Quantifying Information Leakage of Randomized Protocols*

Fabrizio Biondi[1], Axel Legay[2], Pasquale Malacaria[3], and Andrzej Wąsowski[1]

[1] IT University of Copenhagen, Denmark
[2] INRIA Rennes, France
[3] Queen Mary University of London, United Kingdom

Abstract. The quantification of information leakage provides a quantitative evaluation of the security of a system. We propose the usage of Markovian processes to model and analyze the information leakage of deterministic and probabilistic systems. We show that this method generalizes the lattice of information approach and is a natural framework for modeling refined attackers capable to observe the internal behavior of the system. We also use our method to obtain an algorithm for the computation of channel capacity from our Markovian models. Finally, we show how to use the method to analyze timed and non-timed attacks on the Onion Routing protocol.

1 Introduction

Quantification of information leakage is a recent technique in security analysis that evaluates the amount of information about a secret (for instance about a password) that can be inferred by observing a system. It has sound theoretical bases in Information Theory [1,2]. It has also been successfully applied to pratical problems like proving that patches to the Linux kernel effectively correct the security errors they address [3]. It has been used for analysis of anonymity protocols [4,5] and analysis of timing channels [6,7]. Intuitively, *leakage of confidential information of a program is defined as the difference between the attacker's uncertainty about the secret before and after available observations about the program* [1].

The underlying algebraic structure used in leakage quantification for *deterministic* programs is the *lattice of information* (LoI) [1]. In the LoI approach an attacker is modelled in terms of possible observations of the system she can make. LoI uses an equivalence relation to model how precisely the attacker can distinguish the observations of the system. An execution of a program is modeled as a relation between inputs and observables. In this paper we follow the LoI approach but take a process view of the system. A process view of the system is a more concise representation of behaviour than an observation relation. Moreover a process view does not require that the system is deterministic, which allows us

* The research presented in this paper has been partially supported by MT-LAB, a VKR Centre of Excellence for the Modelling of Information Technology.

to handle randomized protocols—for the first time using a generic, systematic and implementable LoI-based methodology.

We use Markov Decision Processes to represent the probabilistic partial-information semantics of programs, using the nondeterminism of the model for the choices that depend on the unknown secret. We define the leakage directly on such model. With our method we can distinguish the inherent randomness of a randomized algorithm from the unpredictability due to the lack of knowledge about the secret. We exploit this distinction to quantify leakage only for the secret, as the information leakage about the random numbers generated is considered uninteresting (even though it is an information in information theoretical sense). We thus work with both deterministic and randomized programs, unlike the previous LoI approach.

We give a precise encoding of an *attacker* by specifying her prior knowledge and observational capabilities. We need to specify which of the logical states of the system can be observed by the attacker and which ones he is able to distinguish from each other. Given a program and an attacker we can calculate the leakage of the program to the attacker.

We also show how to turn the leakage computation into leakage optimization: we compute the maximum leakage over all possible prior information of attackers *ceteris paribus*, or in other words, the leakage for the worst possible attacker without specifying the attacker explicitly. This maximum leakage is known as the *channel capacity* of the system [8]. Since we are able to model a very large class of attackers the obtained channel capacity is robust. Computing channel capacity using this method requires solving difficult optimization problems (as the objective is nonlinear), but we show how the problem can be reduced to standard reward optimization techniques for Markovian models for a class of interesting examples.

Our method can be applied to finite state systems specified using a simple imperative language with a randomization construct. It can also be used for systems modeled directly as Markov Decision Processes. We demonstrate the technique using an MDP model of the known Onion Routing protocol [9], showing that we can obtain the channel capacity for a given topology from an appropriate Markov Decision Process describing the probabilistic partial information behavior of the system. Also, our behavioral view of the system allows us to encode an attacker with time-tracking capabilities and prove that such an attacker can leak more information than the canonical attacker that only observes the traffic on the compromised nodes. Timing-based attacks to the Onion Routing protocol have been implemented before [10,11], but to our best knowledge the leakage of timing-attacks has not been quantified before.

Our contributions include:

– A method for modeling attack scenarios consisting of process models of systems and observation models of attackers, including a simple partial-observability semantics for imperative programs, so that these models can also be obtained from code.

– A definition of leakage that generalizes the LoI approach to programs with randomized choices (strictly including the class of deterministic programs), and dually the first application of the LoI approach to process specifications of systems.
– A method for computing leakage for scenarios modeled as described above. The method is fully implementable.
– A method to parameterize the leakage analysis on the attacker's prior information about the secret, to allow the computation of channel capacity by maximizing an equation characterizing leakage as a function of prior information.
– The worst-case analysis of the Onion Routing protocol when observed by non time-aware and time-aware attackers able to observe the traffic passing through some compromised nodes.

The paper proceeds as follows. Section 2 provides the core background on probabilistic systems and the LoI approach. Section 3 gives an overview of our new leakage quantification method. The non-obvious steps are further detailed in Sections 4–6. In Sect. 7 we explain how to use the method for computing channel capacity, and we use this technique to analyze leakage in the onion routing protocol against untimed and timing attacks (Sect. 8). We discuss the related work (Sect. 9) and conclude (Sect. 10).

2 Background

2.1 Markovian Models

Definition 1. *A tuple $\mathcal{C} = (S, s_0, P)$ is a Markov Chain (MC), if S is a finite set of states, $s_0 \in S$ is the initial state and P is an $|S| \times |S|$ probability transition matrix, so $\forall s, t \in S.\ P_{s,t} \geq 0$ and $\forall s \in S.\ \sum_{t \in S} P_{s,t} = 1$.*

The probability of transitioning from any state s to a state t in k steps can be found as the entry of index (s, t) in P^k [12]. We call $\pi^{(k)}$ the probability distribution vector over S at time k and $\pi_s^{(k)}$ the probability of visiting the state s at time k; note that $\pi^{(k)} = \pi_0 P^k$, where $\pi_s^{(0)}$ is 1 if $s = s_0$ and 0 otherwise.

A state $s \in S$ is *absorbing* if $P_{s,s} = 1$. In the figures we will not draw the looping transition of the absorbing states, to reduce clutter.

Let $\xi(s, t)$ denote the *expected residence time* in a state t in an execution starting from state s given by $\xi(s, t) = \sum_{n=0}^{\infty} P_{s,t}^n$. We will write ξ_s for $\xi(s_0, s)$.

Given a Markov chain $\mathcal{C} = (S, s_0, P)$ let a *discrimination relation* $\mathcal{R}$ be an equivalence relation over S. Given $\mathcal{C}$ and $\mathcal{R}$ define the *quotient of $\mathcal{C}$ by $\mathcal{R}$* as a new Markov chain $\mathcal{C}/\mathcal{R} = (S/\mathcal{R}, s_0', P')$ where

– $S/\mathcal{R}$ is the set of the equivalence classes of S induced by $\mathcal{R}$
– s_0' is the equivalence class of s_0

– $P' : S/\mathcal{R} \times S/\mathcal{R} \to [0,1]$ is a probability transition function between equivalence classes of $S/\mathcal{R}$ such that

$$\forall c, d \in S/\mathcal{R}.\ P'_{c,d} = \frac{1}{|c|} \sum_{\substack{s \in c \\ t \in d}} P_{s,t}$$

Given k Markov chains $\mathcal{C}^1 = (S^1, s_0^1, P^1)$,...,$\mathcal{C}^k = (S^k, s_0^k, P^k)$ their synchronous *parallel composition* is a MC $\mathcal{C} = (S, s_0, P)$ where S is $S^1 \times ... \times S^k$, s_0 is $s_0^1 \times ... \times s_0^k$ and $P_{s^1 \times ... \times s^k, t^1 \times ... \times t^k} = \prod_{i=1}^{k} P_{s^i,t^i}$.

Definition 2. *A* Markov Decision Process *(MDP) is a tuple $\mathcal{P} = (S, s_0, P, \Lambda)$ where S is a finite set of states containing the initial state s_0, Λ_s is the finite set of available actions in a state $s \in S$ and $\Lambda = \bigcup_{s \in S} \Lambda_s$, and $P : S \times \Lambda \times S \to [0,1]$ is a transition probability function such that $\forall s, t \in S. \forall a \in \Lambda_s.\ P(s,a,t) \geq 0$ and $\forall s \in S. \forall a \in \Lambda_s.\ \sum_{t \in S} P(s,a,t) = 1$.*

We we will write $s \xrightarrow{a} [P_1 \mapsto t_1, ..., P_n \mapsto t_n]$ to denote that in state $s \in S$ the system can take an action $a \in \Lambda_s$ and transition to the states $t_1, ..., t_n$ with probabilities $P_1, ..., P_n$.

We will enrich our Markovian models with a finite set $\mathtt{V}$ of integer-valued variables, and an assignment function $A : S \to \mathbb{Z}^{|\mathtt{V}|}$ assigning to each state the values of the variables in that state. We will use the expression $\mathtt{v}_s$ to denote the value of the variable $\mathtt{v} \in \mathtt{V}$ in the state $s \in S$. Later we will use the values of the variables to define the discrimination relations, as explained in Section 6.

2.2 Reward and Entropy of a Markov Chain

A real-valued reward functions on the transitions of a MC $\mathcal{C} = (S, s_0, P)$ is a function $R : S \times S \to \mathbb{R}$. Given a reward function on transitions, the expected reward $R(s)$ for a state $s \in S$ can be computed as $R(s) = \sum_{t \in S} P_{s,t} R(s,t)$, and the expected total reward $R(\mathcal{C})$ of $\mathcal{C}$ as $R(\mathcal{C}) = \sum_{s \in S} R(s) \xi_s$.

The entropy of a probability distribution is a measure of the unpredictability of the events considered in the distribution [13]. Entropy of a discrete distribution over the events $x \in X$ is computed as $\sum_{x \in X} \mathbf{P}(x) \log_2 \frac{1}{\mathbf{P}(x)}$=-$\sum_{x \in X} \mathbf{P}(x) \log_2 \mathbf{P}(x)$. We will sometimes write $H(\mathbf{P}(x_1), \mathbf{P}(x_2), .., \mathbf{P}(x_n))$ for the entropy of the probability distribution over $x_1, ..., x_n$.

Since every state s in a MC $\mathcal{C}$ has a discrete probability distribution over the successor states we can calculate the entropy of this distribution. We will call it *local entropy*, $L(s)$, of s: $L(s) = -\sum_{t \in S} P_{s,t} \log_2 P_{s,t}$. Note that $L(s) \leq \log_2(|S|)$.

As a MC $\mathcal{C}$ can be seen as a discrete probability distribution over all of its possible traces, we can assign a single entropy value $H(\mathcal{C})$ to it. The global entropy $H(\mathcal{C})$ of $\mathcal{C}$ can be computed by considering the local entropy $L(s)$ as the expected reward of a state s and then computing the expected total reward of the chain [14]:

$$H(\mathcal{C}) = \sum_{s \in S} L(s) \xi_s$$

2.3 Lattice of Information

Let Σ be a finite set of observables over a deterministic program $\mathcal{P}$. Consider all possible equivalence relations over Σ; each of them represents the discriminating power of an attacker. Given two equivalence relations $\approx, \sim$ over Σ define a refinement ordering as

$$\approx \sqsubseteq \sim \quad \text{iff} \quad \forall \sigma_1, \sigma_2 \in \Sigma \ (\sigma_1 \sim \sigma_2 \Rightarrow \sigma_1 \approx \sigma_2) \tag{1}$$

The ordering forms a *complete lattice* over the set of all possible equivalence relations over Σ [15]: the Lattice of Information (abbreviated as `LoI`).

If $\approx \sqsubseteq \sim$ then classes in $\sim$ refine (split) classes in $\approx$, thus $\sim$ represents an attacker that can distinguish more while $\approx$ represents an attacker that can distinguish less observables.

By equipping the set Σ with a probability distribution we can see an equivalence relation as a random variable (technically it is the set theoretical kernel of a random variable but for information theoretical purposes can be considered a random variable [1]). Hence the `LoI` can be seen as a lattice of random variables.

The connection between `LoI` and leakage can be illustrated by this simple example: consider a password checking program checking whether the user input is equal to the secret `h`. Then an attacker observing the outcome of the password check will know whether the secret is `h` or not, hence we can model the leakage of such a program with the equivalence relation $\{\{\mathtt{h}\}, \{x | x \neq \mathtt{h}\}\}$.

More generally, observations over a deterministic program $\mathcal{P}$ form an equivalence relation over the possible states of $\mathcal{P}$. A particular equivalence class will be called an observable. Hence an observable is a set of states indistinguishable by an attacker making that observation. If we consider an attacker able to observe the outputs of a program then the random variable associated to a program $\mathcal{P}$ is given by the equivalence relation on any two states σ, σ' from the universe of program states Σ defined by

$$\sigma \simeq \sigma' \iff [\![\mathcal{P}]\!](\sigma) = [\![\mathcal{P}]\!](\sigma') \tag{2}$$

where $[\![\mathcal{P}]\!]$ represents the denotational semantics of $\mathcal{P}$ [16]. Hence the equivalence relation amounts to "having the same observable output". This equivalence relation is nothing else than the set-theoretical kernel of the denotational semantic of $\mathcal{P}$ [17].

Given a random variable associated to an attacker's observations of a deterministic program $\mathcal{P}$ the *leakage* of P is then defined as the Shannon entropy of that random variable. It is easy to show that for deterministic programs such entropy is equal to the difference between the attacker's a priori and a posteriori uncertainty about the secret and that it is zero if and only if the program is secure (i.e. non interferent) [1].

More intentional attackers in the LoI setting are studied in [18,7], however this is the first work where LoI is used to define leakage in a probabilistic setting.

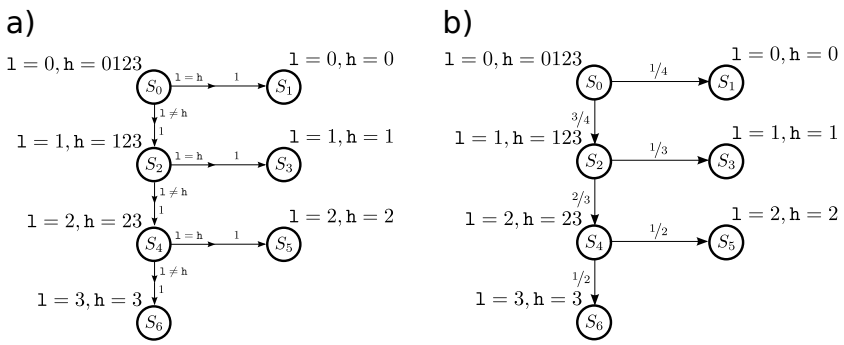

Fig. 1. Simple loop example a) MDP semantics b) MC model

3 Information Leakage of Markov Chains

We begin with an overview of the proposed technique for leakage quantification. It proceeds in five steps, that are all fully automatable for finite state programs. Let a *scenario* be a pair $(\mathcal{P}, \mathcal{A})$, where $\mathcal{P}$ is the system we want to analyze and $\mathcal{A}$ is an attacker. We will call $\mathcal{P}$ the *program*, even if it can be any system suitably modeled as an MDP as explained in Sect. 4.

Step 1: Define a MDP representing $\mathcal{P}$ (Sections 4, 8). We first give a probabilistic semantics to the program in the form of an MDP, in which probabilistic choices are represented by successor state distributions and branching is represented by decision states. This is more or less standard definition of operational semantics for randomized imperative programs.

Example [17]. A program has two variables `l` and `h`. Variable `h` is 2-bit long and private, while variable `l` is public. The attacker can read `l` but not `h`:

```
l = 0; while (l != h) do l = l + 1;
```

The MDP representing the probabilistic partial information semantics of the program is depicted in Fig. 1a. The states in which the system stops and produces an output are encoded with the absorbing states of the MDP, i.e. the states with a probability of transitioning to themselves equal to 1. In the MDP in Fig. 1a states S_1, S_3, S_5 and S_6 are absorbing states.

Step 2: Define the attacker $\mathcal{A}$. An attacker is an external agent observing the system to infer information about its private data. We assume that the attacker knows the implementation of the system (white-box), but is not necessarily able to observe and discriminate all the logical states of the system at runtime. We specify the prior information about the system that the attacker might have, and which system states she can observe and discriminate at runtime.

Definition 3. *An attacker is a triple $\mathcal{A} = (\mathcal{I}, \mathcal{R}_{\mathcal{A}}, \mathcal{T}_{\mathcal{A}})$ where $\mathcal{I}$ is a probability distribution over the possible values of the secret encoding the attacker's prior*

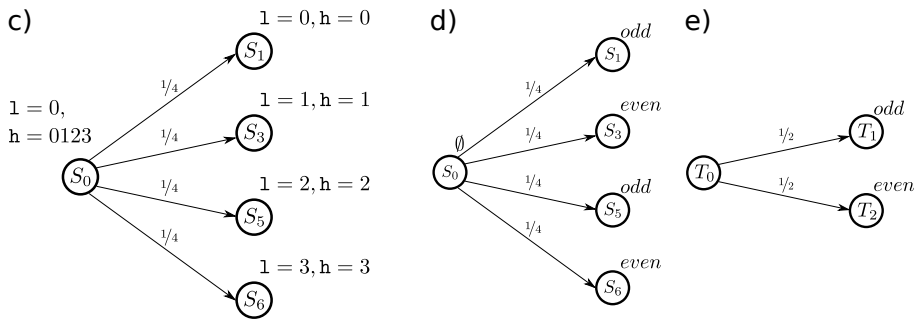

Fig. 2. Simple loop example c) Observable reduction d) Relabeling e) Quotient

information about it, $\mathcal{R}_\mathcal{A}$ is a discrimination relation over the states of the system in which two states are in the same class iff the attacker cannot discriminate them, and $\mathcal{T}_\mathcal{A} \subseteq S$ is the set of states hidden to the attacker.

Example. In our example we will use the following attacker: $\mathcal{I} = (1/4, 1/4, 1/4, 1/4)$ (no prior information), $\mathcal{T}_\mathcal{A} = (S_2, S_4)$ (cannot observe internal states) and $\mathcal{R}_\mathcal{A} = \{(S_1, S_5), (S_3, S_6)\}$ (cannot distinguish states S_1 from S_5 and S_3 from S_6).

Step 3: Resolve the nondeterminism in the MDP. To transform the MDP in a MC, an thus compute leakage, we need to exploit the prior information $\mathcal{I}$ of the attacker. We use it to compute a probability distribution over possible values of private variables in each states of the MDP. To do this for a given state s we just need to normalize $\mathcal{I}$ on the allowed values of the private variables for the state. The probability of the each action $a \in \Lambda_s$ is computed as the probability of the event labelling a given the probability distribution over the values of the secret in s. We will denote the obtained MC by $\mathcal{C}$.

Example. In state S_0 the probability distribution over `h` is $\mathcal{I} = (1/4, 1/4, 1/4, 1/4)$ and `l=0`. The program transitions to state S_1 if `h=l` and to state S_2 if `h`$\neq$`l`. We have that P_{S_0,S_1} is $\mathbf{P}(\mathtt{h} = \mathtt{l}|S_0) = 1/4$ and the probability distribution on `h` in S_1 is $(1, 0, 0, 0)$. Complementarily, P_{S_0,S_2} is $3/4$ and the probability distribution on `h` in S_2 is $(0, 1/3, 1/3, 1/3)$. Figure 1b shows the outcome after repeating this step in all states of the MDP of Fig. 1a.

Step 4: Hide non-observable states (Sect. 5). In the above example the attacker cannot observe the internal states of the system. We expressed this by taking $\mathcal{T}_\mathcal{A} = (S_2, S_4)$. Since these states are not observable, we remove them from the MC and redistribute the probability of visiting them to their successors. If a hidden state has no or only hidden successors, it will never produce any observable—we call this event *divergence*. In general we assume that the observer can understand if the program diverges, so divergence is one of the possible outputs of the system. We write $\mathbb{C}$ for the MC resulting from hiding in $\mathcal{C}$ the states of $\mathcal{T}_\mathcal{A}$. We call $\mathbb{C}$ the *observable reduction* of the scenario.

Example. Figure 2c presents the observable reduction for the running example.

Step 5: Compute the leakage (Sect. 6). From the observable reduction $\mathbb{C}$ and the attacker's discrimination relation $\mathcal{R}_\mathcal{A}$ we can compute the leakage for the scenario $(\mathcal{P}, \mathcal{A})$. The definition of leakage for this model is based on the quotient operator for Markov chains. A quotiented MC $\mathcal{C}/\mathcal{R}$ captures the view of the chain when observed by an agent able to distinguish equivalence classes of $\mathcal{R}$. Let $\mathcal{R}_h$ be a discrimination relation that relates states with the same possible values of the secret that is finer than probabilistic bisimulation. Then leakage is the mutual information between the attacker and the system as seen by an agent able to discriminate only states with different values of the secret:

Definition 4. *Let $(\mathcal{P}, \mathcal{A})$ be a scenario, $\mathcal{A} = (\mathcal{I}, \mathcal{R}_\mathcal{A}, \mathcal{T}_\mathcal{A})$ an attacker, $\mathbb{C}$ the observable reduction of the scenario and $\mathcal{R}_h = \{(s,t) \in S | h_s = h_t\}$. Then the information leakage of $\mathcal{P}$ to $\mathcal{A}$ is*

$$I(\mathbb{C}/\mathcal{R}_h; \mathbb{C}/\mathcal{R}_\mathcal{A}) = H(\mathbb{C}/\mathcal{R}_h) + H(\mathbb{C}/\mathcal{R}_\mathcal{A}) - H(\mathbb{C}/\mathcal{R}_\mathcal{A} \cap \mathcal{R}_h).$$

Corollary 1. *If $\mathcal{P}$ is a deterministic program, then the leakage is $H(\mathbb{C}/\mathcal{R}_\mathcal{A})$.*

Example. Recall that in the running example the attacker is only able to read the parity of `l`. We have that $\mathcal{R}_\mathcal{A} = \{(S_1, S_5), (S_3, S_6)\}$. We name the equivalence classes *even* and *odd* and relabel the state with the classes (see Fig. 2d). The quotient $\mathbb{C}/\mathcal{R}_\mathcal{A}$ is depicted in Fig. 2e. The program is deterministic, so by Corollary 1 the leakage of the scenario is equivalent to the entropy of such quotient, or 1 bit [14].

4 Handling Randomized Imperative Programs

We give a simple probabilistic partial-observation semantics for an imperative language with randomization. This semantics, akin to abstract interpretation, derives Markovian models of finite state programs automatically. Let all variables be integers of predetermined size and class (public, private) declared before execution. Private variables are read-only, and cannot be observed externally. Denote by l (resp. h) names of public (resp. private) variables; by p reals from $[0; 1]$; by `label` all program points; by `f` (`g`) pure arithmetic (Boolean) expressions. Assume a standard set of expressions and the following statements:

$$\begin{aligned} \textit{stmt} ::= \;& l \texttt{ := f(}l\ldots\texttt{)} \mid l \texttt{ := rand } p \mid \texttt{skip} \mid \texttt{goto label} \mid \\ & \texttt{return} \mid \texttt{if g(}l\ldots\texttt{,}h\ldots\texttt{) then } \textit{stmt-list} \texttt{ else } \textit{stmt-list} \end{aligned}$$

The first statement assigns to a public variable the value of expression `f` depending on other public variables. The second assigns zero with probability p, and one with probability $1-p$, to a public variable. The `return` statement outputs values of all public variables and terminates. A conditional branch first evaluates an expression `g` dependent on private and public variables; the first list of statements is executed if the condition holds, and the second otherwise. For simplicity, all statement lists must end with an explicit jump, as in: `if g(l,h) then ...; goto`

$$\frac{pc\text{: } \texttt{skip}}{(pc, L, H) \xrightarrow{\top} [1 \mapsto (pc+1, L, H)]} \qquad \frac{pc\text{: } \texttt{v} := \texttt{f(l)}}{(pc, L, H) \xrightarrow{\top} [1 \mapsto (pc+1, L[^{\texttt{f(l)}}/_{\texttt{v}}], H)]}$$

$$\frac{pc\text{: } \texttt{v := rand } p}{(pc, L, H) \xrightarrow{\top} [p \mapsto (pc+1, L[^{0}/_{\texttt{v}}], H), (1-p) \mapsto (pc+1, L[^{1}/_{\texttt{v}}], H)]}$$

$$\frac{pc\text{: } \texttt{goto label}}{(pc, L, H) \xrightarrow{\top} [1 \mapsto (\texttt{label}, L, H)]} \qquad \frac{pc\text{: } \texttt{return}}{(pc, L, H) \xrightarrow{\top} [1 \mapsto (pc, L, H)]}$$

$$\frac{pc\text{: } \texttt{if g(l,h) then la: A else lb: B}}{(pc, L, H) \xrightarrow{\texttt{g(l,h)}} [1 \mapsto (\texttt{la}, L, H|\texttt{g(l,h)})]}$$

$$\frac{pc\text{: } \texttt{if g(l,h) then la: A else lb: B}}{(pc, L, H) \xrightarrow{\neg\texttt{g(l,h)}} [1 \mapsto (\texttt{lb}, L, H|\neg\texttt{g(l,h)})]}$$

Fig. 3. Execution rules in probabilistic partial information semantics.

`done; else` ...`; goto done; done:` ... . Each program can be easily transformed to this form. Loops can be added in a standard way as a syntactic sugar.

The probabilistic partial-information semantics assumes an external view of the program, so private variables are not visible. A state in this view is a triple (pc, L, H), where pc is the current program counter, L maps public variables to integer values of the appropriate size, and H maps private variables to sets of their possible values. If the observer knows nothing about a private variable h, the set $H(h)$ holds all the values of h's type. If the observer holds some prior information, or learns through interaction with the system, this set is smaller.

The semantics (Fig. 3) is a small-step operational semantics with transitions from states to distributions over states, labeled by expressions dependent on h (only used for the conditional statement). It generates an MDP over the reachable state space. In Fig. 3, `v`, `l` are public variables and `h` is a private variable. Expressions in rule consequences stand for values obtain in a standard way. $L[^{X}/_{l}]$ denotes substition of X as the new value for l in mapping L. Finally, $H|\texttt{g}$ denotes a restriction of each set of possible values in a mapping H, to contain only values that are consistent with Boolean expression `g`. Observe that the `return` rule produces an absorbing state—this is how we model termination in an MDP. The `rand` rules produces a proper distribution, unlike the other Dirac distributions. The `if` rule produces a nondeterministic decision state.

In the obtained MDP states are labelled by values of public variables and sets of values of private variables. Actions from each state represent the secret-dependent events for the state. Our leakage quantification technique works for any MDP of this shape, even the ones not necessarily obtained from code. In Sect. 8 we will create such a model directly from a topology of the Onion Routing protocol.

1. Take $\mathcal{C} \setminus \mathcal{T} = (S, s_0, \mathbb{P})$ and $\mathbb{P} = P$
2. Add to the MC the divergence state $\uparrow$ with $\mathbb{P}_{\uparrow,\uparrow} = 1$
3. Choose a hidden state $t \in \mathcal{T}$, or terminate if $\mathcal{T}$ is empty
4. Let $Pred(t) = \{s \in S \setminus \{t\} \mid \mathbb{P}_{s,t} > 0\}$ be the set of predecessors of t
5. Let $Succ(t) = \{u \in S \setminus \{t\} \mid \mathbb{P}_{t,u} > 0\}$ be the set of successors of t
6. If $\mathbb{P}_{t,t} = 1$:
   (a) For each state $s \in Pred(t)$ set $\mathbb{P}_{s,\uparrow} = \mathbb{P}_{s,t}$
   (b) Remove t from S and $\mathcal{T}$ and go back to step 3
7. Else
   (a) For each $u \in Succ(t)$ set $\mathbb{P}_{t,u} := \frac{\mathbb{P}_{t,u}}{1-\mathbb{P}_{t,t}}$
   (b) Set $\mathbb{P}_{t,t} = 0$
   (c) For each $s \in Pred(t)$ and $u \in Succ(t)$ set $\mathbb{P}_{s,u} := \mathbb{P}_{s,u} + \mathbb{P}_{s,t}\mathbb{P}_{t,u}$
   (d) Remove t from S and $\mathcal{T}$ and go back to step 3

Fig. 4. Computing $\mathcal{C} \setminus \mathcal{T} = (S \setminus \mathcal{T}, s_0, \mathbb{P})$ for a MC $\mathcal{C} = (S, s_0, P)$ and hidden states $\mathcal{T} \subset S$

5 Hiding Non-observable States

In the simple loop example of Sect. 3 the attacker is unable to observe states S_2 and S_4; we call these non-observable states *hidden*. His view of the system is thus adequately represented by the MC in Fig. 2c. In this figure the probability of transferring from the state S_0 to state S_5 is the probability of reaching S_5 from S_0 in the MC of Fig. 1b *eventually*, so after visiting zero or more hidden states.

Note that the initial state cannot be hidden, as we assume the attacker knows that the system is running. This assumption does not restrict the power of the approach, since one can always model a system, whose running state is unknown, by prefixing its initial state by a pre-start state, making it initial, and hiding the original initial state.

We present the hiding algorithm in Fig. 4. We will overload the symbol $\setminus$ to use for the hiding operation: we write $\mathcal{C} \setminus \mathcal{T}$ for the observable MC obtained from $\mathcal{C}$ by hiding the states in set $\mathcal{T}$. If a system stays in a hidden state forever, we say it *diverges*. Divergence will be symbolized by a dedicated absorbing state named $\uparrow$. Otherwise, we compute the new successor probabilities for t; we accomplish this by setting the probability of transitioning from t to itself to 0 and normalizing the other probabilities accordingly. Then we compute the probability that each of its predecessors s would transition to each of its successors u via t and add it to the transition probability from s to u, and finally we remove t from the MC.

The difference between states that cannot be discriminated and hidden states is of primary importance. The former assumes that the attacker is aware of the existence of such states, and thus knows when the system is in one of them, but is not able to discriminate them because they share the same observable properties. For instance, if the attacker can only read the system's output he will not be able to discriminate between different states that produce the same output. In contrast the attacker has no way to observe the behavior of the system when it is in an hidden state, not even by indirect methods like keeping track of the

discrete passage of time. For instance, if the attacker can only read the system's output, the states of the system that produce no output will be hidden to him.

6 Collapsing Non-discriminable States

Discrimination relations are equivalence relations that we use to encode the fact that some states cannot be observed separately by the attacker, since they share some observable properties. Different attackers are able to observe different properties of the states, and thus discriminate them differently.

The discrimination relation $\mathcal{R}_{\mathcal{A}}$ represents the attacker's inability to determine when the system is in a particolar state due to the fact that different states have the same observable properties. We define equivalence classes based on $\mathcal{R}_{\mathcal{A}}$, and the attacker knows that the system is in one of these classes but not in which state. This is encoded by relabelling the states of the MC with their equivalence classes in $\mathcal{R}_{\mathcal{A}}$ and then quotienting it by $\mathcal{R}_{\mathcal{A}}$.

We need to impose a restriction to $\mathcal{R}_{\mathcal{A}}$, since not all discrimination relations are suitable for encoding attackers: the attacker is always able to discriminate states if they behave differently in the relabelled model. Let $\mathcal{C}^{\mathcal{R}_{\mathcal{A}}}$ be the MC $\mathcal{C}$ in which the states are labeled with their equivalence class in $S/\mathcal{R}_{\mathcal{A}}$. Then $\mathcal{R}_{\mathcal{A}}$ encodes the discrimination relation of an attacker only if the states with the same label in $\mathcal{C}^{\mathcal{R}_{\mathcal{A}}}$ are probabilistically bisimilar.

As a result of this condition, all traces in $\mathcal{C}/\mathcal{R}_{\mathcal{A}}$ are relabelled projections of traces in $\mathcal{C}$. This is fundamental to prevent the attacker from expecting traces that do not appear in the actual computation. It also allows us to generalize the discrimination relation ordering used in the `LoI` approach [1]. Let $\mathcal{A}_1 = (\mathcal{I}_1, \mathcal{T}_{\mathcal{A}_1}, \mathcal{R}_{\mathcal{A}_1})$ and $\mathcal{A}_2 = (\mathcal{I}_2, \mathcal{T}_{\mathcal{A}_2}, \mathcal{R}_{\mathcal{A}_2})$ be two attackers, and define

$$\mathcal{A}_1 \sqsubseteq \mathcal{A}_2 \quad \text{iff} \quad \mathcal{I}_1 = \mathcal{I}_2 \wedge \mathcal{T}_{\mathcal{A}_1} = \mathcal{T}_{\mathcal{A}_2} \wedge \mathcal{R}_{\mathcal{A}_1} \subseteq \mathcal{R}_{\mathcal{A}_2}$$

Theorem 1. *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two attackers such that $\mathcal{A}_1 \sqsubseteq \mathcal{A}_2$. Then for any program $\mathcal{P}$, the leakage of the scenario $(\mathcal{P}, \mathcal{A}_1)$ is greater or equal then the leakage of the scenario $(\mathcal{P}, \mathcal{A}_2)$.*

Effectively, the attacker that is able to discriminate more states (a language-like qualitative property) is able to leak more information (an information-theoretical quantitative property). The attacker with the highest leakage can discriminate all states, thus its discrimination relation is the identity; the attacker with the lowest leakage cannot discriminate any state from any other, and thus has leakage 0.

The common definition of leakage of the `LoI` approach [2] assumes that the attacker can observe the different output of a deterministic system. It can be easily encoded in our method. Consider a deterministic program $\mathcal{P}$ with a low-level variable $\mathtt{o}$ encoding the output of the program. Let the an attacker $\mathcal{A}_{I/O}$ have $\mathcal{R}_{\mathcal{A}_{I/O}} = \{(s,t) \in S \times S \mid \mathtt{o}_s = \mathtt{o}_t\}$ and $\mathcal{T}_{\mathcal{A}_{I/O}}$ being the set of all internal states of the MDP semantics of $\mathcal{P}$. The following proposition states that such attacker is the one considered in [2]:

Theorem 2. *Let $(\mathcal{P}, \mathcal{A}_{I/O})$ be a scenario, $\mathcal{A}_{I/O}$ being the attacker defined above. Then $H(\mathcal{C}/\mathcal{R}_{\mathcal{A}_{\mathcal{I}/\mathcal{O}}}) = Leakage(\mathcal{P})$.*

7 Computing Channel Capacity

The method we presented computes the leakage for a scenario, but it is common in security to ask what is the leakage of a given program in the worst-case scenario, i.e. for the scenario with the highest leakage. We consider the maximum leakage over all the attackers with the same discrimination relation $\mathcal{R}_{\mathbb{A}}$ and hidden states $\mathcal{T}_{\mathbb{A}}$ but different prior information $\mathcal{I}$. We define a class of attackers this way because maximizing over all discrimination relations would just conclude that the attacker able to discriminate all states leaks all the information in the system. The maximum leakage for a class of attackers is known as channel capacity, and it is the upper bound to the leakage of the system to any attacker [8]:

Definition 5. *Let $\mathcal{P}$ be a program and $\mathbb{A}$ the class of all attackers with discrimination relation $\mathcal{R}_{\mathbb{A}}$ and hidden states $\mathcal{T}_{\mathbb{A}}$. Let $\hat{\mathcal{A}} \in \mathbb{A}$ be the attacker maximizing the leakage of the scenario $(\mathcal{P}, \mathcal{A})$ for all $\mathcal{A} \in \mathbb{A}$. Then the channel capacity of $\mathcal{P}$ is the leakage of the scenario $(\mathcal{P}, \hat{\mathcal{A}})$.*

To compute it we procede as follows. We first transform the MDP semantics of $\mathcal{P}$ in a parameterized MC with constraints. Then we define a MC and a reward function from it such that the expected total reward of the MC is equivalent to the leakage of the system. Then we extract an equation with constraints characterizing this reward as a function of the prior information $\mathcal{I}$ of the attacker. Finally, we maximize the equation and obtain the maximum leakage, i.e. the channel capacity. In the next Section we will apply this method to compute the channel capacity of attacks to the Onion Routing protocol.

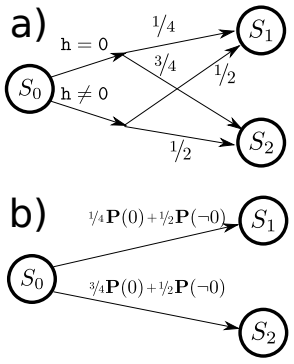

Fig. 5. Reduction from MDP to parameterized MC

Step 1: Find the parameterized MC. We abuse the notation of Markov chain allowing the use of variables in the transition probabilities. This allows us to transform the MDP semantics of a program $\mathcal{P}$ in a MC with the transition probabilities parameterized by the probability of choosing the actions in each state.

Consider the MDP in Fig 5a; in state S_0 either $\mathtt{h} = \mathtt{0}$ or $\mathtt{h} \neq \mathtt{0}$ and the system moves to the next state with the appropriate transition probability. Let $\mathbf{P}(0)$ and $\mathbf{P}(\neg 0)$ be $\mathbf{P}(\mathtt{h} = \mathtt{0}|S_0)$ and $\mathbf{P}(\mathtt{h} \neq \mathtt{0}|S_0)$ respectively; then we can transform the MDP in the MC in Fig 5b, with the constraint $\mathbf{P}(0) + \mathbf{P}(\neg 0) = 1$.

We hide the states in $\mathcal{T}_{\mathbb{A}}$ in the MC obtaining the observational reduction $\mathbb{C}$, as described in Sect. 5.

Step 2: Define a reward function for leakage. We want to define a reward function on the parameterized MC such that the expected total reward of the chain is equivalent to the leakage of the system. This step can be skipped if the leakage equation can be obtained directly from the model, like in the examples in the next Section. In the example in Fig. 5 the system is deterministic, so its leakage is equal to its entropy by Corollary 1, and we just need to define the entropy reward function on transitions $R(s,t) = -\log_2 P_{s,t}$, as explained in [14].

For a probabilistic system we need to build another MC by composing $\mathbb{C}/\mathcal{R}_h$, $\mathbb{C}/\mathcal{R}_{\mathbb{A}}$ and $\mathbb{C}/\mathcal{R}_{\mathbb{A}} \cap \mathcal{R}_h$, and we define the leakage reward function on the composed chain:

Theorem 3. *Let $\mathcal{C}$ be the parallel composition of $\mathbb{C}/\mathcal{R}_h$, $\mathbb{C}/\mathcal{R}_{\mathbb{A}}$ and $\mathbb{C}/\mathcal{R}_{\mathbb{A}} \cap \mathcal{R}_h$. Let R be a reward function on the transitions of $\mathcal{C}$ such that*

$$R(s_1 \times s_2 \times s_3, t_1 \times t_2 \times t_3) = \log_2 \frac{P_{s_1,t_1} P_{s_2,t_2}}{P_{s_3,t_3}}.$$

Then the expected total infinite time reward of $\mathcal{C}$ with the reward function R is equivalent to $H(\mathbb{C}/\mathcal{R}_h) + H(\mathbb{C}/\mathcal{R}_{\mathbb{A}}) - H(\mathbb{C}/\mathcal{R}_{\mathbb{A}} \cap \mathcal{R}_h)$ and thus to the leakage.

Step 3: Extract the leakage as an equation. Now that we have a reward function R on the transitions of a MC characterizing the leakage of the system, we need to maximize it. One possible strategy is to extract the explicit equation of the reward of the chain as a function of the transition probabilities, which themselves are a function of the prior information $\mathcal{I}$. For a reward function $R(s,t)$ on transitions the reward for the MC is

$$R(\mathcal{C}) = \sum_{s \in S} R(s)\xi_s = \sum_{s \in S} \left(\sum_{t \in S} P_{s,t} R(s,t) \cdot \sum_{k=0}^{\infty} P_{s_0,s} \right)$$

Since for the leakage reward function $R(s,t)$ is a function of $P_{s,t}$, the transition probabilities are the only variables in the equation.

In the example in Fig. 5 the leakage is equal to the entropy, so the reward function is $R(s,t) = -\log_2 P_{s,t}$ and the leakage equation is

$$\begin{aligned} \mathcal{R}(\mathcal{C}) = &- (\mathbf{P}(0)/4 + \mathbf{P}(\neg 0)/2) \log \left((\mathbf{P}(0)/4 + \mathbf{P}(\neg 0)/2) \right) - \\ &- (3\mathbf{P}(0)/4 + \mathbf{P}(\neg 0)/2) \log \left((3\mathbf{P}(0)/4 + \mathbf{P}(\neg 0)/2) \right) \end{aligned} \quad (3)$$

under the constraint above.

Step 4: Maximize the leakage equation Maximizing the extracted constrained leakage equation computes the channel capacity of the system. This can be done with any maximization method. Note that in general the strategy maximizing this reward function will be probabilistic, and thus will have to be approximated

numerically. In the cases in which the maximum leakage strategy is deterministic, an analytical solution can be defined via Bellman equations. This case is more complex that standard reward maximization for MDPs, since the strategy in every state must depend on the same prior information $\mathcal{I}$, and this is a global constraint that cannot be defined in a MDP. A theoretical framework to automate this operation is being studied, but most cases are simple enough to not need it, like the examples in the next Section.

8 Onion Routing

8.1 Case: Channel Capacity of Onion Routing

Onion Routing [9] is an anonymity protocol designed to protect the identity of the sender of a message in a public network. Each node of the network is a router and is connected to some of the others, in a directed network connection topology; the topology we consider is the depicted in Fig. 6. When one of the nodes in the topology wants to send a message to the receiver node R, it initializes a path through the network to route the message instead of sending it directly to the destination. The node chooses randomly one of the possible paths from itself to R, respecting the following conditions:

1. No node can appear in the path twice.
2. The sender node cannot send the message directly to the receiver.
3. All paths have the same probability of being chosen.

If some nodes are under the control of an attacker, he may try to gain information about the identity of the sender. In this example node 3 is a compromised node; the attacker can observe the packets transitioning through it, meaning that when a message passes through node 3 the attacker learns the previous and next node in the path. The goal of the attacker is to learn the identity of the sender of the message; since there are 4 possible senders, this is a 2-bit secret.

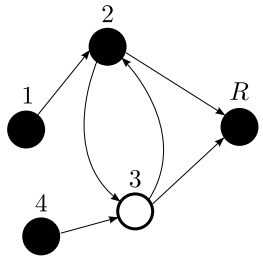

Fig. 6. Network topology for Onion Routing

$\mathtt{h}$	Path	$\mathtt{o}$	$\mathrm{P}(\mathtt{O}\|\mathtt{h})$
$1(h_1)$	$1 \to 2 \to R$	NN	$\frac{1}{2}$
	$1 \to 2 \to 3 \to R$	$2R$	$\frac{1}{2}$
$2(h_2)$	$2 \to 3 \to R$	$2R$	1
$3(h_3)$	$3 \to 2 \to R$	$N2$	1
$4(h_4)$	$4 \to 3 \to R$	$4R$	$\frac{1}{2}$
	$4 \to 3 \to 2 \to R$	42	$\frac{1}{2}$

Fig. 7. Onion Routing paths, observations and probabilities

Figure 7 summarizes the possible secrets of the protocol, the corresponding paths, the observation for each path assuming node 3 is compromised and the probability that a given sender will choose the path.

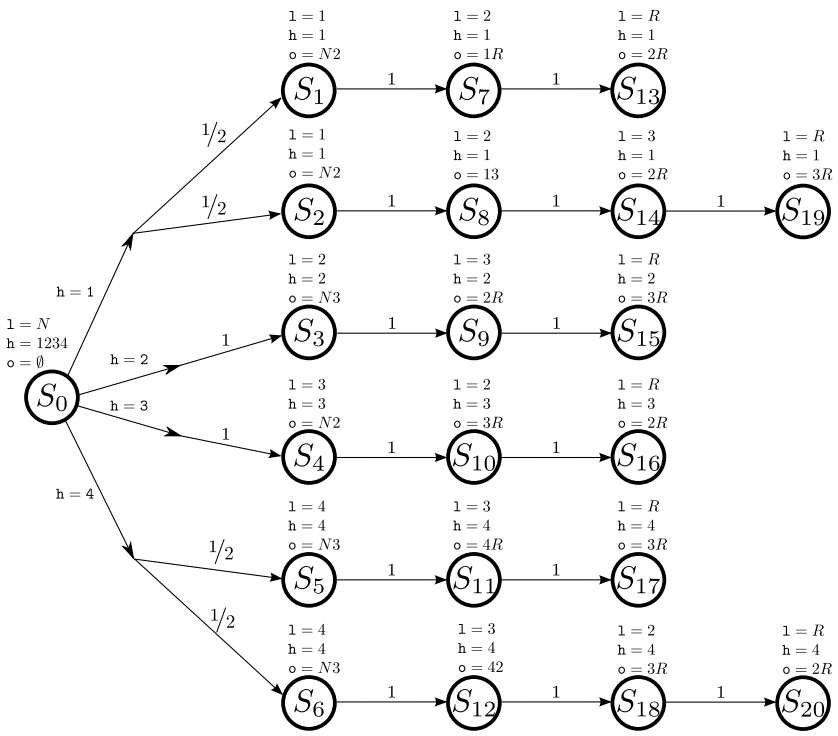

Fig. 8. Markov Decision Process for Onion Routing

We give directly the MDP semantics of the system in Fig. 8; its WHILE code is not shown for simplicity. The prior information $\mathcal{I}$ of the attacker consists of the prior probabilities he assigns to the identity of the sender; we use h_i to denote $\mathbf{P}(\mathtt{h} = i)$, for $i = 1...4$. Clearly $h_1 + h_2 + h_3 + h_4 = 1$. The full system is represented in Fig. 8, parameterized on the h_i parameters. Each state is labelled with the low-level variables $\mathtt{l}$ and $\mathtt{o}$ and the confidential variable $\mathtt{h}$. Variable $\mathtt{l}$ represents the name of the node being visited in the Onion Routing topology, $\mathtt{o}$ represents the observables in that node (the nodes before and after it in the path), and $\mathtt{h}$ the name of the sender of the message.

Since the attacker can observe only node 3, all states with $\mathtt{l} \neq 3$ except the initial state are unobservable τ-states. We reduce the chain accordingly; the resulting observational reduction is shown in Fig. 9a. We call it $\mathbb{C}$. Note that one of the paths does not pass through node 3, so if that path is chosen the attacker will never observe anything; in that case the system diverges. We assume that the attacker can recognize this case, using a timeout or similar means.

To compute the leakage we need also to define $\mathcal{R}_h$ and $\mathcal{R}_\mathcal{A}$. This is straightforward; $\mathcal{R}_h$ is $((s,t) \in (S \times S) | \mathtt{h}_s = \mathtt{h}_t)$ and $\mathcal{R}_\mathcal{A}$ is $((s,t) \in (S \times S) | \mathtt{o}_s = \mathtt{o}_t)$. The resulting MCs $\mathbb{C}/\mathcal{R}_h$ and $\mathbb{C}/\mathcal{R}_\mathcal{A}$ are shown in Fig. 9bc. Note that $\mathbb{C}/\mathcal{R}_h \cap \mathcal{R}_\mathcal{A} = \mathbb{C}$.

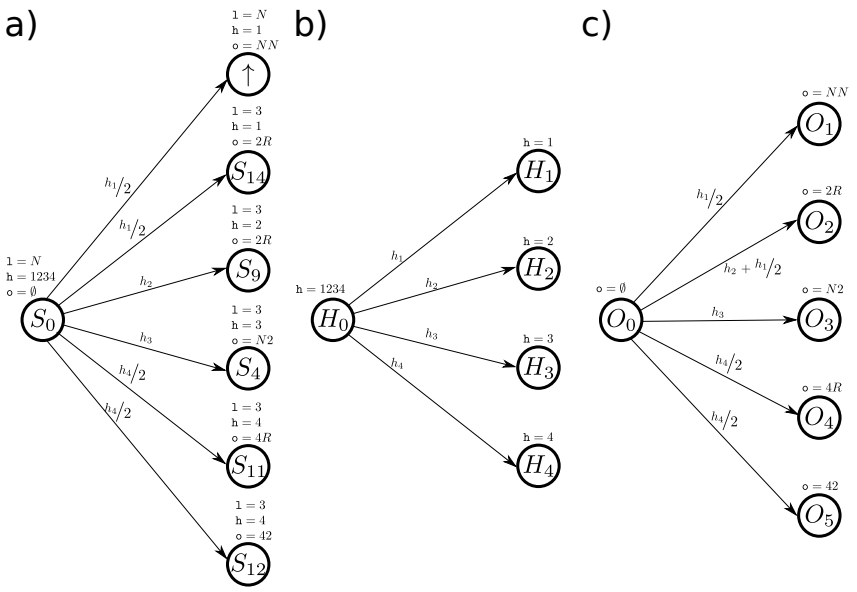

Fig. 9. Markov chains for Onion Routing: a) Observable reduction $\mathbb{C}$ b) $\mathbb{C}/\mathcal{R}_h$ c) $\mathbb{C}/\mathcal{R}_{\mathcal{A}}$

Since the system is very simple, we can extract the leakage equation directly from Def. 4. The leakage parameterized on $\mathcal{I}$ is

$$\begin{aligned} &H(\mathbb{C}/\mathcal{R}_h) + H(\mathbb{C}/\mathcal{R}_{\mathcal{A}}) - H(\mathbb{C}/\mathcal{R}_{\mathcal{A}} \cap \mathcal{R}_h) = \\ &= H(h_1, h_2, h_3, h_4) + H(\frac{h_1}{2}, \frac{h_1}{2} + h_2, h_3, \frac{h_4}{2}, \frac{h_4}{2}) - \\ &\quad H(\frac{h_1}{2}, \frac{h_1}{2}, h_2, h_3, \frac{h_4}{2}, \frac{h_4}{2}) \end{aligned} \tag{4}$$

Under constraints $0 \leq h_i \leq 1$ and $h_1 + h_2 + h_3 + h_4 = 1$ it has its maximum of 1.819 bits at $h_1 = 0.2488$, $h_2 = 0.1244$, $h_3 = 0.2834$, $h_4 = 0.2834$, thus these are the channel capacity and the attacker with highest leakage.

8.2 Case: Channel Capacity of Discrete Time Onion Routing

Due to our intensional view of the system, we can naturally extend our analysis to integrate timing leaks. Time-based attacks on the Tor implementation of the Onion Routing network have been proven to be effective, particularly in low-latency networks [10,11]. We show how to quantify leaks for an attacker capable to make some timing observations about the network traffic.

In this example there are two compromised nodes, A and B, and the attacker is able to count how many time units pass between the message being forwarded by A and the message arriving in B. The topology of the network is shown in Fig. 10 and the relative paths, observations and probabilities in Fig. 11. We will ignore messages departing from the compromised nodes A and B for simplicity.

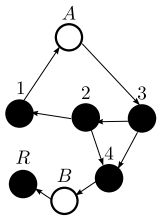

Fig. 10. Network topology for Timed Onion Routing

h	Path	o	P(o|h)
$1(h_1)$	$1 \to A \to 3 \to 4 \to B \to R$	$13, 4R$	$\frac{1}{2}$
	$1 \to A \to 3 \to 2 \to 4 \to B \to R$	$13, 4R$	$\frac{1}{2}$
$2(h_2)$	$2 \to 4 \to B \to R$	$NN, 4R$	$\frac{1}{2}$
	$2 \to 1 \to A \to 3 \to 4 \to B \to R$	$13, 4R$	$\frac{1}{2}$
$3(h_3)$	$3 \to 4 \to B \to R$	$NN, 4R$	$\frac{1}{2}$
	$3 \to 2 \to 4 \to B \to R$	$NN, 4R$	$\frac{1}{2}$
$4(h_4)$	$4 \to B \to R$	$NN, 4R$	1

Fig. 11. Timed Onion Routing paths, observations and probabilities

We add to the system a low-level variable $\mathtt{t}$ that represents the passage of the time between the message passing by A and passing by B. Variable $\mathtt{t}$ is initialized to 0 when the message passes by A and increased by 1 at each subsequent step. We will analyze the difference of leakage between the attacker $\mathcal{A}_\mathcal{T}$ that can discriminate states with different values of $\mathtt{t}$ and the attacker $\mathcal{A}_\mathcal{N}$ that does not have this power.

Both attackers are able to observe nodes A and B, so they have the same hidden states. Their observable reduction $\mathbb{C}$ of the system is the same, depicted in Fig. 12a. The secret's discrimination relation is also the same: $\mathcal{R}_h$ is $((s,t) \in (S \times S) | \mathtt{h}_s = \mathtt{h}_t)$, and the resulting quotient $\mathbb{C}/\mathcal{R}_h$ is depicted in Fig. 12b.

The two attackers have two different discrimination relations. For the attacker $\mathcal{A}_\mathcal{N}$, who is not able to keep count of the discrete passage of time, the relation is $\mathcal{R}_{\mathcal{A}_\mathcal{N}} = ((s,t) \in (S \times S) | \mathtt{o}_s = \mathtt{o}_t)$, while for the time-aware attacker $\mathcal{A}_\mathcal{T}$ it is $\mathcal{R}_{\mathcal{A}_\mathcal{T}} = ((s,t) \in (S \times S) | \mathtt{o}_s = \mathtt{o}_t \wedge \mathtt{t}_s = \mathtt{t}_t)$. The resulting MCs $\mathbb{C}/\mathcal{R}_{\mathcal{A}_\mathcal{N}}$ and $\mathbb{C}/\mathcal{R}_{\mathcal{A}_\mathcal{T}}$ are shown in Fig. 13.

Note that since the time-aware attacker has strictly more discriminating power, since $\mathcal{R}_{\mathcal{A}_\mathcal{T}} \subseteq \mathcal{R}_{\mathcal{A}_\mathcal{N}}$, we expect that he will leak more information. We show now how to validate this intuition by computing the difference of the leakage between $\mathcal{A}_\mathcal{T}$ and $\mathcal{A}_\mathcal{N}$. The difference of the leakage between the two attackers is

$$
\begin{aligned}
&I(\mathbb{C}/\mathcal{R}_h; \mathbb{C}/\mathcal{R}_{\mathcal{A}_\mathcal{T}}) - I(\mathbb{C}/\mathcal{R}_h; \mathbb{C}/\mathcal{R}_{\mathcal{A}_\mathcal{N}}) = \\
&H(\mathbb{C}/\mathcal{R}_h) + H(\mathbb{C}/\mathcal{R}_{\mathcal{A}_\mathcal{T}}) - H(\mathbb{C}/\mathcal{R}_{\mathcal{A}_\mathcal{T}} \cap \mathcal{R}_h) - H(\mathbb{C}/\mathcal{R}_h) - \\
&\qquad - H(\mathbb{C}/\mathcal{R}_{\mathcal{A}_\mathcal{N}}) + H(\mathbb{C}/\mathcal{R}_{\mathcal{A}_\mathcal{N}} \cap \mathcal{R}_h) = \\
&H(\mathbb{C}/\mathcal{R}_{\mathcal{A}_\mathcal{T}}) - H(\mathbb{C}/\mathcal{R}_{\mathcal{A}_\mathcal{N}}) = \\
&H\left(h_1 + \frac{h_2}{2}, \frac{h_2}{2} + h_3 + h_4\right) + \left(h_1 + \frac{h_2}{2}\right) H\left(\frac{1}{3}, \frac{2}{3}\right) - \\
&\qquad - H\left(h_1 + \frac{h_2}{2}, \frac{h_2}{2} + h_3 + h_4\right) = \\
&\left(h_1 + \frac{h_2}{2}\right) H\left(\frac{1}{3}, \frac{2}{3}\right) \approx \\
&0.91829 \left(h_1 + \frac{h_2}{2}\right)
\end{aligned}
\tag{5}
$$

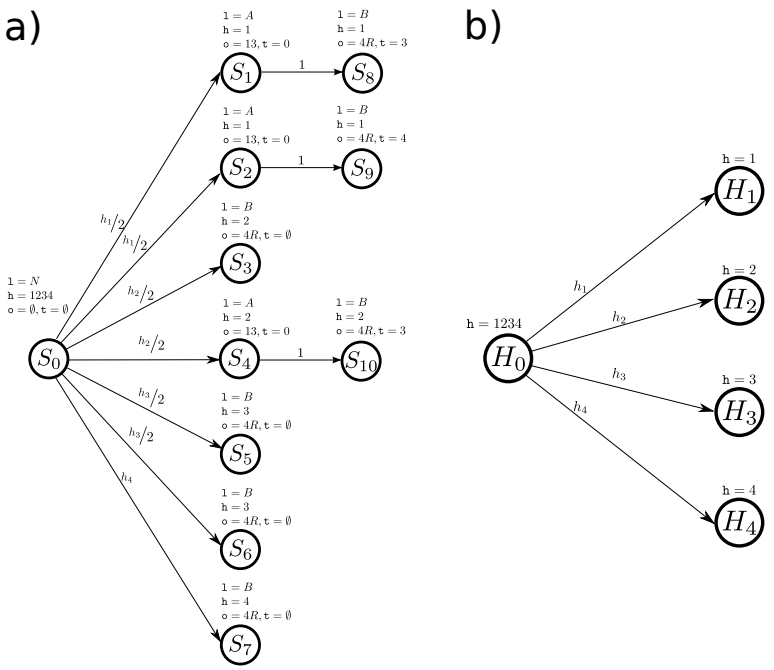

Fig. 12. Markov chains for Timed Onion Routing: a) Observable reduction $\mathbb{C}$ b) $\mathbb{C}/\mathcal{R}_h$

showing that the time-aware attacker $\mathcal{A}_{\mathcal{T}}$ leaks $\approx 0.91829\left(h_1 + \frac{h_2}{2}\right)$ bits of information more than the time-unaware attacker $\mathcal{A}_{\mathcal{N}}$.

9 Related work

Alvim, Andrés and Palamidessi [19] study leakage and channel capacity of interactive systems where secrets and observables can alternate during the computation.

Chen and Malacaria study leakage and channel capacity of traces and subtraces of programs [18], and, in [20], consider transition systems with particular attention to multi-threaded programs. They use Bellman equations to determine the minimal and maximal leakage. None of these works however deal explicitly with Markov Chains and randomized systems.

Intensional aspects of systems like timing leaks have been investigated by Köpf et al. in [7,6] and more recent work by Köpf, Mauborgne and Ochoa has investigated caching leaks [21].

Channel capacity for the Onion Routing protocol has been first characterized by Chen and Malacaria using Lagrange multipliers [5].

Recently Alvim et al. [22] have proposed a generalization of min-leakage by encapsulating it in problem-dependent gain functions. They suggest a generalization of LOI which would be interesting to compare with our work. On the other hand the use of alternative measure of leakage like g-leakage is a relatively orthogonal

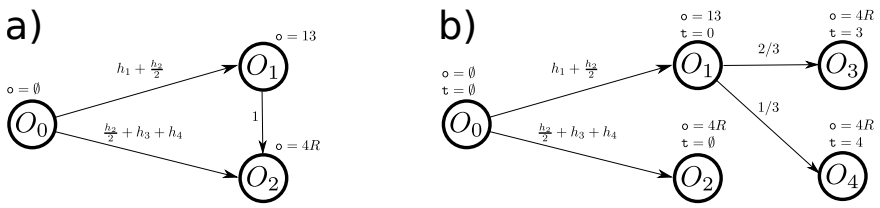

Fig. 13. Markov chains for Timed Onion Routing: a) $\mathbb{C}/\mathcal{R}_{\mathcal{A}_\mathcal{N}}$ b) $\mathbb{C}/\mathcal{R}_{\mathcal{A}_\mathcal{T}}$

idea and could be applied to our approach as well, substituting min-leakage with Shannon leakage.

The Lattice of Information approach to security seems to be related to the Abstract Interpretation approach to code obfuscation investigated by Giacobazzi et al. [23]; it would be interesting to further understand the connection between these approaches.

10 Conclusion

We presented a method to quantify the information leakage of a probabilistic system to an attacker. The method considers the probabilistic partial information semantics of the system and allows to encode attackers that can partially observe the internal behavior of the system. The method presented can be fully automated, and an implementation is being developed. The paper extends the consolidated LoI approach for leakage computation to programs with randomized behavior.

We extended the method to compute the channel capacity of a program, thus giving a security guarantee that does not depend on a given attacker, but considers the worst case scenario. We show how this can be obtained by maximizing an equation parameterized on the prior information of the attacker. The automatization of this computation raises interesting theoretical problems, as it requires to encode the property that all probability distributions on state must be derived from the same prior information, and thus involves a global constraint. We intend to work further on identifying suitable optimizations for constraints arising in this problem.

Finally, we analyzed the channel capacity of the Onion Routing protocol, encoding the classical attacker able to observe the traffic in a node and also a new attacker with time-tracking capacilities, and we proved that the time-tracking attacker is able to infer more information about the secret of the system.

References

1. Malacaria, P.: Algebraic foundations for information theoretical, probabilistic and guessability measures of information flow. CoRR **abs/1101.3453** (2011)
2. Clark, D., Hunt, S., Malacaria, P.: A static analysis for quantifying information flow in a simple imperative language. Journal of Computer Security **15** (2007) 321–371

3. Heusser, J., Malacaria, P.: Quantifying information leaks in software. In Gates, C., Franz, M., McDermott, J.P., eds.: ACSAC, ACM (2010) 261–269
4. Chatzikokolakis, K., Palamidessi, C., Panangaden, P.: Anonymity protocols as noisy channels. Inf. Comput. **206** (2008) 378–401
5. Chen, H., Malacaria, P.: Quantifying maximal loss of anonymity in protocols. In Li, W., Susilo, W., Tupakula, U.K., Safavi-Naini, R., Varadharajan, V., eds.: ASIACCS, ACM (2009) 206–217
6. Köpf, B., Smith, G.: Vulnerability bounds and leakage resilience of blinded cryptography under timing attacks. In: CSF, IEEE Computer Society (2010) 44–56
7. Köpf, B., Basin, D.A.: An information-theoretic model for adaptive side-channel attacks. In Ning, P., di Vimercati, S.D.C., Syverson, P.F., eds.: ACM Conference on Computer and Communications Security, ACM (2007) 286–296
8. Millen, J.K.: Covert channel capacity. In: IEEE Symposium on Security and Privacy. (1987) 60–66
9. Goldschlag, D.M., Reed, M.G., Syverson, P.F.: Onion routing. Commun. ACM **42** (1999) 39–41
10. Murdoch, S.J., Danezis, G.: Low-cost traffic analysis of tor. In: Proceedings of the 2005 IEEE Symposium on Security and Privacy. SP '05, Washington, DC, USA, IEEE Computer Society (2005) 183–195
11. Abbott, T.G., Lai, K.J., Lieberman, M.R., Price, E.C.: Browser-based attacks on tor. In Borisov, N., Golle, P., eds.: Privacy Enhancing Technologies. Volume 4776 of Lecture Notes in Computer Science., Springer (2007) 184–199
12. Cover, T., Thomas, J.: Elements of information theory. Wiley, New York (1991)
13. Shannon, C.E.: A mathematical theory of communication. The Bell system technical journal **27** (1948) 379–423
14. Biondi, F., Legay, A., Nielsen, B.F., Wąsowski, A.: Maximizing entropy over markov processes. Under review; available at www.itu.dk/people/fbio/maxent.pdf (2012)
15. Landauer, J., Redmond, T.: A lattice of information. In: CSFW. (1993) 65–70
16. Winskel, G.: The formal semantics of programming languages - an introduction. Foundation of computing series. MIT Press (1993)
17. Malacaria, P.: Risk assessment of security threats for looping constructs. Journal of Computer Security **18** (2010) 191–228
18. Malacaria, P., Chen, H.: Lagrange multipliers and maximum information leakage in different observational models. In ÃŽlfar Erlingsson, Pistoia, M., eds.: PLAS, ACM (2008) 135–146
19. Alvim, M.S., Andrés, M.E., Palamidessi, C.: Quantitative information flow in interactive systems. Journal of Computer Security **20** (2012) 3–50
20. Chen, H., Malacaria, P.: The optimum leakage principle for analyzing multi-threaded programs. In Kurosawa, K., ed.: ICITS. Volume 5973 of Lecture Notes in Computer Science., Springer (2009) 177–193
21. Köpf, B., Mauborgne, L., Ochoa, M.: Automatic quantification of cache side-channels. In Madhusudan, P., Seshia, S.A., eds.: CAV. Volume 7358 of Lecture Notes in Computer Science., Springer (2012) 564–580
22. Alvim, M.S., Chatzikokolakis, K., Palamidessi, C., Smith, G.: Measuring information leakage using generalized gain functions. In: CSF. (2012)
23. Preda, M.D., Giacobazzi, R.: Semantics-based code obfuscation by abstract interpretation. Journal of Computer Security **17** (2009) 855–908