Verifying Opacity Properties in Security Systems

Chunyan Mu✉ and David Clark

Abstract—We delineate a methodology for the specification and verification of flow security properties expressible in the opacity framework. We propose a logic, OpacTL, for straightforwardly expressing such properties in systems that can be modelled as partially observable labelled transition systems. We develop verification techniques for analysing property opacity with respect to observation notions. Adding a probabilistic operator to the specification language enables quantitative analysis and verification. This analysis is implemented as an extension to the PRISM model checker and illustrated via a number of examples. Finally, an alternative approach to quantifying the opacity property based on entropy is sketched.

Index Terms—opacity, logic, security, verification

1 Introduction

Security of software can be stated and investigated only with respect to a specification of what security means in a given context. The proliferation of software has lead to a profusion of security properties, many of which exhibit a great deal of similarity. This has lead to a search for common abstractions, and general formalisms for expressing security properties. *Opacity*, first introduced by Mazaré [1], is a promising approach for describing and unifying security properties. The key insight behind opacity is twofold:

- Distinguish between system behaviour, and observations of the system's behaviour.
- A security property φ is opaque, provided that for every behaviour π satisfying φ there is another behaviour π', not satisfying φ, such that π and π' give rise to identical observations.

Clearly, when φ is opaque in this sense, observers cannot be sure if φ holds of the observed system, or not. Many familiar security definitions such as non-interference, declassification, and knowledge-based security can be obtained by suitable choices of predicate φ [2], and notion of observable behaviour.

Absolute guarantees of security are often impractical, and we may tolerate violation of security properties, as long as it happens with sufficiently low probability. So verifying and analysing security properties can also usefully take quantitative aspects into account. For this reason, opacity has been extended to quantitative opacity [3], [4].

While opacity successfully unifies many concrete notions of software security, it is a semantic concept, typically described informally using the mathematical vernacular. There are not yet dedicated development of tools for automatic verification of opacity properties. The present paper asks several questions:

- Can we build a sufficiently expressive *logic* for opaque predicates?
- Can we build automated verification tools for opaque predicates?
- Can we extend the logic and automated verification for opacity to probabilistic opacity?

We answer all questions in the affirmative.

First, we propose a logic OpacTL for expressing opacity and internalise its definition in the logic: for this purpose we extend conventional temporal logic with a predicate

$$\odot[\psi]$$

which expresses that property ψ is opaque. OpacTL allows us to specify the more general opacity property in a *straightforward* way, and the property required to be secret can be defined flexibly regarding users' requirements. We also present OpacPTL, a probabilistic generalisation of OpacTL, which enables us to express probabilistic opacity, and allows us to reason about the *degree* of satisfaction or violation of the security property of interest. We demonstrate the expressibility of both OpacTL and OpacPTL through examples of opacity, respectively probabilistic opacity, from the literature. We automate both logics by translation to the PRISM model checker. We also introduce an alternative approach to measure the level of opacity of the system, based on the notion of entropy.

Outline. This paper is organised as follows. In Section 2 we recall the definition of labelled transition system, observations, and opacity. Section 3 introduces a temporal logic OpacTL for the formal specification of opacity. Section 4 presents the probabilistic extension of the logic OpacPTL, and a prototype model checker for the (probabilistic) opacity fragment of the logic. Section 5 describes the implementation of our techniques for analysing (quantitative) opacity flow properties of systems in a prototype tool, and demonstrates its applicability using several case studies. Section 6 provides an alternative measurement of the opacity formula based on the notion of *entropy*. Section 7 briefly reviews

- *C. Mu is with the Department of Computing and Games, Teesside University, Middlesbrough, UK.*
  E-mail: c.mu@tees.ac.uk
- *D. Clark is with Department of Computer Science, University College London, London, UK.*
  E-mail: david.clark@ucl.ac.uk

Manuscript received 14 Oct. 2021; revised 15 Feb. 2022; accepted 24 Feb. 2022.

literature in related areas, and Section 8 draws conclusions and points out some future directions.

2 Labelled transition systems and observations

Let $\mathbb{N}$ be the set of natural numbers, assume $0 \in \mathbb{N}$. An *alphabet* Σ is a non-empty, finite set, $|\Sigma|$ is its cardinality. Σ^* denotes the set of all *finite* words over Σ including the *empty word* ε, Σ^n denotes the set of all *finite* words over Σ and the length of the words is n, $\Sigma^+ = \Sigma^* \backslash \{\varepsilon\}$, Σ^ω denotes the set of all *infinite* words, Σ^∞ denotes the set of all *finite* and *infinite* words. The subsets of Σ^*:$L \subseteq \Sigma^*$ are called languages, and $L \subseteq \Sigma^\infty$ are called ω-languages.

Definition 1. A *labelled transition system* (LTS) is a tuple $\mathcal{L} = (S, \Sigma, \rightarrow, F)$, where:

- Σ is a finite alphabet of *labels*, or *actions*, including a distinguished element $\perp$ representing termination (staying there forever).
- S is a finite set of *states*.
- $\rightarrow \subseteq S \times \Sigma \times S$ is the *transition relation*.
- $F \subset S$ is the set of final states, a (possibly empty) subset of S.

Definition 2. We say $\mathcal{L}$ is *deterministic* iff $\forall s, t \in S$, whenever $s \xrightarrow{a} t$ and $s \xrightarrow{a} t'$ then $t = t'$. We say $\mathcal{L}$ is *circular*, if each state has an outgoing transition, i.e., for all $s \in S$ there is $s \xrightarrow{a} t$. A *path in* $\mathcal{L}$ is a total map $\pi : \mathbb{N} \to (\Sigma \cup S)$, subject to the following constraints:

- Whenever $i \in \mathbb{N}$ is even, then $\pi(i)$ is a state. Otherwise $\pi(i)$ is a label.
- For all even i we have $\pi(i) \xrightarrow{\pi(i+1)} \pi(i+2)$ and the triple satisfies the transition relation.

The set of all $\mathcal{L}$'s paths is denoted by $\mathsf{path}(\mathcal{L})$. The set of paths of $\mathcal{L}$ starting from state s is denoted by $\mathsf{path}(\mathcal{L}, s)$. We write $\pi[i \ldots]$ to denote $\pi(i)\pi(i+1)\ldots$, and write $\mathsf{erase}(\pi)$ for the map that erases all the states from π: $\mathsf{erase}(\pi) = \pi(1)\pi(3)\ldots$, in other words, $n \mapsto \pi(2n+1)$, defined for all $n \in \mathbb{N}$. A *trace in* $\mathcal{L}$ is a map $tr : \mathbb{N} \to \Sigma$, such that $tr = \mathsf{erase}(\pi)$ for some path π.

We are often sloppy when talking about paths and traces, in particular, we often elide the indices, writing e.g. $ab\perp\perp$... for a trace $\{(0, a), (1, b)\} \cup \{(n, \perp) \mid n > 1\}$.

Definition 3. A trace tr is *well-structured* iff for all suitable $i < j$ we have: $tr(i) = \perp \Rightarrow tr(j) = \perp$. Such the smallest i is denoted by *last*. A path π is *well-structured* iff $\mathsf{erase}(\pi)$ is well-structured, $\pi(last * 2)$ is called a *final state*. $\mathcal{L}$ is *well-structured* iff all of its paths are well-structured. Traces and paths are *semantically finite* iff they are well-structured, and at least one of their labels is $\perp$.

Note that a well-structured LTS won't change a state after a termination. A key idea in modelling security properties is the observation power of the attacker. We use a set of *observables*, distinct from the states and actions of the LTS, for this purpose. Actions, states and observables are connected by an observation function.

Definition 4. Let Θ be a finite alphabet for observables. We write $\Theta_\perp$ for $\Theta \cup \{\perp\}$, assuming that $\perp \notin \Theta$. A *observation function* is a function $obs : \mathsf{path}(\mathcal{L}) \to \Theta_\perp^*$, subject to the additional constraint that $obs(\perp) = \perp$: i.e., for all paths π of the form $\pi = \pi_L \perp \pi_R$ we have $obs(\pi) = obs(\pi_L)\perp obs(\pi_R)$. Observation functions on traces are defined similarly.

The intuition is that Θ contains all possible projections of paths where a projection of a sequence A is a sequence B that can be obtained from A by deleting members of A, e.g. ad is a projection of $abcd$. Note that projections (observations) of paths are not themselves paths. In practice many observation functions will have additional structure, e.g., $obs(\pi) = obs'(\pi(1))obs'(\pi(3))obs'(\pi(5))\ldots$, for some function $obs' : \Sigma \to \Theta_\perp$ on transition labels.

Opacity is a general framework for formulating and unifying security properties expressed as predicates. Here, a predicate is simply a subset of $\mathsf{path}(\mathcal{L})$. A predicate is *opaque* if, given any path of the system, an adversary's observation is unable to determine whether the path satisfies the predicate. In comparison with other security policies such as non-interference, the opacity framework allows a more flexible specification of both the adversary's power of observation and the confidential properties of the system. In the paper, we use $[\![\varphi]\!]$ to denote a set of paths satisfying φ.

Definition 5. [Opacity and observability] A *predicate* φ over $\mathsf{path}(\mathcal{L})$ is a subset of $\mathsf{path}(\mathcal{L})$. Given an observation function obs, a predicate φ is *opaque w.r.t. obs* iff: for every path $\pi \in [\![\varphi]\!]$, there is a path $\pi' \notin [\![\varphi]\!]$ such that $obs([\![\pi]\!]) = obs([\![\pi']\!])$, i.e., all paths in $[\![\varphi]\!]$ are covered (observationally equivalent) by paths in $[\![\overline{\varphi}]\!]$ (where $[\![\overline{\varphi}]\!]$ denotes the complement of $[\![\varphi]\!]$): $obs([\![\varphi]\!]) \subseteq obs([\![\overline{\varphi}]\!])$. Paths in $[\![\varphi]\!]$ are called *observable* paths iff there is at least one path in $[\![\varphi]\!]$ which is not covered by paths in $[\![\overline{\varphi}]\!]$: $[\![\varphi]\!] \setminus obs^{-1}(obs([\![\overline{\varphi}]\!])) \neq \emptyset$.

Many special cases of opacity are easily definable in our approach, such as *initial-state opacity*, *final-state opacity*, *initial-final-state opacity* [5], *total-opacity* [6] and *language-based opacity* [7], [8]. In Section 4 we will generalise opacity to *probabilistic opacity* [4].

3 The Opaque Temporal Logic OpacTL

The behaviour of a state transition system is described as sequences of labels during the possible executions. The labels indicate the valuations of the input/output variables of the system. The temporal properties we specify are upon such behaviours over the same alphabet. We study here OpacTL, which is a temporal logic [9] with an opacity operator $\odot$ over *semantically finite* traces for specification of security properties.

3.1 Formulae

Let Asp be a fixed set, the *atomic state propositions*, ranged over by α. We have two classes of formulae given by the grammar below: *state formulae* and *path formulae* ranged over by ϕ and ψ respectively.

$$\begin{array}{lcl} \phi & ::= & \texttt{true} \mid \texttt{false} \mid \alpha \mid \neg\phi \mid \phi \wedge \phi \mid \odot[\psi] \\ \psi & ::= & \mathbf{X}\phi \mid \phi\mathbf{U}\phi \mid \phi\mathbf{R}\phi \mid \perp \mid \neg\psi \end{array}$$

Note that an OpacTL formula is defined relative to a state and always a state formula. Path formulae only appear inside the $\odot[\cdot]$ operator.

3.2 Model

A *model* is a tuple $(\mathcal{L}, \eta, obs)$ where:

- $\mathcal{L} = (\Sigma, S, \rightarrow, F)$ is an LTS, that is circular, deterministic and well-structured.
- $\eta : S \rightarrow \mathfrak{P}(\mathsf{Asp})$ is the state labelling function, where $\mathfrak{P}(\mathsf{Asp})$ is the powerset of Asp.
- $obs : \mathsf{path}(\mathcal{L}) \rightarrow \Theta^*_\perp$ is an observation function.

3.3 Satisfaction relations

As we have two notions of formulae (state and path formulae), we need two satisfaction relations. Let $\mathcal{M} = (\mathcal{L}, \eta, obs)$ and $s \in S$. We now define the *satisfaction relation* $\mathcal{M} \models_s \phi$ for state formulae.

- $\mathcal{M} \models_s \mathtt{true}$ always holds.
- $\mathcal{M} \models_s \alpha$ iff $\alpha \in \eta(s)$.
- $\mathcal{M} \models_s \neg\phi$ iff $\mathcal{M} \not\models_s \phi$.
- $\mathcal{M} \models_s \phi \wedge \phi'$ iff $\mathcal{M} \models_s \phi$ and $\mathcal{M} \models_s \phi'$.
- $\mathcal{M} \models_s \odot[\psi]$ iff for all paths $\pi \in \mathsf{path}(\mathcal{L}, s)$ whenever $\mathcal{M} \models_\pi \psi$ then there exists a path $\pi' \in \mathsf{path}(\mathcal{L}, s)$ s.t. $\mathcal{M} \not\models_{\pi'} \psi$ and $obs(\pi) = obs(\pi')$.

Now let π be a path in $\mathcal{L}$. Then we define:

- $\mathcal{M} \models_\pi \mathbf{X}\phi$ iff $\mathcal{M} \models_{\pi(2)} \phi$.
- $\mathcal{M} \models_\pi \phi\mathbf{U}\phi'$ iff $\exists i \in \mathbb{N}.\mathcal{M} \models_{\pi(2i)} \phi'$ and $\forall 0 \leq j < i.\mathcal{M} \models_{\pi(2j)} \phi$.
- $\mathcal{M} \models_\pi \phi\mathbf{R}\phi'$ iff either $\exists i \in \mathbb{N}.\mathcal{M} \models_{\pi(2i)} \phi \wedge \forall 0 \leq j \leq i.\mathcal{M} \models_{\pi(2j)} \phi'$ or $\forall j \in \mathbb{N}.\mathcal{M} \models_{\pi(2j)} \phi'$.
- $\mathcal{M} \models_\pi \perp$ iff $\pi(1) = \perp$.
- $\mathcal{M} \models_\pi \neg\psi$ iff $\mathcal{M} \not\models_\pi \psi$.

Note that the opacity operator is general and can be used to express initial-opacity (only $\pi(0)$ is sensitive), final-opacity (only $\pi(last * 2)$ is sensitive, a focus of this paper), and language-opacity ($\pi(i)$ is sensitive for all i).

Theorem 1. Given a model $\mathcal{M} = (\mathcal{L}, \eta, obs)$ and a state s in $\mathcal{L}$. Let ψ be a path formula. Define:

$$\mathsf{tr}([\![\psi]\!], s) = \{\mathsf{erase}(\pi) : \pi \in \mathsf{path}(\mathcal{L}, s) \mid \mathcal{M} \models_\pi \psi\},$$

then we have:

$$\mathcal{M} \models_s \odot[\psi]$$

iff

$$\mathsf{tr}([\![\psi]\!], s) \text{ is semantically opaque w.r.t. } obs.$$

Here, "semantically opaque" is to be understood as the definition of opaque in Definition 5.

Proof: According to the satisfaction relations, we have:

$$\begin{aligned}
& \mathcal{M} \models_s \odot[\psi] \\
\Leftrightarrow\ & \forall \pi \in \mathsf{path}(\mathcal{L}, s).(\mathcal{M} \models_\pi \psi \ \Rightarrow \\
& \exists \pi' \in \mathsf{path}(\mathcal{L}, s)\ s.t.\ (\mathcal{M} \not\models_{\pi'} \psi \wedge obs(\pi) = obs(\pi'))) \\
\Leftrightarrow\ & \{obs(\pi) \mid \mathcal{M} \models_\pi \psi\} \subseteq \{obs(\pi') \mid \mathcal{M} \models_{\pi'} \neg\psi\} \\
\Leftrightarrow\ & \{\pi \in \mathsf{path}(\mathcal{L}, s) \mid \mathcal{M} \models_\pi \psi\} \\
& \quad \text{are covered by paths violating } \psi. \\
\Leftrightarrow\ & \mathsf{tr}([\![\psi]\!], s) \text{ is semantically opaque.}
\end{aligned}$$

□

The opacity computation problem is the problem of recognising whether there is a path violating ψ but observationally equivalent to each of the ψ satisfying paths.

3.4 Verification of OpacTL

The verification problem of OpacTL is a decision algorithm to check whether $\mathcal{M} \models_s \phi$, for a given model $\mathcal{M}$, and an OpacTL formula ϕ and a starting state s. That is, we need to set up whether the formula ϕ is valid in the initial state s of $\mathcal{M}$. Let $\mathsf{Post}(s)$ denote immediate state successors of s in a path, and $\mathsf{Pre}(s)$ denote the immediate state predecessors of s in a path. The basic procedure follows the conventional CTL model checking [10]:

(i) Convert the OpacTL formulae in a positive normal form, that is, formulae built by the basic modalities $\odot[\mathbf{X}\phi]$, $\odot[\phi\mathbf{U}\phi']$, and $\odot[\phi\mathbf{R}\phi']$, and successively pushing negations inside the formula at hand: $\neg\mathtt{true} \rightsquigarrow \mathtt{false}$, $\neg\mathtt{false} \rightsquigarrow \mathtt{true}$, $\neg\neg\phi \rightsquigarrow \phi$, $\neg(\phi\wedge\phi') \rightsquigarrow \neg\phi\vee\neg\phi'$, $\neg(\phi\vee\phi') \rightsquigarrow \neg\phi\wedge\neg\phi'$, $\neg\mathbf{X}\phi \rightsquigarrow \mathbf{X}\neg\phi$, $\neg(\phi\mathbf{U}\phi') \rightsquigarrow \neg\phi\mathbf{R}\neg\phi'$, $\neg(\phi\mathbf{R}\phi') \rightsquigarrow \neg\phi\mathbf{U}\neg\phi'$;

(ii) Recursively compute the satisfaction sets $\mathsf{Sat}(\phi') = \{s \in S \mid s \models \phi'\}$ for all state subformulae ϕ' of ϕ: the computation carries out a bottom-up traversal of the parse tree of the state formula ϕ starting from the leafs of the parse tree and completing at the root of the tree which corresponds to ϕ, where the nodes of the parse tree represent the subformulae of ϕ and the leafs represent an atomic proposition $\alpha \in \mathsf{Asp}$ or $\mathtt{true}$ or $\mathtt{false}$. All inner nodes are labelled with an operator. For positive normal form formulae, the labels of the inner nodes are $\neg$, $\wedge$, $\odot[\mathbf{X}]$, $\odot[\mathbf{U}]$, $\odot[\mathbf{R}]$. At each inner node, the results of the computations of its children are used and combined to build the states of its associated subformula. In particular, satisfaction sets for conventional state formula are given as follows:

  - $\mathsf{Sat}(\mathtt{true}) = S$,
  - $\mathsf{Sat}(\alpha) = \{t \in S \mid \alpha \in \eta(t)\}$,
  - $\mathsf{Sat}(\neg\phi) = S \setminus \mathsf{Sat}(\phi)$,
  - $\mathsf{Sat}(\phi \wedge \phi') = \mathsf{Sat}(\phi) \cap \mathsf{Sat}(\phi')$,
  - $\mathsf{Sat}(\odot[\psi]) = \{s \in S \mid \mathcal{T}_\odot(\mathcal{M}, s, \psi) = \mathtt{true}\}$;

(iii) Check whether $s \in \mathsf{Sat}(\phi)$.

Furthermore, for the opacity operator $\odot[\psi]$, we compute $\mathsf{Sat}(\odot[\psi])$ as: $s \in \mathsf{Sat}(\odot[\psi])$ iff $\mathcal{T}_\odot(\mathcal{M}, s, \psi) = \mathtt{true}$, $\mathcal{T}_\odot(\mathcal{M}, s, \psi)$ is sketched in Algorithm 1. Specifically, we compute all (regular-expression-like formatted) traces Λ (Λ') starting from s and satisfying (violating) ψ, and check if each such trace in Λ is observationally covered by a such trace in Λ'. Alogrithm 2 $\mathbf{compU}(\mathcal{M}, s, \phi, \phi')$ computes a set of such formatted traces satisfying $\phi\mathbf{U}\phi'$. Similarly, an algorithm $\mathbf{compR}(\mathcal{M}, s, \phi, \phi')$ can be proposed to compute a set of such formatted traces satisfying $\phi\mathbf{R}\phi'$.

Theorem 2. [Soundness of $\odot[\psi]$ translation] Given a model $\mathcal{M}$, a state s, and a path formula ψ:

$$\mathcal{M} \models_s \odot[\psi] \quad \text{iff} \quad \mathcal{T}_\odot(\mathcal{M}, s, \psi) = \mathtt{true}.$$

Algorithm 1: Translating $\odot[\psi]$: $\mathcal{T}_\odot(\mathcal{M}, s, \psi)$

```
Data: $\mathcal{M}, s, \psi$
Result: $\odot[\psi]$
switch $\psi$ do
    case $\mathbf{X}\phi$:  $\mathsf{Sat}(\psi) \leftarrow \cup_{i\in\mathbb{N}}\{tr(s \to s_i) \mid \mathsf{Post}(s) = s_i \wedge s_i \in \mathsf{Sat}(\phi)\}$,
                 $\mathsf{Sat}(\neg\psi) \leftarrow \cup_{i\in\mathbb{N}}\{tr(s \to s_i) \mid \mathsf{Post}(s) = s_i \wedge s_i \in \mathsf{Sat}(\neg\phi)\}$;
    case $\phi\mathbf{U}\phi'$: $\mathsf{Sat}(\psi) \leftarrow \mathbf{compU}(\mathcal{M}, s, \phi, \phi')$,
                 $\mathsf{Sat}(\neg\psi) \leftarrow \mathbf{compR}(\mathcal{M}, s, \neg\phi, \neg\phi')$;
    case $\phi\mathbf{R}\phi'$: $\mathsf{Sat}(\psi) \leftarrow \mathbf{compR}(\mathcal{M}, s, \phi, \phi')$,
                 $\mathsf{Sat}(\neg\psi) \leftarrow \mathbf{compU}(\mathcal{M}, s, \neg\phi, \neg\phi')$;
end
$\Lambda \leftarrow \{\lambda \mid \lambda \in \mathsf{Sat}(\psi)\}$; $\Lambda' \leftarrow \{\lambda \mid \lambda \in \mathsf{Sat}(\neg\psi)\}$;
for each $\lambda \in \Lambda$ do
    find ← false;
    for each $\lambda' \in \Lambda'$ do
        if $obs(\lambda) \subseteq obs(\lambda')$ then
            find ← true; break ;
        end
    end
    if $(\neg$find$)$ then return find;
end
return find.
```

Proof: The proof is obtained by the satisfaction relation of $\odot[\psi]$ and the construction of the translation $\mathcal{T}_\odot(\mathcal{M}, s, \psi)$. Let π (and π') denotes a corresponding path of the regular-expression-like formatted trace λ (and λ' respectively).

$$\begin{aligned}
& \mathcal{T}_\odot(\mathcal{M}, s, \psi) = \mathtt{true} \\
\Leftrightarrow\ & \forall\lambda \in \mathsf{Sat}(\psi) : \exists\lambda' \in \mathsf{Sat}(\neg\psi).obs(\lambda) \subseteq obs(\lambda') \\
\Leftrightarrow\ & \forall\pi.\mathsf{erase}(\pi) \in \lambda \in \mathsf{Sat}(\psi) : \\
& \exists\pi'.\mathsf{erase}(\pi') \in \lambda' \in \mathsf{Sat}(\neg\psi).obs(\pi) = obs(\pi') \\
\Leftrightarrow\ & \forall\pi \in \mathsf{path}(\mathcal{L}, s).\mathcal{M}_\pi \models \psi : \\
& \exists\pi' \in \mathsf{path}(\mathcal{L}, s).\mathcal{M}_\pi \not\models \psi.(obs(\pi) = obs(\pi')) \\
\Leftrightarrow\ & \mathcal{M} \models_s \odot[\psi].
\end{aligned}$$

□

Due to the recursive nature of the CTL model checking algorithm, complexity is linear in the size of the non-opacity state formula ϕ, while the worst case of finding opaque paths for reachability objectives ψ and checking satisfaction of the opacity state formula $\odot[\psi]$, specified in Algorithm 1, is EXPSPACE. Note that the algorithm traverses all traces satisfying ψ and all traces violating ψ, and conducts observation equivalence comparison. So the worst case complexity here follows the complexity of the hyper property model checking problem with two quantifier ($\forall$) alternations, and thus EXPSPACE [11].

The until operator allows to derive the temporal modality $\mathbf{F}$ ("eventually") as usual: $\mathbf{F}\phi \overset{\text{def}}{=} \mathtt{true}\ \mathbf{U}\ \phi$. To simplify expression in the examples, by abuse of notation, we use $\mathbf{F}s$ to denote $\mathbf{F}\phi$ (i.e., ϕ will be eventually true which takes place on state s).

Example 1. Consider the system $\mathcal{M}$ accepting finite inputs presented in Fig. 1 (a). Let s_3 be a sensitive state, s_0 be a starting state, $\{s_3, s_6\}$ be two final states of interests, and the observation function be: $a \to a$, $b \to \epsilon$, $c \to \epsilon$, i.e. b, c are hidden, but a is visible. Consider the security property *eventually reaching* s_3, i.e., $\psi = \mathbf{F}s_3$ in our logic, we have:

$$\mathsf{tr}([\![\psi]\!], s_0) = \{ac(b)^*a(\bot)^*\}, \mathsf{tr}([\![\neg\psi]\!], s_0) = \{aba(c)^*(\bot)^*\}.$$

Here we use e.g. $ac(b)^*a(\bot)^*$ to represent the infinite trace that starts with ac, followed by possibly infinitely many b, followed by a, and followed by possibly in-

Algorithm 2: Computing $\mathsf{Sat}(\phi\mathbf{U}\phi')$: **compU**($\mathcal{M}, s, \phi, \phi'$)

```
Data: $\mathcal{M}, s, \phi, \phi'$
Result: $\mathsf{Sat}(\phi\mathbf{U}\phi')$: a set of regular-expression-like
        formatted traces whose corresponding paths
        satisfying $\phi\mathbf{U}\phi'$
$\Lambda \leftarrow \{\}$; $i \leftarrow 0$ ;
for each $t_i \in \mathsf{Sat}(\phi')$ do
    $T_i \leftarrow \{t_i\}$; $\Pi_i \leftarrow \{\pi \mid \pi(0) = t_i\}$;
    while $\{s_j \in \mathsf{Sat}(\phi) \setminus (T_i \cup \mathsf{Sat}(\phi')) \mid \mathsf{Post}(s_j) \cap T_i \neq \emptyset\} \neq \emptyset$ do
        let $s_j \in \{s_j \in \mathsf{Sat}(\phi) \setminus (T_i \cup \mathsf{Sat}(\phi')) \mid \mathsf{Post}(s_j) \cap T_i \neq \emptyset\}$;
        if $s_j \in \mathsf{Post}(s_j) \cap T_i$ then
            /* There is a self-loop: wrap it with a star
               and concatenate paths starting from a
               state in $\mathsf{Post}(s_j) \cap T_i$ found earlier */
            for each $\pi' \in \Pi_i$ s.t. $\pi'(0) \in \mathsf{Post}(s_j) \cap T_i$ do
                $\Pi_i \leftarrow \Pi_i \cup \{(s_j \xrightarrow{a} s_j)^* + \pi'[1...]\}$;
            end
        end
        for each: $q_1 \in \mathsf{Post}(s_j) \cap T_i, q_2 \in \mathsf{Post}(q_1) \cap T_i, \ldots, q_n \in \mathsf{Post}(q_{n-1}) \cap T_i$ s.t.
          $\mathsf{Post}(q_n) \cap T_i = \emptyset$ do
            if $\mathsf{Pre}(s_j) \notin \{q_1, q_2, \ldots q_n\}$ then
                for each $\pi' \in \Pi_i$ s.t.
                  $\pi'(0) \in \mathsf{Post}(s_j) \cap T_i$ do
                    $\Pi_i \leftarrow \Pi_i \cup \{s_j \xrightarrow{a} q_1 + \pi'[1...]\}$;
                end
            end
            else if $\mathsf{Pre}(s_j) = q_n \wedge s_j \in \mathsf{Post}(q_n)$ then
                /* There is a cycle, wrap it with a star
                   and concatenate paths starting from a
                   state in $\mathsf{Post}(s_j) \cap T_i$ found earlier */
                for each $\pi' \in \Pi_i$ s.t.
                  $\pi'(0) \in \mathsf{Post}(s_j) \cap T_i$ do
                    $\Pi_i \leftarrow \Pi_i \cup \{(s_j \xrightarrow{a_1} q_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n}$
                      $\mathsf{Pre}(s_j) \xrightarrow{a_{n+1}} s_j)^* + \pi'[1...]\}$;
                end
            end
        end
        $T_i \leftarrow T_i \cup \{s_j\}$;
    end
    $\Lambda_i = \{\lambda \leftarrow \mathsf{erase}(\pi) \mid \forall\pi \in \Pi_i\}$; $\Lambda \leftarrow \Lambda \cup \Lambda_i$;
    $i \leftarrow i + 1$;
end
return $\Lambda$.
```

finitely many $\perp$. Clearly:

$$obs(\llbracket\psi\rrbracket) = obs(\llbracket\neg\psi\rrbracket) = \{aa(\perp)^*\},$$

so: $\mathcal{M} \models_{s_0} \odot[\psi]$, i.e., the system is ψ-opaque.

Example 2. Consider the system accepting infinite inputs presented in Fig. 1 (b). Let s_2 be a sensitive state, s_0 be a starting state, $\{s_2, s_6\}$ be two final states of interests, and the observation function be: $a \to \epsilon$, $b \to b$, $c \to \epsilon$, i.e. a and c are hidden, but b isn't. Consider the security property *eventually reaching* s_2, i.e., $\psi = \mathbf{F}s_2$:

$$\mathsf{tr}(\llbracket\psi\rrbracket, s_0) = \{ab(ab)^*\}, \qquad \mathsf{tr}(\llbracket\neg\psi\rrbracket, s_0) = \{acbc(bc)^*\}$$

clearly, $obs(\llbracket\psi\rrbracket) = obs(\llbracket\neg\psi\rrbracket) = \{bb^*\}$, so the system is ψ-opaque, i.e., $\mathcal{M} \models_{s_0} \odot[\psi]$.

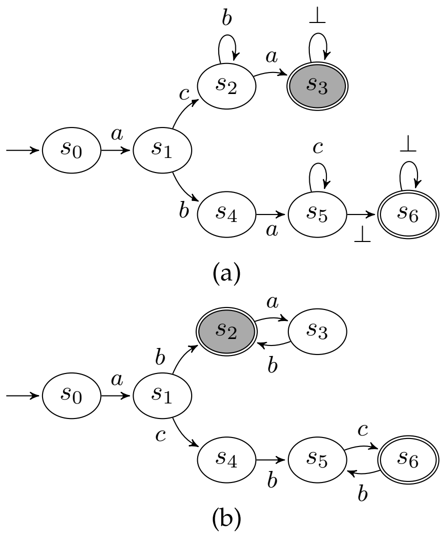


Fig. 1. OpacTL over finite and infinite languages. Grey node denotes a sensitive state.

3.5 Link to non-interference

Information flow policies are designed to ensure that secret data does not influence publicly observable data. Non-interference (NI) [12] essentially says that low security users should not be aware of the activity of high security users. This can be interpreted in many different ways including as a hyperproperty of paths of a system. Appropriate to an LTS is to consider a system with labels classified into two security levels: *low* (L) and *high* (H). The system satisfies non-interference if and only if any sequence of low-level labels that can be produced by the LTS is consistent with all possible sequences of high-level labels. We formalise this notion of NI.

Let $trace(\mathcal{L})$ be the set of all traces produced by paths of $\mathcal{L}$. We consider projections of these traces. Let $\mathcal{L}$ be an LTS over an alphabet Σ which is partitioned into two disjoint sets of labels H and L. A *high projection* of a trace is the sequence of labels in H that remains after all labels in L have been deleted from the original trace, denoted by λ_{H}. Similarly a *low projection* of a trace is the sequence of labels in L that remains after all labels in H have been deleted from the original, denoted by λ_{L}.

Definition 6 (Trace Consistency). Let $t_1, t_2 \in \Sigma^*$ be *consistent* with each other, written $t_1 \approx t_2$ iff $\exists t \in trace(\mathcal{L})$ such that t_1 and t_2 are both projections (can be high or low) of t.

Intuitively, non-interference requires that any observable low trace projection produced by the LTS must be consistent with every high trace projection that can be produced by the LTS.

Definition 7 (Non-interference). Let $\mathcal{L}$ be an LTS, $\Sigma = (\mathsf{H}, \mathsf{L})$. Let $trace_{\mathsf{L}}(\mathcal{L}) = \{l \in \mathsf{L}^* \mid \exists t \in trace(\mathcal{L}) \wedge t_{\mathsf{L}} = l\}$ and similarly define $trace_{\mathsf{H}}(\mathcal{L})$ We say $\mathcal{L}$ satisfies *non-interference* iff

$$\forall l \in trace_{\mathsf{L}}(\mathcal{L}), \forall h \in trace_{\mathsf{H}}(\mathcal{L}) : l \approx h.$$

This is a quite restrictive property but intuitive to derive from the original definition of non-interference. However, relating it to opacity of a trace property has some obstacles. Non-interference is a 2-hyperproperty, i.e., can be expressed as a set of pairs of traces. Opacity is also a 2-hyperproperty but the parameter property, ψ, is not. This means that it is difficult to understand what ψ should be defined. In addition, our definition of non-interference implies for that every high projection and every low projection of a trace, every other possible high projection is a projection from some trace whose low projection is the same. Opacity states something weaker, that there *exists* a trace with the same low projection but a different high projection. This asymmetry means that even if the property of having a certain high trace is opaque, the property of not having that trace is not necessarily opaque.

Consider two LTSs whose trace sets are A and B respectively. Let $A = \{h_1l_1, h_2l_1, h_3l_2, h_4l_2\}$ and let $B = \{h_1l_1, h_2l_1, h_3l_1, h_4l_1, h_1l_2, h_2l_2, h_3l_2, h_4l_2\}$. Here, "high projections of the trace " cannot be expressed as a trace property so we resort to using a family of ψs, one for each high trace. Both A and B satisfy opacity for the family, but only B satisfies non-interference. In particular $h_1 \not\approx l_2$ in A.

Schoepe and Sabelfeld [2] prove equivalence between the two notions for input-output (i.e., length two) traces when the set of opaque properties is strong enough to characterise every possible information leak, i.e., form a family as in our example, and are also symmetric, i.e., both existence and non-existence of the property is opaque. These two requirements coalesce when the family of ψ properties covers all possible high instances. Their proofs specify this family but do not exhibit it.

4 Probabilistic Opaque Temporal Logic OpacPTL

This section extends OpacTL with probabilistic operators that allow the construction of probabilistic models for quantitative opacity analysis and verification. For the purpose of security analysis, the notion of *probabilistic opacity* [4] defines the likelihood of a path in predicate φ which is not covered by a path in $\llbracket\bar{\varphi}\rrbracket$ from the observer's view: the smaller the notion the more secure the system.

Definition 8 (Degree of opacity). The degree of ψ-opaque property is defined as:

$$\mathcal{D}(\odot[\psi]) = \mathsf{Prob}([\![\psi]\!] \setminus obs^{-1}(obs([\![\bar{\psi}]\!]))),$$

i.e., the probability that a ψ-trace is not opaque, where $\mathsf{Prob}(x)$ denotes the probability of x, $[\![\psi]\!]$ and $[\![\bar{\psi}]\!]$ represent the set of traces satisfying and violating property ψ respectively.

We propose to add a probabilistic operator in our logic to capture this definition (the degree of opacity).

4.1 Probabilistic model

We generalise Definition 1 to cater for probabilities. Below $\mathsf{Dist}(X)$ denotes the set of discrete probability distribution over a set X.

Definition 9. A *probabilistic labelled transition system* (pLTS) is a tuple $\mathcal{L} = (S, \Sigma, \mathbb{P}, F)$. Here S and Σ are states, and labels, respectively. $\mathbb{P} : S \rightarrow \mathsf{Dist}(\Sigma \times S)$ is the probabilistic transition relation.

Definition 10. Given a pLTS $\mathcal{L} = (S, \Sigma, \mathbb{P}, F)$, we construct an LTS $(S, \Sigma, \rightarrow, F)$ by setting $\rightarrow = \{(s, l, s') \mid \mathbb{P}(s)(l, s') > 0\}$. By "abus de notation" we will also use $\mathcal{L}$ to refer to the induced LTS $(S, \Sigma, \rightarrow, F)$. We say a pLTS is *deterministic* if the induced LTS is deterministic in the sense of Section 2.

Definition 11. A *probabilistic model* is a tuple $\mathcal{M} = (\mathcal{L}, \eta, obs)$ where: $\mathcal{L}$ is a deterministic pLTS, η is a state labelling function, and *obs* is an observation function. Each probabilistic model $\mathcal{M}$ induces a model in the sense of §3.2, by replacing $\mathcal{M}$'s pLTS with the induced LTS. Abusing notation once more, we will also use $\mathcal{M}$ to denote this induced model.

4.2 OpacPTL

To allow quantitative verification, we add the probabilistic operators to the state formulae:

$$\phi \ ::= \ \ldots \mid \mathbf{P}_{\bowtie p}[\psi] \mid \mathbf{P}_{\bowtie p}[\odot[\psi]] \qquad \psi \ ::= \ \ldots$$

Here $\bowtie \in \{\leq, <, \geq, >\}$, $p \in [0, 1]$. The semantics of the probabilistic operator $\mathcal{M} \models_s \mathbf{P}_{\bowtie p}[\psi]$ means that "the probability, from state s, that ψ is true for an outgoing path satisfies $\bowtie p$": $\mathcal{M} \models_s \mathbf{P}_{\bowtie p}[\psi]$ iff $\mathsf{Prob}([\![\psi]\!]) \bowtie p$. It is standard to extend $\mathbf{P}_{\bowtie p}$ to path formulae ψ as PCTL, i.e., $\mathbf{P}_{\bowtie p}[\psi]$, the procedure can be found in e.g., [10]. A state satisfies a probabilistic operator $\mathbf{P}_{\bowtie p}[\odot[\psi]]$ if the quantity of $\odot[\psi]$ is $\bowtie p$. The semantics of the probabilistic opacity operator is given as:

$$\mathcal{M} \models_s \mathbf{P}_{\bowtie p}[\odot[\psi]] \qquad \text{iff} \qquad \mathcal{D}(\odot[\psi]) \bowtie p$$

Note that the satisfaction relations $\models_\pi$ and $\models_s$ work on the induced model in the sense of §3.2, not the probabilistic model itself. This is standard in probabilistic model checking, see e.g. [10].

Theorem 3. Given a probabilistic model $\mathcal{M} = (\mathcal{L}, \eta, obs)$ and a state s in $\mathcal{L}$. Let ψ be a path formula, and: $\Pi = \{\pi \in \mathsf{path}(\mathcal{L}, s) \mid \mathcal{M} \models_\pi \psi \wedge \pi$ is semantically observable$\}$, then we have:

$$\mathcal{M} \models_s \mathbf{P}_{\bowtie p}[\odot[\psi]] \qquad \text{iff} \qquad \mathsf{Prob}(\Pi) \bowtie p$$

Here semantically observable is to be understood in the sense of Definition 5.

Proof: By the definition of the semantics of the probabilistic operator, we have:

$$\begin{aligned}
& \mathcal{M} \models_s \mathbf{P}_{\bowtie p}[\odot[\psi]] \\
\Leftrightarrow\ & \mathcal{D}(\odot[\psi]) \bowtie p \ \Leftrightarrow\ \mathsf{Prob}([\![\psi]\!] \setminus obs^{-1}(obs([\![\bar{\psi}]\!]))) \bowtie p \\
\Leftrightarrow\ & \mathsf{Prob}(\{\pi \in \mathsf{path}(\mathcal{L}, s) \mid \mathcal{M} \models_\pi \psi\} \ \setminus \\
& \quad \{obs^{-1}(obs(\pi')) \in \mathsf{path}(\mathcal{L}, s) \mid \mathcal{M} \models_{\pi'} \neg\psi\}) \ \bowtie p \\
\Leftrightarrow\ & \mathsf{Prob}(\{\pi \in \mathsf{path}(\mathcal{L}, s) \mid \mathcal{M} \models_\pi \psi \ \wedge \\
& \quad \nexists \pi'.(\mathcal{M} \models_{\pi'} \neg\psi \wedge obs(\pi) = obs(\pi'))\}) \bowtie p \\
\Leftrightarrow\ & \mathsf{Prob}(\{\pi \in \mathsf{path}(\mathcal{L}, s) \mid \mathcal{M} \models_\pi \psi \ \wedge \\
& \quad \pi \text{ is not covered by a path violating } \psi\}) \bowtie p \\
\Leftrightarrow\ & \mathsf{Prob}(\Pi) \bowtie p
\end{aligned}$$

□

Example 3. Consider the model presented in Fig. 2. Let s_3 be a sensitive state, s_0 be a starting state, $\{s_3, s_6\}$ be final states of interests, and the observation function be: $a \rightarrow a$, $b \rightarrow b$, $c \rightarrow \epsilon$. Consider the security property be *eventually reaching* s_3, i.e., $\psi = \mathbf{F} s_3$, so:

$$tr([\![\psi]\!]) = \{ac(b)^* a(\bot)^*\}, \qquad tr([\![\neg\psi]\!]) = \{aba(c)^*(\bot)^*\}.$$

Let 0.1 be the threshold quantity, $obs(\psi) = \{a(b)^* a(\bot)^*\}$, $obs(\neg\psi) = \{aba(\bot)^*\}$, so:

$$\begin{aligned}
& \mathbf{P}_{\leq 0.1}(\odot[\psi]) \\
\Leftrightarrow\ & \mathbf{P}_{\leq 0.1}\{[\![\psi]\!] \setminus obs^{-1}(obs([\![\neg\psi]\!]))\} \\
\Leftrightarrow\ & \mathbf{P}_{\leq 0.1}(\{aca\bot^*, acbba\bot^*, \ldots\}) \\
\Leftrightarrow\ & \frac{1}{3} * \frac{1}{2} + \frac{1}{3} * (\frac{1}{2})^2 * \frac{1}{2} + \cdots + \frac{1}{3} * (\frac{1}{2})^n \frac{1}{2} \leq 0.1 \\
\Leftrightarrow\ & \frac{1}{4} \leq 0.1 = \mathtt{false}
\end{aligned}$$

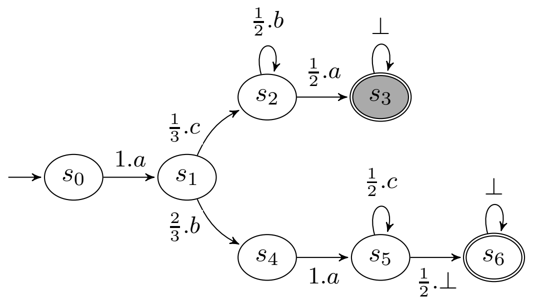


Fig. 2. Example: OpacPTL

4.3 Verification of OpacPTL

Intuitively, verification of probabilistic opacity answers the question "is the system opaque?" quantitatively, relative to a secret property ψ and the observability of the adversary given a threshold probability p. Given a probabilistic model

$\mathcal{M}$, a starting state s, and a property ψ required to be secure, the probabilistic verification problem of opacity property $\odot[\psi]$ is to decide whether $\mathcal{M} \models_s \mathbf{P}_{\bowtie p}(\odot[\psi])$ holds or not. Algorithm 3 presents the procedure of finding all *probabilistic non-opaque (observable)* traces $p\Lambda(s, \bar{\odot}\psi)$ starting at s. Note that the probability of each formatted traces satisfying ψ is also calculated and associated with the trace for quantitative purpose. Then we can calculate:

$$\mathcal{D}(\odot[\psi]) = \sum_{p\lambda \in p\Lambda(s,\bar{\odot}\psi)} \mathsf{Prob}(p\lambda).$$

Algorithm 3: $\mathcal{PT}_{\odot}(\mathcal{M}, s, \psi)$: finding probabilistic non-opaque traces

```
Data: $\mathcal{M}, s, \psi$
Result: probabilistic non-opaque traces $p\Lambda(s, \bar{\odot}\psi)$
switch $\psi$ do
    case $\mathbf{X}\phi$:  $\mathsf{Sat}(\psi) \leftarrow \cup_{i\in\mathbb{N}}\{tr(s \rightarrow s_i) \mid \mathsf{Post}(s) = s_i \wedge s_i \in \mathsf{Sat}(\phi)\}$,
                $\mathsf{Sat}(\neg\psi) \leftarrow \cup_{i\in\mathbb{N}}\{tr(s \rightarrow s_i) \mid \mathsf{Post}(s) = s_i \wedge s_i \in \mathsf{Sat}(\neg\phi)\}$;
    case $\phi\mathbf{U}\phi'$: $\mathsf{Sat}(\psi) \leftarrow \mathbf{compU}(\mathcal{M}, s, \phi, \phi')$,
                $\mathsf{Sat}(\neg\psi) \leftarrow \mathbf{compR}(\mathcal{M}, s, \neg\phi, \neg\phi')$;
    case $\phi\mathbf{R}\phi'$: $\mathsf{Sat}(\psi) \leftarrow \mathbf{compR}(\mathcal{M}, s, \phi, \phi')$,
                $\mathsf{Sat}(\neg\psi) \leftarrow \mathbf{compU}(\mathcal{M}, s, \neg\phi, \neg\phi')$;
end
$p\Lambda \leftarrow \{p\lambda \mid p\lambda.tr \leftarrow \lambda \wedge p\lambda.pr \leftarrow \mathsf{Prob}(\lambda)$ for $\lambda \in \mathsf{Sat}(\psi)\}$;
$p\Lambda' \leftarrow \{p\lambda \mid p\lambda.tr \leftarrow \lambda \wedge p\lambda.pr \leftarrow \mathsf{Prob}(\lambda)$ for $\lambda \in \mathsf{Sat}(\neg\psi)\}$;
$p\Lambda'' = \{\}$;
for each $p\lambda \in p\Lambda$ do
    for each $p\lambda' \in p\Lambda'$ do
        if $obs(p\lambda.tr) \subseteq obs(p\lambda'.tr)$ then
            $p\Lambda'' \leftarrow p\Lambda'' \cup \{p\lambda\}$; break ;
        end
    end
end
$p\Lambda_{\bar{\odot}} \leftarrow p\Lambda \setminus p\Lambda''$;     /* non-opaque probabilistic traces */;
return $p\Lambda_{\bar{\odot}}$.
```

Theorem 4 (Soundness of $\mathbf{P}_{\mathcal{M}}(\odot[\psi])$ translation). Given a model $\mathcal{M}$, starting state s, a probability threshold p, and a security property ψ:

$$\mathcal{M} \models_s \mathbf{P}_{\bowtie p}(\odot[\psi]) \qquad \text{iff} \qquad \mathsf{Prob}(\mathcal{PT}_{\odot}(\mathcal{M}, s, \psi)) \bowtie p.$$

Proof: The proof is obtained by the satisfaction relation of $\mathbf{P}_{\bowtie p}(\odot[\psi])$ and the construction translation of $\mathcal{PT}_{\odot}(\mathcal{M}, s, \psi)$ described in Algorithm 3. The algorithm will terminate since $\mathsf{Sat}(\psi)$ are computed as a set of regular-expression-like formatted traces satisfying ψ as Algorithm 1. Probability of such a trace is calculated by multiplication of the probability of each transition label for non-cycle part, and multiplication of $p/(1-p)$ for a cycle with probability p. □

We have implemented our translation as an extension of the PRISM model checker [13]. Example 4 presents the result generated by the prototype tool.

Example 4. Consider the following example used in [4] presented in Fig. 3. Let s_0 be a starting state, $\{t_0, t_1, t_2, t_3, t_4, t_5\}$ be final states, observation function be: $a \rightarrow a$, $b \rightarrow \epsilon$, $c \rightarrow c$, and $x \rightarrow \epsilon$. Assume the property of interest is the system *terminating at sensitive states* $\{t_2, t_3, t_5\}$. PRISM also allows to directly specify properties which evaluate to a value using $\mathbf{P} =?[\psi]$. The property specification is given as:

$$\mathbf{P} =?[\odot\ \mathbf{F}\ (((s = 2) \vee (s = 3) \vee (s = 5)) \wedge (t = 1))],$$

where $s = i$ denotes t_i is a sensitive state, $t = 1$ denotes the status of terminating. We can automatically calculate the probabilistic opacity of the system as:

```
Result: 0.026.{0.01046:bcax:ca,0.0156:ca(b)*x:ca}
        (value in the initial state)
```

where 0.026 denotes the probability of opacity of the system, i.e., the probability of traces satisfying ψ but not covered by those observationally equivalent traces violating ψ, which include: $bcax$ with probability 0.0104 and $ca(b)^*x$ with probability 0.0156, both observed as ca.

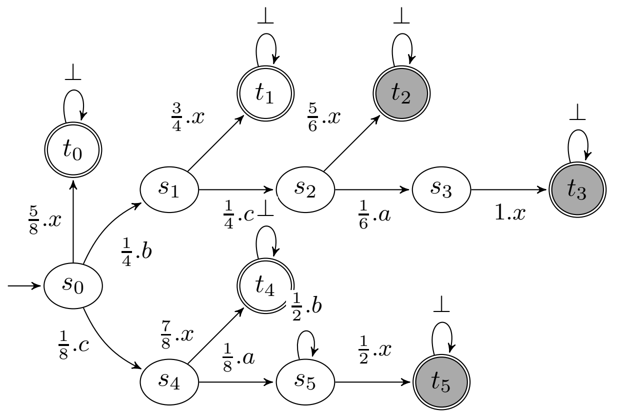


Fig. 3. Example: probabilistic opacity, grey node denotes a sensitive state.

5 Implementation and Examples

We have built a prototype tool for verification of opacity as an extension of the PRISM model checker [14]. Models are described in an extension of the PRISM modelling language with observations and transition labels. The new model type is denoted as "ldtmc". Properties are described in an extension of the PRISM's property specification language with the opacity operator. The tool and details of all examples and case studies are available from [13].

5.1 Modelling probabilistic opacity in PRISM

Models in PRISM are described in a state-based language based on guarded commands. A model is constructed as a number of *modules* which can interact. Each module contains a set of finite-valued variables which define the state of the module. The behaviour of each module is described by a set of guarded commands in the form of:

```
[<action>] <guard> -> <prob> : <update> + ...
```

```
                + <prob> : <update>;
```

The *guard* is a predicate over the variables of all the modules in the model. Each *update* describes a probabilistic transition which specifies how the variables of the module are updated if the guard is true. The *prob* attached with each update specifies the probability that the corresponding state transition takes place. The *action* label is optional which allows modules to synchronise over commands.

We have extended the existing modelling language for model type "ldtmc" to allow: (i) the definition of *observation functions*:

```
observations
 <label> -> <observable>, ... <label> -> <observable>;
endobservations
```

which defines each label and its observation through the keyword `observations`; (ii) and the specification of *transition labels* over updates is in the form:

```
[]<guard> -> <prob>:<LABEL>:<update> + ...
                    + <prob>:<LABEL>:<update>;
```

5.2 Modelling dining cryptographers

Anonymity is an important concept in security. To illustrate the versatility of our work, we use our logic to express anonymity of the dining cryptographers protocol [15]. Consider that three cryptographers 1, 2 and 3 are sharing a meal at a restaurant. At the end of the meal, at most one cryptographer will pay the bill, and they would like to check whether the bill has been paid or not, but the cryptographers respect each other's right to make an anonymous payment. A two-stage protocol is performed to solve the problem: (i) every two cryptographers establish a shared one-bit secret: each of them flips a coin, the outcome is only visible to himself and the cryptographer on his right; (ii) each cryptographer publicly announces whether the two outcomes agree or disagree, if the cryptographer is not the payer, he says the truth, otherwise, he states the opposite of what he sees. When all cryptographers have announced, they count the number of disagrees. If that number is odd, then one of them has paid, but no other cryptographer is able to deduce who is the payer.

Without loss of generality, let us assume cryptographer 3 is the observer who tries to know which of the other two paid the bill if the bill has been paid. The observer does not know the initial state of the cryptographer $i = 1, 2$ being the payer (p_i), he knows the outcome of the flipping coins of himself and of the cryptographer-1: *head* (h_i) or *tail* (t_i) where $i = 1, 3$ but does not know that of the cryptographer-2, he knows the procedure of the protocol and he can hear what cryptographer i says: *agree* (a_i) or *disagree* (d_i), and therefore he knows the outcome of the disagreement counting is *even* (e) or *odd* (o). In other words, labels p_i, h_2, t_2 are invisible to him, while labels h_1, t_1, h_3, t_3, d_i, a_i, e and o are visible to him. Therefore the set of observables includes: $\Theta = \{d_i, a_i, e, o \mid 1 \le i \le 3\} \cup \{h_i, t_i \mid i = 1, 3\}$, and the observation function on the transition labels of the model is specified as: $p_1, p_2, t_2, h_2 \to \epsilon$; $a_i \to a_i$, $d_i \to d_i$; $h_1 \to h_1$; $t_1 \to t_1$, $h_3 \to h_3$; $t_3 \to t_3$; $e \to e$; $o \to o$. Our tool can automatically check the property "cryptographer 1 is the payer" ($\psi = \mathbf{X}(payer = c_1)$) is *opaque* if the bill has been paid. The property specification is given as: $\mathbf{P} =?[\odot\, \mathbf{X}\, (payer = c_1)]$. Our tool answers the question "c_1 *is the payer* is ψ-opaque" as follows:

```
Result: 0.0.{} (value in the initial state)
```

The result shows that there are no observable traces found.

5.3 Modelling a location privacy example

Consider a simple example of location privacy releasing by credit card records presented in Fig. 4 which describes the card holder's activities. Assume the adversary can observe the credit card records to track partial location information and the observations are given as: station $\to s$, work $\to \epsilon$, travel $\to \epsilon$, office $\to \epsilon$, coffeshop $\to c$, bankA $\to b$, bankB $\to b$, airport $\to a$, home $\to \epsilon$, L1 $\to \epsilon$, L2 $\to \epsilon$, L3 $\to \epsilon$. Suppose that the states leading to the final location L1, L2 and L3 are sensitive. Then we have the property specification as: $\mathbf{P} =?[\odot\ \mathbf{F}\ dest]$, where $dest$ denotes states q_9, q_{10} and q_{11}, led by locations L1, L2 and L3 respectively. The result generated by the tool is as follows:

```
Result: 0.3333.
        {0.1667:stationtravelbankBairportL1:sba,
        0.1667:stationtravelbankBairportL2:sba}
        (value in the initial state)
```

The result meets our intuition, that the traces leading to L1 and L2 with observation sba and sba are not covered by traces leading to insensitive final location states.

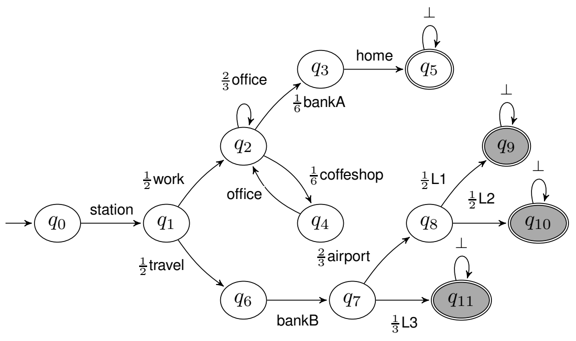


Fig. 4. Modelling a location privacy example

6 Entropy of the Opacity Formula

Probabilistic specification $\mathbf{P}(\odot[\psi])$ calculates the probability of non-ψ-opaque behaviours of a model $\mathcal{M}$. In this section, we provide some discussions on an alternative measurement of the opacity formula based on the notion of *entropy*:

$$\mathcal{H}(\odot[\psi]).$$

We adapt the definition of the entropy of a language (of finite words) [16] to calculate this.

Definition 12 ($\mathcal{H}[\odot[\psi]]$). Given a model $\mathcal{M} = (\mathcal{L}, \eta, obs)$. Let ψ be a path formula, the entropy of the opacity formula $\odot[\psi]$ is defined as:

$$\mathcal{H}(\odot[\psi]) = \limsup_{n\to+\infty} \frac{\log_2(1 + \mid [\![\psi]\!] \setminus obs^{-1}(obs([\![\bar{\psi}]\!]))]_n \mid)}{n},$$

where:

$$\begin{aligned} \mid [\![\psi]\!] \setminus obs^{-1}(obs([\![\bar{\psi}]\!]))]_n \mid \quad = \quad & \mid \{\pi \in \mathsf{path}(\mathcal{L}, s) \mid \mathcal{M} \models_\pi \psi \\ & \wedge \mid \mathsf{erase}(\pi) \mid = n \\ & \wedge\ \pi \text{ is transparent}\} \mid . \end{aligned}$$

Intuitively, the entropy of $\odot[\psi]$ can be understood as the amount of information (in bits per symbol) in typical words of (the language of) ψ-transparent.

It is easy to slightly change Algorithm 1 to calculate the size of transparent traces. We can then take the prefix of those traces with length of n and compute the entropy of ψ-opaque property using Definition 12.

Example 5. Consider again the models presented in Example 1. It is easy to see that for any $n \in \mathbb{N}$, $|\,[\odot[\mathbf{F}s_i]]_n\,| = 0$, so:

$$\mathcal{H}[\odot[\mathbf{F}s_i]] = \limsup_{n\to+\infty} \frac{\log_2(1+0)}{n} = 0,$$

where $i = 3$ for model (a) and $i = 2$ for model (b). The result shows the degree of transparency of traces satisfying $\psi = \mathbf{F}s_i$ is 0 in the notion of entropy, and implies the model is $[\mathbf{F}s_i]$-opaque.

Example 6. Consider again the models presented in Fig. 2. It is easy to see that for any $n \in \mathbb{N}$, $|\,[\odot[\mathbf{F}s_3]]_n| = n$, so:

$$\mathcal{H}[\odot[\mathbf{F}s_3]] = \limsup_{n\to+\infty} \frac{\log_2(1+n)}{n} = 0.$$

The result shows the degree of transparency of traces satisfying $\psi = \mathbf{F}s_3$ is 0 in the notion of entropy, and implies the model is $[\mathbf{F}s_3]$-opaque. Note that, the entropy-based measurement is not as precise as the probabilistic-based measurement. But it somehow meets our intuition: when $n \to \infty$, most of the traces satisfying $[\mathbf{F}s_3]$ are covered by traces violating $[\mathbf{F}s_3]$ (only $acba\perp^*$ is not covered) from the observer's view.

[17] formulated the basic entropy-based properties in model checking context. We adapt their discussions here to our scenario for opacity properties, to illustrate some intuition of entropy-based measurement. Consider a model $\mathcal{M}$, and an opacity formula $\odot[\psi]$. Let Π and $\Pi(\odot[\psi])$ denote all the behaviours of the model and all the (in)finite paths satisfying $\odot[\psi]$, intuitively:

- $\mathcal{H}(\Pi)$ measures the quantity of all the behaviours of the system,
- $\mathcal{H}(\Pi \cap \Pi(\odot[\psi]))$ measures the quantity of the ψ-opaque behaviours of the system,
- $\mathcal{H}(\Pi) - \mathcal{H}(\Pi \cap \Pi(\odot[\psi]))$ quantifies how difficult to steer the system to be ψ-opaque,
- and $\mathcal{H}(\Pi \setminus \Pi(\odot[\psi])) = \mathcal{H}(\odot[\psi])$ measures the quantity of non-ψ-opaque (transparent) behaviours of the system.

In general, for any language accepted by a given finite well-structured model $\mathcal{M}$, its entropy can be effectively computed using linear algebra [16]. Let $A(\mathcal{M})$ denote the extended adjacency matrix of $\mathcal{M}$:

$$A(\mathcal{M}) = |\{a \in \Sigma \mid s \xrightarrow{a} t \in \rightarrow\}|.$$

Theorem 5. [16] For any finite deterministic trimmed model $\mathcal{M}$, the entropy of the paths accepted by $\mathcal{M}$ can be calculated as:

$$\mathcal{H}(\mathsf{path}(\mathcal{M})) = \log_2 \rho(A(\mathcal{M})),$$

where $\rho(A)$ is the spectral radius the matrix A, i.e., maximal modulus of its eigenvalues.

If we can find a finite deterministic model accepting the specified paths satisfying the property (this is out range of this paper), we can then calculate the entropy of the property as the logarithm of a spectral radius using Theorem 5.

Example 7. Consider a model presented in Fig. 5 (a) named $\mathcal{M}_a$. Let s_3 be a sensitive state, and s_0 be the starting state, and the observation function be: $a \to \epsilon$, $b \to b$, $c \to c$. Consider the security property *eventually reaching* s_3, i.e., $\psi = \mathbf{F}s_3$ in our logic. It is easy to see that the model presented in Fig. 5 (b), say $\mathcal{M}_b$, accepts all $\mathbf{F}s_3$-transparent paths. We can then calculate $\mathcal{H}(\odot[\mathbf{F}s_3])$ by Theorem 5 as:

$$\mathcal{H}(\odot[\mathbf{F}s_3]) = \log_2 \rho(A(\mathcal{M}_b)) = \log_2 \frac{1+\sqrt{5}}{2}.$$

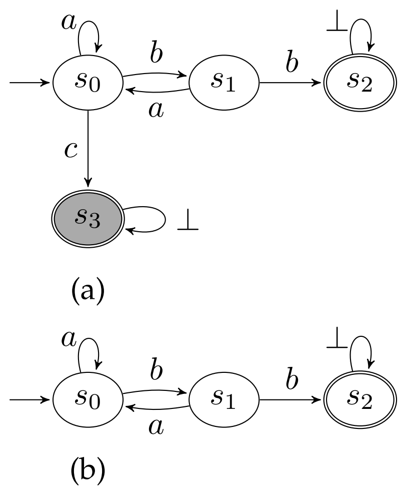


Fig. 5. Example: Entropy of $\odot[F\ s_3]$ for model (a). Grey node denotes a sensitive state. (b) is the model accepting all $F\ s_3$-transparent paths.

7 Related Work

We present related literature from the perspective of specification and verification of information security policies. This work relates to the specification of information security policies and (quantitative) verification of opacity properties.

Security policies. There are a number of information security policies for confidentiality: non-deducibility [18] is designed to keep attacker observable events consistent with possible variations of secret inputs, but this policy is not able to protect secret outputs and to address covert channels; non-interference [12] is one of the most popular flow policies but it is too strong for practical applications; quantified non-interference [19] is then introduced to relax the absolute non-interference policy by computing the amount of the interference; declassification policies [20] control the release of information; (quantified) opacity [3], [4], [6], [21] is considered as a more general property where the sensitive information can be contained in the input, output, and each transition step, which can lead to the definition of initial, final, and language-based opacity. Opacity is a promising approach for describing and unifying security properties.

Recently, quantified opacity [3], [4], [21] has been studied in terms of probability and information entropy.

This work focuses on general policies using opacity. Opacity has reasonable potential being a good choice for specifying flow security properties for modern communication systems due to the feature of partial observability and uncertainty of the environment.

Verification of opacity. Opacity was first introduced in the context of cryptographic protocols in [22], [23]. Later on, Bryans et al investigated opacity properties in systems modelled as Petri nets [24] and labelled transition systems [6], where the secret was specified as predicates over system runs. Generally speaking, the opacity properties can be classified into two families: *language-based opacity* [25], [26], [27] where the secret predicate is regarding to a subset of system runs; and *state-based opacity* [5], [6], [28], [29], [30] where the secret predicate is referring to a subset of states. Intuitively, the system is language-based opaque if for any word w in the secret language L_S, there is at least one other word w' in the non-secret language L_{NS} equivalent to w based on the adversary's observations; the system is considered as state-based opaque if the adversary is not able to induce whether the initial state (initial-state opacity), current state (current-state opacity [6], [28]), the state a few steps ago (K-step opacity [28]), or the initial-final state pair (Initial-and-Finial state opacity [5]) is a secret state or not. Dubreil [26] studied opacity verification of infinite-state systems using *approximate* models. Kobayashi and Hiraishi [31] investigated the approach of verification of opacity for infinite-state discrete event systems modelled by pushdown automaton. They showed that opacity of pushdown systems is undecidable. The relationship among variant notions of opacity has been studied [5], [29], [32] and transformation mappings between them described – enabling efficient use of existing verification approaches.

This paper focuses on an easy-expressing logic OpacTL and OpacPTL for *specifying* opacity, and applies probabilistic model checking techniques for automatically *verifying* opacity properties with guarantees. Automated verification approach is built on solid foundations and provides the rigorous guarantees needed to give confidence and identify subtle flaws in a security system. Quantitative aspects enable designers to effectively weigh and arbitrate between concerns such as security and performance. Quantitative verification therefore turns to be a good fit for the analysis of security property. We build the prototype tool as an extension of PRISM [14] for the quantitative verification perspective, and will consider to integrate opacity enforcement mechanism into our framework as a future work.

Logics for security properties. Linear temporal logic formulas cannot express properties of sets of execution paths, i.e. hyperproperties are required for specifying information flow policies, such as non-interference properties. Computational tree logic formulas cannot express observational determinism even if path quantifiers are defined, hence the need to develop specialised logics that characterise information flow properties for security policies. Dimitrova [33] proposed SecLTL, an extension of LTL with a hide operator, for specifying path-based integration of information flow policies in reactive systems. The hide operator specifies requirements such that the observable behaviour of a system needs to be independent of the choice of a *secret*. Clarkson et. al [34] proposed HyperLTL and HyperCTL* as an extension of propositional linear-time temporal logic (LTL) and branching-time temporal logic (CTL*) respectively for the purpose of specifying hyperproperties. HyperLTL and HyperCTL* specified information flow policies by explicit quantification over multiple traces. Epistemic logic [35] is a different approach to reason about information flow properties from perspective of *knowledge*, rather than *secrets* as other logics for information flow properties. Specifically, *knowledge operators* are introduced to temporal logics to express knowledge-based information flow properties for imperative programs [36].

Comparing with the above works, we focus on elementarily expressing and verifying observability properties for security systems. Logic for Hyperproperties can be more expressive than our logic, but we aim to propose easy-expressing specification of opacity and observability property for information security concerns in particular. The logic OpacTL proposed in this paper can be used to specifying the opacity property in a very straightforward way for security systems, and the property required to be secret can be defined flexibly regarding users' requirements. For instance, the user can require that a particular state such as initial, next or final of the system should be kept secret. In addition, our logic OpacPTL can specify opacity security properties in a quantitative way, allowing us to reason about the degree of satisfaction or violation of the security property of interest. Such a degree is measured based on both *probability* and *entropy* of observable behaviours of the model, which is novel and promising.

Quantitative security properties Opacity and related concepts were first studied and related to information flow properties in a *qualitative* context in [6]. In the probabilistic context, opacity has been studied in [3], [4], [37]. [37] studied the notion of opacity in the probabilistic computational world. There opacity was based on the probabilities of observer's pre-beliefs on the truth of the predicate. The work in [3] presents a quantitative information leakage analysis in terms of probabilistic opacity. A number of quantitative opacity notions are introduced in [4] which can be applied in information flow security analysis.

In this paper, the measurement of opacity has been studied in two perspectives. We measure the probability of observable behaviours to which extent the model satisfies a sensitive property, and apply probabilistic model checking technique to automate the approach. In many situations probabilistic verification is highly relevant, but there is an important limitation: for many interesting properties, the probability is either 0 or 1 (too precise) and thus no quantitative sense analysis [17]. We therefore also study the entropy of observable behaviours regarding to a sensitive property of interest. This allows us to measure the amount of information in bits per symbol in typical behaviours.

8 Conclusions

We have proposed a novel, probabilistic logic for simply expressing the opacity of labelled transition system properties and demonstrated how opacity in this context can be checked using an extension of the PRISM model checker.

We also provide a discussion towards an entropy-based measurement of the opacity formulae. Given the flexibility of opacity as a security property there are many possible directions for future research. Promising directions include applying our technique to location privacy protocols and generalising the opacity framework to games and robotic systems modelled as partially observable transition systems in order to provide better decision procedures. We also propose to develop approaches towards an entropy-based measurement of the opacity formulae.

References

[1] L. Mazaré, "Using unification for opacity properties," in *In Proceedings of the Workshop on Issues in the Theory of Security (WITS'04*, 2004, pp. 165–176.

[2] D. Schoepe and A. Sabelfeld, "Understanding and enforcing opacity," in *CSF*, 2015, pp. 539–553.

[3] B. Bérard, J. Mullins, and M. Sassolas, "Quantifying opacity," in *QEST*, 2010, pp. 263–272.

[4] J. W. Bryans, M. Koutny, and C. Mu, "Towards quantitative analysis of opacity," in *TGC*, 2012, pp. 145–163.

[5] Y. Wu and S. Lafortune, "Comparative analysis of related notions of opacity in centralized and coordinated architectures," *Discrete Event Dynamic Systems*, vol. 23, no. 3, pp. 307–339, 2013.

[6] J. Bryans, M. Koutny, L. Mazaré, and P. Y. A. Ryan, "Opacity generalised to transition systems," *Int. J. Inf. Sec.*, vol. 7, no. 6, pp. 421–435, 2008.

[7] A. Saboori and C. N. Hadjicostis, "Verification of initial-state opacity in security applications of discrete event systems," *Inf. Sci.*, vol. 246, pp. 115–132, 2013.

[8] ——, "Current-state opacity formulations in probabilistic finite automata," *IEEE Trans. Automat. Contr.*, vol. 59, no. 1, pp. 120–133, 2014.

[9] E. M. Clarke and E. A. Emerson, "Design and synthesis of synchronization skeletons using branching-time temporal logic," in *Logics of Programs*, 1981, pp. 52–71.

[10] C. Baier and J.-P. Katoen, *Principles of Model Checking*. The MIT Press, 2008.

[11] B. Bonakdarpour and B. Finkbeiner, "The complexity of monitoring hyperproperties," *CoRR*, vol. abs/2101.07847, 2021.

[12] J. A. Goguen and J. Meseguer, "Security policies and security models," in *S & P*, 1982, pp. 11–20.

[13] "Prototype tool and case studies," https://bitbucket.org/cmu777/prism-opac.git, 2020.

[14] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, ser. LNCS, G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806. Springer, 2011, pp. 585–591.

[15] D. Chaum, "The dining cryptographers problem: Unconditional sender and recipient untraceability," *Journal of Cryptology*, vol. 1, pp. 65–75, 1988.

[16] N. Chomsky and G. A. Miller, "Finite state languages," *Information and Control*, vol. 1, no. 2, pp. 91–112, 1958.

[17] E. Asarin, M. Blockelet, A. Degorre, C. Dima, and C. Mu, "Entropy model checking," in *QAPL*, 2014.

[18] D. Sutherland, "A model of information," in *CCS*, 1986, pp. 1113–1119.

[19] D. Clark, S. Hunt, and P. Malacaria, "Quantitative analysis of the leakage of confidential data," *Electr. Notes Theor. Comput. Sci.*, vol. 59, no. 3, pp. 238–251, 2001.

[20] A. Sabelfeld and D. Sands, "Dimensions and principles of declassification," in *CSFW*, 2005, pp. 255–269.

[21] B. Bérard, K. Chatterjee, and N. Sznajder, "Probabilistic opacity for Markov decision processes," *Inf. Process. Lett.*, vol. 115, no. 1, pp. 52–59, 2015.

[22] A. Boisseau, "Abstractions pour la vérification de propriétés de sécurité de protocoles cryptographiques," Ph.D. dissertation, École Normale Supérieure de Cachan, France, 2003.

[23] L. Mazaré, "Decidability of opacity with non-atomic keys," in *FAST*, 2004, pp. 71–84.

[24] J. Bryans, M. Koutny, and P. Y. A. Ryan, "Modelling dynamic opacity using petri nets with silent actions," in *FAST*, 2004, pp. 159–172.

[25] E. Badouel, M. A. Bednarczyk, A. M. Borzyszkowski, B. Caillaud, and P. Darondeau, "Concurrent secrets," *Discrete Event Dynamic Systems*, vol. 17, no. 4, pp. 425–446, 2007.

[26] J. Dubreil, "Opacity and abstractions," in *APNOC'09*, Paris, France, June 2009.

[27] M. Ben-Kalefa and F. Lin, "Supervisory control for opacity of discrete event systems," in *CCC*, 2011, pp. 1113–1119.

[28] A. Saboori and C. N. Hadjicostis, "Notions of security and opacity in discrete event systems," in *CDC*, 2007, pp. 5056–5061.

[29] ——, "Verification of k-step opacity and analysis of its complexity," *IEEE Trans. Automation Science and Engineering*, vol. 8, no. 3, pp. 549–559, 2011.

[30] Y. Falcone and H. Marchand, "Runtime enforcement of k-step opacity," in *CDC*, 2013, pp. 7271–7278.

[31] K. Kobayashi and K. Hiraishi, "Verification of opacity and diagnosability for pushdown systems," *J. Applied Mathematics*, vol. 2013, pp. 654 059:1–654 059:10, 2013.

[32] F. Cassez, J. Dubreil, and H. Marchand, "Synthesis of opaque systems with static and dynamic masks," *Formal Methods in System Design*, vol. 40, no. 1, pp. 88–115, 2012.

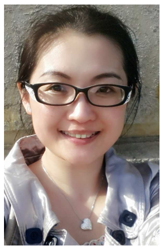
[33] R. Dimitrova, B. Finkbeiner, M. Kovács, M. N. Rabe, and H. Seidl, "Model checking information flow in reactive systems," in *VMCAI*, 2012, pp. 169–185.

[34] M. R. Clarkson, B. Finkbeiner, M. Koleini, K. K. Micinski, M. N. Rabe, and C. Sánchez, "Temporal logics for hyperproperties," in *POST*, 2014, pp. 265–284.

[35] J. W. G. III and P. F. Syverson, "A logical approach to multilevel security of probabilistic systems," pp. 164–176, 1992.

[36] M. Balliu, M. Dam, and G. L. Guernic, "Epistemic temporal logic for information flow security," in *PLAS*, 2011, p. 6.

[37] Y. Lakhnech and L. Mazaré, "Probabilistic opacity for a passive adversary and its application to chaum's voting scheme," *IACR Cryptology ePrint Archive*, vol. 2005, p. 98, 2005.



Chunyan Mu is a senior lecturer at Teesside University. Her research interests include information flow security, language-based security, and quantitative verification.

David Clark is a reader at University College London. His research interests include the use of information theory in both software testing and information flow control.