sigma_L \cup \Sigma_H$ (\(\cup\) denotes disjoint union). The restriction of $\mathcal{A}$ to low-level actions, denoted by $\mathcal{A}_{|\Sigma_L}$, is defined as the TA identical to $\mathcal{A}$ except that any edge of the form $(\ell, g, a, R, \ell')$ with $a \notin \Sigma_L$ (i.e., $a \in \Sigma_H$) is discarded.

The hiding of high-level actions in a TA replaces all edges labeled with high-level actions with unobservable transitions.

**Definition 2.4 (Hiding).** Let $\mathcal{A} = (\Sigma, \mathcal{L}, \ell_0, \mathcal{X}, I, E)$ be a TA with $\Sigma = \Sigma_L \cup \Sigma_H$. The hiding of high-level actions in $\mathcal{A}$, denoted by $\mathcal{A}_{|\Sigma_H}$, is defined as the TA identical to $\mathcal{A}$ except that any edge of the form $(\ell, g, a, R, \ell')$ with $a \in \Sigma_H$ is replaced with an edge $(\ell, g, e, R, \ell')$, where $e$ is the special silent (unobservable) action.

**Example 2.5.** Consider again the TA $\mathcal{A}$ in Figure 1(a). Assume $\Sigma_L = \{l_1, l_2\}$ and $\Sigma_H = \{h\}$. Then the restriction of $\mathcal{A}$ to $\Sigma_L$ is given in Figure 1(b). The hiding of $\Sigma_H$ in $\mathcal{A}$ is given in Figure 1(c).

2.2.2 Concrete Semantics of TAs. We recall below the concrete semantics of TAs.

**Definition 2.6 (Semantics of a TA).** Given a TA $\mathcal{A} = (\Sigma, \mathcal{L}, \ell_0, \mathcal{X}, I, E)$, the semantics of $\mathcal{A}$ is given by the timed transition system $(S, s_0, \rightarrow)$, with

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

Moreover, we write $(\ell, \mu) \xrightarrow{(d,e)} (\ell', \mu')$ for a combination of a delay and discrete transition if $\exists \mu'': (\ell, \mu) \xrightarrow{e} (\ell', \mu'') \xrightarrow{d} (\ell', \mu')$.

Given a TA $\mathcal{A}$ with concrete semantics $(S, s_0, \rightarrow)$, we refer to the states of $S$ as the concrete states of $\mathcal{A}$. A concrete run of $\mathcal{A}$ is an alternating sequence of concrete states of $\mathcal{A}$ and pairs of delays and edges starting from the initial state $s_0$ of the form $s_0, (d_0, e_0), s_1, \ldots$, with $i = 0, 1, \ldots, e_i \in E, d_i \in \mathbb{R}_{\geq 0}$ and $s_i \xrightarrow{(d_i,e_i)} s_{i+1}$.

Given a state $s = (\ell, \mu)$, we say that $s$ is reachable in $\mathcal{A}$ if $s$ appears in a run of $\mathcal{A}$; and by extension, we say that $\ell$ is reachable in $\mathcal{A}$.

Given a concrete run $(\ell_0, \mu_0), (d_0, e_0), (\ell_1, \mu_1), \ldots$, the associated timed word is $(a_0, \tau_0), (a_1, \tau_1), \ldots$, where $a_i$ is the action of edge $e_i$, and $\tau_i = \sum_{0 \leq j \leq i} d_j$, for $i = 1, 2, \ldots$. The timed language of $\mathcal{A}$, denoted by $L(\mathcal{A})$, is the set of timed words associated with all concrete runs of $\mathcal{A}$. Silent actions ($e$) are omitted from the timed words and the subsequent timed language.

**Example 2.7.** Let us first come back to the TA in Figure 1(a). A possible run for this TA is $(\ell_0, 0, 0) \xrightarrow{(2,e_1)} (\ell_0, 0, 2) \xrightarrow{(3,e_2)} (\ell_0, 0, 5) \xrightarrow{(0,e_1)} (\ell_1, 0, 5)$. Note that, as an abuse of notation, we write $(\ell_0, 0, 2)$ for $(\ell_0, \mu_0)$ such that $\mu_0(x) = 0$ and $\mu_0(y) = 2$.

\[\text{\textsuperscript{2}Several works related to TAs add a notion of acceptance, i.e., the run must end in a final location to be accepting and be part of the language.}\]
Since $\ell_1$ belongs to a concrete run of $\mathcal{A}$, $\ell_1$ is reachable in $\mathcal{A}$. In addition, the timed word associated to the aforementioned run is $((h, 2), (h, 5), (l_1, 5))$.

Let us now consider the TA in Figure 1(b). The timed word $((h, 2), (h, 5), (l_1, 5))$ is not part of this TA, and in fact, it can be shown that $\ell_1$ is unreachable, i.e., it cannot be reached along any run. As the only edge that can be taken is $e_4$ (to $\ell_2$), the only concrete run is $((\ell_0, (0, 0)) \rightarrow (\ell_0, (5, 5)))$ and the timed language is exactly $\{(l_1, 5)\}$.

Let us now consider the TA in Figure 1(c). The timed language of this TA is made of two timed words of length 1: $\{(l_1, 5), (l_2, 4)\}$. Observe that silent actions are omitted. Also, the first timed word $(l_1, 5)$ can correspond to either a run reaching $\ell_1$ (via $e_3$), or a run reaching $\ell_2$ (via $e_4$).

### 2.3 Extensions of Timed Automata

In the following, we briefly review two extensions of timed automata, both being considered relatively often when discussing security issues.

#### 2.3.1 Probabilistic Timed Automata

Probabilistic timed automata extend TAs with probabilities along edges [81, 116].

A probabilistic TA is a tuple $\mathcal{A} = (\Sigma, \mathcal{L}, \ell_0, \mathcal{X}, I, E, \pi)$, where $\pi : E \to [0, 1]$ is a probability function. For $e \in E$, $\pi(e)$ is the probability of taking edge $e$. Let $\text{out}(\ell) \subseteq E$ be the set of outgoing edges from location $\ell$; then

$$\forall \ell \in \mathcal{L} : \sum_{e \in \text{out}(\ell)} \pi(e) = 1.$$  

#### 2.3.2 Parametric Timed Automata

Parametric TA extend TAs with timing parameters that can be used instead of integer constants in guards and invariants [14]. These parameters are in general rational valued, even though some restrictions may exist (see a survey in Reference [17]).

Formally, given a set of timing parameters $P$, a parametric clock guard $g$ is a constraint over $\mathcal{X} \cup P$ defined by a conjunction of inequalities of the form $x \triangleright p$ or $x \triangleright c$, with $x \in \mathcal{X}$, $p \in P$, and $c \in \mathbb{Z}$.

A parametric TA is a tuple $\mathcal{A} = (\Sigma, \mathcal{L}, \ell_0, \mathcal{X}, P, I, E)$, where $I(\ell)$ assigns to location $\ell$ a parametric clock guard, and $E$ is a set of edges $e = (\ell, g, a, R, \ell')$, where $g$ is a parametric clock guard.

Given a parameter valuation $v : P \to \mathbb{Q}$, we denote by $v(\mathcal{A})$ the non-parametric structure where each occurrence of a parameter $p_i$ is replaced with $v(p_i)$. This structure is usually a timed automaton as defined in Reference [12].

**Example 2.8.** Consider the PTA $\mathcal{A}$ in Figure 3, featuring a single parameter $p$. Given $v$ such that $v(p) = 5$, $v(\mathcal{A})$ corresponds to the TA in Figure 1(a).

### 2.4 Model Checking Tools for (Extensions of) Timed Automata

Model checking [53] involves building a (usually finite) model of a system and then checking whether a given property holds on this model by traversing (a portion of) the (finite or infinite) set of the reachable states. Model checking has the advantages of being fully automatic and of usually generating concrete witnesses (“traces”) in the case of a successful reachability property. However, it suffers from the infamous state space explosion problem. To cope with this problem, model checking tools often rely on specific abstractions that reduce the size of the state space, i.e., the number of states, (usually) without losing critical information.

The most frequently used model checker among the works surveyed here is UPPAAL [122], a verification tool supporting an extension of TAs that has been used by the community since the

---

3With some assumptions such as $v(p) \in \mathbb{Z}$ for all $p$, or assuming an appropriate rescaling of constants.
early 2000’s. Uppaal extends TAs with features such as integer variables, data structures, functions, and synchronization channels. The tool allows us to verify a subset of TCTL (mainly reducing to reachability) and generates concrete execution traces in case of successful reachability.

One of the many extensions of Uppaal, Uppaal-SMC [59], is mentioned a few times in this survey. Uppaal-SMC supports probabilistic TAs, and uses statistical model checking (SMC) to avoid the exhaustive exploration of the state space. Basically, SMC consists in a statistical approach that allows us to know whether a given property is true with a given degree of confidence. Although it lacks exhaustiveness, it is generally well suited for checking stochastic properties.

IMITATOR [18] is mentioned in this survey when reviewing works in which some timing parameters are unknown. IMITATOR takes as input extensions of TAs with timing parameters and performs parametric timed model checking: rather than answering a binary answer (“yes”/“no”) to a given property, it synthesizes a set of parameter valuations enforcing the property.

VerICS [95] is used in the context of communication protocols. It is a SAT-based verification tool (i.e., the model and the property are translated into a Boolean formula that holds if the property holds) that supports TAs. It allows bounded model checking (BMC [39]).

Finally, the following other model checking tools are mentioned in some of the works surveyed in the following: Kronos [190], MiniSAT [167], Modest [41], PAT [126], Uppaal-CORA [1], and Uppaal-Stratego [58].

3 NON-INTERFERENCE

In this section, we survey methods related to the timed modeling of information flow, allowing us to verify non-interference properties. Non-interference concerns systems where different levels of security are present in the system (typically, normal users and administrators). Such systems satisfy non-interference if there is no information flow from the high security level to the low security level. Intuitively, actions taken on the higher security level (including the absence of actions) does not impact in any way the lower level. Equivalently, there is no way for a low-level user to infer information on the higher level.

Non-interference can become more challenging when timing information are involved. Without the non-interference property, systems are vulnerable to attacks that exploit the timing of message flows to infer information. Typically, it is possible to infer the content of some memory space from the access times of a cryptographic module. Such attacks are surveyed in Reference [40].

Time particularly influences the non-interference of a system. It has been shown in Reference [74] that a non-interferent system can become interferent when timing constraints are added.

In this entire section, the attacker (if any) has access to the entire system model and can observe at runtime some partial behavior (for example, low-level actions together with their timing).

Example 3.1. Let us informally illustrate the notion of non-interference for timed systems.

Consider the TAs in Figure 2, featuring a single clock $x$ and two actions: $l$ is a low-level action, while $h$ is a high-level action. When referring to non-interference as a notion related to the observable timed language (i.e., only the low-level actions, with their timestamp, are observable to an external observer), then the TA in Figure 2(b) is not non-interferent. Indeed, if an attacker (that
has access to the TA model) observes $l$ at time 1.5, then they can deduce that the edge from $\ell_0$ to $\ell_2$ and then to $\ell_3$ was necessarily taken, and therefore $h$ has necessarily occurred.

In contrast, the TA in Figure 2(c) is non-interferent: It is not possible to deduce information on the occurrence of high-level actions by only observing low-level actions (with their timestamp).

As a side-note on this example, also observe that, as Reference [74] mentioned, Figure 2(a) is an untimed non-interferent automaton, but adding timing constraints (Figure 2(b)) makes it interferent.

**Outline of the section.** In this section, we review various notions of non-interference for timed automata, namely $n$-non-interference (Section 3.1), non-interference and simulation (Section 3.2), non-interference and probabilities (Section 3.3), timed opacity (Section 3.4), and non-interference and information flow (Section 3.5). We also provide a local summary and discussion in Section 3.6.

### 3.1 $n$-non-interference

A first occurrence of the use of TAs in the context of non-interference is found in Reference [69], where Focardi et al. study a known privacy breach on websites, where a timed attack allows a malicious website to determine whether another unrelated website is present in the user’s Web-browsing history. The website, the cache, and the user requests are modeled using TAs, where actions are divided between high-level (the user) and low-level (the malicious website) ones. A model is obtained by doing the parallel composition of these automata; then, the authors deduce whether there is a possibility of information flow from high-level actions to low-level ones. The results highlight that the attack is possible only if the requests from the user are separated by less than a certain amount of time. Those results are compared with both symbolic model checking based on binary decision diagram [39] and with process algebras [70]. The TAs approach has the advantage of using continuous time, but verifying properties takes longer.

**Timed $n$-non-interference.** Barbuti et al. are the first to propose in Reference [29] a method using TAs to model and analyze timed non-interference. As usual, the set of actions in the considered TAs is supposed to be partitioned into high-level and low-level actions. Non-interference is defined in regard to a minimum delay $n$ between consecutive high-level actions. A given timed automaton $\mathcal{A}$ is $n$-non-interfering if the language of $\mathcal{A}$ subject to the hiding of its high-level actions $\Sigma_H$ (i.e., $\mathcal{A}_{(\Sigma_H)}$, see Definition 2.4) is equal to the language of the TA based on $\mathcal{A}$ but where high-level actions are allowed if being separated by at least $n$ time units. This latter condition is ensured by taking the parallel composition of $\mathcal{A}$ with a gadget TA ensuring that any two high-level actions are separated by at least $n$ time units. Formal details are given about the transformation of the TA, along with an example on a simplified airplane controller. However, checking equivalence between two timed languages is undecidable for general TAs [12], and therefore the practical side of checking $n$-non-interference is not addressed in this work.

In Reference [30], the authors adapt the approach from Reference [29] with a different notion of non-interference. The difference is that the equivalence is now checked on the set of reachable locations (called “states” in Reference [30]) rather than on the timed language. This equivalence is decidable as it reduces to the reachability problem for TAs, which is decidable [12]. An example of this new method is illustrated on Fischer’s mutual exclusion protocol [117]. Two processes and an intruder able to rewrite a shared variable are modeled using TAs. With this new definition, proving $n$-non-interference reduces to checking whether locations that are not reachable without the intruder are also not reachable with the intruder but with actions of the two processes separated by time intervals of at least $n$ time units. It is then possible to infer a minimum waiting time between two high-level actions of the process, such that the system verifies non-interference.
Example 3.2. Consider again the TA in Figure 1(a), featuring two low-level actions $l_1$ and $l_2$ and one high-level action $h$. In absence of high-level actions (i.e., restricted to low-level actions, see Figure 1(b)), note that, since $x$ is never reset, only the edge $e_1$ (from $\ell_0$ to $\ell_2$ with action $l_1$) can be taken. Therefore, the timed language of this TA restricted to low-level actions is \{($l_1, 5$)\} and the only reachable locations are \{$\ell_0, \ell_2$\}. However, when one allows high-level actions (Figure 1(a)), the transitions resetting $x$ (edges $e_1$ and $e_2$) can be taken every 2 or 3 time units, therefore potentially modifying the observable behavior.

Let us first illustrate non-interference on timed language [29]. Recall that the only timed word in the absence of $h$ (i.e., in the TA restricted to low-level actions, in Figure 1(b)) is ($l_1, 5$). Note that taking the edge $e_2$ to $\ell_1$ does not change this timed word. The only way to obtain a different timed word is to take the lower transition $e_3$ from $\ell_0$ to $\ell_2$, labelled with $l_2$. This transition can only be taken after taking twice the lower self-loop ($e_2$) below $\ell_0$—which requires a minimum delay of 2 time units. Therefore, the system is not 2-non-interferent but is $n$-non-interferent for any $n \geq 3$. Note that Reference [29] does not perform a parameterized verification (for any $n$) but requires a different check for any valuation of $n$.

Concerning non-interference w.r.t. the reachability of locations [30], the only way to obtain a different set of reachable locations is to take the transition $e_3$ to $\ell_1$. To do so, one needs to take the self-loop $e_1$ above $\ell_0$ (i.e., after 2 time units) and then the self-loop $e_2$ below $\ell_0$ (i.e., after 3 time units)—or vice versa—so that eventually the guard $y = 5 \land x = 0$ is satisfied. This is possible whenever the frequency of the high-level action (i.e., $h$) is high enough: For $n = 1$ or $n = 2$, this is possible. And, therefore, the system is neither 1-non-interferent nor 2-non-interferent. However, the system is 3-non-interferent, because, even though the high-level actions are allowed, there is no way to perform twice $h$ within five time units (required by the guard $y = 5$), with a minimum duration of three time units in between (required by the 3-non-interference property). Again, a parameterized verification (for any $n$) is not discussed in Reference [30] (it will be addressed in Reference [19], see below).

Generalized abstract non-interference. Giacobazzi and Mastroeni [78] propose a notion of generalized abstract non-interference (GANI), extending abstract non-interference from Reference [77]. This notion applies to any kind of finite-state machine models. An abstraction of the observable information flow and of a restricted information flow (hiding what must be kept private) must be given. The system verifies GANI if the observation of the attacker on the observable information flow is equivalent to its observation on the restricted information flow. The authors prove that, in the context of TAs, GANI includes the $n$-non-interference from Reference [29].

Parametric $n$-non-interference. André and Kryukov [19] later extend the decidable $n$-non-interference on reachable locations from Reference [30] to the formalism of parametric timed automata [14]. They define the $n$-location-non-interference synthesis problem on parametric TAs as the computation of all timed intervals ($n$) and parameter values for which the system is non-interferent with respect to the set of reachable locations. The synthesis is done using the parametric timed model checker IMITATOR [18]. The procedure may not terminate but is exact (sound and complete) if it does; if the synthesis is interrupted before termination (e.g., using a timeout), then the result is an over-approximation of the set of conditions for which the system is non-interfering. The method is used on a model of the Fischer’s mutual exclusion protocol extended from Reference [30].

Example 3.3. First consider again the TA in Figure 1(a). Assume that $n$ is now a parameter. The method proposed in Reference [19] can show that the TA is non-interferent (w.r.t. locations) iff $n \geq 4$. 
Now consider the PTA $A$ in Figure 3, which acts as a parametric version of Figure 1(a). The PTA is non-interferent iff

$$\left( p \in \{0, 3, 8\} \right) \lor \left( n \leq 2 \land p \in \{4, 5\} \right) \lor \left( n \leq 3 \land p \in \{2, 6, 7, 9\} \right).$$

Observe that this constraint involves both the timing parameter from the model ($p$) and the frequency ($n$).

3.2 Non-interference and Simulation

Gardey et al. [74] propose several definitions of non-interference, related to various notions of simulation. They consider not only the verification problem (“is the system non-interferent?”) but also the control problem (“synthesize a controller that will restrict the system to obtain non-interference”).

They define a notion of timed strong non-deterministic non-interference (timed SNNI) based on timed language equivalence between the automaton with hidden high-level actions ($A_{\Sigma_H}$) and the automaton with removed high-level actions ($A_{\Sigma_L}$). This notion is similar to $n$-non-interference from Reference [29] when $n = 0$. As seen previously, this problem is undecidable, which mainly comes from the undecidability of the language inclusion checking in timed automata.

In contrast, three decidable notions are proposed:

**Timed cosimulation-based SNNI (“timed CSNNI”).** The TA obtained by hiding high-level actions ($A_{\Sigma_H}$) can be weakly simulated by the original TA $A$. It is proved that this is equivalent to checking that $A_{\Sigma_H}$ can be weakly simulated by $A_{\Sigma_L}$ and reciprocally.

**Timed bisimulation-based strong non-deterministic non-interference (“timed BSNNI”).** This notion extends timed CSNNI to (weak) bisimulation (i.e., $A_{\Sigma_H}$ is weakly bisimilar to $A_{\Sigma_L}$).

**Timed state SNNI (“timed StSNNI”).** The set of reachable locations in $A_{\Sigma_L}$ and $A$ are identical. This definition is equivalent to $n$-non-interference from Reference [30] when $n = 0$.

Decidability of the non-interference verification for these three notions comes from the decidability of simulation [7], bisimulation [121], and location reachability [12] in TAs, respectively.

It is proved in Reference [74] that timed BSNNI implies timed CSNNI, which itself implies timed SNNI.

**Example 3.4.** Assuming $\Sigma_H = \{h\}$ and $\Sigma_L = \{l_1, l_2\}$, the TA $A$ in Figure 4 is SNNI, because the timed language is the same when hiding $h$ (i.e., $A_{\Sigma_H}$) or when pruning $h$ (i.e., $A_{\Sigma_L}$), that is removing $h$ and all the subgraph from $\ell_5$. It is not cosimulation-based SNNI, because the automaton obtained when pruning $h$ (i.e., $A_{\Sigma_L}$) cannot simulate the one when hiding $h$ (i.e., $A_{\Sigma_H}$). That is because in this case, location $\ell_6$ cannot be simulated by any reachable location.
Fig. 4. A TA that is SNNI but not cosimulation-based SNNI (inspired by Reference [35, Fig. 5a]).

Fig. 5. Comparing cosimulation-based SNNI with bisimulation-based SNNI (inspired by Reference [35, Fig. 6]).

Example 3.5. The automaton in Figure 5(a) is cosimulation-based SNNI as the automaton obtained by pruning \( h \) (i.e., \( A_{|\Sigma_L} \)) can simulate the one obtained by hiding \( h \) (i.e., \( A_{\setminus L} \))—and reciprocally. It is not bisimulation-based SNNI, since the location \( \ell_2 \) has no equivalent in the automaton obtained by pruning \( h \) (i.e., \( A_{|\Sigma_L} \)). However, the automaton in Figure 5(b) is bisimulation-based SNNI as the automaton obtained when pruning \( h \) (i.e., \( A_{|\Sigma_L} \)) can simulate the one obtained by hiding \( h \) (i.e., \( A_{\setminus L} \))—and reciprocally.

Control. The timed non-interference control problem is then addressed in Reference [74] for those definitions, in the form of an algorithm generating a controller with the assumption that high-level actions are controllable (i.e., can be enabled or disabled). It is proved that the timed SSSNNI and the timed CSNNI control problems are decidable.

Benattar et al. [35] extend Reference [74] by proposing a subclass of TAs (called dTA) for which the timed SNNI verification problem becomes decidable. A model is in the dTA class if the TA obtained from \( A \) by removing high-level actions (i.e., \( A_{|\Sigma_L} \)) is deterministic. It is proved that the timed SNNI control problem is decidable on the dTA class, and the synthesized controller is the most permissive one (the one that forbids the least actions); note that the set of controllable actions is not necessarily \( \Sigma_H \), in contrast to Reference [74].

It is possible to synthesize a most permissive controller for timed BSSNNI and CSSNNI on the dTA class but not in the general TA class. The generation of the most permissive controller (for dTAs) is done by solving safety timed games [51].

3.3 Non-interference and Probabilistic Timed Automata

A first attempt to adapt timed non-interference to probabilistic timed automata was performed in Reference [118]. Lanotte et al. prove the decidability of weak bisimulation for probabilistic TAs and give a correct algorithm for this problem. It is then applied to a cryptographic protocol of probabilistic non-repudiation.

Lanotte et al. continue this work in References [119, 120], where they propose a framework capturing both timed and probabilistic aspects of non-interference. Modeling is performed with probabilistic TAs, allowing us to analyze information knowledge on probabilistic protocols, where information can be known modulo some probability. For that purpose, a notion of **Probabilistic**
Timed Non-Interference (PTNI) is defined based on weak bisimulation: This notion of PTNI is decidable. It is basically the same notion than the timed bisimulation-based SNNI of Reference [74], with the addition of the probabilistic behavior. A notion of Probabilistic Timed Non Deducibility on Composition (PTNDC) is also defined such that it is satisfied on a system where the observational behavior of the isolated system (where high-levels actions are hidden) is equal to the behavior of the system communicating with any high-level agent. An application of the framework is performed on a network device, and PTNDC is checked on the system, allowing us to identify covert channels that could not have been detected without capturing both probabilistic and timed aspects.

Example 3.6. Figure 6, taken from Reference [120, Fig. 6], presents a system where probabilistic timed non-interference is violated. If the high-level action $h$ is not taken, then one can observe that the timed language is either $\{(l, 3)\}$ or $\{(l, 4)\}$, with a probability of $\frac{1}{2}$ in both cases. However, if high-level action $h$ is taken, then the probability of observing $\{(l, 3)\}$ is $\frac{9}{10}$, and the probability of observing $\{(l, 4)\}$ is $\frac{1}{10}$. Thus, $h$ interferes with the probability of observing $l$. Note that the system becomes non-interferent if either probabilities or time are not considered.

3.4 Timed Opacity

The notion of timed opacity is fairly close to timed non-interference.

Opacity was initially introduced in References [44, 131] to model information leaks from a system to an attacker; that is, it expresses the power of the attacker to deduce some secret information based on some publicly observable behaviors. Cassez [50] extends this notion to timed opacity. If an attacker observing a subset of the actions cannot deduce whether a given sequence of actions has been performed, then the system is opaque. Here, the attacker can also observe time, hence untimed systems that are opaque may not be so once time is added. The article proves that, with such a definition, opacity is undecidable, even for the restricted class of event-recording automata (ERAs) [13], a strict subclass of TAs (i.e., strictly less expressive than TAs).

Wang et al. [176] define a notion of initial-state opacity that is proved decidable on real-time automata (RTAs) [63]. The RTA formalism is a strict subclass of TA with a single clock that is reset at each transition, meaning the only timed information expressed by this formalism is the time elapsed in the current location; most properties are decidable, even complement and language inclusion (undecidable for the full class of TAs). In this context, an intruder can observe the elapsed time on a given subset of transitions, and the system is opaque if an intruder cannot determine whether the system starts or not from a given secret state. The method described in this article consists in constructing two RTAs accepting the projection of languages, from secret initial states for the first one, and from non-secret ones for the other. The authors then use a notion of trace equivalence between an RTA and a finite-state automaton to translate the two RTAs into
Fig. 7. Illustrating various notions of non-interference.

their respective trace equivalent (untimed) finite-state automata. If the intersection of those two automata is empty, then the system is initial-state opaque.

In Reference [177] the authors extend this work to timed language-opacity (i.e., the opacity on timed language studied in Reference [50]). The positive decidability properties of RTAs allow us to check the emptiness of the inclusion between the language accepted by the system and the language where secret words are not allowed. Therefore, timed language-opacity is decidable on the RTA subclass. A tool has been developed to check language-opacity on RTA models.

A recent work by Ammar et al. [16] addresses the timed opacity problem in a similar fashion to Reference [50]: An intruder has access to a subset of actions, along with timed information. The originality of Reference [16] is to consider a time-bounded framework. As in Reference [50], a secret location is timed opaque if the intruder cannot infer from the observation of any execution that the system has reached this particular location. Ammar et al. propose two variants of this definition of opacity. The first one, called bounded timed opacity, implies that the location is opaque up to a given time duration, and is decidable for non-Zeno TA (i.e., TA where it is impossible for an infinity of actions to occur in finite time). The second one, δ-duration bounded opacity, implies that the location must remain secret for at least δ time units after it is reached. To solve the δ-duration bounded opacity, one needs to construct two TAs, with final secret and non-secret locations, respectively, and then apply a reduction such that, for any accepted word w of timed length |w|, all prefixes w’ of that word have a timed length |w’| ≥ |w| − δ. The problem can then be solved by the timed bounded inclusion checking, which is decidable for timed automata [140]; this is in contrast with the (potentially unbounded) language inclusion checking, known to be undecidable [12]. This time-bounded setting is the crux that explains the difference of decidability between Reference [16] and Reference [50]. A case study on a cloud service model illustrates the method by determining the δ value up to which the system is δ-duration bounded opaque. The verification is run using SpaceEx [72], a tool taking as input hybrid automata [85], a formalism of which TAs are a subclass.

Example 3.7. Here, we illustrate the δ-duration bounded opacity from Reference [16]. Consider the TA in Figure 7(a) (taken from Reference [16, Fig. 2a]) where a and b are actions of the model and ℓ1 is a secret location. If we define δ = 4, then ℓ1 is δ-duration bounded opaque: Indeed, for every timed word passing through ℓ1 and ending δ time units after reaching ℓ1, there exists an equivalent timed word not passing through ℓ1. However, if we consider δ = 5 or higher, then ℓ1 is not δ-duration bounded opaque anymore, as there now exist timed words passing through ℓ1 and ending δ time units after reaching ℓ1 such that there exists no equivalent timed word not passing through ℓ1. This is for example the case of the timed word “(a, 2)(b, 7).”

André et al. [20] propose a method to check timed opacity, where the attacker can only observe the “execution time”, i.e., the runs durations from the initial location to a given target location ℓT. This differs from Reference [50], where the attacker was able to observe some actions with their timestamps. More precisely, the definition of Reference [20] states that a system is opaque with
regard to a secret location $\ell_S$ on the way to a target location $\ell_T$ if, for some “executions times,” $\ell_T$ can be reached both with or without passing through $\ell_S$. Two problems are considered:

1. the timed opacity problem, that computes the set of execution times (from the initial location to the target location) for which the system is opaque, i.e., it is not possible to deduce whether the secret location $\ell_S$ was visited; and

2. the full timed opacity problem, that checks whether the system is opaque for all execution times.

These two notions of opacity are proved to be decidable on (non-parametric) TAs. That is, the attacker model of Reference [20] is weaker than that of Reference [50]: While an attacker in Reference [50] can read some actions with their timestamp, the attacker in Reference [20] has access only to the system execution time, which can be seen as a single action with its timestamp (i.e., the action denoting system completion).

**Parametric timed opacity.** The notion of opacity considered in Reference [20] is extended to parametric TAs in the same work. First, the timed opacity emptiness problem is considered: Does there exist a parameter valuation and an execution time for which the system is opaque? This problem is shown to be undecidable for general PTAs but decidable on a sub-class of parametric timed automata called lower-bound automata (L-PTA) and upper-bound automata (U-PTA) [88].

Second, the full timed opacity emptiness problem is considered: Does there exist a parameter valuation for which the system is opaque for all execution times? This second problem is undecidable both for general PTAs and for the aforementioned L/U-PTAs. Third, the timed opacity synthesis problem is addressed, i.e., the computation of both execution times and parameter valuations guaranteeing opacity. Despite undecidability of the emptiness problem, a semi-algorithm is provided (i.e., that may not terminate, but is correct if it does). The method is demonstrated on a set of benchmarks, including standard PTAs benchmarks [23], as well as Java programs from the DARPA Space/Time Analysis for Cybersecurity library, manually translated into PTAs. Experiments are run using IMITATOR and allow us to synthesize the timing parameter valuations for which this definition of opacity holds on the system. The full time opacity synthesis problem is, however, left out. An artifact allowing reproducibility is available online.

**Example 3.8.** Consider the PTA in Figure 7(b) (taken from Reference [20] and inspired by Reference [74, Fig. 2]). Consider the parameter valuation (i.e., replacing a parameter with a concrete valuation) $v$ such that $v(p_1) = 1$ and $v(p_2) = 2$. The resulting TA is time-opaque w.r.t. $\ell_S$ on the way to $\ell_T$ for execution times $[2, 3]$, because, for any duration $d$ with $d \in [2, 3]$, there exists both a run passing by $\ell_S$ and a run not passing by $\ell_S$ and reaching $\ell_T$ in $d$ time units. This is not the case of all execution times: For example, for $d = 1.5$ there exists a run of duration 1.5 passing by $\ell_S$ and reaching $\ell_T$, but no such run not passing by $\ell_S$.

In the parametric setting, the method given in Reference [20] allows us to synthesize parameter valuations for which the TA is (full) time-opaque. Here, for valuations $v$ such that $v(p_1) = v(p_2)$, the system is time-opaque for all execution times.

### 3.5 Non-interference and Information Flow

Vasilikos et al. [139] describe a timed command language adapted from Dijkstra’s guarded command language [62] and show how specifications of this language can be translated to a non-deterministic subclass of TAs. A property of non-interference can be verified on the type system [48] of the specified program, ensuring that there is no flow of information from a high-level action to a low-level one. The approach is demonstrated on a small voting protocol. This method is limited to the subclass of automata than can be described in the command language, and its
scalability is not discussed. UPPAAL is used as the underlying model checker, but no automated translation seems to be proposed.

Vasilikos et al. [172] develop a method to check soundness of timed systems allowing intentional information leaks, such as smart power grids [47]. The system is modeled with TAs where locations are either strongly or weakly observable in regard to an attacker. In that sense, Reference [172] addresses a broader class than Reference [139], that was limited to TAs that can be described by the timed command language. The modeling of Reference [172] describes where it is allowed to leak information and where it is not (weakly observable locations being allowed to bypass the security policy). The information flow is not described through actions but through variables (low-level ones, that are visible, and high-level ones, that must remain hidden). An algorithm is given that checks that no information flow leaks on strongly observable locations. As in Reference [139], no implementation seems to be available.

Example 3.9. Let us briefly illustrate some of the notions from Reference [172]. Consider the TA in Figure 7(c). We assume that \( h \) is a high-level variable, \( x \) is a clock, and \( \ell_2 \) is strongly observable. If \( \ell_1 \) was strongly observable, then non-interference would not hold, as observing the clock \( x \) allows us to deduce whether \( h \) has a negative value. However, the guard from \( \ell_1 \) to \( \ell_2 \) erases this information in such a way that the system respects non-interference.

Vasilikos et al. [173] study the impact of clock granularity on timing channel attacks. Timing side-channels attacks are closely related to non-interference, as an attacker is able to deduce internal information about a system by accessing some publicly available data—notably time. Here, by “clock granularity,” we mean clock as in processor clock that ticks periodically (different from the abstract concept of “timed automata clocks”). The authors of Reference [173] notably propose a modeling based on timed automata with stochastic conditions, and they provide sufficient conditions “for when one can achieve better security by increasing the grain of the clock.”

The authors describe stochastic systems with the help of probabilistic TAs and provide an algorithm for obtaining, given an attack scenario on a deterministic system, the timing channels along with their respective probability of occurrence. Models in TAs are given for attacks techniques from the literature (the one-pad technique, the clock-edge technique and the co-prime technique) [156]. No implementation is provided, but a manual proof on the case study is considered.

Gerking et al. [76] present a method reducing the verification of timed non-interference to a refinement check proposed in Reference [83], itself reducing to checking “testing automata,” i.e., reducing to reachability [6]. Their notion of timed non-interference is equivalent to the SNNI from Reference [74], where it is expressed as a bisimulation of the low-level behavior between the original automaton and the automaton with high-level actions disabled. For the refinement check, the automata of the latter is modified by adding a dedicated error location that is reachable if and only if timed non-interference is violated. If the error location is reachable on the parallel composition of the two automata, then the system is interferent. This technique, however, requires that the modified automaton is deterministic, resulting in the same expressive power as the work of Benattar et al. [35], where the SNNI verification problem is proved to be decidable on the dTA subclass (the automaton obtained by removing high-level actions must be deterministic). The method is used using UPPAAL on a model of a cyber-manufacturing system, where it is able to detect a potential timing channel attack, and we propose a mitigation by adding some temporal constraints on the system. The approach of non-interference based on reachable locations [30] is also used as a comparison, and is not able to detect the attack.

Finally, note that Reference [61] proposes a formal method-based approach not using TAs, and therefore not detailed in this survey, but not entirely unrelated either: Their approach identifies CTL formulas that express paths that could lead to timing attacks. Bounded model checking
Table 1. Comparison of Works on Timed Non-Interference

| Work                        | TNI property           | Class of models | Control | Decidable | Tool support |
|-----------------------------|------------------------|-----------------|---------|-----------|--------------|
| Lanotte et al. [118]        | Weak bisimulation      | Probabilistic TA| ×       | ✓         | ×            |
| Gardey et al. [74]          | Weak bisimulation      | TA              | ✓       | ×         | ×            |
| Benattar et al. [35]        | Weak bisimulation      | TA subclass (dTA)| ✓       | ×         | ×            |
| Barbucci et al. [29]        | TNI on language        | TA              | ×       | ×         | ×            |
| Lanotte et al. [119, 120]   | subset of TNI on Language | Probabilistic TA| ×       | ✓         | ×            |
| Gardey et al. [74]          | Cosimulation TNI       | TA              | ✓       | ✓         | ×            |
| Benattar et al. [35]        | TNI on language        | TA subclass (dTA)| ✓       | ✓         | ×            |
| Gerking et al. [72]         | TNI on language        | TA subclass (dTA)| ×       | ✓         | ×            |
| Barbucci and Yesel [30]     | TNI on reachable states| TA              | ×       | ✓         | ✓            |
| Gardey et al. [74]          | TNI on reachable states| TA              | ✓       | ✓         | ×            |
| André and Kryukov [19]      | TNI on reachable states| Parametric TA   | ×       | ✓         | IMITATOR     |
| Cassez [50]                 | Timed opacity on language | ERA             | ×       | ×         | ×            |
| Wang et al. [177]           | Timed opacity on language | TA subclass (RTA)| ×       | ✓         | ×            |
| Ammar et al. [16]           | Timed opacity on language (bounded) | non-Zeno TA | ×       | ✓         | SpaceEx     |
| André et al. [20]           | Timed opacity (execution times) | TA              | ✓       | ✓         | IMITATOR     |
|                           | Full timed opacity (execution times) | TA              | ✓       | ✓         | IMITATOR     |
|                           | Timed opacity emptiness | Parametric TA subclass | ×       | ✓         | IMITATOR     |
|                           | Full timed opacity (execution times) | Parametric TA subclass | ×       | ✓         | IMITATOR     |
| Nielson et al. [159]        | TNI on variables       | TA subclass     | ×       | ✓         | ×            |
| Vasilikos et al. [172]      | TNI on variables       | TA              | ✓       | ✓         | ×            |
| Vasilikos et al. [173]      | Clock granularity      | Probabilistic TA| ×       | ✓         | ×            |

Techniques can be applied, and UPPAAL (that takes TAs as input formalism) is cited as one of the possible target model checkers.

3.6 Discussion

Table 1 summarizes the results of each paper surveyed in this section. The second column indicates the kind of **timed non-interference (TNI)** property studied by the paper, while the third column indicates the class of models on which the work applies (dTA is a subclass such that the automaton obtained by removing high-level actions is deterministic [35]). “Finite-state automata” indicates that the work applies not only to TAs but to any kind of automata-based formalism. The last three columns indicate whether the approach supports controller synthesis, whether the property studied in the paper is decidable, and whether the work is supported by an automated model checking tool, respectively.

The framework of Reference [35] seems to be one of the most important works so far as it allows us not only to check non-interference on reachable states on TA and non-interference on accepted language for the dTA subclass but also to generate a controller ensuring such properties on the system.

Reference [177] allows us to check language-opacity on real-time automata, made possible by the strong syntactic restrictions on this subclass of TAs, while Reference [16] allows us to check location-based language opacity (over bounded time) on non-Zeno timed automata. Reference [172] allows us to check non-interference on variable flow on TAs. Reference [120] allows us to check a subset of non-interference on accepted language (non-deducibility on composition) on probabilistic sTA. Concerning parametric TAs, Reference [19] addresses the synthesis of timing parameters for which n-non-interference holds, while Reference [20] allows us to check a rather weak notion of timed opacity on TAs and on a subset of parametric TAs. Together, those works combine all that is currently expressible on timed non-interference.

A major outcome from Table 1 is that only few works benefit from an actual implementation. This is not only disappointing but also surprising, as the TA community came out with several efficient reasonably scalable model checkers—the leading one being UPPAAL, already used in a large
number of industrial contexts. Therefore, a gap seems to be still present between the theory of non-interference in the context of TAs ("which problem can be answered?"), and the actual automated verification of non-interference properties of timed systems using TA-based techniques.

Finally, note that only one work uses Uppaal, while others use tools applied to extensions of TAs: SpaceEx (that takes as input hybrid automata) and IMITATOR (that takes as input parametric timed automata). This may come from the fact that opacity problems are in general harder than reachability, and Uppaal is a model checker mainly targeting reachability properties.

4 ATTACK TREES

Attack trees [105] are a formalism inspired by fault trees [149, 174] that represents the possible behaviors of an attacker. An attack tree takes the form of a tree, where the root represents the goal of the attack. Each node, starting by the root, is refined with logical gates (such as "AND"/"OR") into sub-goals, until basic actions are reached, forming the leaves of the tree.

Example 4.1. Figure 8, taken from Reference [109, Fig. 1], depicts an attack tree modeling how one could get access to a software code. Stealing the code (OR gate) can be done in three different ways: Bribing, a Network Attack or a Physical robbery. Each of those possibilities is modeled by a SAND ("sequential AND") gate, meaning that each subgoal must be executed from left to right. For instance, successfully Bribing implies first that a programmer is bribed and then that the bribed programmer obtains the code.

Since 2015, a line of works started to use the theory (and tools) of timed automata to equip attack trees with timing information and model and verify them using TA software. In the following, we review several works encoding attack trees (and their extensions) into timed automata (and their extensions).

4.1 Attack-fault Trees

The first use of TAs as a target formalism for attack trees is described in Reference [109], where Kumar et al. extend attack trees with cost values for several resources, such as time. Their framework allows us to answer cost related security properties. For example, it is possible to know which attack path minimizes some resources, or what are the paths that are more harmful to the system. Pareto-optimal curves are used when several resources are to be optimized. A translation from the extended attack trees to priced timed automata [33] is described. A timed automaton is constructed for each node of the attack tree. The parallel composition of these TAs generates a priced timed automaton encoding the attack tree. The paper then provides guidelines on how to verify the model against Weighted CTL with Uppaal-CORA [1], an extension of Uppaal for priced TAs.
This first work is then extended to **attack-fault trees (AFTs)** in Reference [111]. Logical gates of both fault trees and attack trees are allowed in this formalism, thus increasing the expressiveness compared to standard attack trees. AFTs are translated to probabilistic TAs [31] in a similar way to that of Reference [109], and checking is performed with UPPAAL-SMC [59]. A case study of an oil pipeline from Reference [107] is analyzed.

Valluripally et al. [170] describe a framework for performing model checking on **Virtual Reality Learning Environment (VRLE)** [141], which is a part of the Internet of Things. The framework consists in modeling attack trees and translating them to probabilistic timed automata. Unlike Reference [109], each leaf node is transformed into an automaton that models the corresponding attack scenario. Model checking can be performed with UPPAAL-SMC to determine if the probability of disruption of the system is higher than a set threshold. If it is for some attack scenario, then modification of the system’s design can be proposed, resulting in a novel attack tree, and the process can continue until the system is considered safe. The method is illustrated on iSocial [194], a VRLE for youth with autism spectrum disorder, composed of VR headset devices, handheld controllers, and base stations. The probability of loss of information or privacy leakage can be studied either for each of the attacks scenarios present on the tree (by checking with the appropriate automaton) or by combining them. This allows us to observe that combining denial of service and unauthorized access always leads to a loss of information (a probability of 1), while combining unauthorized access and user location always leads to a privacy leakage. The implementation of some design principles (mainly consisting in limiting authorizations to strict functionality) results in a significant reduction in both loss of information and privacy leakage.

Finally, in Reference [9], Ali translates a so-called extension of attack trees named “simplified timed attack trees” into a network of weighted timed automata. UPPAAL is the underlying model checker. No automated translation tool seems to be available, and the novelty compared to the aforementioned works (notably Reference [109]) remains unclear.

**Testing IoT systems using attack trees.** Krichen and Alroobaea [108] propose a framework for formal testing IoT systems in a realistic environments, using attack trees. It borrows the modeling and translation to priced timed automata from Reference [109]. The resulting network of automata is used as an input for generating abstract test cases. Those test scenarios are then translated to the **Testing and Test Control Notation version 3 (TTCN-3)** [168] specification language for test execution. The tests are run on a cloud testing architecture that will output verdicts according to the TTCN-3 language.

**Adding parameters.** André et al. [22] further extend [111] by devising a method synthesizing the set of all timing parameter and cost parameter valuations leading to a successful attack. To do so, the AFTs are modeled with parametric weighted timed automata that is an extension of parametric timed automata where transitions are associated with possibly parametric (discrete) costs. This formalism allows multiple weights, which can be incremented by constants or variables values. It does not, however, consider stochastic information. The compiler ATTop [110] provides an automated translation from AFTs to parametric weighted timed automata. Similarly to previous approaches, each node of the tree is modeled using a parametric weighted timed automaton. The resulting model can be checked with IMITATOR. The method is used on two case studies.

### 4.2 Attack-defense Trees

Meanwhile, Gadyatskaya et al. [73] extend the attack-defense tree formalism [103, 104] (i.e., attack trees including defender’s actions) with a timed semantics. Stochastic behavior and cost of actions are also added to the semantics. A translation algorithm that generates a network of probabilistic TAs is provided. Unlike Kumar et al. [109], this work actually models the system using the classical
approach where each participant is modeled by an automaton. The attack-defense tree goals are translated into Boolean formulas, the attacker’s goal being to reach a state where the Boolean formula encoding the attack-defense tree is true. The probabilistic extension of timed automata is used both to answer queries on the probability of some event and to model a cost on the actions taken by the attacker. Model checking is done with Uppaal-Stratego \cite{58} and can answer queries on cost or probabilities of an attack, along with timed constraints. The method is illustrated on an example, which allows us to deduce expecting cost of attack.

This work is then extended by Hansen et al. \cite{82}. The cost of attacks is modeled as a function of time, depicting a more realistic system. Also, the attacker is now parametric in a sense that a range of valuations will be checked for parameters defining the probability of non-deterministic behaviors. Analysis of variance (a method allowing us to compare the effect of discrete factors on continuous variables) is proposed to identify the parameter valuations that minimize the cost of an attack. This time, model checking is performed with Uppaal-SMC.

Hermanns et al. \cite{86} propose a formalism that is more expressive than attack-defense tree: attack-defense diagrams (ADDs). Those allow cycles, capturing the dynamic aspects of attack scenarios. A node in an ADD represents not only the actions of the attacker/defender but also independent timed events. The attacker’s goal is to reach a sink node (without outgoing edges) that represents the success of the attack. Likewise, the defender’s goal is to reach another sink node that represents the failure of the attack. Each node of the resulting graph is translated into a probabilistic timed automaton that models either a timed event or a logical gate corresponding to an action. Model checking is performed on the parallel composition of the resulting automata using the stochastic model checking tool Modest \cite{41}. It allows us to check properties expressing cost, time, or probability of success, and the process is illustrated on an ad-hoc case study.

Arias et al. \cite{26} adapt the translation from Reference \cite{22} to attack-defense trees, aiming in particular at modeling coalitions of agents. Attack-defense trees are extended with agents and new gates (such as the SCAND—for sequential AND gate with attacks and defenses as children). The semantics is given in terms of extended asynchronous multi-agent systems (EAMAS), which extend AMAS \cite{92} with attributes in local transitions. Attack-defense trees are translated into the EAMAS formalism by turning each node in the tree as an agent.

A dedicated tool adt2amas\footnote{https://lipn.univ-paris13.fr/adt2amas/} takes as input an attack-defense tree and automatically generates the corresponding EAMAS model. The tool then translates that EAMAS model into modgraph or into a network of parametric timed automata in the IMITATOR format. Case studies illustrate how the method can answer questions regarding cost and time of attacks.

Petrucci et al. \cite{145} extend this latter work with a reduction technique dedicated to trees, allowing for a faster checking process. It consists in using the layered structure present in a tree, in particular an attack tree, to ignore some of the interleavings when exploring the tree. This latter work was also integrated into the adt2amas tool (with option \texttt{--layered-reductions}).

4.3 Discussion

Table 2 summarizes the results of each paper presented in this section. Columns 3, 4, and 5 (from left to right), respectively, indicate whether the tree formalism supports probabilities, cost and/or timing parameters, and agents. Note that the work from Reference \cite{86} not only allows modeling defense aspects but also introduces cycles and new logical gates to the formalism. The sixth column indicates the class of models used as a target formalism, which relates to the level of expressiveness of the properties studied. The rightmost column gives the tools used to perform automated verification, if any.
Table 2. Comparison of Works Using TA as Target Formalism for Attack Trees

| Work                        | Input formalism       | Probabilities | Parameters | Agents | Target formalism       | Tool support         |
|-----------------------------|-----------------------|---------------|------------|--------|------------------------|----------------------|
| Kumar et al. [109]          | Attack-fault trees    | ×             | ×          | ×      | Priced TA              | Uppaal-CORA          |
| Kumar and Stoelinga [111]   | Attack-fault trees    | ✓             | ×          | ×      | Probabilistic TA       | Uppaal-SMC           |
| Valluripally et al. [170]   | Attack-fault trees    | ✓             | ×          | ×      | Stochastic TA          | Uppaal-SMC           |
| André et al. [22]           | Attack-fault trees    | ×             | ✓          | ×      | Parametric weighted TA | IMITATOR             |
| Ali [9]                     | Attack-fault trees    | ×             | ×          | ×      | Weighted TA            | Uppaal               |
| Gadyatskaya et al. [73]     | Attack-defense trees  | ✓             | ×          | ×      | Probabilistic TA       | Uppaal-SMC           |
| Hansen et al. [82]          | Attack-defense trees  | ✓             | ×          | ×      | Probabilistic TA       | Uppaal-SMC           |
| Petrucci et al. [26, 145]   | Attack-defense trees  | ×             | ✓          | ✓      | Parametric TA          | Uppaal/IMITATOR      |
| Hermanns et al. [86]        | Attack-defense diagrams | ✓               | ×          | ×      | Probabilistic TA       |                       |

Using timed automata as a target formalism for attack trees is quite recent; in contrast, the theoretical works on opacity surveyed in Section 3 started as early as 2002. So far, we identified two branches (attack-fault trees and attack-defense trees) whose expressive power differs mainly in terms of the type of the supported logical gates. Concerning attack-fault trees, Reference [111] models stochastic behaviors but does not perform parameter synthesis, while Reference [22] performs parameter synthesis but does not support probabilistic automata. On the topic of attack-defense trees, Reference [86] models stochastic behaviors in the more expressive attack-defense diagrams. However, Reference [26] performs parameter synthesis and models several agents. It is worth mentioning that attack-defense trees were also translated to other (non-TA) timed formalisms, notably to continuous-time Markov chains (e.g., References [93, 127]).

Contrarily to works related to non-interference properties (Section 3), the works surveyed in this section on attack trees are clearly well supported by existing model checkers. With one exception, all of the works in Table 2 use a target model checker, with some reported experiments. Several of these works also feature an automated translation from the input attack tree into the target formalism (i.e., TAs or extensions of TAs).

As a summary, this direction of research seems to be very dynamic, and a clear future work will be to unify the various approaches (featuring faults, defense, agents, parameters, probabilities...) into a unique and fully automated translation.

5 APPLICATIONS IN THE CONTEXT OF CYBER-PHYSICAL SYSTEMS

In this section, we review works where timed automata or their extensions are used to solve concrete problems related to security and that were not covered by the previous two sections of this survey.

Outline of the section. We first review works aiming at formalizing timed extensions of RBAC models (Section 5.1); we then review works related to the verification of controllers in a security context (Section 5.2); we then review a large line of quite different works related to security and communication protocols (Section 5.3); and we finally review some remaining works (Section 5.4) and propose some discussion on these works (Section 5.5).

5.1 Analyses of RBAC Models

Several works performed analyses of RBAC models using timed automata.

Mondal and Sural [134] analyze a TRBAC model [37], the timed version of a Role Based Access Control model [152]. An RBAC system is composed of three components (role, permission, and user). A formal translation from TRBAC to TAs is provided, where each role is modeled by an automaton, along with a controller automaton that manages the activation/deactivation of these roles. To reduce the size of the state space, the model features only one clock. This leads to a fast verification time, but the resulting model is a high-level abstraction of a real TRBAC. Security properties are checked using CTL queries with UPPAAL, ensuring the correct behavior of the system.
Mondal et al. [135] extend Reference [134] to GTRBAC [94], an extension of TRBAC allowing us to differentiate between enabled role (ready for user assignment) and active role (assumed by at least one user). Also, a given user can take on multiple roles. They provide a framework such that a given GTRBAC system is automatically translated into a timed automata network, and CTL queries are generated according to a set of desired properties. The resulting network of timed automata is proved to possess all the properties of the GTRBAC model in which the authors are interested. To face the state space explosion issue, the authors provide some abstractions that may be useful to reduce the size of the model, depending on the property to verify. Experiments with Uppaal are used to illustrate the impact of those abstractions on the size of the state space.

Geevalla et al. [75] use model checking to analyze spatio-temporal RBAC [148] policies. Unlike GTRBAC models, those allow us to model both temporal and spatial aspects of the system. As a proof of concept, the authors model a banking application with timed automata. A property of separation of duty between role (i.e., that a given user cannot be assigned two exclusive roles at the same time) is verified, and a trace of execution that violates such a property is given. The system is quite small and scalability is not discussed. The translation is performed manually—automating the process being left as future work.

Vasilikos et al. [171] propose a method for checking time-dependent information flow on distributed systems, typically for RBAC models. The authors describe a branching-time logic extending CTL, called BTCTL, allowing comparisons between sets of variables, as well as checking if a configuration of variables holds before or after a given event. Given a distributed system modeled with TAs and queries expressed using BTCTL, an algorithm is provided that syntactically modifies the model and queries so that they can be expressed in the subset of CTL supported by Uppaal. Execution traces are proved to be equivalent. A translator from BTCTL to the Uppaal CTL fragment has been implemented by the authors, but no experiments are reported.

5.2 Verification of Controllers

Acampora [5] introduces **TA-based Fuzzy Controllers (TAFC)**, a kind of fuzzy controller [160] modeled by timed automata extended with fuzzy variables. The controller allows for the detection of network intrusions, and it is built based on data collected from routers from the network. The process is used on a denial of service attack, and the generated controller is then compared to a classic, untimed fuzzy controller. The results show that the TAFC makes better prediction than its untimed counterpart.

Wang et al. [178] study a networked water level control system with both numerical and physical aspects. The different parts of the systems are modeled with TAs, along with the controller of the system, and an attacker, each in the form of an automaton. Some security properties are expressed using LTL and checked with the PAT model checker [126]. The results highlight that the system is secure as long as the frequency of the monitoring of the system is greater than or equal to the frequency of the attacks. The method is very ad hoc, and the paper is therefore rather a case study.

Alshalalfah et al. [10] study the resilience of controllers of artificial pancreas. Those devices allow for an automated regulation of glucose level. The system features wireless communication between an insulin pump and a controller that makes decisions about insulin injection. The components of the system are modeled with priced timed automata, including an attacker monitoring the actions of the controller. The attacker can perform a replay attack by intercepting messages from the controller and re-sending them later, disturbing the actions of the controller. Two controllers from the literature are modeled, and stochastic properties are verified with Uppaal-SMC. The first studied controller shows that when attacked, it is still able to maintain safe glucose level in most scenarios, which is not the case for the second one.
5.3 Verification of Security Protocols

Checking formally the correctness of security protocols is an important yet difficult matter (see, e.g., Reference [112]). In addition, verification can be even more challenging in the presence of timing information. We review here works related to the verification of security protocols using (extensions of) timed automata.

Reference [55] and extensions. A method for modeling security protocols via a network of timed automata is proposed for the first time by Corin et al. [55, 56]. The resulting model considers the timed aspects of such protocols, allowing us to model not only timeouts but also the possible actions following a timeout, such as the retransmission of a message. The behavior of each participant, including the intruder, is modeled by an automaton. The formal verification of security properties is done using Uppaal. An example is given with a simplified version of the Needham-Schroeder protocol [128, 136] where a Dolev-Yao intruder [65] successfully intercepts the message if the responder timeout is long enough. The method is then applied to prove that retransmissions are secure on the Yahalom protocol [46]. Only protocols requiring timeouts are considered.

Jakubowska et al. [91] propose a method similar to Reference [55] for modeling security protocols. Timed automata (extended with global variables) are used to model security protocol such that each participant is modeled by an automaton. Unlike Reference [55], this work gives a more formal way of specifying security protocols. While this method does not consider timeout, it models timestamps [137], however. Authentication and integrity properties can be checked either using partition refinement with Kronos or SAT bounded model checking with VerICS, which may return traces of execution leading to states violating such properties. The method is then applied to the Kerberos protocol [34] and to the Wide Mouthed Frog protocol (WMP) [46] and identifies a violation of authentication in both cases. Model checking is used to prove that such an attack is not possible anymore when testing freshness of the session keys. Jakubowska and Penczek [90] extend Reference [91] in various ways. Timeouts are now considered and timed aspects of timing attacks are studied on the Andrew Secure RPC protocol [155], the Telecommunications Management Network protocol [151] and the Needham-Schroeder protocol [136]. In particular, its is described how well-chosen timeout values can prevent an attack on the Needham-Schroeder Public-Key (NSPK) that was described in Reference [128].

Benerecetti et al. [36] propose a framework, named TPMC, for checking security properties of timed security protocols. TPMC extends the specification language HLPSL [27], an established language in the security protocol community. A timed version of HLPSL (namely “THLPSL”) is used to model protocols, and an automatic translation to UPPAAL TAs is provided. The method is exemplified on the Wide Mouthed Frog protocol. The way systems are modeled and verified is quite similar to the work by Corin et al. [56], with the advantage of proposing a formalism that might be easier to use for the security community.

Kurkowski and Penczek [114] use bounded model checking to verify security protocols modeled using TAs. (Bounded model checking for (extensions of) timed automata was considered in, e.g., References [98, 99, 188]). This work is relatively close to Reference [55] and extends to timed automata a previous work by the same authors on finite-state automata [113]. In Reference [114], a formal specification language for timed security protocols is thoroughly described. Knowledge of participants is part of the model, and attacks correspond to states of the system where an intruder possesses certain knowledge. A translation to a network of TAs is then proposed. Checking is done using SAT-based bounded model checking with VerICS. The method is used on several types of security protocols, including WMF, Kerberos, and NSPK and is able to capture known attacks on authentication and secrecy. The computation times are compared to the TMPC framework and show better overall results.
SAT- and SMT-based verification. We review here a line of works by Szymoniak, Kurkowski, and co-authors. First, in Reference [164], the objective is to analyze execution times and to exhibit a time ("timeout") during which a protocol can be executed. An implementation has been carried out, allowing us to “synthesize” a secure execution time for a protocol. Experiments were performed on a modification of the Needham Schroeder authentication protocol [136]. This work is not TA based, but it shares similarities with the TA-based work by André et al. [20], where the authors also synthesize timing parameter valuations (such as timeout) to provide secure execution times for a system.

Szymoniak et al. [165] apply their method from Reference [164] to the WooLamPi protocol [186]. Models of timed automata are generated using a compiler from their previous work and then translated to a Boolean formula to be verified using the SAT-solving tool MiniSAT [167]. This allows us to find a dependency between latency values that forbid man-in-the-middle attack.

Then, in Reference [191], Zbrzezny et al. propose a bounded model checking approach based on Satisfiability Modulo Theory (SMT). The underlying model is networks of communicating timed automata, and the modeling is inspired by References [115, 163]. The experiments in Reference [191] reuse the properties detailed in Reference [164]. Experimental results on several common protocols (NSPK [136], a timed version of Needham-Schroeder Symmetric Key [191, Section 2], WooLamPi [186], and WMF [46]) show that the SMT approach is faster than the previous SAT approach. In particular, their implementation on the SMT-solver Yices2 [66] is at least 10 times faster than its SAT counterpart (implemented on the SAT-solver MiniSAT). An artifact for reproducibility is given.

This former work [191] is the partial basis of VerSecTis [193], a tool dedicated to the verification of timed security protocols with SMT-based techniques. Here, protocols are modeled as Timed Interpreted Systems [187, 192] instead of network of timed automata. Again, instructions to reproduce experiments are available online.

Szymoniak et al. [166] build on previous works [90, 115] and perform a formal modeling and timed analyses of the WooLamPi protocol and of the Sensor Network Encryption Protocol [144]. Reachability using SMT is used to identify the time frames during which a man-in-the-middle behavior (i.e., a passive interception of the data) is possible.

This line of work is continued into Reference [162], where simulations are used to test the influence of time parameters on the correctness of security protocols. Timed automata are, however, not used as the underlying model.

Other works. Koltuksuz et al. [102] describe how to model and verify security protocols using TAs. The method shares similarities with Reference [56], each participant being modeled as an automaton. An example is given using the Neuman-Stubblebine repeated authentication protocol [137] with a Dolev-Yao intruder; an extensive verification is performed using UPPAAL.

Szymoniak et al. [163] propose a method for modeling security protocols with a particular focus on the latency in the network. Beyond the specificity of the networks delays, the approach is quite classical, describing honest users and intruders behaving on a shared network, modeled with TAs. There are no experimental results, and the latency of the communications is modeled by adding some fixed value to the timed constraints of automata.

Piech and Grodzki [146, 147] address the problem of dynamic security estimation and aim at providing algorithms that detect threats and modify attributes of a communication protocol in real time to adapt to the risks. To do so, they model a given communication protocol using a probabilistic timed automaton, such that each location in the automaton corresponds to a security state (i.e., a value modeling the security attributes of the system). However, this TA is only used as a means of simulating the execution of a given communication protocol to show the results
of the provided algorithm. In Reference [147], Petri nets are used as an additional formalisms; experiments are performed, but it remains unclear using which framework.

Siedlecka [157] models security protocols with probabilistic timed automata. The paper extends [158], a previous work where security protocols were modeled with (untimed) probabilistic automata. This work considers both the timed aspects of the protocols (such as the timeframe of tasks or the communication delays) and the probabilistic aspects inherent to the success of the attacks. The framework allows us to know the probability of success of the attacker at a given step of the protocol. A tool has been implemented and is used on the security part of the MobInfoSec protocol [89] as well as on the NSPKL protocol [129].

Finally, in Reference [125], Li et al. extend the formalism of applied $\pi$-calculus [2] with parametric timed constraints. Using this formalism, the authors model security protocols along with an adversary, and they synthesize parameter valuations for which the system is secure. As the expressiveness of such modeling seems close to that of parametric timed automata, it suffers from the same drawbacks: Their algorithms offer no guarantee of termination. Furthermore, timestamps can be expressed in this formalism but not timeouts. The results have been implemented in a tool named Security Protocol Analyzer, which performs an on-the-fly verification of the state space, stopping if no parameter valuations satisfy the property, or whenever the entire state space has been explored. The tool is run on several security protocols, which finds a previously undiscovered time related attack on Kerberos V (i.e., the latest version of this protocol). In Reference [124], the authors use the same method to consider clock drifts (i.e., inaccuracies in clock rates). They show that the TESLA protocol [143] is safe when assuming precise timed constraints but might not be once clock drifts occur. These two works [124, 125] do not consider timed automata, but the methods used share similarities with parameter synthesis for parametric timed automata.

Case studies. Godskesen and Gryn [79] briefly report on the formal verification of the authenticated routing for the ad hoc networks (ARAN) protocol [153]. A TA model of the network, including an attacker, has been implemented and verified with Uppaal, allowing us to confirm an already known attack along with the (also known) correction needed to prevent it. No model, queries nor results are shown.

Tobarra et al. [169] describe an analysis of routing protocols in wireless sensor networks [8]. In this type of infrastructure, an intruder can intercept or even modify messages between nodes of the network. The attacker approach is used on the $\mu$-Tesla authentication protocol [144], which is modeled with timed automata. However, the intruder is not modeled with an automaton but with variables denoting its knowledge, along with functions allowing it to increase its own knowledge on regard of the information it already have access to. Checking is claimed to have been done with Uppaal, but no queries or results are provided.

Andrychowicz et al. [24] model and verify security of Bitcoin contracts. Two contract protocols from Reference [25] are verified: the timed commitment scheme and the simultaneous commitment scheme. A model is given in timed automata for the blockchain algorithm and two users. Both users are in turn considered as dishonest. The resulting network is checked with Uppaal and conditions for validation of security properties are given (typically the respect of some maximum time limit before timeout).

Saffarian and Rafe [150] apply model checking using Uppaal and timed automata on Mobile Internet Protocol version 6 (MIPv6) [142]. Through some level of abstraction, the protocol is modeled, including its real time aspects. Aside from participants, the network and a Dolev-Yao intruder are modeled by one automaton figuring the unprotected communication media. Formal verification cover the security of MIPv6 route optimization and identifies already-known attacks scenarios, along with a new one that was not studied previously.

ACM Computing Surveys, Vol. 55, No. 6, Article 127. Publication date: December 2022.
Lu and Sun [130] study the four-way handshake protocol of the IEEE 802.11i standard. This kind of protocol is used for authentication between two parties. Each participant, including an attacker trying to perturb communications with a denial of service attack, is modeled with a TA. Using UPPAAL, it is formally proved that the key generated during the authentication will be received as planned. However, the specification of the system implies a high level of abstraction that may impact its realism.

Discussion. In general, security protocols are modeled with networks of timed automata such that each participant or component is modeled by one automaton. Among the surveyed frameworks, Reference [157] is the only framework coping with both timed and probabilistic aspects. Probabilities set aside, Reference [191] is among the most expressive as it considers timeouts and timestamps and models the knowledge of the agents. Still, it is worth mentioning the framework TPMC from Reference [36], as it features a specification language that was designed to be easy to use for the security community.

Finally, none of the works consider a parameterized number of participants. In other words, it may happen that some protocols were proved correct for a given number of participants (two, typically), but some larger number of participants might introduce security breaches.

5.4 Other Works

We finally review some miscellaneous works that did not fit in the structure of the survey.

Burmester et al. [45] study security in cyber-physical systems, with multiple attack vectors, i.e., both on the discrete ("cyber") and on the continuous ("physical") aspect of the system. The system is modeled with hybrid automata [85], a formalism that is commonly used when it comes to modeling physical aspects of a system; TAs can be seen as a subclass of hybrid automata. In this work, the intruder is based on the Byzantine faults model [64], where the adversary controls the communication channels of the system. The considered case study is the Russia–Ukraine natural gas grid, a communication network managing the flow of gas in a grid linking Russia to Europe. A flow verification protocol is given that allows Ukraine to verify the correctness of flows while guaranteeing privacy of Russia’s information. The security properties are formally proved using existing cryptographic methodologies.

Sun et al. [161] apply model checking techniques in the context of the Internet of Things, specifically Electronic Product Code networks [154]. They model a supply chain management system using a timed automata network. Various potential attacks are modeled, mainly related to Radio Frequency IDentification tags. The verification is performed with UPPAAL on a concrete system instance and establishes a safe refresh frequency for the detection of tags.

David et al. [60] study socio-technical attacks using formal verification. In particular, a case study of the home payment service of an internet TV service is presented. This service allows the user’s payment card to be read by a card reader built into the TV remote. Typically, an attacker can trick the user to install malicious software. The authors use TAs to model the physical actions possible for the attacker (such as scamming or physically intruding) and for the user (such as enabling a localization or losing the credit card). UPPAAL is used to perform formal verification, and several attacks scenarios are captured that way.

5.5 Discussion

The works surveyed in this section cover multiple application domains. Many of the aforementioned verification techniques reduce to reachability checking, and therefore UPPAAL is often the model checker of choice. In addition, over the past three decades, numerous works addressed the safety of protocols, by modeling these protocols using timed automata, and carrying automated...
verification, often with UPPAAL; it is therefore not surprising that the security of protocols is an application domain of predilection for timed automata.

In terms of concrete application domains, many works addressed application domains related to networks and communications. This is unsurprising considering that security issues are mainly considered in communicating contexts, when intruders can observe or interfere with communications between legitimate stakeholders. More surprising is the very small number of works (mainly [10]) targeting health applications. Digital devices used in a health environment are typical examples of cyber-physical systems; in addition, their functional correctness and security should be of utmost importance considering their safety-critical nature. One may wonder why so few works were performed in this area: Perhaps these communities are not so interested in formal verification (which would be worrying); perhaps the systems are too expressive for being modeled by timed automata (e.g., continuous variables more expressive than simple TA clocks might be necessary); perhaps those systems are too large to be verified by model checking and their designers failed in designing well-balanced abstractions.

6 CONCLUSIONS

In this article, we surveyed works using timed automata as a formalism for expressing, studying or formally verifying various categories of security properties. We notably collected a number of works focusing on theoretical aspects of non-interference and opacity properties, on attack trees and their extensions, on RBAC models, and on various security protocols. In the following, we draw some high-level conclusions and discuss perspectives.

6.1 Theoretical Issues

The discussed theoretical aspects mostly concern opacity and non-interference. They naturally collide with the known (un)decidability results for TAs and their subclasses such as ERAs. It seems that the undecidability of the timed language inclusion in timed automata is a major point in the study of opacity in timed automata. While timed opacity is undecidable for the language for the full class of TAs (and even for ERAs) [50], decidability can be envisioned by weakening the attacker model (that could only measure the full execution time [20]), by restraining the formalism [177] or by restraining the problem by adding a time bound [16]—to go back to the decidability of time-bounded language inclusion checking for TAs.

In addition, extending these problems to parametric timed extensions of TAs, such as the U-PTAs or L-PTAs [43], known for their partially decidable properties (e.g., References [21, 43]), seems an interesting future work.

6.2 Links with Applications

Modeling with TAs. Among the theoretical works on security properties for TAs (mostly surveyed in Section 3), a blind spot seems to be the relationship with concrete applications. Not only few of the surveyed works are equipped with implementations, but the relationship with more concrete or higher-level formalisms is virtually not discussed. That is, while timed automata represent an interesting formalism, and while studying security properties on this formalism is of interest by itself, nevertheless relating these models and properties with actual systems (programming languages, protocols...) was not discussed at all in the works surveyed in Section 3. This is even more frustrating as Sections 4 and 5 convince us that TAs are a suitable formalism to verify security properties in actual systems.

Translations from applications (such as code) to timed automata would be highly interesting, so as to use model checking techniques for TAs as a way to prove opacity of actual programs—or other areas of applications.
Successful research areas. Perhaps surprisingly, two research directions were thoroughly studied (and are still ongoing), namely the translation of attack trees and their extensions, as well as of RBAC models, to (extensions of) timed automata. These works are often complete, in the sense that the input formalism is usually completely supported, and most works come with an automated translation, and a target model checker supporting (extensions of) TAs.

On the opposite, health applications are surprisingly virtually absent from the surveyed works (see discussion in Section 5.5).

Extensions. We mostly limited ourselves to “vanilla” timed automata and considered few extensions (mostly probabilistic, and parametric), mainly because they appeared reasonably frequently in the surveyed works. Other formalisms of interest, potentially used in the context of formally assessing security properties, include hybrid automata [85] or continuous-time Markov chains [127].

Tool support. UPPAAL, which is one of the most popular tools for the verification of TAs, is the most widely used tool among the applications surveyed in Section 5, which can be seen as unsurprising. More surprising, UPPAAL is not much used in the context of attack trees, perhaps due to the need of probabilistic and parametric extensions needed by the surveyed attack tree frameworks. Also surprising at first, UPPAAL is not much used in the context of non-interference and opacity, mainly because these notions are close to language inclusion, a problem undecidable for TAs, while UPPAAL mainly focuses on reachability.

We believe the community should develop new tools efficient at language inclusion for either decidable subclasses (such as event-recording automata, or one-clock automata), or for the whole class of TAs, with the risk that the algorithms do potentially not terminate (this is notably the paradigm of the algorithm proposed in Reference [179], with a very frequent termination in practice).

Tool reliability. A discussion quite absent from the surveyed works is that of the tool reliability, either because of tool failures (due to bugs), or due to modeling issues. For example, in Reference [112], Kurkowski et al. showed that some incorrect protocols can be verified positively by some tools (e.g., TA4SP). Modeling issues can involve Zeno behaviors, i.e., an infinite number of discrete actions in a finite time, or deadlocks or timelocks. While techniques exist to avoid these problems in timed automata (e.g., References [80, 180]), maybe not all surveyed works considered these issues. In addition, the guaranteed absence of bugs in timed automata model checkers remains a real challenge. Only few recent works attempted at a certified verification, notably by Wimmer et al. and the Munta verified model checker for timed automata [182–185]. These works do not seem to have been used for proving security protocols.

Reproducibility. The surveyed works describing implementations and experiments do generally not provide their code, models and experiment results on long-term archiving venues. This may result in loss of data in a couple of years. This is not specific to this area of research but is even more unfortunate, as most data could be archived very easily (usually no sensitive data, reasonably sized models and results, etc.). Fortunately, some recent exceptions do provide online benchmarks with all necessary information for reproducibility (e.g., References [20, 191, 193]).

6.3 Toward Parameterized Verification Using Timed Automata?

A research domain notably absent of our survey on timed automata is that of parameterized models, whereas some works considered timing parameters (e.g., References [19, 20, 26, 125]), parameterized models in terms of number of participants, are absent from the surveyed works. In other words, it may happen that some protocols surveyed in this article were proved correct for a fixed number of participants but could turn insecure for larger number of participants.
This problem was tackled in various areas of security—but, to the best of our knowledge, not using timed automata. This problem was for example tackled in Reference [54] for a class of (untimed) protocols, by projecting all honest users onto one agent and all attackers onto another one—it is therefore sufficient to consider only two agents. Achieving something similar in timed automata is far from trivial as, depending on the modeling choices, we might need to equip each agent with its own clock, which creates an unbounded number of clocks.

Nevertheless, there is a relatively long line of works studying the decidability of "timed network protocols," i.e., networks made of an arbitrary number of timed automata, communicating with various paradigms and restricted by various constraints, such as the number of clocks per agent (e.g., References [3, 4, 38]). An explanation why these results have not been used in the context of security may come from the fact that virtually none of these works led to an implementation.

ACKNOWLEDGMENTS

We thank anonymous reviewers for their useful comments, as well as Jaime Arias and Laure Petrucci for a feedback on their recent works.

REFERENCES

[1] [n.d.]. Uppaal CORA. Retrieved from https://docs.uppaal.org/extensions/cora/.
[2] Martín Abadi, Bruno Blanchet, and Cédric Fournet. 2018. The applied pi calculus: Mobile values, new names, and secure communication. J. ACM 65, 1 (2018), 1:1–1:41. https://doi.org/10.1145/3127586
[3] Parosh Aziz Abdulla, Giorgio Delzanno, Othmane Rezine, Arnaud Sangnier, and Riccardo Traverso. 2016. Parameterized verification of time-sensitive models of ad hoc network protocols. Theor. Comput. Sci. 612 (2016), 1–22. https://doi.org/10.1016/j.tcs.2015.07.048
[4] Parosh Aziz Abdulla and Bengt Jonsson. 2003. Model checking of systems with many identical timed processes. Theor. Comput. Sci. 290, 1 (2003), 241–264. https://doi.org/10.1016/S0304-3975(01)00330-9
[5] Giovanni Acampora. 2010. Exploiting timed automata-based fuzzy controllers and data mining to detect computer network intrusions. In FUZZ-IEEE. IEEE, 1–8. https://doi.org/10.1109/FUZZY.2010.5584893
[6] Luca Aceto, Patricia Bouyer, Augusto Burgueño, and Kim Guldstrand Larsen. 2003. The power of reachability testing for timed automata. Theor. Comput. Sci. 300, 1–3 (2003), 411–475. https://doi.org/10.1016/S0304-3975(02)00334-1
[7] Luca Aceto, Anna Ingólfsdóttir, Mikkel Lykke Pedersen, and Jan Poulsen. 2000. Characteristic formulae for timed automata. RAIRO Theor. Inf. Appl. 34, 6 (2000), 565–584. https://doi.org/10.1051/ita:2001031
[8] Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. 2002. Wireless sensor networks: A survey. Comput. Netw. 38, 4 (2002), 393–422. https://doi.org/10.1016/S1389-1286(01)00302-4
[9] Aliyu Tanko Ali. 2021. Simplified timed attack trees. In RCIS. Lecture Notes in Business Information Processing, Vol. 415, Samira Si-Said Cherfi, Anna Perini, and Selmin Nurcan (Eds.). Springer, 653–660. https://doi.org/10.1007/978-3-030-75018-3_49
[10] Abdel-Latif Alshalalfah, Ghaith Bany Hamad, and Otmane Aît Mohamed. 2019. Towards system level security analysis of artificial pancreas via UPPAAL-SMC. In ISCAS. IEEE, 1–5. https://doi.org/10.1109/ISCAS.2019.8702440
[11] Rajeev Alur, Costas Courcoubetis, and David L. Dill. 1993. Model-checking in dense real-time. Inf. Comput. 104, 1 (1993), 2–34. https://doi.org/10.1016/0019-9916(93)90010-8
[12] Rajeev Alur and David L. Dill. 1994. A theory of timed automata. Theor. Comput. Sci. 126, 2 (April 1994), 183–235. https://doi.org/10.1016/0304-3975(94)90010-8
[13] Rajeev Alur, Limor Fix, and Thomas A. Henzinger. 1999. Event-clock automata: A determinizable class of timed automata. Theor. Comput. Sci. 211, 1–2 (1999), 253–273. https://doi.org/10.1016/S0304-3975(97)00173-4
[14] Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. 1993. Parametric real-time reasoning. In STOC, S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal (Eds.). ACM, New York, NY, 592–601. https://doi.org/10.1145/167088.167242
[15] Rajeev Alur and P. Madhusudan. 2004. Decision problems for timed automata: A survey. In SFM-RT. Lecture Notes in Computer Science, Vol. 3185, Marco Bernardo and Flavio Corradini (Eds.). Springer, 1–24. https://doi.org/10.1007/978-3-540-30800-9_1
[16] Ikhlass Ammar, Yamen El Touati, Moez Yeddies, and John Mullins. 2021. Bounded opacity for timed systems. J. Inf. Secur. Eng. 66 (September 2021), 1–13. https://doi.org/10.1007/jisesa.2021.102926
[17] Étienne André. 2019. What’s decidable about parametric timed automata? Int. J. Softw. Tools Technol. Transf. 21, 2 (April 2019), 203–219. https://doi.org/10.1007/s10009-017-0467-0
Timed Automata as a Formalism for Expressing Security

[18] Étienne André. 2021. IMITATOR 3: Synthesis of timing parameters beyond decidability. In CAV. 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

[19] Étienne André and Aleksandr Kryukov. 2020. Parametric non-interference in timed automata. In IEECCS, Yi Li and Alan Liew (Eds.). 37–42. https://doi.org/10.1109/IEEECS51672.2020.00012

[20] 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.

[21] Étienne André, Didier Lime, and Mathias Ramparison. 2018. TCTL model checking lower/upper-bound parametric timed automata without invariants. In FORMATS. Lecture Notes in Computer Science, Vol. 11022, David N. Jansen and Pavithra Prabhakar (Eds.). Springer, 1–17. https://doi.org/10.1007/978-3-030-00151-3_3

[22] Étienne André, Didier Lime, Mathias Ramparison, and Marielli Stoeinga. 2021. Parametric analyses of attack-fault trees. Fundam. Inf. 182, 1 (30 September 2021), 69–94. https://doi.org/10.3233/FI-2021-2066

[23] Étienne André, Dylan Marinho, and Jaco van de Pol. 2021. A benchmarks library for extended timed automata. In TAP. Lecture Notes in Computer Science, Vol. 12740, Frédéric Louergue and Franz Wotawa (Eds.). Springer, 39–50. https://doi.org/10.1007/978-3-030-79379-1_3

[24] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Łukasz Mazurek. 2014. Modeling bitcoin contracts by timed automata. In FORMATS. Lecture Notes in Computer Science, Vol. 8711, Axel Legay and Marius Bozga (Eds.). Springer, 7–22. https://doi.org/10.1007/978-3-319-10512-3_2

[25] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Łukasz Mazurek. 2016. Secure multiparty computations on bitcoin. Commun. ACM 59, 4 (2016), 76–84. https://doi.org/10.1145/2896386

[26] Jaime Arias, Carlos E. Budde, Wojciech Penczek, Laure Petrucci, Teofil Sidoruk, and Mariëlle Stoelinga. 2020. Hackers vs. security: Attack-defence trees as asynchronous multi-agent systems. In ICFEM. Lecture Notes in Computer Science, Vol. 12531, Shang-Wei Lin, Zhe Hou, and Brendan Mahoney (Eds.). Springer, 3–19. https://doi.org/10.1007/978-3-030-63406-3_1

[27] Alessandro Armando, David A. Basin, Yohan Boichut, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, Luca Cuccu, Paul Hankes Drielsma, Pierre-Cyrille Héam, Olga Kouchnarenko, Jacopo Mantovani, Sebastian Modersheim, David von Oheimb, Michael Rusinowitch, Judson Santiago, Mathieu Turuani, Luca Vigano, and Laurent Vigneron. 2005. The AVISPA tool for the automated validation of internet security protocols and applications. In CAV. Lecture Notes in Computer Science, Vol. 3576, Kousha Etessami and Sriram K. Rajamani (Eds.). Springer, 281–285. https://doi.org/10.1007/11513988_27

[28] Matteo Avalle, Alfredo Pironti, and Riccardo Sisto. 2014. Formal verification of security protocol implementations: A survey. Form. Aspects Comput. 26, 1 (2014), 99–123. https://doi.org/10.1007/s00165-012-0269-9

[29] Roberto Barbuti, Nicoletta De Francesco, Antonella Santone, and Luca Tesi. 2002. A notion of non-interference for timed automata. Fundam. Inf. 51, 1–2 (2002), 1–11.

[30] Roberto Barbuti and Luca Tesi. 2003. A decidable notion of timed non-interference. Fundam. Inf. 54, 2–3 (2003), 137–150.

[31] Danièle Beauquier. 2003. On probabilistic timed automata. Theor. Comput. Sci. 292, 1 (2003), 65–84. https://doi.org/10.1016/S0304-3975(01)00215-8

[32] Maurice H. ter Beek, Gabriele Lenzini, and Marinella Petrocchi. 2005. Team automata for security: A survey. Theor. Comput. Sci. 335, 1–3 (2005), 1–23. https://doi.org/10.1016/j.tcs.2004.11.044

[33] Germán Behrmann, Kim Guldstrand Larsen, and Jacob Illum Rasmussen. 2004. Priced timed automata: Algorithms and applications. In FMCO. Lecture Notes in Computer Science, Vol. 3657, Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem P. de Roever (Eds.). Springer, 162–182. https://doi.org/10.1007/978-3-540-30280-0_11

[34] Giampaolo Bella and Lawrence C. Paulson. 1998. Kerberos version 4: Inductive analysis of the secrecy goals. In ESORICS. Lecture Notes in Computer Science, Vol. 1485, Jean-Jacques Quisquater, Yves Deswarte, Catherine A. Meadows, and Dieter Gollmann (Eds.). Springer, 361–375. https://doi.org/10.1007/BFb0055875

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

[36] Massimo Benerecetti, Gabriele Lenzini, and Adriano Peron. 2009. TPMC: A model checker for time-sensitive security protocols. J. Comput. 4, 5 (2009), 366–377. https://doi.org/10.4304/jcp.4.5.366-377

[37] Elisa Bertino, Piero A. Bonatti, and Elena Ferrari. 2001. TRBAC: A temporal role-based access control model. ACM Trans. Inf. Syst. Secur. 4, 3 (2001), 191–233. https://doi.org/10.1145/501978.501979

[38] Nathalie Bertrand and Paulin Fournier. 2013. Parameterized verification of many identical probabilistic timed processes. In FSTTCS. LIPIcs, Vol. 24, Anil Seth and Nisheeth K. Vishnoi (Eds.). Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 501–513. https://doi.org/10.4230/LIPIcs.FSTTCS.2013.501

[39] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Ofer Strichman, and Yunshan Zhu. 2003. Bounded model checking. Adv. Comput. 58 (2003), 117–148. https://doi.org/10.1007/978-3-540-458003-2

ACM Computing Surveys, Vol. 55, No. 6, Article 127. Publication date: December 2022.
[40] Arnab Kumar Biswas, Dipak Ghosal, and Shishir Nagaraja. 2017. A survey of timing channels and countermeasures. *Comput. Surv.* 50, 1 (2017), 6:1–6:39. https://doi.org/10.1145/3023872

[41] Henrik C. Bohnenkamp, Pedro R. D’Argenio, Holger Hermanns, and Joost-Pieter Katoen. 2006. MODEST: A compositional modeling formalism for hard and softly timed systems. *IEEE Trans. Softw. Eng.* 32, 10 (2006), 812–830. https://doi.org/10.1109/TSE.2006.104

[42] Andrew Bortz and Dan Boneh. 2007. Exposing private information by timing web applications. In *WWW*. Carey L. Williamson, Mary Ellen Zurko, Peter F. Patel-Schneider, and Prashant J. Shenoy (Eds.). ACM, 621–628. https://doi.org/10.1145/1242572.1242566

[43] Laura Bozelli and Salvatore La Torre. 2009. Decision problems for lower/upper bound parametric timed automata. *Form. Methods Syst. Des.* 35, 2 (2009), 121–151. https://doi.org/10.1007/s10703-009-0074-0

[44] Jeremy W. Bryans, Maciej Koutny, Laurent Mazaré, and Peter Y. A. Ryan. 2008. Opacity generalised to transition systems. *Int. J. Inf. Secur.* 7, 6 (2008), 421–435. https://doi.org/10.1007/s10207-008-0058-x

[45] Mike Burmester, Emmanueln Magkos, and Vassilios Chrissikopoulos. 2012. Modeling security in cyber-physical systems. *Int. J. Crit. Infrastruct. Protect.* 5, 3–4 (2012), 118–126. https://doi.org/10.1016/j.ijcip.2012.08.002

[46] Michael Burrows, Martin Abadi, and Roger M. Needham. 1990. A logic of authentication. *ACM Trans. Comput. Syst.* 8, 1 (1990), 18–36. https://doi.org/10.1145/77648.77649

[47] Gungor Vehbi C., Sahin Dilan, Taskin Koçak, Salih Ergürt, Concettina Buccella, Carlo Cecati, and Gerhard P. Hancke. 2011. Smart grid technologies: Communication technologies and standards. *IEEE Trans. Industr. Inf.* 7, 4 (2011), 529–539. https://doi.org/10.1109/TII.2011.2166794

[48] Luca Cardelli. 1996. Type systems. *Comput. Surv.* 28, 1 (1996), 263–264. https://doi.org/10.1145/234313.234418

[49] G. Cassanelli, Giovanna Mura, Fausto Fantini, Massimo Vanzi, and Bernard Plano. 2006. Failure analysis-assisted FMEA. *Microelectron. Reliab.* 46, 9–11 (2006), 1795–1799. https://doi.org/10.1016/j.microrel.2006.07.072

[50] Franck Cassé. 2009. The dark side of timed opacity. In *ISA*. 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

[51] Franck Cassé, Alexandre David, Emmanuel Fleury, Kim Guldstrand Larsen, and Didier Lime. 2005. Efficient on-the-fly algorithms for the analysis of timed games. In *CONCUR*. Lecture Notes in Computer Science, Vol. 3653, Martin Abadi and Luca de Alfaro (Eds.). Springer, 66–80. https://doi.org/10.1007/11359452_9

[52] Edmund M. Clarke and E. Allen Emerson. 1981. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Proceedings of the Workshop on Logics of Programs*. Lecture Notes in Computer Science, Vol. 131, Dexter Kozen (Ed.). Springer, 52–71. https://doi.org/10.1007/BFb0025774

[53] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. 2001. *Model Checking*. MIT Press.

[54] Hubert Comon-Lundh and Véronique Cortier. 2004. Security properties: Two agents are sufficient. *IEEE Trans. Comput.* 53, 2 (2004), 114–121. https://doi.org/10.1109/TC.2004.1302002

[55] Ricardo Corin, Sandro Etalle, Pieter H. Hartel, and Angélica Mader. 2004. Timed model checking of security protocols. In *FMSE*. Vijayalakshmi Atluri, Michael Backes, David A. Basin, and Michael Waidner (Eds.). ACM, 23–32. https://doi.org/10.1145/1029133.1029137

[56] Ricardo Corin, Sandro Etalle, Pieter H. Hartel, and Angelika Mader. 2004. Timed model checking of security protocols. In *FMSE*. Vijayalakshmi Atluri, Michael Backes, David A. Basin, and Michael Waidner (Eds.). ACM, 25–32. https://doi.org/10.1145/1029133.1029137

[57] Ricardo Corin, Sandro Etalle, Pieter H. Hartel, and Angelika Mader. 2007. Timed analysis of security protocols. *J. Comput. Secur.* 15, 6 (2007), 619–645. http://content.iospress.com/articles/journal-of-computer-security/jcs294

[58] Nitisit Dalal, Jenny Shah, Khushboo Hisaria, and Devesh Jinwala. 2010. A comparative analysis of tools for verification of security protocols. *IEEE Trans. Industr. Inf.* 6, 1 (2010), 18–36. https://doi.org/10.1145/1889206.1889213

[59] Alexandre David, Peter Gjøl Jensen, Kim Guldstrand Larsen, Marius Mikucionis, and Jakob Haahr Taaekvist. 2015. Uppaal strategy. In *TACAS*. Lecture Notes in Computer Science, Vol. 9035, Christel Baier and Cesare Tinelli (Eds.). Springer, 206–211. https://doi.org/10.1007/978-3-662-46681-0_16

[60] Nicolas David, Alexandre David, René Rydhof Hansen, Kim Guldstrand Larsen, Axel Legay, Mads Chr. Olesen, and Christian W. Probst. 2015. Modelling social-technical attacks with timed automata. In *MIST*. Elisa Bertino and Ilson You (Eds.). ACM, 21–28. https://doi.org/10.1145/2808783.2808787

[61] Shuwen Deng, Wenjie Xiong, and Jakub Szefer. 2018. Cachet timing side-channel vulnerability checking with computation tree logic. In *ISCA*. Jakub Szefer, Weidong Shi, and Ruby B. Lee (Eds.). ACM, 2:1–2:8. https://doi.org/10.1145/3214292.3214294

[62] Edsger W. Dijkstra. 1975. Guarded commands, nondeterminacy and formal derivation of programs. *Commun. ACM* 18, 8 (1975), 453–457. https://doi.org/10.1145/360933.360975

[63] Catalin Dima. 2001. Real-time automata. *J. Autom. Lang. Comb.* 6, 1 (2001), 3–23. https://doi.org/10.25596/jalc-2001-003
Timed Automata as a Formalism for Expressing Security

[64] Danny Dolev. 1982. The byzantine generals strike again. J. Algor. 3, 1 (1982), 14–30. https://doi.org/10.1016/0196-6774(82)90004-9

[65] Danny Dolev and Andrew Chi-Chih Yao. 1983. On the security of public key protocols. IEEE Trans. Inf. Theory 29, 2 (1983), 198–207. https://doi.org/10.1109/TIT.1983.1056650

[66] Bruno Dutertre. 2014. Yices 2.2. In CAV. Lecture Notes in Computer Science, Vol. 8559, Armin Biere and Roderick Bloem (Eds.). Springer, 737–744. https://doi.org/10.1007/978-3-319-08867-9_49

[67] Ansgar Fehnker. 1999. Scheduling a steel plant with timed automata. In RTCSA. IEEE Computer Society, 280–286. https://doi.org/10.1109/RTCSA.1999.811256

[68] Edward W. Felten and Michael A. Schneider. 2000. Timing attacks on web privacy. In CCS. Dimitris Gritzalis, Sushil Jajodia, and Pierangela Samarati (Eds.). ACM, 25–32. https://doi.org/10.1145/352600.352606

[69] Riccardo Focardi, Roberto Gorrieri, Ruggero Lanotte, Andrea Maggiolo-Schettini, Fabio Martinelli, Simone Tini, and Enrico Tronci. 2001. Formal models of timing attacks on web privacy. Electr. Not. Theor. Comput. Sci. 62 (2001), 229–243. https://doi.org/10.1016/S1571-0661(04)00329-9

[70] Riccardo Focardi, Roberto Gorrieri, and Fabio Martinelli. 2000. Information flow analysis in a discrete-time process algebra. In CSFW. IEEE Computer Society, 170–184. https://doi.org/10.1109/CSFW.2000.856935

[71] Peter Fontana and Rance Cleaveland. 2014. A menagerie of timed automata. Comput. Surv. 46, 3 (2014), 40:1–40:56. https://doi.org/10.1145/2518102

[72] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. 2011. SpaceEx: Scalable verification of hybrid systems. In CAV. Lecture Notes in Computer Science, Vol. 6806, Ganesh Gopalakrishnan and Shaz Qadeer (Eds.). Springer, 379–395. https://doi.org/10.1007/978-3-642-22110-1_30

[73] Olga Gadyatskaya, René Rydhof Hansen, Kim Guldstrand Larsen, Axel Legay, Mads Chr. Olesen, and Danny Bøgsted Poulsen. 2016. Modelling attack-defense trees using timed automata. In FORMATS. Lecture Notes in Computer Science, Vol. 9884, Martin Fränzle and Nicolas Markey (Eds.). Springer, 35–50. https://doi.org/10.1007/978-3-319-44878-7_3

[74] Guillaume Gardey, John Mullins, and Olivier H. Roux. 2007. Non-interference control synthesis for security timed automata. Electr. Not. Theor. Comput. Sci. 180, 1 (2007), 35–53. https://doi.org/10.1016/j.entcs.2005.05.046

[75] Emsaieb Geepalla, Behzad Bordbar, and Kozo Okano. 2012. Verification of spatio-temporal role based access control using timed automata. In NESEA. IEEE Computer Society, 1–6. https://doi.org/10.1109/NESEA.2012.6474023

[76] Christopher Gerkling, David Schubert, and Eric Bodden. 2018. Model checking the information flow security of real-time systems. In ESoS. Lecture Notes in Computer Science, Vol. 10953, Mathias Payer, Awaï Bashid, and Jose M. Such (Eds.). Springer, 27–43. https://doi.org/10.1007/978-3-319-94496-8_3

[77] Roberto Giacobazzi and Isabella Mastroeni. 2004. Abstract non-interference: Parameterizing non-interference by abstract interpretation. In POPL. Neil D. Jones and Xavier Leroy (Eds.). ACM, 186–197. https://doi.org/10.1145/964001.964017

[78] Roberto Giacobazzi and Isabella Mastroeni. 2005. Generalized abstract non-interference: Abstract secure information-flow analysis for automata. In MMM-ACNS. Lecture Notes in Computer Science, Vol. 3685, Vladimir Gorodetsky, Igor V. Kotenko, and Victor A. Skormin (Eds.). Springer, 221–234. https://doi.org/10.1007/11560326_17

[79] Jens Chr. Godskesen and Olena Gryn. 2006. Modelling and verification of security protocols for ad hoc networks using UPPAAL. In NWPT. Reykjavík University.

[80] Rodolfo Gómez and Howard Bowman. 2007. Efficient detection of zeno runs in timed automata. In FORMATS. Lecture Notes in Computer Science, Vol. 4763, Jean-François Raskin and Shaz Qadeer (Eds.). Springer, 379–395. https://doi.org/10.1007/978-3-540-75454-1_15

[81] H. Gregersen and H. E. Jensen. 1995. Formal Design of Reliable Real Time Systems. Master’s Thesis. Department of Mathematics and Computer Science, Aalborg University.

[82] René Rydhof Hansen, Kim Guldstrand Larsen, Axel Legay, and Danny Bøgsted Poulsen. 2017. Quantitative evaluation of attack defense trees using stochastic timed automata. In GraMSec. Lecture Notes in Computer Science, Vol. 10744, Peng Liu, Sjouke Mauw, and Ketil Stølen (Eds.). Springer, 75–90. https://doi.org/10.1007/978-3-319-74860-3_5

[83] Christian Heinzemmann, Christian Brenner, Stefan Dziwok, and Wilhelm Schäfer. 2015. Automata-based refinement checking for real-time systems. Comput. Sci. Res. Dev. 30, 3–4 (2015), 255–283. https://doi.org/10.1007/s00450-014-0257-9

[84] Martijn Hendriks and Marcel Verhoef. 2006. Timed automata based analysis of embedded system architectures. In IPDPS. IEEE. https://doi.org/10.1109/IPDPS.2006.1639422

[85] Thomas A. Henzinger. 1996. The theory of hybrid automata. In LiCS, Moshe Y. Vardi and Edmund M. Clarke (Eds.). IEEE Computer Society, 278–292. https://doi.org/10.1101/LICS.1996.561342
[86] Holger Hermanns, Julia Krämer, Jan Krčál, and Mariélle Stoelinga. 2016. The value of attack-defence diagrams. In POST. Lecture Notes in Computer Science, Vol. 9635, Frank Piessens and Luca Viganò (Eds.). Springer, 163–185. https://doi.org/10.1007/978-3-662-49635-0_9

[87] Kevin J. Hoffman, David Zage, and Cristina Nita-Rotaru. 2009. A survey of attack and defense techniques for reputation systems. Comput. Surv. 42, 1 (2009), 1:1–1:31. https://doi.org/10.1145/1592541.1592542

[88] Thomas Hune, Judi Romijn, Mariélle Stoelinga, and Frits W. Vaandrager. 2002. Linear parametric model checking of timed automata. J. Logic Algebra. Program. 52–53 (2002), 183–220. https://doi.org/10.1016/S1567-8326(02)00037-1

[89] Tomasz Hyla, Jerzy Pejaś, Imed El Fray, Witold Maćków, Włodzimierz Chocianowicz, and Marcin Szulga. 2014. Sensitive information protection on mobile devices using general access structures. In ICONS. IARA.

[90] Gizela Jakubowska and Wojciech Penczek. 2007. Modelling and checking timed authentication of security protocols. Fundam. Inf. 79, 3–4 (2007), 363–378.

[91] Gizela Jakubowska, Wojciech Penczek, and Marian Srebrny. 2005. Verifying security protocols with timestamps via translation to timed automata. In CS&P, L. Czaia (Ed.). Warsaw University Press, 100–115.

[92] Wojciech Jamroga, Wojciech Penczek, Teofil Sidoruk, Piotr Dembinski, and Antoni W. Mazurkiewicz. 2020. Towards partial order reductions for strategic ability. J. Artif. Intell. Res. 68 (2020), 817–850. https://doi.org/10.1613/jair.11936

[93] Ravi Jhawar, Karim Lounis, and Sjouke Mauw. 2016. A stochastic framework for quantitative analysis of attack-defense trees. In STM. Lecture Notes in Computer Science, Vol. 9871, Gilles Barthe, Evangelos P. Markatos, and Pierangela Samarati (Eds.). Springer, 138–153. https://doi.org/10.1007/978-3-319-46598-2_10

[94] James Joshi, Elisa Bertino, Usman Latif, and Arif Ghafoor. 2005. A generalized temporal role-based access control model. IEEE Trans. Knowl. Data Eng. 17, 1 (2005), 4–23. https://doi.org/10.1109/TKDE.2005.1

[95] Magdalena Kacprzak, Sjouke Mauw, Sasa Radomirovic, and Patrick Schweitzer. 2014. Attack-defence trees. In STS. Lecture Notes in Computer Science, Vol. 8971, Gilles Barthe, Evangelos P. Markatos, and Pierangela Samarati (Eds.). Springer, 138–153. https://doi.org/10.1007/978-3-319-46598-2_10

[96] Ahmet Koltuksuz, Burcu Külahçıoğlu, and Murat Ozkan. 2010. Utilization of timed automata as a verification tool for security protocols. In SSS. Lecture Notes in Computer Science, Vol. 6561, Pierpaolo Degano, Sandro Etalle, and Joshua D. Guttman (Eds.). Springer, 84–100. https://doi.org/10.1007/978-3-642-30793-5_6

[97] Jeroen J. A. Keiren, Peter Fontana, and Rance Cleaveland. 2017. Corrections to "a menagerie of timed automata". Fundam. Inf. 85, 1-4 (2008), 313–328. http://content.iospress.com/articles/fundamenta-informatica/fi85-1-4-22

[98] Roland Kindermann, Tommi A. Junttila, and Ilkka Niemelä. 2012. Beyond lassos: Complete SMT-based bounded model checking for timed automata. In FMOODS-FORTE. Lecture Notes in Computer Science, Vol. 7273, Holger Giese and Grigore Rosu (Eds.). Springer, 84–100. https://doi.org/10.1007/978-3-642-30793-5_6

[99] Michał Knapik and Wojciech Penczek. 2012. Bounded model checking for parametric timed automata. Trans. Petri Nets Other Models Concurr. 5 (2012), 141–159.

[100] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2020. Spectre attacks: Exploiting speculative execution. Commun. ACM 63, 7 (2020), 93–101. https://doi.org/10.1145/3399742

[101] Paul C. Kocher. 1996. Timing attacks on implementations of diffie-hellman, RSA, DSS, and other systems. In CRYPTO. Lecture Notes in Computer Science, Vol. 1109, Neal Koblitz (Ed.). Springer, 104–113. https://doi.org/10.1007/3-540-49964-7_22

[102] Gizela Jakubowska and Wojciech Penczek. 2007. Verifying security protocols with timestamps via translation to timed automata. In CS&P, L. Czaia (Ed.). Warsaw University Press, 100–115.

[103] Barbara Kordy, Sjouke Mauw, Sasa Radomirovic, and Patrick Schweitzer. 2014. Foundations of attack-defense trees. In FMOODS-FORTE. Lecture Notes in Computer Science, Vol. 7273, Holger Giese and Grigore Rosu (Eds.). Springer, 84–100. https://doi.org/10.1007/978-3-642-30793-5_6

[104] Michal Knapik and Wojciech Penczek. 2012. Bounded model checking for parametric timed automata. Trans. Petri Nets Other Models Concurr. 5 (2012), 141–159.

[105] Gizela Jakubowska and Wojciech Penczek. 2007. Modelling and checking timed authentication of security protocols. Fundam. Inf. 79, 3–4 (2007), 363–378.

[106] Robert Kotcher, Yutong Pei, Pranjal Jumde, and Collin Jackson. 2013. Cross-origin pixel stealing: Timing attacks using CSS filters. In CCS. Ahmed-Reza Sadeghi, Virgil D. Gligor, and Moti Yung (Eds.). ACM, 1055–1062. https://doi.org/10.1145/2508859.2516712

[107] Siwar Kriaa, Marc Bouissou, Frederic Colin, Yoran Halgand, and Ludovic Piétre-Cambacédès. 2014. Safety and security interactions modeling using the BDMP formalism: Case study of a pipeline. In SAFECOMP. Lecture Notes in Computer Science, Vol. 8666, Andrea Bondavalli and Felicita Di Giandomenico (Eds.). Springer, 326–341. https://doi.org/10.1007/978-3-319-10506-2_22
Timed Automata as a Formalism for Expressing Security

[108] Moez Krichen and Roobaea Alroobaea. 2019. A new model-based framework for testing security of IoT systems in smart cities using attack trees and price timed automata. In ENASE. Ernesto Damiani, George Spanoudakis, and Leszek A. Maciażek (Eds.). SciTePress, 570–577. https://doi.org/10.5220/0007830605700577

[109] Rajesh Kumar, Enno Ruijters, and Mariëlle Stoelinga. 2015. Quantitative attack tree analysis via priced timed automata. In FORMATS. Lecture Notes in Computer Science, Vol. 9268, Sriman Sankaranarayanan and Enrico Vicario (Eds.). Springer, 156–171. https://doi.org/10.1007/978-3-319-22975-1_11

[110] Rajesh Kumar, Stefano Schivo, Enno Ruijters, Bugra Mehmet Yildiz, David Huistra, Jacco Brandt, Arend Rensink, and Mariëlle Stoelinga. 2018. Effective analysis of attack trees: A model-driven approach. In FASE. Lecture Notes in Computer Science, Vol. 10802, Alessandra Russo and Andy Schürr (Eds.). Springer, 56–73. https://doi.org/10.1007/978-3-319-89363-1_4

[111] Rajesh Kumar and Mariëlle Stoelinga. 2017. Quantitative security and safety analysis with attack-fault trees. In HASE. IEEE, 25–32. https://doi.org/10.1109/HASE.2017.12

[112] Mirosław Kurkowski, Adam Kozakiewicz, and Olga Siedlecka-Lach. 2016. Some remarks on security protocols verification tools. In ISAT, Part II. AISc, Vol. 522, Adam Grzech, Jerzy Swiatek, Zofia Wilimowska, and Leszek Borzemski (Eds.). 65–75. https://doi.org/10.1007/978-3-319-46586-9_6

[113] Mirosław Kurkowski and Wojciech Penczek. 2007. Verifying security protocols modelled by networks of automata. Fundam. Inf. 79, 3–4 (2007), 453–471. http://content.iosrjournals.com/articles/fundamenta-informaticae/F79-3-4-15

[114] Mirosław Kurkowski and Wojciech Penczek. 2009. Timed automata based model checking of timed security protocols. Fundam. Inf. 93, 1–3 (2009), 245–259. https://doi.org/10.3233/FI-2009-0100

[115] Mirosław Kurkowski and Wojciech Penczek. 2012. Applying timed automata to model checking of security protocols. In Handbook of Finite State Based Models and Applications, Jiacun Wang (Ed.). Chapman & Hall/CRC, 223–254. https://doi.org/10.1007/978-1-4398-1105-12

[116] Marta Z. Kwiatkowska, Gethin Norman, Roberto Segala, and Jeremy Sproston. 2002. Automatic verification of real-time systems with discrete probability distributions. Theor. Comput. Sci. 282, 1 (2002), 101–150. https://doi.org/10.1016/S0304-3975(01)00046-9

[117] Leslie Lamport. 1987. A fast mutual exclusion algorithm. ACM Trans. Comput. Syst. 5, 1 (1987), 1–11. https://doi.org/10.1145/7351.7352

[118] Ruggero Lanotte, Andrea Maggiolo-Schettini, and Angelo Troina. 2003. Weak bisimulation for probabilistic timed automata and applications to security. In SEFM. IEEE Computer Society, 34–43. https://doi.org/10.1109/SEFM.2003.1236205

[119] Ruggero Lanotte, Andrea Maggiolo-Schettini, and Angelo Troina. 2005. Information flow analysis for probabilistic timed automata. In FAST. IFIP, Vol. 173, Theodosis Dimitrakos and Fabio Martinelli (Eds.). Springer, 13–26. https://doi.org/10.1007/3-87-24098-5_2

[120] Ruggero Lanotte, Andrea Maggiolo-Schettini, and Angelo Troina. 2010. Time and probability-based information flow analysis. IEEE Trans. Softw. Eng. 36, 5 (2010), 719–734. https://doi.org/10.1109/TSE.2010.4

[121] François Laroussinie, Kim Gulstrand François, and Carsten Weise. 1995. From timed automata to logic—and back. In MFCS. Lecture Notes in Computer Science, Vol. 969, Jiri Wiedermann and Petr Hájek (Eds.). Springer, 529–539. https://doi.org/10.1007/3-540-60246-1_158

[122] Kim Gulstrand Larsen, Paul Pettersson, and Wang Yi. 1997. UPPAAL in a nutshell. Int. J. Softw. Tools Technol. Transf. 1, 1–2 (1997), 134–152. https://doi.org/10.1007/s100090050010

[123] Elizabeth LeMay, Michael D. Ford, Ken Keefe, William H. Sanders, and Carol Muehrcke. 2011. Model-based security metrics using ADversary view security evaluation (ADVISE). In QEST. IEEE Computer Society, 191–200. https://doi.org/10.1109/QEST.2011.34

[124] Li Li, Jun Sun, and Jin Song Dong. 2016. Automated verification of timed security protocols with clock drift. In FM. Lecture Notes in Computer Science, Vol. 9995, John S. Fitzgerald, Constance L. Heitmeyer, Stefania Gnesi, and Anna Philippou (Eds.). 513–530. https://doi.org/10.1007/978-3-319-48989-6_31

[125] Li Li, Jun Sun, Yang Liu, Meng Sun, and Jin Song Dong. 2018. A formal specification and verification framework for timed security protocols. IEEE Trans. Softw. Eng. 44, 8 (2018), 725–746. https://doi.org/10.1109/TSE.2017.2712621

[126] Yang Liu, Jun Sun, and Jin Song Dong. 2011. PAT 3: An extensible architecture for building multi-domain model checkers. In ISSRE, Tadashi Dohi and Bojan Cukic (Eds.). IEEE Computer Society, 190–199. https://doi.org/10.1109/ISSRE.2011.19

[127] Karim Lounis and Samir Ouchani. 2020. Modeling attack-defense trees’ countermeasures using continuous time markov chains. In ASYDE. Lecture Notes in Computer Science, Vol. 12524, Loek Cleophas and Mieke Massink (Eds.). Springer, 30–42. https://doi.org/10.1007/978-3-030-67220-1_3

[128] Gavin Lowe. 1995. An attack on the needham-schroeder public-key authentication protocol. Inform. Process. Lett. 56, 3 (1995), 131–133. https://doi.org/10.1016/0020-0190(95)00144-2

ACM Computing Surveys, Vol. 55, No. 6, Article 127. Publication date: December 2022.
[129] Gavin Lowe. 1996. Breaking and fixing the needham-schroeder public-key protocol using FDR. Softw. Concepts Tools
17, 3 (1996), 93–102.
[130] Yuteng Lu and Meng Sun. 2018. Modeling and verification of IEEE 802.11i security protocol in UP-PAAAL for internet of things. Int. J. Softw. Eng. Knowl. Eng. 28, 11–12 (2018), 1619–1636. https://doi.org/10.1142/S021919401840020X
[131] Laurent Mazarré. 2004. Using unification for opacity properties. In WITS, Peter Ryan (Ed.). 165–176.
[132] Fujio Miyawaki, Ken Masumune, Satoshi Suzuki, Kitaro Yoshimitsu, and Juri Vain. 2005. Scrub nurse robot system-intraoperative motion analysis of a scrub nurse and timed-automata-based model for surgery. IEEE Trans. Industr. Electr. 52, 5 (2005), 1227–1235. https://doi.org/10.1109/TIE.2005.855692
[133] Chirag Modi, Dhiren R. Patel, Bhavesh Borisaniya, Hiren Patel, Avi Patel, and Muttrukrishnan Rajaran. 2013. A survey of intrusion detection techniques in cloud. J. Netw. Comput. Appl. 36, 1 (2013), 42–57. https://doi.org/10.1016/
jnca.2012.05.003
[134] Samrat Mondal and Shamik Sural. 2008. Security analysis of temporal-RBAC using timed automata. In IAS, Massimiliano Rak, Ajith Abraham, and Valentina Casola (Eds.). IEEE Computer Society. 37–40. https://doi.org/10.1109/IAS.2008.10
[135] Samrat Mondal, Shamik Sural, and Vijayalakshmi Atluri. 2011. Security analysis of GTRBAC and its variants using model checking. Comput. Secur. 30, 2–3 (2011), 128–147. https://doi.org/10.1016/j.cose.2010.09.002
[136] Roger M. Needham and Michael D. Schroeder. 1978. Using encryption for authentication in large networks of computers. Commun. ACM 21, 12 (1978), 993–999. https://doi.org/10.1145/356957.356959
[137] B. Clifford Neuman and Stuart G. Stubblebine. 1993. A note on the use of timestamps as nonces. ACM SIGOPS Operat. Syst. Rev. 27, 2 (1993), 10–14. https://doi.org/10.1145/155848.155852
[138] Brian Nielsen and Arne Skou. 2003. Automated test generation from timed automata. Int. J. Softw. Tools Technol. Transf. 5, 1 (2003), 59–77. https://doi.org/10.1007/s10009-002-0094-1
[139] Flemming Nielsen, Hanne Riis Nielson, and Panagiotis Vasilikos. 2017. Information flow for timed automata. In Models, Algorithms, Logics and Tools, Lecture Notes in Computer Science, Vol. 10460, Luca Aceto, Giorgio Bacci, Giovanni Bacci, Anna Ingólfsdóttir, Axel Legay, and Radu Mardare (Eds.). Springer. 3–21. https://doi.org/10.1007/978-3-319-63121-9_1
[140] Joel Ouaknine, Alexander Rabinovich, and James Worrell. 2009. Time-bounded verification. In CONCUR. Lecture Notes in Computer Science, Vol. 5710, Mario Bravetti and Gianluigi Zavattaro (Eds.). Springer. 496–510. https://doi.org/10.1007/978-3-642-04081-8_33
[141] Zhigeng Pan, Adrian David Cheok, Hongwei Yang, Jiejie Zhu, and Jiaoying Shi. 2006. Virtual reality and mixed reality for virtual learning environments. Comput. Graph. 30, 1 (2006), 20–28. https://doi.org/10.1016/j.cag.2005.10.004
[142] Charles E. Perkins, David B. Johnson, and Jari Arkko. 2011. Mobility support in IPv6. RFC 6275 (2011), 1–169. https://doi.org/10.17487/rfc6275
[143] Adrian Perrig, Ran Canetti, Dawn Xiaodong Song, and J. D. Tygar. 2001. Efficient and secure source authentication for multicast. In NDSS. The Internet Society.
[144] Adrian Perrig, Robert Szewczyk, J. D. Tygar, Victor Wen, and David E. Culler. 2002. SPINS: Security protocols for sensor networks. Wireless Netw. 8, 5 (2002), 521–534. https://doi.org/10.1023/A:1016598314198
[145] Laure Petrucci, Michał Knapik, Wojciech Szczypinski, and Grzegorz Falonec. 2018. A spatio-temporal access control model supporting delegation for pervasive computing applications. In TrustBus’08. Lecture Notes in Computer Science, Vol. 5185, Steven Furnell, Bogdan Franczyk (Eds.). Springer. 185–198. https://doi.org/10.1007/978-3-319-39426-8_13
[146] Indrakshi Ray and Manachai Toachchoodee. 2008. A spatio-temporal access control model supporting delegation for pervasive computing applications. In TrustBus’08. Lecture Notes in Computer Science, Vol. 5185, Steven Furnell, Bogdan Franczyk (Eds.). Springer. 185–198. https://doi.org/10.1007/978-3-319-39426-8_13
[147] Enno Ruijters and Mariëlle Stoelinga. 2015. Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. Comput. Sci. Rev. 15 (2015), 29–62. https://doi.org/10.1016/j.cosrev.2015.03.001
[148] Zohreh Saffarian Eidgahi and Vahid Rafe. 2016. Security analysis of network protocols through model checking: A case study on mobile IPv6. Secur. Commun. Netw. 9, 10 (2016), 1072–1084. https://doi.org/10.1002/sec.1401
[149] Veli Sahin, Cambyse Guy Omidyar, and Timothy M. Bauman. 1988. Telecommunications management network (TMN) architecture and interworking designs. IEEE J. Select. Areas Commun. 6, 4 (1988), 685–696. https://doi.org/10.1109/49.1940
[150] Ravi S. Sandhu, Edward J. Feinstein, Hal L. Feinstein, and Charles E. Youman. 1996. Role-based access control models. Computer 29, 2 (1996), 38–47. https://doi.org/10.1109/2.485845
Timed Automata as a Formalism for Expressing Security

[153] Kimya Sanzgiri, D. LaFlamme, Bridget Dahill, Brian Neil Levine, Clay Shields, and Elizabeth M. Belding-Royer. 2005. Authenticated routing for ad hoc networks. *IEEE J. Select. Areas Commun.* 23, 3 (2005), 598–610. https://doi.org/10.1109/JSAC.2004.842547

[154] Sanjay E. Sarma, David Brock, and Daniel W. Engels. 2001. Radio frequency identification and the electronic product code. *IEEE Micro* 21, 6 (2001), 50–54. https://doi.org/10.1109/40.977758

[155] Mahadev Satyanarayanan. 1989. Integrating security in a large distributed system. *ACM Trans. Comput. Syst.* 7, 3 (1989), 247–280. https://doi.org/10.1145/65000.65002

[156] Michael Schwarz, Clémentine Maurice, Daniel Gruss, and Stefan Mangard. 2017. Fantastic timers and where to find them: High-resolution microarchitectural attacks in javascript. In *Proc. Lecture Notes in Computer Science*, Vol. 10322, Aggelos Kiayias (Ed.). Springer, 247–267. https://doi.org/10.1007/978-3-319-70972-7_13

[157] Olga Siedleca-Lamch. 2020. Probabilistic and timed analysis of security protocols. In *CISIS*. AISC, Vol. 1267, Álvaro Herrero, Carlos Cambra, Daniel Urda, Javier Sedano, Héctor Quintián, and Emilio Corcho (Eds.). Springer, 142–151. https://doi.org/10.1007/978-3-030-57805-3_14

[158] Olga Siedleca-Lamch, Miroslaw Kurkowski, and Jacek Piątkowski. 2016. Using probabilistic automata for security protocols verification. *J. Appl. Math. Comput. Mech.* 15, 2 (2016).

[159] Alireza Souri and Monire Norouzi. 2019. A state-of-the-art survey on formal verification of the internet of things applications. *J. Serv. Sci. Res.* 11, 1 (2019), 47–67. https://doi.org/10.1145/32927-019-0003-8

[160] Michio Sugeno. 1985. An introductory survey of fuzzy control. *Inf. Sci.* 36, 1–2 (1985), 59–83. https://doi.org/10.1016/0020-0255(85)90026-X

[161] Yan Sun, Tin-Yu Wu, Xiaojiong Ma, and Han-Chieh Chao. 2015. Modeling and verifying EPC network intrusion system based on timed automata. *Perv. Mobile Comput.* 24 (2015), 61–76. https://doi.org/10.1016/j.pmjc.2015.06.007

[162] Sabina Szymoniak. 2021. Time influence on security protocol. In ENASE, Raian Ali, Hermann Kaindl, and Leszek A. Maciaszek (Eds.), SCITEPRESS, 181–188. https://doi.org/10.5220/0010388701810188

[163] Sabina Szymoniak, Miroslaw Kurkowski, and Jacek Piątkowski. 2015. Timed models of security protocols including delays in the network. *J. Appl. Math. Comput. Mech.* 14, 3 (2015), 127–139. https://doi.org/10.17512/jamcm.2015.3.14

[164] Sabina Szymoniak, Olga Siedleca-Lamch, and Miroslaw Kurkowski. 2016. Timed analysis of security protocols. In *ISAT, Part II.* AISC, Vol. 522, Adam Grzech, Jerzy Świątek, Zofia Wilimowska, and Leszek Borzems (Eds.), 53–63. https://doi.org/10.1007/978-3-319-46586-9_5

[165] Sabina Szymoniak, Olga Siedleca-Lamch, and Miroslaw Kurkowski. 2019. Network’s delays in timed analysis of security protocols. In *ISAT.* AISC, Vol. 852, Leszek Borzems, Jerzy Świątek, and Zofia Wilimowska (Eds.), 19–29. https://doi.org/10.1007/978-3-319-99981-4_3

[166] Sabina Szymoniak, Olga Siedleca-Lamch, Agnieszka M. Zbrzezny, Andrzej Zbrzezny, and Mirosław Kurkowski. 2021. SAT and SMT-based verification of security protocols including time aspects. *Sensors* 21, 9 (2021), 3053. https://doi.org/10.3390/s21093055

[167] Niklas Sörensson and Niklas Een. 2005. Minisat v1.13-a SAT solver with conflict-clause minimization. In *SAT Competition* 2005.

[168] Scott R. Tilley and Tauhida Parveen. 2012. *Software Testing in the Cloud—Migration and Execution*. Springer. https://doi.org/10.1007/978-3-642-32122-1

[169] Llanos Tobarra, Diego Cazorla, Fernando Cuartero, and Juan José Pardo. 2008. Modelling secure wireless sensor networks routing protocols with timed automata. In *PM2HW2N*, Maria Blanca Caminero, Francisco M. Delicado Martínez, and Richard Werner Nelem Pazzi (Eds.). ACM, 51–58. https://doi.org/10.1145/1454630.1454638

[170] Samaikya Valluripally, Aniket Gulhane, Reshmi Mitra, Khaza Anuarul Hoque, and Prasad Calyam. 2020. Attack trees for security and privacy in social virtual reality learning environments. In *CCNC*. IEEE, 1–9. https://doi.org/10.1109/CCNC46108.2020.9045724

[171] Panagiotis Vasilikos, Flemming Nielson, and Hanne Riis Nielson. 2017. Time dependent policy-based access control. In *TIME.* LIPICS, Vol. 90, Sven Schewe, Thomas Schneider, and Jef Wijsen (Eds.), Schloss Dagstuhl - Leibniz-Zentrum f"ur Informatik, 21:1–21:18. https://doi.org/10.4230/LIPIcs.TIME.2017.21

[172] Panagiotis Vasilikos, Flemming Nielson, and Hanne Riis Nielson. 2018. Secure information release in timed automata. In *POST.* 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

[173] Panagiotis Vasilikos, Hanne Riis Nielson, Flemming Nielson, and Boris Köpf. 2019. Timing leaks and coarse-grained clocks. In *CSF.* IEEE, 32–47. https://doi.org/10.1109/CSF.2019.00010

[174] William E. Vesely, Francine F. Goldberg, Norman H. Roberts, and David F. Haasl. 1981. *Fault Tree Handbook*. Technical Report. Nuclear Regulatory Commission Washington DC.

[175] Md Tawhid Bin Waex, Jürgen Dingel, and Karen Rudie. 2013. A survey of timed automata for the development of real-time systems. *Comput. Sci. Rev.* 9 (2013), 1–26. https://doi.org/10.1016/j.cosrev.2013.05.001

ACM Computing Surveys, Vol. 55, No. 6, Article 127. Publication date: December 2022.
[176] 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

[177] Lingtai Wang, Naijun Zhan, and Jie An. 2018. The opacity of real-time automata. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst.* 37, 11 (2018), 2845–2856. https://doi.org/10.1109/TCAD.2018.2857363

[178] Ting Wang, Qi Su, and Tieming Chen. 2017. Formal analysis of security properties of cyber-physical system based on timed automata. In *DSC*. IEEE Computer Society, 534–540. https://doi.org/10.1109/DSC.2017.44

[179] Ting Wang, Jun Sun, Yang Liu, Xinyu Wang, and Shanping Li. 2014. Are timed automata bad for a specification language? language inclusion checking for timed automata. In *TACAS*. Lecture Notes in Computer Science, Vol. 8413, Erika Abrahám and Klaus Havelund (Eds.). Springer, 310–325. https://doi.org/10.1007/978-3-642-54862-8_21

[180] Ting Wang, Jun Sun, Xinyu Wang, Yang Liu, Yuanjie Si, Jin Song Dong, Xiaohu Yang, and Xiaohong Li. 2015. A systematic study on explicit-state non-zenoness checking for timed automata. *IEEE Trans. Softw. Eng.* 41, 1 (2015), 3–18. https://doi.org/10.1109/TSE.2014.2359893

[181] Wojciech Widel, Maxime Audinot, Barbara Fila, and Sophie Pinchinat. 2019. Beyond 2014: Formal methods for attack tree-based security modeling. *Comput. Surv.* 52, 4 (2019), 75:1–75:36. https://doi.org/10.1145/3331524

[182] Simon Wimmer. 2019. Munta: A verified model checker for timed automata. In *FORMATS*. Lecture Notes in Computer Science, Vol. 11750, Étienne André and Mariëlle Stoelinga (Eds.). Springer, 236–243. https://doi.org/10.1007/978-3-030-29662-9_14

[183] Simon Wimmer, Frédéric Herbreteau, and Jaco van de Pol. 2020. Certifying emptiness of timed Büchi automata. In *FORMATS*. Lecture Notes in Computer Science, Vol. 12288, Nathalie Bertrand and Nils Jansen (Eds.). Springer, 58–75. https://doi.org/10.1007/978-3-030-57628-4_4

[184] Simon Wimmer and Peter Lammich. 2018. Verified model checking of timed automata. In *TACAS, Part I*. Lecture Notes in Computer Science, Vol. 10805, Dirk Beyer and Mariëlle Huisman (Eds.). Springer, 61–78. https://doi.org/10.1007/978-3-319-89960-2_4

[185] Simon Wimmer and Joshua von Mutius. 2020. Verified certification of reachability checking for timed automata. In *TACAS, Part I*. Lecture Notes in Computer Science, Vol. 12078, Armin Biere and David Parker (Eds.). Springer, 425–443. https://doi.org/10.1007/978-3-030-45190-5_24

[186] Thomas Y. C. Woo and Simon S. Lam. 1994. A lesson on authentication protocol design. *ACM SIGOPS Operat. Syst. Rev.* 28, 3 (1994), 24–37. https://doi.org/10.1145/182110.182113

[187] Bozena Wozna-Szczesniak and Andrzej Zbrzezny. 2016. Checking EMTLK properties of timed interpreted systems via bounded model checking. *Stud. Logic.* 104, 4 (2016), 641–678. https://doi.org/10.1007/s11225-015-9637-9

[188] Bozena Wozna-Szczesniak, Agnieszka M. Zbrzezny, and Andrzej Zbrzezny. 2017. SMT-based searching for k-quasi-optimal runs in weighted timed automata. *Fundam. Inf.* 152, 4 (2017), 411–433. https://doi.org/10.3233/FI-2017-1527

[189] Kai Xu, Loon Ching Tang, Min Xie, S. L. Ho, and M. L. Zhu. 2002. Fuzzy assessment of FMEA for engine systems. *Reliab. Eng. Syst. Safety* 75, 1 (2002), 17–29. https://doi.org/10.1016/S0951-8320(01)00101-6

[190] Sergio Yovine. 1997. KRONOS: A verification tool for real-time systems. *Int. J. Softw. Tools Technol. Transf.* 1, 1-2 (1997), 123–133. https://doi.org/10.1007/s100090050009

[191] Agnieszka M. Zbrzezny, Sabina Szymoniak, and Miroslaw Kurkowski. 2019. Efficient verification of security protocols time properties using SMT solvers. In *CSIS-ICEUTE*. AISC, Vol. 951, Francisco Martínez-Álvarez, Alicia Troncoso Lora, José António Sáez Muñoz, Héctor Quintián, and Emilio Corchado (Eds.). Springer, 25–35. https://doi.org/10.1007/978-3-030-20005-3_3

[192] Agnieszka M. Zbrzezny, Sabina Szymoniak, Olga Siedlecka-Lamch, and Miroslaw Kurkowski. 2020. VerSecTis—An agent based model checker for security protocols. In *AAMAS*. Amal El Fallah Seghrouchni, Gita Sukthankar, Bo An, and Neil Yorke-Smith (Eds.). International Foundation for Autonomous Agents and Multiagent Systems, 2123–2125.

[193] Chiara Zizza, Adam Starr, Devin Hudson, Sai Shreya Nuguri, Prasad Calyam, and Zhizhai He. 2018. Towards a social virtual reality learning environment in high fidelity. In *CCNC*. IEEE, 1–4. https://doi.org/10.1109/CCNC.2018.8319187