Markovian Processes for Quantitative Information Leakage

Fabrizio Biondi

IT UNIVERSITY OF COPENHAGEN

Markovian Processes for Quantitative Information Leakage

A dissertation submitted to the IT University of Copenhagen
in partial fulfillment of the requirements for the degree
of Doctor of Philosophy in Computer Science by

Fabrizio Biondi

Process and System Models Group
Software and Systems Section

Copenhagen, January 2014
Revised April, 2014

Abstract

Quantification of information leakage is a successful approach for evaluating the security of a system. It models the system to be analyzed as a channel with the secret as the input and an output as observable by the attacker as the output, and applies information theory to quantify the amount of information transmitted through such channel, thus effectively quantifying how many bits of the secret can be inferred by the attacker by analyzing the system's output.

Channels are usually encoded as matrices of conditional probabilities, known as channel matrices. Such matrices grow exponentially in the size of the secret and observables, are cumbersome to compute and store, encode both the behavior of the system and assumptions about the attacker, and assume an input-output behavior of the system. For these reasons we propose to model the system-attacker scenario with Markovian models.

We show that such models are more compact and treatable than channel matrices. Also, they clearly separate the behavior of the system from the assumptions about the attacker, and can represent even non-terminating behavior in a finite model.

We provide techniques and algorithms to model and analyze both deterministic and randomized processes with Markovian models and to compute their information leakage for a very general model of attacker. We present the QUAIL tool that automates such analysis and is able to compute the information leakage of an imperative WHILE language. Finally, we show how to use QUAIL to analyze some interesting cases of secret-dependent protocols.

Acknowlegdments

The research in this thesis would not have been accomplished without the three years of constant support from my excellent supervisor Andrzej Wąsowski. Also, without his unerring eye it would be full of my mistakes and distractions, and without his patience and teaching I would still be the amateur researcher that he took under his wing on February 2011. I count working with him among the most fortunate circumstances of my career.

I am also deeply grateful to Axel Legay for being my *de facto* co-supervisor, reading my drafts and sharing with me his encyclopedic knowledge of current and past research. I also thank him and his research group in Rennes for the hospitality and warmth during the months in which I have been visiting them.

I want to thank Bo Friis Nielsen for helping me handle probabilistic processes and for the fruitful discussions on the axiomatic bases of probability. I am also grateful to Pasquale Malacaria for teaching me information theory and leakage theory during the months in which I have been visiting him in London.

I am grateful to the colleagues in the Software and Systems and Theoretical Computer Science sections of the IT University of Copenhagen for the mutual support and guidance while we were all traveling this rugged but exhilarating path, and particularly Paolo Tell for the coffee that is allowing me to write these lines right now. I want to hug all my friends in Copenhagen, all of which I will dearly miss, for the lunches, beers, dinners, movies and for the great times we shared.

Finally, I want to thank my family and friends for their huge support and patience in these three years, and particularly during the months it took me to write this thesis. I promise that I will be more available and less cantankerous from now on, or at least I will find an equally valid excuse to continue being insufferable.

Contents

Chapter 1

Introduction

> “ *What man art thou that, thus bescreened in night,*
> *So stumblest on my counsel?*
>
> What kind of man are you, that hides in the darkness,
> and listens to my private thoughts? ”
>
> William Shakespeare, *Romeo and Juliet*

Data security is one of the main concerns of computer science. Cases of vulnerabilities in protocols being exploited by individuals and governments to access private information of individuals have been multiplying in the last years, even leading to international scandals and friction between allied countries. It is clearly the responsibility of computer scientists and software developers to provide systems that protect the privacy of user data as strongly as mathematically possible.

One limitation to the privacy of data is that many services need private data to function properly. For instance, a delivery company needs to know a person's address and when the person is at home to deliver the person a parcel, but the same information would be very dangerous in the hands of a band of robbers. The highest standard we can require is that a service provider knows only the information that it strictly needs to complete a task. To bring an example closer to computer science, an authentication system must be able to verify if a string entered by the user corresponds to a stored password, but it does not necessarily have to know the password itself.

Guaranteeing that a system does not possess, use or reveal more secret data than it needs to, requires formal approaches and models of the secret-system-attacker scenarios. Some of these approaches are based on some logical property that must not be violated to guarantee security. Since the property can either be guaranteed or violated, these are known as *qualitative* approaches. Consider an encrypted transmission from a sender to a receiver via a messenger. The property we can formalize is that the messenger is not able to read the message, or more properly that the information that he has about the message before and after it passes through his hands is the same.

Alas, this kind of approach is very hard to apply in most real cases, since *some* very small amount of information will always be learned by the messenger; for instance if the message is short he knows that it cannot transmit a large amount of information, or

if the encryption is a word-substitution scheme then he can infer the length of the single words in the message. Qualitative approaches tend to either guarantee security only for a very weak attacker model, or reject as unsafe any protocol that leaks any amount of information, no matter how small. To go back to the authentication example, an attacker that tries a random password (e.g. "12345") and gets rejected by the system learns *some* positive amount of information about the actual password stored in the system (e.g. that the password is not "12345"). Finally, when a qualitative property is violated, there is no measure of *how bad* the violation is: the cases in which the attacker learns the whole password and the case in which he learns some very vague information about the password are indistinguishable.

These considerations gave rise to the *quantitative* approach to security. In a quantitative approach we do not ask *if* a security property is violated in a particular case, but *how much* it is violated. This also means that a secret is not seen as some atomic object that can be either known or unknown, but as a given number of secret bits, and we ask how many of these bits are inferred by the attacker through his observation of the system. This approach is particularly fitting for authentication and communication scenarios, where the secret is stored in a variable with a known bit length.

Coming back to the authentication example, we also expect that any measure of how much a secret is violated should be intuitively correlated with any similar measure e.g. how easier it gets for the attacker to guess the password, or how many attempts it will take him, or how much computational power he will require. In easier terms, if an attacker A is more dangerous than an attacker B according to one of these measures, then he must be more dangerous according to all of them.

Many measures of the effectiveness of an attacker, including Shannon leakage [38, 39, 47, 66], min-entropy leakage [84], g-leakage [6] and Bayes risk [24, 25], respect this principle. Attackers that are ordered according to one of the measures will be ordered according to all of them [61]. Furthermore, the ordering is a complete lattice, known as the Lattice of Information [57]. We will explain this concept more precisely in Chapter 3. In this thesis we will measure information leakage with Shannon leakage. Some different measures of information leakage are presented in Section 3.2.

To compute this value it is necessary to model somehow the secret-system-attacker scenario. The model should be as compact as possible and should allow for efficient computation of the leakage. The state of the art model used to represent a scenario is called a *channel matrix*. In a channel matrix the idea is that the system is seen as an input-output probabilistic channel with the secret as the input and the system's observable output as the output. Each row of the matrix corresponds to a possible value of the secret, each column to a possible output, and the value in the cell corresponding to the secret value s and output o is the probability that o is produced if the value of the secret is s. Then the leakage can be computed in time linear in the number of cells of the matrix.

One of the problems with channel matrices is that they necessarily have a number of rows exponential in the size of the secret and a number of columns exponential in the size of the output. For instance, for an authentication with a 64-bit secret password the matrix will have $\approx 1.8 \cdot 10^{19}$ rows. This is often an unnecessarily large representation of the system.

This representation also assumes that the the system has a finite number of possible outputs and secret values, which may not be true for non-terminating systems. Finally, the matrix models together properties of the system and of the attacker, while it would be preferable to keep them separated.

For these reasons we propose to model the secret-system-attacker model with Marko-

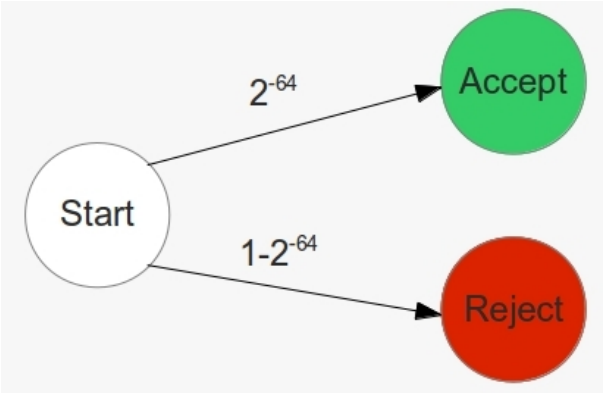


Figure 1.1: Markov chain model for a 64-bit authentication example

vian models, in particular Markov Decision Processes and Markov chains. In Chapters 4 and 5 we describe in details the modeling procedure and how to compute information leakage on the models. Since every state of a Markov chain can be equivalently modeled as a channel, this approach can also be seen as a generalization from a single channel to a Markovian sequence of channels. The advantages of this approach are described in the following statement:

Thesis Statement *Using Markovian models instead of channel matrices for modeling system-attacker scenarios for information leakage has the following advantages:*

- *The Markovian model can be directly obtained from the source code of the system*
- *The model is more compact and consequently the computation of the leakage is faster*
- *The system and attacker are modeled separately, allowing for model reuse*
- *It is possible to model and analyze systems with non-terminating behavior*

For instance, the 64-bit authentication example can be represented by the Markov chain with 3 states in Fig 1.1 instead of a channel matrix with 10^{19} rows.

We have implemented the complete analysis technique, from source code of a process to final result, in the QUAIL (QUantitative Analyzer for Imperative Languages) tool. QUAIL can be used to analyze different kinds of security protocols, including authentication, anonymity and voting protocols. We present the tool in Chapter 6 and some case studies in Chapter 7.

1.1 Thesis Contributions

The main contributions of this thesis are as follows:

Encoding a secret-dependent program with a Markovian model We provide a procedure to encode a process written in a simple imperative language with a Markov Decision Process. The encoding assumes no probability distribution on the values of the secret, thus being independent from the encoding of an attacker.

Encoding an attacker We show how to encode an attacker for a given secret-dependent program to account for prior knowledge of the attacker about the secret and discrimination capabilities of the attacker.

Encoding a process-attacker scenario We show how to merge a process model and an attacker model to obtain a Markov chain so that the chain represents the interaction of the process and the attacker. The amount of information that the attacker infers about the secret that the process depends on can be computed on such model.

Computing entropy of a Markov chain We provide a polynomial-time procedure to compute the entropy of a Markov chain, by reduction to computing the expected infinite-horizon reward for an appropriate reward function on transitions. Computing entropy of a Markov chain model is a necessary step to computing the leakage of the system.

Computing information leakage of a Markov chain Given a Markov chain where the states are appropriately annotated with allowed values of variables, we show how to compute the mutual information between two or more such variables. Mutual information between the observables and the secret corresponds to the information leakage of the system.

Determining whether a specification allows for unsafe implementations Given a specification encoded in an appropriate Markovian model, we provide algorithms to determine whether any implementation of the specification exists such that the implementation would reveal the whole secret that the process depends on.

Computing channel capacity of a specification We provide a different technique at a higher abstraction level to compute the maximum leakage over all implementations of a specification and any prior information of attackers.

Introducing the QUAIL anayzer We introduce the QUAIL tool for the analysis of information leakage of a simple imperative language, discuss which assumption and choices we chose to implement, and show how to analyze a number of different examples of deterministic and randomized secret-dependent protocols.

1.2 Main Contributions

1.2.1 Paper A

Bibliographical entry: Fabrizio Biondi, Axel Legay, Bo Friis Nielsen, and Andrzej Wasowski. Maximizing entropy over Markov processes. *Journal of Logic and Algebraic Programming*, under revision

In this paper we introduce the computation of entropy on Markov chain as a way to compute information leakage of deterministic processes. We present Markov chains as process models and both Interval Markov Chains and Markov Decision Processes as specification models, and we discuss how to find a process maximizing leakage among all respecting a given specification. We provide algorithms to recognize when a specification permits non-terminating or arbitrarily unsafe implementations.

1.2.2 Paper B

Bibliographical entry: Fabrizio Biondi, Axel Legay, Pasquale Malacaria, and Andrzej Wasowski. Quantifying information leakage of randomized protocols. In Roberto Giacobazzi, Josh Berdine, and Isabella Mastroeni, editors, *VMCAI*, volume 7737 of *Lecture Notes in Computer Science*, pages 68–87. Springer, 2013

In this paper we formalize the modeling of a system-attacker scenario with a Markov chain. We also extend the entropy computation on Markov chains to mutual information computation, allowing us to model and analyze randomized processes as well. We provide an algebraic way to find the channel capacity of a protocol by extracting a system of equations from the model and maximizing over it. We show how the technique can be extended to topological models like the Onion Routing protocol with an appropriate Markovian encoding of the observables, and we use the same technique to prove the effectiveness of time-aware attackers for the protocol.

1.2.3 Paper C

Bibliographical entry: Fabrizio Biondi, Axel Legay, Louis-Marie Traonouez, and Andrzej Wasowski. QUAIL: A quantitative security analyzer for imperative code. In Sharygina and Veith [83], pages 702–707

In this paper we introduce the QUAIL (QUantitative Analyzer for Imperative Languages) tool. QUAIL implements our modeling and leakage computation theories, and is able to compute the leakage for a process written in a simple Turing-complete imperative language. We explain how to use QUAIL, how it works and which assumptions it is based on. Then we analyze a number of different protocols and explain how to interpret the result.

1.3 Other Contributions

1.3.1 Paper D

Bibliographical entry: Fabrizio Biondi, Axel Legay, Bo Friis Nielsen, and Andrzej Wasowski. Maximizing entropy over Markov processes. In Adrian Horia Dediu, Carlos Martín-Vide, and Bianca Truthe, editors, *LATA*, volume 7810 of *Lecture Notes in Computer Science*, pages 128–140. Springer, 2013

This is the conference version of Paper A, presented at the 7th International Conference on Language and Automata Theory and Applications (2013).

1.3.2 Paper E

Bibliographical entry: Fabrizio Biondi, Axel Legay, Bo Friis Nielsen, and Andrzej Wasowski. Maximizing entropy over Markov processes. In *24nd Nordic Workshop on Programming Theory - NWPT*, 2012

This is the extended abstract version of Paper A, presented at the 24th Nordic Workshop on Programming Theory (2012).

1.4 Structure of the Thesis

This thesis is organized as follows.

Chapter 2 defines common concepts in probability theory and information theory that are used throughout the thesis. This chapter is mostly based on the fundamental texts by Bertsekas and Tsitsiklis [11] and Cover and Thomas [36].

Chapter 3 introduces the state of the art in the quantitative approach to security via information theory. We show the different approaches to the quantification of security, including Shannon leakage, min-entropy leakage and Bayes risk, and how they give rise to the same ordering on the effectiveness of the attackers. This order is a lattice, known as the Lattice of Information.

Chapter 4 introduces our method to model processes as Markovian models so that the information leakage of the process can be computed on the model. In the chapter we model terminating processes, covering the same cases that are commonly modeled as channel matrices (see Chapter 3). The assumption is that the process starts, terminates, and upon terminating produces some output that is readable to the attacker. We also show how to raise the abstraction level and compute the maximum leakage over all implementations of a protocol. This chapter is based on Papers A and B.

Chapter 5 extends the modeling and leakage computation to non-terminating processes, i.e. processes where the secret, the observation or both do not terminate. We provide a way to quantify leakage by separating the non-terminating processes by cases according to whether the secret, the observation or both do not terminate, and show how to compute meaningful measures of leakage in each case. This chapter is unpublished research.

Chapter 6 presents the QUantitative Analyzer for Imperative Languages (QUAIL) that we wrote to implement the basic theory presented in this work. We show how to encode and analyze different kinds of protocols with QUAIL, including anonymity, and authentication protocols. For each protocol we also show how to interpret the numerical result obtained through QUAIL. This chapter is based on Paper C.

Chapter 7 presents solved cases of secret-dependent algorithms analyzed with QUAIL. It is based on paper C and on the encoding of topological anonymity protocols in QUAIL developed with Mads Tinggaard Pedersen and Andrzej Wąsowski [75].

Chapter 8 contains the conclusions of the thesis and an outline of future work on the subject of Markovian modeling for the quantification of information leakage.

Finally, the Appendix contains papers A to C.

Chapter 2

Background

> *In omnibus negotiis prius quam aggrediare, adhibenda est præparatio diligens.*
>
> In all matters, before beginning, a diligent preparation should be made.
>
> Cicero, *De Officiis*, I.21

2.1 Discrete Probability Theory

A *sample space* Ω is the finite or countable set of all possible outcomes for an experiment we want to model. It is assumed that the experiment produces exactly one of the outcomes. An *event* is a subset $E \subseteq \Omega$ of the sample space.

Definition 2.1.1. *A* probability distribution π *on* Ω *is a function* $\pi : \Omega \to [0, 1]$ *such that*

$$\sum_{x \in \Omega} \pi(x) = 1$$

Definition 2.1.2. *The* probability *of an event is* $P(E) = \sum_{x \in E} \pi(x)$.

The following properties of probability are immediate:

Nonnegativity For any event E: $P(E) \geq 0$;

Additivity For any two events E, F: $P(E \cup F) = P(E) + P(F) - P(E \cap F)$;

Normalization $P(\emptyset) = 0$, $P(\Omega) = 1$.

The *conditional probability* of an event E given an event F is

$$P(E|F) = \frac{P(E \cap F)}{P(F)}$$

if $P(F) > 0$, and is undefined otherwise. It follows from the definition above that $P(E \cap F) = P(F)P(E|F)$. This can be expanded in the *chain rule* of probability:

$$P(E_1 \cap E_2 \cap E_3 \cap ... \cap E_n) = P(E_1)P(E_2|E_1)P(E_3|E_2 \cap E_1)...P(E_n|E_{n-1} \cap ... \cap E_1)$$

The conditional probabilities $P(E|F)$ and $P(F|E)$ are related by *Bayes' theorem*:

$$P(E|F) = \frac{P(F|E)P(E)}{P(F)}$$

Two events E and F are said to be *independent* iff $P(E \cap F) = P(E)P(F)$.Given a set of events $S = \{E_1, ..., E_n\}$ they are independent if $P(E_1 \cap E_2 \cap ... \cap E_n) = P(E_1)P(E_2)...P(E_n)$ and all proper subsets of S are independent.

A *random variable* X is a real-valued function of the outcome of an experiment. It is said to be *discrete* if its image is finite or countable. A *probability mass function* (PMF) $p_X(x)$ is the probability of the event $\{X = x\}$ consisting of all outcomes for which the value of X is x: $p_X(x) = P(\{X = x\})$. Note that $\sum_{x \in X} p_X(x) = 1$. We will just write $p(x)$ when the random variable is clear from the context. We will sometimes write explicitly the PMF of a random variable X as $\{x_1 \mapsto p(x_1), x_2 \mapsto p(x_2), ..., x_n \mapsto p(x_n)\}$.

A function of a random variable is also a random variable. The *expectation* or *expected value* $E[X]$ or a random variable X with PMF $p(x)$ is

$$E[X] = \sum_{x \in X} xp(x)$$

More generally, the *nth moment* of a random variable X is the expected value of X^n.

2.2 Probabilistic Processes

A *discrete time-indexed probabilistic process* is an infinite sequence of random variables $X_0, X_1, ...$ on the same sample space S, known as the *state space*. Each element $s \in S$ is said to be a *state* that the process can visit during its infinite life. The subscript index represents discrete time steps, so X_i is the random variable modeling the process at time step $i \in \mathbb{N}$.

2.2.1 Markov Chains

The PMF on each random variable X_i can in general depend on the PMF of all other X_j. A probabilistic process is said to be a *Markov chain* (MC) if the PMF of the variable X_i depends only on the PMF of the variable X_{i-1}, i.e.

$$\forall s_0, .., s_i \in S.\ P(X_i = s_i | X_{i-1} = s_{i-1}, ..., X_0 = s_0) = P(X_i = s_i | X_{i-1} = s_{i-1})$$

A Markov chain is said to be *time-invariant* if the probabilities of transitioning from one state to another do not depend on the time, i.e.

$$\forall i, j \in \mathbb{N}, s, t \in S.\ P(X_i = t | X_{i-1} = s) = P(X_j = t | X_{j-1} = s)$$

We will always assume that our probabilistic processes are time-invariant. In this case we can just write $P_{s,t}$ for $P(X_i = t | X_{i-1} = s)$, since it does not depend on the time step i. We store the *transition probabilities* $P_{s,t}$ in a *transition probability matrix* P of size $|S| \times |S|$. Similarly, we can draw a *transition probability graph* where the

nodes are the states of the chain and there is a directed edge labeled with $P_{s,t}$ from the node corresponding to state s to the node corresponding to state t iff $P_{s,t} > 0$. The probability of transitioning from a state s to a state t in exactly k steps can be found as the entry on row s column t of the matrix P raised to power k, as shown in the *Chapman-Kolmogorov equation*:

$$P(X_{i+k} = t | X_i = s) = P^k_{s,t}$$

Let $\pi^{(i)}$ be the PMF of the random variable X_i. We will assume for simplicity that $\pi^{(0)}$ assigns probability 1 to a given state s_0. Then applying the Chapman-Kolmogorov equation for a time-invariant MC it is possible to compute any $\pi^{(i)}$ as

$$\pi^{(i)} = \pi^{(0)} P^i$$

Thus a MC can be fully characterized by the triple (S, s_0, P). A *path* of length n in a MC is a sequence of states $s_1, ..., s_n$, and its probability is

$$P(s_1, ..., s_n) = \prod_{i=1}^{n-1} P_{s_i, s_{i+1}}$$

A state $s \in S$ of a Markov chain is said to be *deterministic* if there exists a state t such that $P_{s,t} = 1$, *stochastic* otherwise. A state t is *reachable* from a state s if there exists a time k such that $P^k_{s,t} > 0$. We will usually consider MCs in which all states are reachable from s_0. Two states $s, t \in S$ are said to *communicate*, written $s \leftrightarrow t$, if s is reachable from t and vice versa.

A subset of states $R \subseteq S$ is sometimes called a *component*. A component R is *strongly connected* (SCC), or a *communicating class*, if for each couple of states $s, t \in R$ it holds that $s \leftrightarrow t$. A SCC R is said to be *closed* or a *closed communicating class* (CCC) or an *end component* if no state in the component has transitions with nonzero probability to states not in the component, *open* otherwise:

Definition 2.2.1. *Given a MC $\mathcal{C} = (S, s_0, P)$, a component $R \subseteq S$ is a* closed communicating class *if $P(X_n \notin R | X_{n-1} \in R) = 0$.*

A MC can be divided in its transient states and closed communicating classes in polynomial time by Tarjan's algorithm.

For a state $s \in S$ we define the *expected residence time* ξ_s of s as

$$\xi_s = \sum_{n=0}^{\infty} P^n_{s_0,s}$$

A state s is *recurrent* iff $\xi_s = \infty$, *transient* otherwise. States in closed SCCs of a finite MC are all recurrent, while all other states are transient.

A state s is *absorbing* if $P_{s,s} = 1$. A Markov chain is said to be absorbing if its recurrent states are absorbing. In Section 4.2.1 we will show that absorbing chains have finite entropy.

We will enrich our Markovian models with a finite set $\mathtt{V}$ of natural-valued variables, and for simplicity we assume that there is a very large finite bit-size M such that a variable is at most M bit long. We define an assignment function $A : S \to [0, 2^M - 1]^{|\mathtt{V}|}$ 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 $s \in S$. This allows us to focus on the stochastic process representing the value of any variable $\mathtt{v}$ at any moment in time, by summing together the probabilities of being in states with the same values of $\mathtt{v}$. We will call this process the *marginal process*, or just *marginal*, $\mathcal{C}_{|\mathtt{v}}$ on $\mathtt{v}$:

Definition 2.2.2. *Let $\mathcal{C} = (S, s_0, P)$ be a Markov chain and $\mathtt{v} \in \mathtt{V}$ a variable. Then we define the* marginal process $\mathcal{C}_{|\mathtt{v}}$ of $\mathcal{C}$ on $\mathtt{v}$ *as a stochastic process $(\mathtt{v}_1, \mathtt{v}_2, ...)$ where*

$$\forall n.\ P(\mathtt{v}_k = n) = \sum_{\forall s.\ \mathtt{v}(s)=n} \pi_s^{(k)}$$

We will just call the marginal process $\mathtt{v}$ when it is clear from the context that we refer to it. Note that $\mathcal{C}_{|\mathtt{v}}$ is not necessarily a Markov chain. In the paper we will allow assignments of sets of values to variables and marginals on sets of variables; such extensions are straightforward.

Assume that the system modeled by $\mathcal{C}$ has a single secret variable $\mathtt{h}$ and a single observable variable $\mathtt{o}$. Then the distributions over the marginal processes $\mathcal{C}_{|\mathtt{h}}$ and $\mathcal{C}_{|\mathtt{o}}$ model the behavior of the secret and observable variable respectively at each time step, and their correlation quantifies the amount of information about the secret that can be inferred by observing the observable variable.

Given a Markov chain $\mathcal{C} = (S, s_0, P)$ let a *discrimination relation* $\mathcal{R}$ be an equivalence relation over S. A discrimination relation is used to represent the fact that an observer may not be able to distinguish the single states of the chain.

A *quotient* of a Markov chain by a discrimination relation represents the expected behavior that the chain according to an attacker that is able to discriminate only the states in separate classes of the discrimination relation.

Definition 2.2.3. *Given a Markov chain $\mathcal{C} = (S, s_0, P)$ and a discrimination relation $\mathcal{R}$ 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*

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

We will usually compute quotients by defining a discrimination relation that discriminates states with different values of variables we are interested in, e.g. secret and observable variables. Nonetheless it must be noted that in general a marginal process and a quotient representing the same varaible do not coincide, since the quotient is always a Markov chain while the marginal process may not be. We will mostly use quotients in Chapter 4 and marginals in Chapter 5. The reason for this distinction is that in Chapter 4 we will reduce the modeling of a terminating program to a single probability distribution over the possible observations and secrets, and in this case the quotient approach always gives the correct result and is simpler to compute than marginals. On the other hand, in Chapter 5 we will consider Markov chains with internal observable state to characterize a concept of discrete time, so the marginals and the quotients may not coincide and we have to use the former.

2.2.2 Interval Markov Chains

Definition 2.2.4. *A closed-interval* Interval Markov Chain *(IMC) is a tuple* $\mathcal{I} = (S, s_0, \check{P}, \hat{P})$ *where* S *is a finite set of states containing the initial state* s_0*,* $\check{P}$ *is an* $|S| \times |S|$ *bottom transition probability matrix,* $\hat{P}$ *is a* $|S| \times |S|$ *top transition probability matrix, such that for each pair of states* $s, t \in S$ *we have* $0 \leq \check{P}_{s,t} \leq \hat{P}_{s,t} \leq 1$.

An IMC is used to model a specification, i.e. an infinite set of Markov chains with a similar structure. We say that a Markov chain *implements* an IMC if it is contained in this set.

Definition 2.2.5. *A Markov chain* $\mathcal{C} = (S, s_0, P)$ implements *an Interval Markov Chain* $\mathcal{I} = (S, s_0, \check{P}, \hat{P})$ *in the Uncertain Markov Chain (UMC) semantics [22], written* $\mathcal{C} \models \mathcal{I}$*, if* $\forall s, t \in S.\ \check{P}_{s,t} \leq P_{s,t} \leq \hat{P}_{s,t}$.

We assume without loss of generality that our Interval MCs are *coherent*, meaning that every value for each transition interval is attained by some implementation. Coherence can be established by checking that both following conditions hold [55]:

1. $\forall s, t \in S. \check{P}_{s,t} \geq (1 - \sum_{u \neq t} \hat{P}_{s,u})$
2. $\forall s, t \in S. \hat{P}_{s,t} \leq (1 - \sum_{u \neq t} \check{P}_{s,u})$

Any Interval MC $\mathcal{I} = (S, s_0, \check{P}, \hat{P})$ can be transformed into a coherent Interval MC $\mathcal{I}' = (S, s_0, \check{P}', \hat{P}')$ by changing the top and bottom transition probability matrices to the following:

1. $\check{P}'_{s,t} = \max(\check{P}_{s,t}, 1 - \sum_{u \neq t} \hat{P}_{s,u})$
2. $\hat{P}'_{s,t} = \min(\hat{P}_{s,t}, 1 - \sum_{u \neq t} \check{P}_{s,u})$

The resulting coherent Interval MC $\mathcal{I}'$ is unique and has the same implementations as the original incoherent Interval MC $\mathcal{I}$ [55], so in particular it has an implementation iff $\mathcal{I}$ has at least one implementation.

A state s of an Interval MC is *deterministic* if $\exists t.\ \check{P}_{s,t} = 1$, *stochastic* otherwise. We say that a state t is *reachable* from a state s if $\exists s_1, s_2, ..., s_n \in S. s_1 = s \wedge s_n = t \wedge \hat{P}_{s_i, s_{i+1}} > 0$ for $1 \leq i < n$. We say that a component $R \subseteq S$ is *strongly connected* if $\forall s, t \in R$. t is reachable from s.

Note that if there is an implementation in which a subset of states $R \subseteq S$ is strongly connected, then R must be strongly connected in the IMC.

2.2.3 Markov Decision Processes

Definition 2.2.6 ([78])**.** *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 write* $P_{(s,a),t} = x$ *for* $P(s, a, t) = x$.

An MDP makes a nondeterministic choice of an action in each state. A selection of an action determines a probability distribution over the successor states.

Definition 2.2.7. *A* probabilistic strategy *is a function $\sigma : S \to \delta(\Lambda_s)$ assigning to a state s a probability distribution over the actions in Λ_s. A probabilistic strategy is* positional *if its probability distributions assign probability 1 to some action.*

The (infinite) set of strategies of the process $\mathcal{P}$ is denoted $\Sigma_{\mathcal{P}}$.

A strategy σ resolves an MDP $\mathcal{P} = (S, s_0, P, \Lambda)$ into an MC $\mathcal{C} = (S, s_0, P')$ where $P'_{s,t} = \sum_{a \in \Lambda_s} \sigma(s, a) P_{(s,a),t}$. In this sense MDPs, similarly to IMCs, can also be used as specification models for MCs.

2.2.4 Reward Functions

Definition 2.2.8. *A* non-negative reward function *over the transitions of a Markov chain is a function $R : S \times S \to \mathbb{R}^+$ assigning a non-negative real value, called reward, to each transition.*

Given a reward function R we can compute the value of the reward for a concrete execution of a chain by summing reward values for the transitions exercised in the execution. More interestingly, we can compute the expected reward of each state $s \in S$ as $R(s) = \sum_{t \in S} P_{s,t} R_{s,t}$, and then the expected reward over the infinite behavior of the chain $\mathcal{C}$ is $R(\mathcal{C}) = \sum_{s \in S} R(s) \xi_s$ [78, Chpt. 5]. The expected reward of a Markov chain is finite iff the expected reward of each recurrent state is zero.

An IMC represents an infinite set of Markov chains, and thus does not have in general a single expected reward, since each implementation can have a different expected reward. Similarly, in an MDP strategies that resolve the MDP in different Markov chains will in general have different rewards. Thus for specification models like IMCs and MDPs it is interesting to determine which implementation or strategy maximizes or minimizes a given reward function of interest.

2.3 Information Theory

Definition 2.3.1. *The* entropy *$H(X)$ of a random variable X with PMS $p(x)$ is defined as*

$$H(X) = -\sum_{x \in X} p(x) \log(p(x))$$

where all logarithms are taken in base 2, thus measuring entropy in bits.

Entropy is a measure of the uncertainty in the state of information represented by the PMS $p(x)$. Generally, an entropy of n bits means that n bits of information are still required to be able to know exactly the value of X.

It can be shown that entropy has a minimum of 0 if the PMF of X has a value $\bar{x}$ with $p(\bar{x}) = 1$ and a maximum of $\log(|X|)$ if the PMF is uniform. Entropy can equivalently be defined as the expectation of the random variable $\log(1/p(X))$.

Definition 2.3.2. *The* joint entropy *$H(X, Y)$ of two random variables X, Y with joint PMF $p(x, y)$ is*

$$H(X, Y) = -\sum_{x \in X} \sum_{y \in Y} p(x, y) \log(p(x, y))$$

Definition 2.3.3. *The* conditional entropy $H(Y|X)$ *of two random variables* X, Y *with joint PMF* $p(x, y)$ *is*

$$H(Y|X) = -\sum_{x \in X} \sum_{y \in Y} p(x, y) \log(p(y|x))$$

Definition 2.3.4. *The* mutual information $I(X, Y)$ *of two random variables* X, Y *with joint PMF* $p(x, y)$ *is*

$$I(X; Y) = \sum_{x \in X} \sum_{y \in Y} p(x, y) \log \left(\frac{p(x, y)}{p(x)p(y)} \right)$$

Entropy, conditional entropy and mutual information are always nonnegative. The following equations hold:

- $H(X, Y) = H(X) + H(Y|X) = H(Y) + H(X|Y)$
- $I(X; Y) = H(Y) - H(Y|X) = H(X) - H(X|Y)$
- $I(X; X) = H(X)$
- $I(X; Y) + H(X, Y) = H(X) + H(Y)$

The last equation is particularly interesting, as it shows that the sum of the entropies of two random variable is greater or equal than their joint entropy. In particular

$$H(X) + H(Y) = H(X, Y) \Leftrightarrow I(X; Y) = 0 \Leftrightarrow X \text{ and } Y \text{ are independent}$$

showing that $I(X; Y)$ is a measure of the dependence of X and Y. We will use this intuition in Chapter 3 to quantify the dependence between an observable variable O and a secret variable h, thus quantifying how many bits of h an attacker learns by knowing the value of O.

Entropy and mutual information can also be defined for probabilistic processes, as the limits of the entropy and mutual informations for a given amount of time.

Definition 2.3.5. *Let* $X_1, X_2, ...$ *be a discrete time-indexed probabilistic process. Then we define its entropy as the limit of the joint entropy of its random variables:*

$$H(X_1, X_2, ...) = \lim_{n \to \infty} H(X_1, X_2, ..., X_n)$$

Note that since this is the limit of a monotonic non-decreasing non-negative sequence, it either converges to a non-negative real number or diverges to positive infinity.

Definition 2.3.6. *Let* $X_1, X_2, ...$ *and* $Y_1, Y_2, ...$ *be two disctere time-indexed probabilistic processes. Then we define their mutual information as*

$$I(X_1, X_2, ...; Y_1, Y_2, ...) = \lim_{n \to \infty} I(X_1, X_2, ..., X_n; Y_1, Y_2, ..., Y_n)$$

This is also the limit of a monotonic non-decreasing non-negative sequence, so it either converges to a non-negative real number or it diverges to positive infinity.

Since the entropy of probabilistic processes is potentially infinite, it is common for these kind of processes to compute the average entropy for each computation step, known as *entropy rate*.

Definition 2.3.7. *Consider a Markov chain $\mathcal{C} = (X_0, X_1, X_2, ...)$. Then the entropy rate of $\mathcal{C}$ is defined as*

$$\bar{H}(\mathcal{C}) = \lim_{n\to\infty} \frac{1}{n} H(X_0, X_1, ..., X_n)$$

when such limit exists.

The entropy rate of a process is clearly zero if the entropy of the process converges. We can similarly define mutual information rate:

Definition 2.3.8 ([82])**.** *The* mutual information rate *(or* rate of actual transmission*) between two time-indexed probabilistic processes X_i and Y_i is*

$$\bar{I}(X;Y) = \lim_{n\to\infty} \frac{1}{n} I(X_0, X_1, ..., X_n; Y_0, Y_1, ..., Y_n)$$

when such limit exists.

2.3.1 Memoryless Discrete Channels

A *discrete channel* is a system with an input alphabet X, an output alphabet Y and a conditional probability matrix $p(y|x)$ storing the probability of observing each output y for each input x. A channel is *memoryless* if the probability distribution on the output is independent of any input and output except the last input.

Definition 2.3.9. *The* channel capacity *C of a discrete memoryless channel is the maximum mutual information of X and Y over all possible input distributions $p(x)$:*

$$C = \max_{p(x)} I(X;Y)$$

Chapter 3

State of the Art

> *Nemo solus satis sapit.*
>
> No man is wise enough by himself.
>
> Plautus, *Miles Gloriosus*, III.3.12

Quantitative approaches to security measurements have been subject to an intensive amount of research in the last dozen of years. Insightful theoretical results and important practical applications have convinced a growing number of researchers to investigate quantitative measures to understand and guarantee security. As a consequence, many such measures have been proposed by different groups. Fortunately, many of these measures have been shown to be strictly correlated, allowing us to obtain results for one of them and be sure that the same results apply to the others.

A generic *quantitative security scenario* is a scenario in which a given *system* is under attack by a given *attacker*, and a given *quantitative measure* is used to quantify the effectiveness of the attack. The quantification is sometimes performed automatically by a *tool*.

We discuss the most common system model, the channel matrix, in Section 3.1. In Section 3.2 we discuss the main measures of security based on probability or information theory. In Section 3.3 we present many possible variants for the attacker model. Finally, in Section 3.4 we introduce some of the current qualitative and quantitative tools for automated security analysis.

3.1 System Model

It is common to model a system as a channel. In information-theoretical terms, a channel is anything that depends on some input and produces some output according to the input. Information theory is concerned with analyzing channels and measuring how much information they can transmit.

For the security scenario, the system under analysis depends on the value of the secret and produces some output, so it is natural to model it as a channel with the secret as an input and the output of the system as the output. As explained in Chapter 2,

memoryless discrete channels are encoded as a matrix, aptly known as the *channel matrix* $\mathcal{C}$.

The input alphabet X of the channel matrix $\mathcal{C}$ corresponds to the possible values of the secret, while the output alphabet Y corresponds to the possible observables. The value in the matrix corresponding to the row $x \in X$ and column $y \in Y$ is the conditional probability $P(y|x)$ that the observable y will be observed if the value of the secret is x. A matrix is said to be *deterministic* if all of its values are either 0 or 1, *probabilistic* otherwise. Intuitively, deterministic matrices model deterministic programs, i.e. programs in which for each value of the secret the program deterministically outputs one and only one observable.

In practice we are actually modeling the attack of the system by an attacker, so the output of the channel must be not just what the system produces, but which different outputs the attacker can discriminate: if the system produces a natural number but the attacker can only distinguish the parity of the output, the channel has actually only two possible outputs, *even* and *odd*.

The goal of this thesis is to propose a different system model than the channel matrix, so let us point out some limitations of this model:

- The size of the channel matrix is always exponential in the bit size of the secret and in the bit size of the output. If a system has a 64-bit secret, the matrix has $2^{64} \approx 1.8 \cdot 10^{19}$ rows; if it also has a 64-bit output it has the same amount of columns and $\approx 10^{38}$ cells. The computational problems in just storing such a model in a computer are evident.
- The channel matrix assumes an input-output behavior for the system: the system has some secret, does some computation producing some output, and terminates. Only after termination can the attacker analyze the data and try to infer the value of the secret. This assumption is very limiting, as it does not permit to analyze systems that produce output forever: the channel matrix for such a system would have an infinite number of columns. Similarly, it is not possible to encode an attacker that can continuously receive data from the system and infers information before the system terminates.
- The channel matrix model encodes assumptions about both the system and the attacker, as explained by the example above. Encoding the system and the attacker separately would allow to reuse the system model for different attackers and would provide a better understanding on how the observational properties of the attacker influence his effectiveness.
- Obtaining the channel matrix from a system is not an easy task, as shown by the large amount of effort that has been spent in developing techniques to compute [32] or approximate [23, 30, 31] it. The size of the matrix is not the only problem: determining the probability of an output for a program in a Turing-equivalent language is in general not decidable, as it would decide Turing's halting lemma.

Our proposal, detailed in Chapters 4 and 5, is to substitute the channel matrix with Markovian process models.

3.2 Security Measures

We provide a quick review of some different measures of information leakage that have been proposed.

3.2.1 Shannon Leakage

When we introduced entropy in Chapter 2, we explained that it quantifies the uncertainty expressed by a probability distribution: $H(X)$ quantifies how many bits of information we do not know about the random variable X. Imagine that X represents a secret value, and Y an observable value. Then the conditional entropy $H(X|Y)$ quantifies how many bits of information we do not know about X after observing Y, and the mutual information

$$I(X;Y) = H(X) - H(X|Y)$$

quantifies how many bits of information we have learned about X by observing Y.

This is the basic idea behind *Shannon leakage*: when X is a secret and Y an observable then $I(X;Y)$ quantifies how many bits of X are learned by observing Y, and consequently what is the leakage of information about X to an attacker able to observe Y. The idea of using information theory to define quantitative information flow on programs can be traced to the work of Denning [38, 39]. McLean introduced time in the model, defining a low-level variable at time t as secure if it does not depend on any high-level variable before time t [66]. The idea of using mutual information to quantify leakage was proposed by Gray [47]. Clark, Hunt and Malacaria presented a way to automatically compute Shannon leakage from the syntax of a simple language without loops [33] and of a complete imperative language with loops [34, 60]. They also discuss their approach in relation with the previous information-theoretical ones [34]. Boreale has shown how to compute Shannon leakage for processes specified in process calculi [17]. The work of Köpf et al. on information leakage of side channels [52, 53, 54] mostly uses Shannon leakage as security measure. The work of Chothia et al. on approximating channel matrices [23, 30] is usually based on Shannon leakage, but can be used for any measure and thus considered measure-agnostic.

In this thesis we will always define leakage as Shannon leakage, and consequently focus on how to compute mutual information in a particular scenario. Nonetheless, we are mostly focused on proposing an alternate system model, so we do not particularly advocate for one measure over another.

3.2.2 Bayes Risk

Bayes risk measures the probability that an attacker will fail to guess the secret after using Bayesian hypothesis testing to try to infer it.

Bayesian hypothesis testing is an inference technique to determine which of a finite set of hypotheses about a system is most likely to be true after observing a given amount of data produced by the system [48, Chapter 4]. In the system-attacker scenario each possible value of the secret corresponds to a different hypothesis and the observations correspond to the available data. The hypothesis most supported by the data thus corresponds to the most likely value of the secret. This strategy is also known as Maximum A Posteriori (MAP).

Let a channel C have input alphabet X, output alphabet Y and conditional probabilities $p(y|x)$ for each $x \in X, y \in Y$ and let $p(x)$ be a prior distribution over X. Then Bayes risk is computed as

$$P_e = 1 - \sum_{y \in Y} \max_{x \in X} p(y|x)p(x)$$

The use of Bayesian hypothesis testing as a strategy for the attacker is common in security literature [64, 77]. The application of Bayes risk analysis to database

security has been explored by Chatzikokolakis, Palamidessi and Panangaden [24, 25]. In particular it is interesting to notice that there is a connection between the conditional entropy of a system and the upper bound on its Bayes risk. Such bonds have been proposed by Rényi, improved by Hellman and Raviv and by Santhi and Vardy and more recently improved by Chatzikokolakis, Palamidessi and Panangaden [25, 44, 81].

Recall the formula for conditional entropy:

$$H(X|Y) = -\sum_{y \in Y} p(y) \sum_{x \in X} p(x|y) \log p(y|x)$$

Then the Hellman-Raviv bound for Bayes risk is

$$P_e \leq \frac{1}{2} H(A|O)$$

and the Santhi-Vardy bond is

$$P_e \leq 1 - 2^{-H(A|O)}$$

and we note that the Hellman-Raviv bound is tighter in case of a binary hypothesis, i.e. $|X| = 2$, and the Santhi-Vardy bound is tighter otherwise. Chatzikokolakis et al. improve the bounds by considering in which points the risk function changes inclination, using them to find a finite number of "corner points" that characterize a hyperplane defining an even tighter bounds than the ones above [25].

Bayes risk and min-entropy leakage are both useful measures of leakage when the attacker can perform only one observation and then has to choose a value for the secret. Lin and Kifer show the importance of Bayes risk in their work about formalizing and measuring security guarantees for algorithms working on databases [58, 59]. In particular they show that estimators not based on the Bayesian MAP strategy have higher probabilities of failing to guess the secret than the Bayesian MAP, motivating the choice of Bayes risk as a measure for this scenario. Nonetheless we believe that this optimality depends on the choice of the prior information of the attacker, as argued by Malacaria [61]. Also, the comparison is made against non-axiomatic estimators (least-squares and maximum likelihood) and not against information-theoretical measures.

3.2.3 Min-entropy Leakage

Min-entropy leakage was proposed by Smith [84] to better quantify the vulnerability of a system against an attacker that is able to observe the system only once before having to choose a password, what is known as a *one-try attack* scenario.

Let X be a random variable representing a secret value and π a prior probability distribution on its support. Let C be a channel from X as input and a random variable Y representing an observable output, as explained in Section 3.1.

Then the *prior vulnerability* of π is

$$V(\pi) = \max_{x \in X} \pi(x)$$

and the *posterior vulnerability* of π and C is

$$V(\pi, C) = \sum_{y \in Y} \max_{x \in X} \pi(x) C(x, y)$$

Note that prior and posterior vulnerabilities are probabilities, and in particular vulnerability is the complement of Bayes risk. We want to transform them in entropies to be able to quantify the difference in bits. The *prior min-entropy* is

$$H_\infty(\pi) = -\log V(\pi)$$

while the *posterior min-entropy* is

$$H_\infty(\pi, C) = -\log V(\pi, C)$$

and the *min-entropy leakage* is the difference of the two:

$$\mathcal{L}(\pi, C) = H_\infty(\pi) - H_\infty(\pi, C)$$

Min-entropy is meant to capture the change in the probability that the attacker will be able to guess the value of the secret in one attempt. Smith claims that min-entropy leakage better represents the vulnerability of a system in the one-try attack scenario [41, 84, 85] but Malacaria warns that Smith's argument only holds for an attacker with no prior information on the secret, and it is always possible to find a probabilistic attacker such that Smith's argument does not hold, since the two channels are unrelated in the Lattice of Information [61, Section 4.5]. The contributions of this thesis are measure-agnostic, so we refer the interested reader to the cited papers for the details.

3.2.4 g-leakage

Alvim, Chatzikokolakis, Palamidessi and Smith have recently proposed a generalization of min-entropy leakage called *g-leakage* [6]. The idea behind g-leakage is that in real cases it is not necessarily true that all bits of the secret have the same weight. For instance, consider a secret of length k^2 bit consisting of k passwords of length k for the same system. It is intuitive that leaking a full password and leaking the first bit of each password both amount to leaking k bits. Nonetheless, an attacker with a full password can access the system in one attempt with probability 1, while an attacker with the first bit of each password can access the system in one attempt with probability $2^{-(k-1)}$.

The generalization depends on a *gain function* on the leaked bits that abstracts knowledge about the specific system. While it is proposed as a generalization of min-entropy leakage, g-leakage can be used to generalize Shannon entropy or any other information-theoretical measure by considering expert knowledge about the system. As such it is not necessarily a measure *per se*: we can consider it as an additional layer that can be used to improve the results of a given measure when applied to a specific problem.

3.2.5 Other Measures and Ordering

Many other measures have been proposed to evaluate the effectiveness of an attacker against a system. These include the change in the expected probability that the attacker has of guessing the secret in one try, the expected number of guesses that it will take for the attacker to guess the secret, and more. Fortunately, it has been proved [61, 88] that all these measures preserve the same ordering of attackers.

Consider two different attackers, modeled by the two channels A and B. Say that $A \sqsubseteq B$ if, for any given prior information on the secret, the attack of A is never more effective than the attack of B according to a given measure m (more on prior information

of the attackers will be presented in the next Section). For instance, it is intuitive that an attacker able to observe strictly more details about the system's behavior will usually learn more information. This means that the attack of A is not more effective than the attack of B for any prior information according to m. It has been proved that this ordering is preserved for most of the measures presented, and that furthermore it forms a complete lattice, known as the Lattice of Information [57].

3.3 Attacker Model

The encoding of the *attacker* is another important part of the analysis. Determining which capabilities the attacker has is fundamental to understand how reliable the security analysis is. Many security measures involve quantifying the amount of knowledge about the secret that the attacker gains by observing the system, defined as the difference between the knowledge the attacker has after and before the attack. Clearly this requires a definition of what the attacker knew before attacking the system, known as *prior information* of the attacker. Since probability theory is designed to encode partial information, the prior information is usually modeled as a *prior probability distribution* over the possible values of the secret.

Also part of the encoding of the attackers are two additional parameters: whether the attacker is able to interact with the system, and whether the attacker is able to perform multiple observations on the system. In some case different measures are more fitting to capture the effectiveness of the attacker according to these parameters.

3.3.1 Prior Information

Prior information encodes what the attacker knows about the secret before performing the attack. It is usually modeled as the prior probability distribution over the values of the secret.

Probabilistic Attacker

The probabilistic attacker has some information, in the form of constraints, about the values of the secret, including the secret's size. For instance an attacker could know that the secret is a number from 0 to 7, that it is not 3 or 6, that the probability that it is 4 is 0.4, and that the probability that the secret is 0 is twice the probability that it is one. The prior probability distribution is constructed by taking the distribution with the highest entropy among all those respecting the constraints, in accordance with the Maximum Entropy Principle proposed by Jaynes [48].

The prior distribution encoding the constraints above is $\{0 \mapsto 0.2, 1 \mapsto 0.1, 2 \mapsto 0.1, 3 \mapsto 0, 4 \mapsto 0.4, 5 \mapsto 0.1, 6 \mapsto 0, 7 \mapsto 0.1\}$. We assume that the information the attacker has about the secret is accurate. The case in which this does not necessarily hold has been studied by Clarkson et al. [35, 46].

It is important to notice that the value of the leakage depends on the prior distribution on the secret, and thus on the prior information about the secret possessed by the attacker; it is not just a function of the program or channel analyzed. We show this with a simple example, using Shannon entropy as the security measure.

Consider an authentication protocol with a 2-bit password, i.e. the password has four possible values: either 0, 1, 2 or 3. Four attackers A_1, A_2, A_3 and A_4 are observing the system. The attackers enter 0 as password, and the system rejects them.

Attacker A_1 had no prior information on the value of the password, except that it is 2 bit long. His prior distribution is uniform: $\{0 \mapsto 1/4, 1 \mapsto 1/4, 2 \mapsto 1/4, 3 \mapsto 1/4\}$ with an entropy of 2 bits. After the attack he knows that the secret is not 0, so his posterior distribution is $\{0 \mapsto 0, 1 \mapsto 1/3, 2 \mapsto 1/3, 3 \mapsto 1/3\}$ with an entropy of $\log 3 = 1.5849...$ bits. By observing the response of the system he learned $2 - 1.5849... = 0.4150...$ bits of information.

Attacker A_2 knows that the first bit of the password is a 0, so it is either 0 (00 in binary) or 1 (10). His prior distribution is $\{0 \mapsto 1/2, 1 \mapsto 1/2, 2 \mapsto 0, 3 \mapsto 0\}$ with an entropy of 1 bit. After the attack he knows that the secret is not 0, so his posterior distribution is $\{0 \mapsto 0, 1 \mapsto 1, 2 \mapsto 0, 3 \mapsto 0\}$ with an entropy of 0 bits, i.e. he knows that the secret is 1. By observing the response of the system he learned $1 - 0 = 1$ bits of information.

Attacker A_3 knows that the second bit of the password is a 1, so it is either 1 (01 in binary) or 3 (11). His prior distribution is $\{0 \mapsto 0, 1 \mapsto 1/2, 2 \mapsto 0, 3 \mapsto 1/2\}$ with an entropy of 1 bit. After the attack he knows that the secret is not 0, but he knew that already, so his posterior distribution remains $\{0 \mapsto 0, 1 \mapsto 1/2, 2 \mapsto 0, 3 \mapsto 1/2\}$ with an entropy of 1 bit, i.e. he did not learn anything. By observing the response of the system he learned $1 - 1 = 0$ bits of information.

Finally, attacker A_4 already knew that the password is 1, so his prior distribution is $\{0 \mapsto 0, 1 \mapsto 1, 2 \mapsto 0, 3 \mapsto 0\}$ with an entropy of 0 bits. After the attack he knows that the secret is not 0, but he knew that already, so his posterior distribution remains $\{0 \mapsto 0, 1 \mapsto 1, 2 \mapsto 0, 3 \mapsto 0\}$ with an entropy of 0 bit, i.e. he did not learn anything. By observing the response of the system he learned $0 - 0 = 0$ bits of information.

This is the case if, for instance, A_4 is actually the user that should be authorized to access the system. Note that after the attack both A_2 and A_4 know the secret password, but A_2 has a higher leakage (in fact, the highest of the four attackers) because he learned the secret by observing the system, contrarily to A_4 that knew it before. This shows how prior information is important in characterizing how successful an attack on the system is.

Ignorant Attacker

The ignorant attacker is the case of the probabilistic attacker in which the attacker has no prior information about the value of the secret except its size. This is encoded as a uniform prior distribution on the possible values of the secret. The QUAIL tool that we introduce in Chapter 6 assumes the ignorant attacker scenario.

The ignorant attacker is a common base case for attacker knowledge, since the uniform distribution it is simple to encode. Smith's example of a case in which min-entropy leakage is more relevant than Shannon leakage [84] is valid only for an ignorant attacker, as explained by Malacaria [61].

In most of the example of this thesis it will be assumed for simplicity that the attacker is ignorant. The theory we propose, however, can be used to examine any probabilistic attacker.

Demonic Attacker

The demonic attacker has exactly the prior information that maximizes the amount of leakage of information from the system. If the system is seen as a channel between the secret and the attacker, the amount of leakage for the demonic attacker corresponds to

the maximum amount of information that the channel can transmit, known as *capacity* of the channel [69].

The channel capacity of a system is important because it can be used as a guarantee for the reliability of the system: if the demonic attacker leaks 3 bits of information, we can guarantee that 3 bit is the maximum amount that any attacker can leak, over all possible prior information of attackers. This allows us to produce more general results that do not depend on the choice of a given attacker.

Different measures compute channel capacity differently, but keeping the same concept that capacity is the maximum leakage over all prior distributions of attackers. Among the others, Chen and Malacaria used Bellman equations to compute the channel capacity of traces, subtraces and multithreaded programs [27, 29, 62] and Lagrange multipliers to compute the channel capacity of various protocols including Onion Routing and Crowds [28]. Channel capacity for interactive systems has been studied by Alvim et al. [5].

Side-channel Information

Up to now we have considered the prior information that the attacker has about the secret before performing the attack. In security analysis, and particularly in cryptoanalysis, it has also been considered that the attacker could infer information about the secret by analyzing other channels, like computation time and power consumption [21, 50, 51, 68].

Recent progress has been made in formalizing this side-channel information and automatically quantifying the amount of information that the attacker gains from it, mainly due to the work of Köpf et al. [52, 53, 54]. Side-channel studies are not in the scope of this thesis.

3.3.2 Interactivity

In case the system can input some information from the attacker itself, it is necessary also to encode this additional behavior. In general it can be considered that there is another prior distribution (separated from the one on the secret) on the input variable. This allows to keep the model probabilistic. The problem has been considered by Volpano et al. in their information leakage type system [87]. O'Neill et al. [72] consider a similar encoding of the behavior of the user in their user strategies, also considering the case in which the input strategy is unknown to the attacker.

Alternatively, it can be considered that the attacker inputs the worst possible input, similarly to what is done for the demonic attacker for the prior information about the secret. Alvim et al. recently proposed to use channels with feedback to encode this behavior [5], also pointing out that in this case mutual information is not appropriate to compute leakage, and directed information should be used instead. Chothia et al. prove that an input strategy leaking information exists if and only if the uniform input strategy does, and also note that the input strategy with maximum leakage may not exist if the program does not terminate [32].

We believe that the Markovian encoding of processes we propose in this thesis will be of help in analyzing this problem, as it handles naturally non-terminating programs, as shown in Chapter 5. Nonetheless, in this thesis the attacker will always be assumed to be non-interactive.

3.3.3 Repeated Attacks

A scenario may allow the attacker to run and observe the system only once or more than once. For instance, if the attacker wants to break the anonymity of the votes of an election he cannot rerun the election and accumulate data, while if the attacker is trying to break an authentication scheme he normally can try more than one password.

Probably the most important works on repeated attacks are those by Boreale et al. [18, 19]. It has been noted by Smith [84] that the choice of a repeated or single-attack scenario can influence the relevance of the security measure used. Different notions of leakage for single-attack scenarios have been studied by Braun et al. [20]. Repeated attacks have been considered by Clarkson et al. [35], by Malacaria et al. [61, 63] and by Barthe et al. [10].

Repeated attacks are not discussed in this thesis.

3.4 Tools

We list some of the tools that have been developed to analyze different aspects of system security.

Some tools that are not explicitly developed for security can be used to obtain security-related results. For instance, APEX [49] and PRISM [56] can be used to test language equivalence. These tools can be used for instance to verify whether the output language of a system is independent from a particular secret variable. This corresponds to a test of whether the leakage of the system is exactly zero or not. Such a qualitative analysis is not able to discriminate different systems with positive leakage, and cannot be used to asses whether the leakage of a system is small enough to be insignificant or large enough to be a risk.

Most quantitative tools computing information leakage are able to analyze only deterministic programs. This includes the proposal by Backes et al. [8] to use a model checker to iteratively refine discrimination relations and compute the leakage from the size of the relations, and the QIF analyzer by Mu and Clark [70]. Similarly, the quantitative taint analysis techniques developed by McCamant et al. [65, 71] do not extend to the precise analysis of probabilistic programs.

A common approach is to trade precision for efficiency by implementing estimation tools like LeakWatch, LeakiEst and Weka [31, 32, 43]. Such tools usually work by estimating the channel matrix for the system, and thus are unable to compute results when the size of the secret is large, since the channel matrix has exponential size in the size of the secret.

The JPF-QIF [76] tool by Phan et al. computes only an upper bound for the channel capacity of a system, but is able to handle programs written in Java, being an extension of the Java Pathfinder verification environment.

Finally, the CH-IMP tool developed by Chothia et al. [32] is the closest relative to the QUAIL tool we present in Chapter 6. CH-IMP allows the user to declare explicitly secrets and observables in the code of the system, then uses a Markov chain semantics to construct a channel matrix for these secrets and observations. Like all channel matrix-based tools, CH-IMP is not able to handle efficiently cases in which the secret is larger than ≈ 20 bits. We also note that in QUAIL it is also possible to declare secret and observable variables in any point of the code but at the cost of not being able to guarantee the termination of the analysis.

Chapter 4

Information Leakage of Terminating Processes

> *When you know a thing, to hold that you know it; and when you do not know a thing, to allow that you do not know it; this is knowledge.*

Confucius, *Analects*, 2:17

We show how to model a process with a Markov chain such that it is possible to compute the information leakage of the process by operating on the chain. This modeling allows us to use graph-theoretical insight to address problems that would be impossible to represent in a strictly input-output model like a channel matrix, for instance non-termination.

This chapter is focused on modeling terminating processes, i.e. processes that assume a deterministic behavior with probability 1. The case of handling non-terminating processes will be addressed in Chapter 5.

We will show how to model a program-attacker scenario with a Markov chain and how to compute the information leakage of the scenario. This requires an an algorithm to compute the entropy of a Markov chain.

Entropy of a Markov chain can be computed by defining a slightly non-standard reward function on the transition probabilities of the chain, and then computing the expected long-term reward for this reward function. Casting entropy as a reward function allows us to use machinery usually meant for the optimization and computation of such rewards.

4.1 Markovian Modeling of Processes and Attackers

A Markov chain is a probabilistic process respecting the Markov property, meaning that the distribution over the states at a certain time depends only on the distribution at the time immediately before. This means that in each state of the Markov chain there must be enough information to determine the probability distribution on the next time step.

We need to keep this idea in mind when encoding a process as a Markov chain. We are considering processes written in an imperative language, so each state must contain

```
...
42 x := x +3;
43 if (x<8) then x := x + 1;
...
```

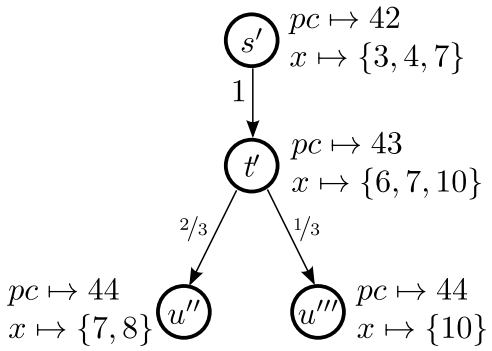


Figure 4.1: Markov chain semantics for a snippet of imperative code

all the information the compiler needs to determine what the process will do next. This leads naturally to having a state of the Markov chain for each possible assignment of values to the variables of the program, including which line of code is currently being processed with the assignment. Since the code is not being modified during the execution of the program, we can just use the line number in the program counter as one of the variables recorded in the state and assume that we can reference the source code by line.

4.1.1 A Simple Example

As an example consider that line number 42 in the source code is `x := x + 3;`, and that state s is the assignment $\{pc \mapsto 42, x \mapsto 3\}$, where pc is the value of the program counter. Then in the model there will be a transition with probability 1 to a state t where $\{pc \mapsto 43, x \mapsto 6\}$: the program counter is increased by 1 as a default behavior, and the variable x is increased by 3 according to the instruction we are processing.

Assume that the program is written in an imperative language. In the following, a state of the Markov chain corresponds to one assignment of values to the variables and program counter, also meaning that no two distinct states have the same assignment.

Since imperative programming languages are deterministic, for each state there will be exactly one other state that will follow with probability 1. This would lead to a trivial and uninteresting Markov chain. The reason why we are using probabilistic processes as models is their capability of representing imperfect information about the behavior of a system. We use this to represent what an attacker with imperfect information about the process knows about the process' behavior. In practice we lift the states to represent not just an assignment of values to the variables, but a level of information of an attacker about these assignments.

Imagine that we allow states to represent assignment of sets of values to each variable except the program counter. Then for the example above we may have a state $s' = \{pc \mapsto 42, x \mapsto \{3,4,7\}\}$ representing the fact that the variable x could have value either 3, 4 or 7. From s' we would have a transition with probability 1 to state $t' = \{pc \mapsto 43, x \mapsto \{6,7,10\}\}$, representing the fact that variable x has been

increased by 3 and is now either 6, 7 or 10.

Now imagine that line 43 of the source code is `if (x<8) then x := x + 1;`. Then from state t' we could only have a transition with probability 1 to state $u' = \{pc \mapsto 44, x \mapsto \{7, 8, 10\}\}$. But we could actually define two more precise successor states for t': $u'' = \{pc \mapsto 44, x \mapsto \{7, 8\}\}$ is visited if the guard is evaluated to be true and $u'' = \{pc \mapsto 44, x \mapsto \{10\}\}$ is visited if the guard is evaluated to false.

The probability of transitioning from t' to u'' and u''' is the probability that the guard is true or false in t', respectively. This means that if we have in each state a probability distribution over possible values of each variable we can have two possible successor states each time a conditional statement is evaluated, and in the state we have enough information to compute the probability that the guard will be true or false. If in t' we have that variable x has a uniform distribution over the three values 6, 7 and 10 then we can immediately compute that the guard `x<8` will be true with probability $2/3$ and false with probability $1/3$, thus the probability of transitioning from t' to u'' is $2/3$ and the probability of transitioning from t' to u''' is $1/3$. The snippet of code and resulting Markov chain are depicted in Fig. 4.1.

Note that, since we assume a distribution on the variable x in every state of the Markov chain, we could have split t' or even s' in three different states, one for each possible assignment of values to x, and given each of them a starting probability depending on the initial distribution over x. This would have meant encoding a different level of information about the attacker, and would also have multiplied the number of states of the chain. How do we choose at which level of abstraction to model the behavior of the system?

That depends on what we want to obtain from our model. Our goal in the end is to be able to compute how much information can be inferred from some known observable variable to some unknown secret variable, where "known" and "unknown" refer to the information available to the attacker, and "observable" depends on what the process outputs at the end of its computation. This means that our Markov chain model will depend both on the process itself and on the knowledge available to the attacker, and in this sense it models not only the process but the observation by a given attacker of a given process.

4.1.2 Markov Decision Process Semantics

Now that we have given an intuition of what the method produces, we shall define it more formally. We start by producing a Markov Decision Process $\mathcal{P}$ encoding the behavior of the system alone. Then we will use the encoding of the attacker to transform $\mathcal{P}$ into a Markov chain.

We distinguish a list of variables (pc, L, H) where pc is the program counter, L is the set of public variables and H is the set of private variables. The secret is one of the private variables. In each state a given value is assigned to each variable in L and to the program counter, while a probability distribution over a set of possible values is assigned to each variable in H. Assume that all variables are integer of fixed size. If we want to guarantee termination of the construction we can force all variables to be declared at the beginning of the code, thus restricting the variables to finite domains, but this is not necessary for the correctness of the procedure. High-level variables are read-only and cannot be accessed externally; their value changes only through observation via evaluation of `if` guards. This models the fact that the attacker cannot directly print or modify a secret value, but only learn about it by observing the decisions that the process takes that depend on it.

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

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

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

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

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

Figure 4.2: Execution rules in probabilistic partial information semantics.

Let l (resp. h) range over names of public (resp. private) variables and p range over reals from $[0; 1]$. Let `label` denote program points and `f` (`g`) pure arithmetic (Boolean) expressions. Assume a standard set of expressions and the following statements:

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

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 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 semantics in Fig. 4.2 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. 4.2, `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. In the rest of the thesis we will conventionally not draw the probability 1 loops in absorbing states to avoid cluttering. The `if` rule produces a nondeterministic decision state.

4.1.3 Modeling the Attacker

The MDP we obtained models the behavior of the process at the required abstraction level, meaning that low-level variables are given values and high-level variables are given probability distributions. Now we need to also consider the model of the attacker to transform the MDP in the Markov chain modeling the process when observed by the attacker.

Definition 4.1.1. *An attacker is a pair $\mathcal{A} = (\mathcal{I}_\mathcal{A}, \mathcal{R}_\mathcal{A})$ where $\mathcal{I}_\mathcal{A}$ is a probability distribution over the possible values of the secret encoding the attacker's prior information about it and $\mathcal{R}_\mathcal{A}$ 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.*

The attacker has some prior information about the secret before observing the process. This is encoded as the prior $\mathcal{I}_\mathcal{A}$, that assigns probability distributions to all high-level variables. In particular, since we assume that the attacker has access to the source code of the process, he will know the length in bit of the secret variables, since he can read the variable declarations. For instance if the secret is declared as a 16-bit integer value the attacker will know that it is an integer from 0 to 65535. Any additional information the attacker possesses, like that the secret is not a number from 18 to 556 or that the probability that the secret is 12 is three times the probability that it is 16, is encoded in the prior distribution as well, as explained in Section 3.3.

The discrimination relation $\mathcal{R}_\mathcal{A}$ encodes what states of the MDP the attacker can discriminate. The usual case is that some of the low-level variables are observable by the attacker, and thus the attacker can discriminate two states iff they differ in the value of at least one of the observable variables. States with the same values for all the observable variables are indistinguishable to the attacker. Remember from Chapter 3 that an attacker with strictly greater discriminatory power leaks more information than an attacker with less discriminatory power.

4.1.4 Building the Markov Chain from the MDP Semantics and Attacker Model

Let a *scenario* be a pair $(\mathcal{P}, \mathcal{A})$ with an MDP and an attacker. We say that a Markov chain $\mathcal{C}$ *models* a scenario $(\mathcal{P}, \mathcal{A})$, written $\mathcal{C} \models (\mathcal{P}, \mathcal{A})$, when it is obtained by the procedure described below.

Step A: Apply Prior Information

Observe that the only nondeterministic transitions not labeled with $\top$ in the MDP semantics in Fig. 4.2 are labeled with the `if` conditions. Since there is no probability distribution over the values of the high-level variables in the process $\mathcal{P}$ we cannot, in general, determine the probability that a guard will be true. However, the probability distributions of the high-level variables are provided in $\mathcal{I}_\mathcal{A}$, which allows us to assign the probability distributions in the initial state. When a guard is evaluated, these probability distributions are propagated to consider the fact that each guard partitions the probability space in two.

For instance, consider that in a certain state the distribution over a variable `h` is uniform over the values $\{0, ..., 23\}$ and the guard `h<11` is evaluated. Then in the successor corresponding to the guard being true the new distribution over `h` will be uniform over $\{0, ..., 10\}$, while in the successor corresponding to the guard being false

Data: A Markov chain $\mathcal{C} = (S, s_0, P)$, the set $\mathcal{T}_\mathcal{A} \subseteq S$ of states to hide.
Result: The Markov chain $\mathcal{C} = (S \cup \{\uparrow\} \setminus \mathcal{T}_\mathcal{A}, s_0, P)$

Add to S the divergence state $\uparrow$ with $P_{\uparrow,\uparrow} = 1$ and $\pi^{(0)}_\uparrow = 0$;
while $T \neq \emptyset$ **do**
  Choose a state $t \in \mathcal{T}_\mathcal{A}$;
  if $P_{t,t} = 1$ **then**
    $\pi^{(0)}_\uparrow \leftarrow \pi^{(0)}_\uparrow + \pi^{(0)}_t$
    foreach $s \in S$ **do**
      $P_{s,\uparrow} \leftarrow P_{s,\uparrow} + P_{s,t}$
      $P_{s,t} \leftarrow 0$
    end
  else
    foreach $u \in S$ **do**
      $P^{\square}_{t,u} \leftarrow \frac{P_{t,u}}{1-P_{t,t}}$
      $\pi^{(0)}_u \leftarrow \pi^{(0)}_u + \pi^{(0)}_t P^{\square}_{t,u}$
      foreach $s \in S$ **do**
        $P_{s,u} \leftarrow P_{s,u} + P_{s,t} P^{\square}_{t,u}$
        $P_{s,t} \leftarrow 0$
      end
    end
  end
  $S \leftarrow S \setminus \{t\}$
  $\mathcal{T}_\mathcal{A} \leftarrow \mathcal{T}_\mathcal{A} \setminus \{t\}$
end

Algorithm 1: Hide a subset of the states of a Markov chain.

the new distribution over $\mathtt{h}$ will be uniform over $\{11, ..., 23\}$. Note that the probability distribution on each state can be obtained by just normalizing the initial distribution $\mathcal{I}_\mathcal{A}$ to the values allowed in the state.

Since we can now assign a probability to each transition, we can transform the MDP model in a Markov chain: 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.

Step B: Hide Unobservable States

We want to model the fact that most of the states of the system are not observable to the attacker. We assume that the attacker can start the process and then observe the value of the output after the process terminates, so all states except the initial state and the terminal states must be hidden. Let $\mathcal{T}$ be the set of all states of the Markov chain except the initial state and the terminal states.

The distinction between the indistinguishable states encoded in $\mathcal{R}_\mathcal{A}$ and the hidden states encoded in $\mathcal{T}$ is fine but fundamental. While the attacker can observe whether the system is in one of the equivalence classes of the discrimination relation $\mathcal{R}_\mathcal{A}$, he is not even aware of the existence of the hidden states in the set $\mathcal{T}$. For instance this means that he can count how many time steps the chain spends in an equivalence class

of $\mathcal{R}_{\mathcal{A}}$ but not how many steps the chain spends in the hidden states. To encode the attacker that only observes the output of the process after its termination, we hide all states except the starting state and the absorbing states of the chain: the attacker knows when the process starts and ends but cannot perform observations about what happens during the computation.

The fact that the hidden states in $\mathcal{T}$ cannot be observed raises a question: what happens if the process terminates in one of these states? In this case the attacker would not be able to observe the termination of the program. Similarly, what if the process does not terminate at all, and its behavior loops forever within these hidden states? It can be seen that the two cases are equivalent from the point of view of the attacker: he observes that the program gets to a certain point and that nothing else happens, including termination. We answer this question by assuming that the attacker can determine, likely via a timeout, whether the program's behavior is not terminating, and thus we consider termination as one of the possible observable outputs of the program.

We want to hide the states in the Markov chain to model the fact that the attacker is not able to observe them. The hidden states are removed from the chain one by one, and each time the transition probability of the other states are modified to consider the probability of eventually transitioning from one state to the other through the hidden state. The initial probability $\pi^{(0)}$ is similarly updated, in case the initial state is among the ones to hide; in this case the resulting Markov chain will in general have not only a single initial state, but a probability distributions over initial states. Also, we add a divergence state $\uparrow$ to represent behavior that stays forever in the hidden states. The procedure is implemented in Algorithm 1.

Step C: Compute the Quotients

Having modeled the problem scenario as a Markov chain, we now want to compute the information leakage of the scenario. To compute it we apply the formula

$$I(O,h) = H(O) + H(h) - H(O,h)$$

where (O,h) is the joint distribution of the secret and observable variables and (O) and (h) are the marginal distributions. At this step all three are modeled as one-step Markov chains, respectively $\mathcal{C}_{O,h}$, $\mathcal{C}_O$ and $\mathcal{C}_h$. We need to produce the three Markov chains and compute their entropy.

The three Markov chains are obtained from the scenario model by quotienting it by appropriate discrimination relations, as explained in Definition 2.2.3. The discrimination relation to obtain $\mathcal{C}_O$ is $\mathcal{R}_{\mathcal{A}}$, as it represents the attacker's discrimination on the secret. We will call $\mathcal{C}_O$ the *attacker's quotient*.

When computing a quotient all states in the same equivalence class are lumped together with the same weight, according to the fact that the attacker has no information about which state in the class he is actually visiting.

The transition probabilities are computed averaging the probabilities of each state in the same equivalence class, to model the fact that the attacker does not know in which of those states the process is. In our case the chain has been reduced by the hiding algorithm to a single 1-step transition, simplifying the quotienting process. The only transitions are the ones from the starting states to the absorbing states and the looping transitions with probability 1 of the absorbing states, so the quotienting formula presenting in Chapter 2 reduces to collapsing together absorbing states in the same equivalence class. The transition probability from the starting state to a class is the sum

of the transition probabilities from the starting state to the states composing the class. An example is shown in Section 4.2.2, in particular in Fig. 4.4.

The *secret's quotient* $\mathcal{C}_h$ represents the marginal distribution of the secret variable. Its entropy is also a measure of how many bits of the secret variable are actually used by the process. Let $\mathcal{R}_h$ be a discrimination relation such that two states are in the same class if and only if they assign the same set of possible values to the secret variables. Then $\mathcal{C}_h$ can be computed by quotienting the scenario Markov chain model by $\mathcal{R}_h$.

Finally we compute the *joint quotient* $\mathcal{C}_{O,h}$ representing the joint behavior of the secret and observable variables. It is obtained by quotienting the scenario Markov chain model by $\mathcal{R}_{\mathcal{A}} \cap \mathcal{R}_h$, thus lumping in the same class only states that agree both on the values of the observable variables and on the set of possible values for the secret variables.

Step D: Compute the Leakage

Finally, we compute the information leakage by computing $H(O)$, $\mathcal{C}_h$ and $\mathcal{C}_{O,h}$ and applying the formula $I(O,h) = H(\mathcal{C}_O) + H(\mathcal{C}_h) - H(\mathcal{C}_{O,h})$.

This requires us to be able to compute the entropy of Markov chains. Since in this procedure the chains are reduced by the hiding algorithm to a single step it reduces to computing the entropy of the distribution of the successors of the initial state, but in the rest of the chapter and in the next chapter we will have to compute the entropy of generic Markov chains. For this reason we discuss the problem more in depth in Section 4.2.

Note that if a process is deterministic, i.e. never uses the random assignment function, then $I(O,h) = H(\mathcal{C}_O)$ since $H(\mathcal{C}_h) = H(\mathcal{C}_{O,h})$. In this case we can simply compute the attacker's quotient and its entropy, as it corresponds with the information leakage.

4.2 Computing the Entropy of a Markov Chain

A Markov chain is an infinite probabilistic process, meaning that it can be considered as a single probability distribution over a potentially infinite number of paths, where each path is a sequence of visit to the states of the chain.

Entropy is a measure of the amount of information required to completely specify the behavior of a system. Since in the case of a Markov chain the system can have an infinite number of behaviors, we expect that the entropy of a Markov chain will in general be infinite. We will show how to characterize in polynomial time the cases in which entropy is infinite.

We define a reward function (see Chapter 2) on the transition of a Markov chain such that the expected reward over the infinite lifetime of the chain corresponds to its entropy. This allows us to compute the entropy of a chain in polynomial time in its size.

For a state s of a Markov chain (S, s_0, P) we define local entropy as follows:

Definition 4.2.1. *Let (S, s_0, P) be a Markov chain and $s \in S$ one of its states. Then the* local entropy *$L(s)$ of s is defined as*

$$L(s) = -\sum_{t \in S} P_{s,t} \log P_{s,t}$$

The local entropy of a state s is the entropy of the probability distribution over the successors of s. Intuitively, it corresponds to the amount of entropy that the system accumulates each time it visits s. For this reason, the entropy of the chain corresponds to the local entropy of each state multiplied by the expected time that the chain spends in that state:

Theorem 4.2.2. *For an MC $\mathcal{C} = (S, s_0, P)$ we have that $H(\mathcal{C}) = \sum_{s \in S} L(s)\xi_s$.*

Since it is known that the expected reward of a chain can be computed in polynomial time [78], we can also compute entropy in polynomial time thanks to Theorem 4.2.2.

4.2.1 Process Termination and Finiteness of Entropy

Since all and only the recurrent states have infinite expected residence time, then the entropy of the chain will be infinite if any of them has positive local entropy:

Corollary 4.2.3. *The entropy $H(\mathcal{C})$ of a chain $\mathcal{C}$ is finite iff the local entropy of all its recurrent states is zero.*

Since computing the local entropy of one state is linear in the size of the chain and the recurrent states can be identified in polynomial time by Tarjan's algorithm, then it is possible to determine in polynomial time whether the entropy of a chain is finite or not.

For a state to have zero local entropy it must have only one transition with probability 1, to itself or another state. Also, states in the same recurrent class have the same recurrence, so they are either all recurrent or all transient. This allows us to conclude that if a chain has finite entropy, each of its recurrent classes are either a single absorbing state or a cycle of deterministic states. In particular, all absorbing Markov chains have finite entropy.

On a similar intuition, whenever the chain visits a state with zero local entropy we can predict perfectly where it will be on the next step. If with probability 1 from a certain time step onwards the chain visits only such states, then we can predict exactly what the chain will do for the rest of its infinite life. Since entropy is the amount of information required to predict the behavior of the chain, for it to be finite it must be that after a certain point the behavior of the chain is deterministic, i.e. there is no ulterior information to discover. This intuition is important because we use it to characterize terminating behavior.

Remember that the Markov chain represents the attacker-system scenario, i.e. also considers some part of the encoding of the attacker. Since we are interested in how much information can be inferred about the secret, as far as we are concerned a scenario terminates in the moment in which its behavior does not reveal any more information. This means that the attacker already has all the information he needs to predict the behavior of the system onwards and, since he has access to the source code of the system, he knows this is the case. At such point the attacker can safely stop his observation of the system, and thus in this sense the scenario terminates.

On the extreme this means that a program that outputs the product of 3 and 7 is already terminated, since it does not depend on any secret and its behavior is completely predictable. Similarly, a password authentication does not leak any information to an attacker that knows the password, so in this sense the scenario is completely predictable to the attacker and can be already considered to be terminated. This does not hold for an attacker that does not know the password.

This simplification allows us to abstract many uninteresting details of the computation and to avoid known problems in characterizing non-termination of programs. Note

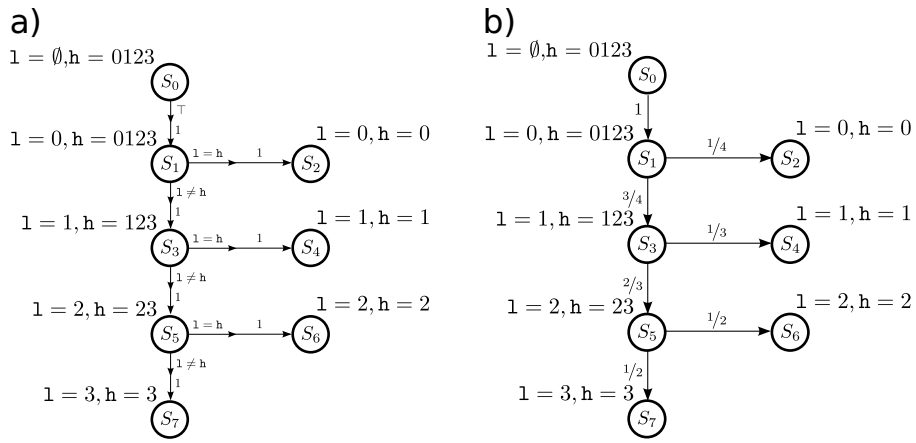


Figure 4.3: Implicit flow example. a) MDP semantics b) MC model

that in this sense the non-terminating program example proposed by Smith [86] actually terminates, since its Markov chain representation has finite entropy.

4.2.2 An Implicit Flow Example

We apply the modeling technique above to a classical academic example of information leakage [63]. A process has two variables, the low-level variable `l` and the high-level variable `h`, both of size 2 bits. The source code of the process is the following:

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

When the process terminates, `l` will have exactly the same value as `h`, even if no assignment like `l := h` exists. This is known as implicit flow, and is what we are interested in capturing.

We model the process as an MDP with the semantics explained above. The resulting MDP $\mathcal{P}$ is shown in Fig. 4.3a, where we are showing only the `if` states and the terminating states to avoid clutter[1]. The notation `h=0123` means that the set of allowed values for `h` in the state is $\{0, 1, 2, 3\}$; remember that at this point we still do not have a probability distribution on the values of `h`, as it is part of the encoding of the attacker. Similarly, the transitions from the `if` states are labeled with the guard of the `if` and its negation, but we cannot compute the probability for such guards yet.

Now we specify the attacker. We encode an attacker that has no prior information about the values of the secret except that is contained in a 2-bit variable. The attacker can observe the value of the variables only at termination, so all states except the starting state and the terminating states are hidden to him. Finally, the attacker can only observe the parity of the variable `l`, so he can discriminate only states where `l` is even from states where `l` is odd. This means that the attacker is $\mathcal{A} = (\mathcal{I}_\mathcal{A}, \mathcal{R}_\mathcal{A})$ where $\mathcal{I}_\mathcal{A} = (1/4, 1/4, 1/4, 1/4)$ and $\mathcal{R}_\mathcal{A} = \{(S_0), (S_1, S_2, S_5, S_6), (S_3, S_4, S_7)\}$.

[1] the full MDP has 16 states, most of which have a single transition with probability 1.

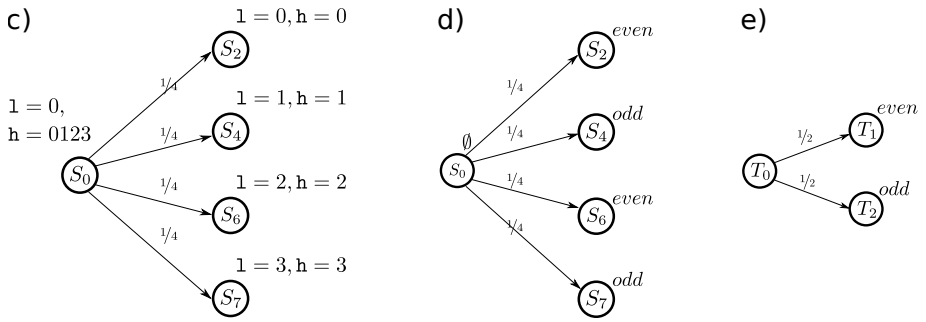


Figure 4.4: Implicit flow example. c) After hiding d) Relabeling e) Quotient

First we apply the prior information $\mathcal{I}_{\mathcal{A}}$. Having a probability distribution over the values of $\mathtt{h}$ allows us to compute the distribution at each state by normalizing the prior distribution on the allowed values of the state. Consequently we can compute the probability for each guard to be true in each state and transform the MDP in the Markov chain $\mathcal{C}$ shown in Fig. 4.3b.

We hide the states in $\mathcal{T} = (S_1, S_3, S_5)$ and redistribute the transition probabilities accordingly between the remaining states, obtaining the reduced model in Fig. 4.4c. Note that since we have hidden all internal states what is left is just a single step distribution from the starting state to the final states.

We want to compute the attacker's quotient according to the discrimination relation $\mathcal{R}_{\mathcal{A}}$. It has three equivalence classes, corresponding to the states in which $\mathtt{l}$ is undefined (labeled $\emptyset$), even (labeled *even*) and odd (labeled *odd*). We label the states of the chain according to their equivalence class; this is shown in Fig. 4.4d. Finally we compute the actual quotient, depicted in Fig. 4.4e.

Since the process is deterministic, the leakage is the entropy of the attacker's quotient, in this case 1 bit.

4.3 Channel Capacity of a Specification

Channel capacity is the maximum amount of information that a channel can transfer. In information leakage it is used to compute the maximum amount of information that an attacker can infer among all attackers with the same observational powers but different prior information about the secret.

We want to raise the level of abstraction to reason at the level of the protocol specification. A specification is more general than a process in the sense that it describes the behavior that the system should have but leaves space for a developer to choose how to implement it. For instance, in our models above we have assumed that the secret has a fixed length, known to the attacker. Such assumption is appropriate for a process-attacker scenario, but not general enough to be considered a specification: the implementer should be free to choose the length of the secret to fit his needs.

4.3.1 IMC Model of a Specification

We need different models to model protocol specification, since the Markov chains and MDPs we used until now assume a fixed secret length. For instance, it is evident that

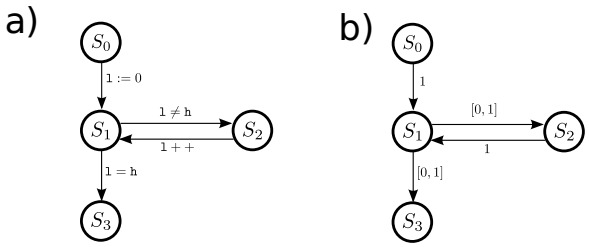


Figure 4.5: Implicit flow specification. a) Control flow graph b) IMC

the size of the MDP model in Fig. 4.4a depends on the size of the secret. How would we specify the behavior of the system independently of the size of the secret?

What the system does is initialize a low-level variable `l` to zero and then increase it until it matches a high-level variable `h`. Its control flow graph is shown in Fig. 4.5a.

The only nondeterministic state is S_1, where the transition depends on the values of `l` and `h`. Alas, we have no information about the values of the two variables in this state, so it is impossible to define transition probabilities for the two transitions. The only information we have is that $P(\texttt{l} = \texttt{h}) + P(\texttt{l} \neq \texttt{h}) = 1$ from probability theory. The actual probabilities will depend on how the implementer implements the specification and on the prior information the attacker knows about the secret variable.

Still, it is possible to obtain interesting results even from such an abstract model of the system. Figure 4.5b shows the specification encoded as an Interval Markov Chain. IMCs are commonly used to model protocol specifications. The transition probabilities are left underspecified to model the fact that some choices about how to implement the system are left open to the implementer.

For an IMC specification there exist an infinite number of Markov chain implementations, as recalled in Chapter 2. Each of the MC implementations represent a scenario, meaning the choice of the underspecified elements of the secret (like the length of the secret) and the choice of an attacker to observe the system. In our subsequent analysis we will obtain results that are valid over all scenarios, thus all possible implementations when observed by all possible attackers. We will assume that the IMCs we work with are coherent a defined in Section 2.2.2, as this does not change the implementations and can be enforced in polynomial time.

Our goal is to compute the channel capacity of the IMC specification, meaning the highest leakage than can be attained by any system-attacker scenario. For simplicity, in this section we will only work with deterministic models. Thus leakage corresponds to entropy, and we want to find the Markov chain implementation with maximal finite entropy among the ones implementing the IMC. We start by determining whether such implementation actually exists, and what does it mean when it doesn't.

4.3.2 Existence of a Maximum Entropy Implementation

Since a specification has an infinite set of possible implementation, it is possible that an implementation maximizing entropy does not exist. We provide an algorithm to determine whether a given IMC has a maximum entropy implementation with finite entropy.

The algorithm is based on the concept of end component. In a Markov chain, an end component is a subset of states of the chain such that once one of them is visited then no state outside the subset will ever be visited again [67]. End components are crucial

```
1  Tag all states of S as UNCHECKED;
2  Find the strongly connected components (SCCs) of the IMC (e.g. with Tarjan's
   algorithm) and tag any state not in any SCC as TRANSIENT;
3  repeat
4      foreach SCC C do
5          Select a state s ∈ C tagged UNCHECKED;
6          if ∀t ∈ S\C.P̌_{s,t} = 0 ∧ Σ_{u∈C} .P̂_{s,u} ≥ 1 then
7              Tag s as ENDCOMPONENTSTATE;
8          else
9              C ← C \ {s};
10             Tag s as TRANSIENT;
11             foreach state r ∈ C do
12                 if P̂_{r,s} ≥ 0 then
13                     Tag r as UNCHECKED;
14                 end
15             end
16         end
17     end
18 until all states in any non-empty SCC are tagged as ENDCOMPONENTSTATE;
```

Algorithm 2: Find all maximal end components of an IMC.

to understanding the long-term behavior of Markov chains, since all states outside an end component will be visited only for a finite time [67]. An end component is said to be *maximal* if no other end components include it.

We extend the definition of end components to Interval Markov Chain. Intuitively, an end component of an IMC is a subset of the states of the IMC such that the subset is an end component of at least one of the implementations of the IMC. For the definition of connectedness in IMCs refer to Section 2.2.2; intuitively, two states are connected in an IMC if they are connected in at least one implementation.

Definition 4.3.1. *Given an IMC $\mathcal{I} = (S, s_0, \check{P}, \hat{P})$, a set of states $R \subseteq S$ is an* end component *of $\mathcal{I}$ if the following holds:*

1. *R is strongly connected;*
2. *$\forall s \in R, t \in S \backslash R . \check{P}_{s,t} = 0$;*
3. *$\forall s \in R . \sum_{u \in R} \hat{P}_{s,u} \geq 1$.*

Corollary 4.3.2. *For an IMC $\mathcal{I} = (S, s_0, \check{P}, \hat{P})$, if $R \subseteq S$ is an end component of $\mathcal{I}$ then there is an implementation of $\mathcal{I}$ in which $\mathbf{P}(X_{n+1} \notin R \mid X_n \in R) = 0$.*

While finding the end components of a Markov chain can be easily accomplished by Tarjan's algorithm [67], for an IMC the solution is not immediate, since a subset of states may be an end component for some implementations but not for others.

Algorithm 2 finds all maximal end components of a coherent IMC. It first identifies all candidate end-components and their complement—the obviously transient states; then it propagates transient states backwards to their predecessors who cannot avoid reaching them. The predecessors are pruned from the candidate end-components and the procedure is iterated until a fixed point is reached.

Lemma 4.3.3. *Algorithm 2 runs in polynomial time, and upon termination the states of the IMC are tagged as* ENDCOMPONENTSTATE *if they are part of any maximal end component, and tagged as* TRANSIENT *otherwise.*

Remember from Chapter 2 that a state s of an IMC is said to be *deterministic* if there exist a state t such that $\check{P}_{s,t} = 1$, and *stochastic* otherwise. A state is deterministic iff its local entropy equals to zero in any implementation of the IMC.

Now that we have determined which states of the IMC are in an end component, we can determine whether there is a maximum entropy implementation for the IMC. For each state in an end component there is at least an implementation in which the state gets visited an infinite number of times, thus having infinite residence time. If the state can have positive local entropy in one of these implementation then such implementation will have infinite entropy by Corollary 4.2.3, thus clearly no maximum entropy implementation with finite entropy exists.

Note that we have no guarantee that a state in an end component will have positive local entropy and infinite residence time in the same implementation. Consider state S_1 in Fig. 4.5b. State S_1 has infinite residence time iff $P_{S_1,S_2} = 1$, and positive local entropy iff $0 < P_{S_1,S_2} < 1$, so there is no implementation in which both conditions are true. When this happens it is also true that there is no maximum entropy implementation with finite entropy, even though no single implementation has infinite entropy: in fact, the set of attainable entropies is unbounded. We will discuss this in Section 4.3.4.

For now we are interested on characterizing whether a maximum entropy implementation with finite entropy exists, and this is accomplished by the following theorem:

Theorem 4.3.4. *Let $\mathcal{I} = (S, s_0, \check{P}, \hat{P})$ be an IMC and S_ω the union of all its end components. Then $\mathcal{I}$ has no Maximum Entropy implementation iff a state $s \in S_\omega$ is stochastic.*

Since Algorithm 2 is polynomial, the condition in Theorem 4.3.4 can be verified in polynomial time.

4.3.3 Numerical Synthesis of a Maximum Entropy Implementation

If after verifying the condition in Theorem 4.3.4 we know that a maximum entropy implementation with finite entropy exists, we conclude that its entropy is the channel capacity of the specification, so we want to compute it.

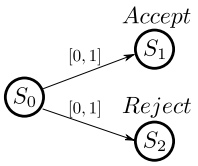


Figure 4.6: Simple authentication IMC

Consider the simple authentication protocol of Fig. 1.1: the user inputs a password, and he gets accepted if the password corresponds to the secret held by the system and rejected otherwise. The probability that the password will get accepted depends on the size of the secret, i.e. on the number of possible passwords, and on the information that the attacker has about the secret. In the example of Fig. 1.1 the probabilities are computed assuming that the password is 64 bit long and the attacker has no prior knowledge over it. In the specification we do not want to fix any of these details, so we leave the probabilities undefined. The IMC model for this specification is shown in Fig. 4.6.

The maximum entropy implementation exists and is the one in which $P_{S_0,S_1} = P_{S_0,S_2} = 0.5$, with 1 bit of entropy. This proves that in no implementation and for

no attacker can this protocol specification leak more than 1 bit of information to the attacker.

Note that this does not guarantee that there is a single scenario with this leakage. In this example the leakage is 1 bit both if the secret is 1 bit and the attacker has no knowledge of it, and if the secret is 64-bit long and the attacker knows 63 of them.

Computing a maximum entropy implementation in general is not an easy task. Linear optimization algorithms like Bellman equations cannot be applied since the maximum entropy solution is in general not positional, as witnessed by the simple authentication example. We propose to use a numerical method to approximate it to an arbitrary precision.

Consider the transition probability $P_{s,t}$ for each pair of states s, t as a dimension in a polytope. Allow it to take values in the interval $[\check{P_{s,t}}, \hat{P_{s,t}}]$. We have produced a convex polytope with a quadratic number of dimensions in the size of the IMC. We add the constraints $\forall s \in S.\ \sum_{t \in S} P_{s,t} = 1$ to make sure that every points in the polytope corresponds to a Markov chain. Since the constraints are linear, the domain remains a convex polytope.

Note that since the IMC is coherent, the polytope will be nonempty. Apply a numerical method to maximize the entropy objective function with an arbitrary precision. Once a global maximum is found, the parameters $P_{s,t}$ interpreted as a Markov chain give a maximum entropy implementation.

4.3.4 Infinite and Unbounded Entropy Implementation

We have shown that there are two different cases in which no maximum entropy implementation with finite entropy exists: when there are implementations with infinite entropy, and when every implementation has finite entropy but the set of attainable entropies is unbounded.

We can discriminate the infinite case from the unbounded case with a simple polynomial-time procedure. Consider the IMC to characterize. For each of its end components $\mathcal{R}$ and each pair of states $s \in \mathcal{R}, t \notin \mathcal{R}$ set $\hat{P}_{s,t} = 0$, then make the IMC coherent again with the coherence algorithm. This way we restrict the implementations to only the ones in which all connected components that can be closed are closed. Let S_ω be the union of all end components in the resulting IMC. If all states in S_ω are deterministic then we are in the unbounded case, otherwise we are in the infinite case. The idea is that since we forced all potentially recurrent states to be recurrent, if this forces them to be deterministic it means that there is no implementation in which they are both recurrent and non-deterministic.

Consider the infinite case, i.e. the specification allows for implementations with infinite entropy. Since we know from Section 4.2 that infinite entropy characterizes non-terminating processes, it means that some of the implementations of the specification are not guaranteed to terminate. Since the processes are deterministic, this means that the protocol depends and reveals an infinite amount of information, which is not possible with a finite secret. Thus either some implementations are non-terminating or they depend on an infinite secret; in both cases we cannot give guarantees on the amount of secret information leakage, thus the specification should be restricted to exclude these implementations.

Consider the unbounded case, i.e. every implementation has finite entropy but there is no maximum. Recall that the IMC specification in Fig. 4.5b is one such case. In these cases the unboundedness comes from the fact that the amount of information revealed is proportional to the size of the secret, and thus tends to infinity as the secret grows. In the

example in Fig. 4.5b the whole secret is always revealed via implicit flow, so for every implementation there is one with a larger secret that leaks more. These specifications should also be restricted to exclude implementations with unbounded entropy, as we are not able to give a security guarantee for the protocol.

4.4 Bibliographic Note

The modeling of protocols and specifications with Markovian models has been proposed in Paper A, limited to probabilistic models of deterministic processes, and formalized and extended to randomized protocols in Paper B. The rest of the chapter, including entropy computation for Markov chains and finding the maximum entropy implementation of a specification, is an extension of material presented in Paper A. Discussion about using MDPs for specification models instead of IMCs is also found in Paper A, together with some examples and proofs of all theorems and lemmata.

Chapter 5

Information Leakage of Non-terminating Processes

> *The idea of eternity is an illness of the spirit.*
>
> Lev Nikolàievič Tolstòj, cited in Maksim Gorkij, *"Tales of Unrequited Love", 1923*

The entropy of a Markov chain is finite if and only if the local entropy of its recurrent states is zero, as shown in Section 4.2.1. Absorbing Markov chains model terminating processes. In Chapter 4 we have shown how to compute the information leakage for terminating processes.

Non-terminating processes are modeled by non-absorbing Markov chain, thus having potentially infinite entropy. Nonetheless, it is not necessarily the case that the leakage of a non-terminating process is infinite. The finiteness of leakage depends on whether the secret is finite and whether the observation is finite. Only if both the secret and the observation are infinite can the leakage be infinite.

If the leakage of a non-terminating process is finite it can be computed. We show a procedure to compute it in Section 5.2. If the leakage is infinite it can still be interesting to know what is the average amount of leakage per time unit, known as *leakage rate*. We show how to compute it in Section 5.3. In Section 5.4.1 we provide an algorithm to compute amount of information that the system leaks in a given time. In Section 5.4.2 we prove that computing the amount of time required to leak a given amount of information is hard by reduction to Skolem's problem.

This chapter contains unpublished research developed with Andrzej Wąsowski, Axel Legay, Pasquale Malacaria and Bo Friis Nielsen.

Consider the snippet of imperative code in Fig. 5.2. There is a secret bit `h`. If `h` is 0 the program produces an infinite string of zeroes and ones with the same probability 0.5. If `h` is 1 the program also produces a string of zeroes and ones, but the probability that it will produce a zero is 0.75. An attacker may be able to observe this infinite string and infer information about the secret by studying the frequencies of zeroes and ones. Note that this program cannot be encoded as a finite channel matrix, as it has an infinite amount of possible outputs. This further shows the expressive power

Secret size	Observation length	Leakage	Leakage rate	Reference
Finite	Finite	Finite	0	Chapter 4
Finite	Infinite	Finite	0	Section 5.2
Infinite	Finite	Finite	0	Section 5.2
Infinite	Infinite	Pot. infinite	Finite	Section 5.3

Figure 5.1: Leakage of various process topologies

```
1  secret int1 h; // bit h is the secret
2  if (h==0) then
3    observable int1 o:=0; // bit o is observable
4    while (0==0) do
5      random o:=randombit(0.5);
6    od
7  else
8    observable int1 o:=0; // bit o is observable
9    while (0==0) do
10     random o:=randombit(0.75);
11   od
12 fi
13 return;
```

Figure 5.2: A non-terminating leaking program.

of the Markovian modeling we propose, as it is able to model both terminating and non-terminating processes.

In Chapter 4 non-terminating behaviour was just considered as a single observable "divergence" state. Here we want to refine this simplification and quantify what the attacker can infer about the secret by observing the infinite behavior of the system.

We distinguish four possible scenarios, according to whether the observation by the attacker is finite or infinite and whether the secret itself is finite or infinite. The case with finite observation over a program depending on a finite secret is the terminating case we considered in Chapter 4, while the others will be considered in this chapter.

When both observation and secret are infinite, we can compute the rate of leakage, i.e. the amount of information leaked for each observation. Much like we define information leakage in terms of mutual information, we will define information leakage rate in terms of mutual information rate. Intuitively, this quantifies the average amount of information the attacker infers for each observation over an infinite time. This is presented in Section 5.3. When only one of observation or secret is finite the leakage is finite but cannot be computed using the method of in Chapter 4, thus we provide a new technique in Section 5.2.

The cases are summarized in Fig. 5.1. In all cases we start the analysis by computing the Markov chain modeling the attacker-system scenario, first by using the MDP semantics as explained in Section 4.1.2, then modeling the attacker as explained in Section 4.1.3 and applying the prior information as explained in Step A of Section 4.1.4

5.1 Finiteness of Leakage

The Markov chain semantics of the system models the joint behavior of all variables. To compute information leakage we are only interested in the secret and observable variables, so we can restrict to them only. We will assume for simplicity that the system has a single secret variable $\mathtt{h}$ with uniform prior distribution and a single observable variable $\mathtt{o}$, but the procedure does not change for multiple secret or observable variables. We remark that, even though the attacker can perform multiple observations, we do not model the case in which the attacker actually interacts with the system. In such case directed information would have to be used as the leakage metric instead of mutual information. We refer to Alvim et al. [4] for the details.

The behavior of the secret variable $\mathtt{h}$ is modeled by the secret's marginal process $\mathcal{C}_{|\mathtt{h}}$, and similarly the behavior of $\mathtt{o}$ is modeled by the observer's marginal process $\mathcal{C}_{|\mathtt{o}}$ and the joint behavior of the two variables by the joint marginal process $\mathcal{C}_{|\mathtt{o},\mathtt{h}}$. The following lemma proves that such marginals have a definite entropy value (eventually infinite). This is important because we will have to actually compute such entropies later.

Lemma 5.1.1. *Let $\mathcal{C} = (S, s_o, P)$ be a Markov chain and $\mathtt{v} \in \mathtt{V}$ a variable. Then $H(\mathcal{C}_{|\mathtt{v}})$ exists, either converging to a non-negative real number or diverging to positive infinity.*

Proof. Since $H(\mathcal{C}_{|\mathtt{v}}) = \lim_{k\to\infty} H(\mathtt{v}_1, \mathtt{v}_2, ..., \mathtt{v}_k)$ and $H(\mathtt{v}_1, \mathtt{v}_2, ..., \mathtt{v}_k)$ is a monotonic non-decreasing non-negative sequence on k. □

We write $H(\mathcal{C}_{|\mathtt{v}}) < \infty$ if the limit converges, $H(\mathcal{C}_{|\mathtt{v}}) = \infty$ otherwise. Since leakage is computed as the mutual information $I(\mathcal{C}_{|\mathtt{o}}; \mathcal{C}_{|\mathtt{h}}) = \lim_{k\to\infty} I(\mathtt{o}_1, ..., \mathtt{o}_k; \mathtt{h}_1, ..., \mathtt{h}_k)$ and the sequence $I(\mathtt{o}_1, ..., \mathtt{o}_k; \mathtt{h}_1, ..., \mathtt{h}_k)$ is non-negative and monotonic non-decreasing in k, then leakage can also diverge to positive infinity.

Lemma 5.1.2. *Let $\mathcal{C} = (S, s_o, P)$ be a Markov chain and $\mathtt{o}, \mathtt{h} \in \mathtt{V}$ two variables. Then $I(\mathcal{C}_{|\mathtt{o}}; \mathcal{C}_{|\mathtt{h}})$ exists, either converging to a non-negative real number or diverging to positive infinity.*

Proof. Since $I(\mathcal{C}_{|\mathtt{o}}; \mathcal{C}_{|\mathtt{h}}) = \lim_{k\to\infty} I(\mathtt{o}_1, ..., \mathtt{o}_k; \mathtt{h}_1, ..., \mathtt{h}_k)$ and $I(\mathtt{o}_1, ..., \mathtt{o}_k; \mathtt{h}_1, ..., \mathtt{h}_k)$ is a monotonic non-decreasing non-negative sequence on k. □

We write $I(\mathcal{C}_{|\mathtt{o}}; \mathcal{C}_{|\mathtt{h}}) < \infty$ if the limit converges, $I(\mathcal{C}_{|\mathtt{o}}; \mathcal{C}_{|\mathtt{h}}) = \infty$ otherwise. Chothia et al. proved that leakage is finite when the observation is infinite and the secret is finite in the point-to-point leakage setting [32, Theorem 1]. We state a similar result in our setting by showing that the leakage is finite when either the secret or the observation are finite:

Lemma 5.1.3. *Let $\mathcal{C} = (S, s_0, P)$ be a Markov chain with secrets and observations and $\mathcal{C}_{|\mathtt{o}}$ and $\mathcal{C}_{|\mathtt{h}}$ its marginals on the observable and secret variables, respectively. Then*

$$H(\mathcal{C}_{|\mathtt{o}}) < \infty \vee H(\mathcal{C}_{|\mathtt{h}}) < \infty \Rightarrow I(\mathcal{C}_{|\mathtt{o}}, \mathcal{C}_{|\mathtt{h}}) < \infty$$

Proof. Recall that both $H(\mathtt{o}_1, \mathtt{o}_2, ..., \mathtt{o}_k)$ and $H(\mathtt{h}_1, \mathtt{h}_2, ..., \mathtt{h}_k)$ are monotonic non-decreasing non-negative sequences on k. Then if any of the two sequences converge, then the sequence $\min(H(\mathtt{o}_1, \mathtt{o}_2, ..., \mathtt{o}_k), H(\mathtt{h}_1, \mathtt{h}_2, ..., \mathtt{h}_k))$ is also a convergent monotonic non-decreasing non-negative sequence on k. Since $I(\mathtt{o}_1, ..., \mathtt{o}_k; \mathtt{h}_1, ..., \mathtt{h}_k)$ is also a monotonic non-decreasing non-negative sequence on k and $\forall k \in \mathbb{N}$.

$$I(\mathtt{o}_1, \mathtt{o}_2, ..., \mathtt{o}_k; \mathtt{h}_1, \mathtt{h}_2, ..., \mathtt{h}_k) \leq \min(H(\mathtt{o}_1, \mathtt{o}_2, ..., \mathtt{o}_k), H(\mathtt{h}_1, \mathtt{h}_2, ..., \mathtt{h}_k))$$

then the sequence $I(\mathtt{o}_1, \mathtt{o}_2, ..., \mathtt{o}_k; \mathtt{h}_1, \mathtt{h}_2, ..., \mathtt{h}_k)$ converges also. □

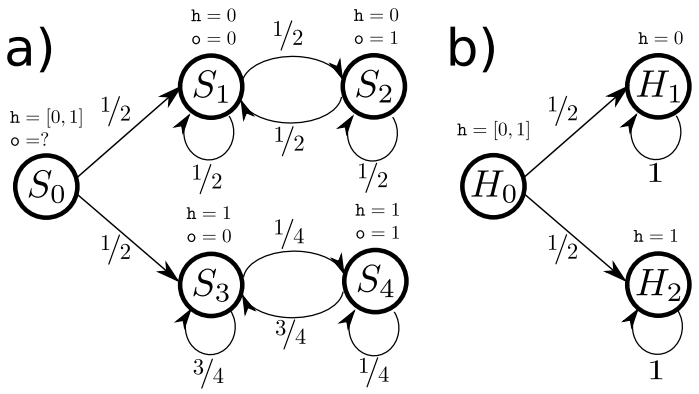


Figure 5.3: Markov chains semantics for the example in Fig. 5.2.

Intuitively, if $H(\mathcal{C}_{|\mathtt{o}}) < \infty$ and $H(\mathcal{C}_{|\mathtt{h}}) = \infty$ then there is an infinite number of secret bits but only a finite amount of observations we can analyze. In the opposite case where $H(\mathcal{C}_{|\mathtt{o}}) = \infty$ and $H(\mathcal{C}_{|\mathtt{h}}) < \infty$ we can analyze an infinite number of observations, but there is only a finite amount of secret bits to be discovered.

If either $H(\mathcal{C}_{|\mathtt{o}}) = \infty$ or $H(\mathcal{C}_{|\mathtt{h}}) = \infty$ then $H(\mathcal{C}_{|\mathtt{o},\mathtt{h}}) = \infty$, since $H(X, Y) \geq \max(H(X), H(Y))$. It follows that if either the observation or the secret are infinite, the formula $I(\mathcal{C}_{|\mathtt{o}}, \mathcal{C}_{|\mathtt{h}}) = H(\mathcal{C}_{|\mathtt{o}}) + H(\mathcal{C}_{|\mathtt{h}}) - H(\mathcal{C}_{|\mathtt{o},\mathtt{h}})$ will produce an indeterminate form $\infty - \infty$ and thus cannot be directly used to compute the leakage. Nonetheless, in both cases leakage has a finite value by Lemma 5.1.3.

5.2 Non-terminating Processes with Finite Leakage

To compute the value of the leakage, consider that the entropies of the marginals are limit computations, since $H(\mathcal{C}_{|\mathtt{v}}) = \lim_{k\to\infty} H(\mathtt{v}_1, ..., \mathtt{v}_k)$. This allows us to compute mutual information using the limit of the entropies of the marginal processes:

$$\begin{aligned} I(\mathcal{C}_{|\mathtt{o}}; \mathcal{C}_{|\mathtt{h}}) &= \lim_{k\to\infty} I(\mathtt{o}_1, ..., \mathtt{o}_k; \mathtt{h}_1, ..., \mathtt{h}_k) \\ &= \lim_{k\to\infty} (H(\mathtt{o}_1, ..., \mathtt{o}_k) + H(\mathtt{h}_1, ..., \mathtt{h}_k) - H((\mathtt{o}, \mathtt{h})_1, ..., (\mathtt{o}, \mathtt{h})_k)) \end{aligned}$$

The limit above computes information leakage in any case, but is not always the most efficient option available. When it is known that the secret (resp. observation) is finite, it is more efficient to use the formula $I(\mathcal{C}_{|\mathtt{o}}; \mathcal{C}_{|\mathtt{h}}) = H(\mathcal{C}_{|\mathtt{h}}) - H(\mathcal{C}_{|\mathtt{h}}|\mathcal{C}_{|\mathtt{o}})$ (resp. $I(\mathcal{C}_{|\mathtt{o}}; \mathcal{C}_{|\mathtt{h}}) = H(\mathcal{C}_{|\mathtt{o}}) - H(\mathcal{C}_{|\mathtt{o}}|\mathcal{C}_{|\mathtt{h}})$). Also, in some cases the marginals may actually be Markov chains. In these cases it is possible to compute their entropy in polynomial time in the size of the chain [14], thus avoiding the computation of one of the limits above.

5.2.1 Solved Example: a Non-terminating Program on a Finite Secret

We now solve a case in which the secret is finite and Markovian and the observation infinite. Consider again the program in Fig. 5.2 with a secret bit $\mathtt{h}$. If $\mathtt{h}$ is 0 the program produces an infinite string of zeroes and ones with the same probability 0.5, starting

a)

		$\mathtt{o}_1$	
		0	**1**
$\mathtt{h}$	**0**	$1/2$	0
	1	$1/2$	0

b)

		$\mathtt{o}_1\mathtt{o}_2$			
		00	**01**	**10**	**11**
$\mathtt{h}$	**0**	$1/4$	$1/4$	0	0
	1	$3/8$	$1/8$	0	0

c)

		$\mathtt{o}_1\mathtt{o}_2\mathtt{o}_3$			
		000	**001**	**010**	**011**
$\mathtt{h}$	**0**	$1/8$	$1/8$	$1/8$	$1/8$
	1	$9/32$	$3/32$	$3/32$	$1/32$

d)

		$\mathtt{o}_1\mathtt{o}_2\ldots\mathtt{o}_{k-1}\mathtt{o}_k$					
		$\overbrace{\mathbf{00..00}}^{k}$	$\overbrace{\mathbf{00..01}}^{k}$	$\overbrace{\mathbf{00..10}}^{k}$	$\cdots$	$\overbrace{\mathbf{01..10}}^{k}$	$\overbrace{\mathbf{01..11}}^{k}$
$\mathtt{h}$	**0**	$1/2^k$	$1/2^k$	$1/2^k$	$\cdots$	$1/2^k$	$1/2^k$
	1	$\frac{(3/4)^{k-1}}{2}$	$\frac{(3/4)^{k-2}\cdot 1/4}{2}$	$\frac{(3/4)^{k-2}\cdot 1/4}{2}$	$\cdots$	$\frac{3/4\cdot(1/4)^{k-2}}{2}$	$\frac{(1/4)^{k-1}}{2}$

Figure 5.4: Non-terminating example: joint distribution of the secret $\mathtt{h}$ and observables $\mathtt{o}_k$: a) for 1 step; b) for 2 steps; c) for 3 steps; d) for k steps.

with a zero. If $\mathtt{h}$ is 1 the program also produces a string of zeroes and ones starting with a zero, but the probability that it will produce a zero is 0.75.

An attacker may be able to observe this infinite string and infer information about the secret by studying the frequencies of zeroes and ones. Reasonably, an attacker observing the output for an infinite time would be able to decide whether the frequency of zeroes is 0.5 or 0.75 and infer the value of $\mathtt{h}$ consequently.

The Markov chain semantics for it is shown in Fig. 5.3a.

Since conveniently $\mathtt{h}_1 = \mathtt{h}_2 = \mathtt{h}_3 = ...$ we will just call it $\mathtt{h}$. The behavior of $\mathtt{h}$ is modeled by the Markov chain in Fig. 5.3b, and its entropy is $H(\mathtt{h}) = 1$ bit. Then $H(\mathcal{C}_{|\mathtt{h}}|\mathcal{C}_{|\mathtt{o}})$ corresponds to

$$\lim_{k\to\infty} H(\mathtt{h}|\mathtt{o}_1, ..., \mathtt{o}_k)$$

We compute $H(\mathtt{h}|\mathtt{o}_1, ..., \mathtt{o}_k)$ for $k \to \infty$. Note that at time 1 $\mathtt{o}$ is always 0, then it changes randomly depending on the value of $\mathtt{h}$. We will write down the joint distribution of $\mathtt{h}$ and $\mathtt{o}$ as a function of k and use it to compute the marginal over $\mathtt{o}$ and finally the conditional entropy.

The joint distribution of $\mathtt{h}$ and $\mathtt{o}$ is shown in Fig. 5.4. For compactness we do not represent the cases with probability 0 in Fig. 5.4cd, i.e. all the cases in which $\mathtt{o}_1 = 1$.

Now let $w^k \in \{0,1\}^k$ be a sequence of k bits. Consider the formula for conditional entropy:

$$H(\mathtt{h}|\mathtt{o}_1, ..., \mathtt{o}_k) = \sum_{w^k\in\{0,1\}^k} P(\mathtt{o}_1, ..., \mathtt{o}_k = w^k)H(\mathtt{h}|\mathtt{o}_1, ..., \mathtt{o}_k = w^k))$$

In our case it holds that

$$H(\mathtt{h}|\mathtt{o}) = \lim_{k\to\infty} H(\mathtt{h}|\mathtt{o}_1, ..., \mathtt{o}_k) = 0$$

thus

$$I(\mathtt{o};\mathtt{h}) = H(\mathtt{h}) - H(\mathtt{h}|\mathtt{o}) = 1 - 0 = 1 \text{ bit}$$

The leakage of the program in Fig. 5.2 is 1 bit, proving that an attacker able to analyze the bit streams produced by the system will eventually learn the value of the secret h with an arbitrary confidence. Note that this considers an attacker able to observe the system for an infinite amount of time.

More importantly, note that the marginal distribution $\mathcal{C}_{|\mathtt{o}}$ of o is *not* a Markov chain. This depends on the fact that the joint distribution depends also on the information that the attacker has about h, so while the attacker gathers information about o and h the joint distribution changes and thus the marginal distribution of o changes also. Nonetheless, the marginal distribution is a sum of Markovian processes, and thus can be represented in a closed form like the one in Fig. 5.4d.

In Section 5.3 we study the average leakage per time unit, and show that it is a useful measure only if both the secret and the observation are infinite. When either is finite it is more interesting to ask how long will it take for the attacker to gather enough information to convince himself of the value of the secret, or, similarly, how much leakage we can expect from this program in a given amount of time. Both questions are investigated in Section 5.4.

5.3 Leakage Rate of a Markov Chain

In the case in which $H(\mathcal{C}_{|\mathtt{o}}) = \infty$ and $H(\mathcal{C}_{|h}) = \infty$, i.e. when the secret is an infinite number of bits and the observer can observe it for an infinite time, then the leakage $I(\mathtt{o},\mathtt{h})$ can be infinite. In this case we do not try to calculate it, but instead we calculate how much information the process leaks for each time step. This quantity is known as *leakage rate*, and corresponds to the *mutual information rate* of the secret and observable. Mutual information rate is introduced by Shannon as *rate of actual transmission* [82] and studied in the fields of data transmission and dynamical systems [7, 9].

Note that the computation of leakage as a rate over time assumes that the attacker is able to keep track of the discrete time, so in this section we will assume that every constant-time operation takes 1 time step. This can be equivalently stated as saying that all transitions between states of the Markov chain semantics represent observable steps.

To compute it, encode the process-attacker scenario with a Markov chain as shown in Section 4.1.2 and compute the joint, secret and attacker's marginal. Then we can use the marginals to compute leakage rate by applying the following definition:

Definition 5.3.1. *Let $\mathcal{C} = (S, s_0, P)$ be a Markov chain and $\mathcal{C}_{|\mathtt{o},\mathtt{h}}$, $\mathcal{C}_{|\mathtt{o}}$ and $\mathcal{C}_h$ its marginals on* `(o,h)`*,* o *and* h *respectively. Then the* leakage rate $\bar{I}$ *is defined as*

$$\bar{I}(\mathcal{C}_{|\mathtt{o}};\mathcal{C}_h) = \bar{H}(\mathcal{C}_{|\mathtt{o}}) + \bar{H}(\mathcal{C}_{|\mathtt{h}}) - \bar{H}(\mathcal{C}_{|\mathtt{o},\mathtt{h}})$$

Leakage rate can also be computed as a limit, since

$$\begin{aligned}\bar{I}(\mathcal{C}_{|\mathtt{o}};\mathcal{C}_{|\mathtt{h}}) &= \lim_{k\to\infty} \frac{I(\mathtt{o}_1,...,\mathtt{o}_k;\mathtt{h}_1,...,\mathtt{h}_k)}{k} \\ &= \lim_{k\to\infty} \frac{(H(\mathtt{o}_1,...,\mathtt{o}_k) + H(\mathtt{h}_1,...,\mathtt{h}_k) - H((\mathtt{o},\mathtt{h})_1,...,(\mathtt{o},\mathtt{h})_k))}{k}\end{aligned}$$

when the limit exists.

Data: A Markov Chain $\mathcal{C} = (S, s_0, P)$, its set $T \subseteq S$ of transient states, one of its closed communicating classes R_i.
Result: The expected residence time rate $\bar{\xi}_r$ for each state $r \in R_i$

Let Q_{s,R_i} be the probability of eventually visiting R_i from state $s \in S$;
foreach $s \in S \setminus T, R_i$ **do**
  $Q_{s,R_i} \leftarrow 0$;
end
foreach $r \in R_i$ **do**
  $Q_{r,R_i} \leftarrow 1$;
end
Let E be a system of linear equations;
foreach $t \in T$ **do**
  Add to E the equation $Q_{t,R_i} = \sum_{r \in R_i} P_{t,r} + \sum_{u \in T} P_{t,u} Q_{u,R_i}$;
end
Solve the system E to obtain Q_{t,R_i} for each $t \in T$;
Let $\pi^{(\infty)}_{R_i} = \sum_{s \in S} \pi^{(0)}_s Q_{s,R_i}$ be the probability that the Markov chain will eventually visit R_i;
Let E' be a system of linear equations;
foreach $r \in R_i$ **do**
  Add to E' the equation $\bar{\xi}_r = \sum_{r' \in R_i} \bar{\xi}_{r'} P_{r',r}$;
end
Solve the system E' under the condition $\sum_{r \in R_i} \bar{\xi}_r = \pi^{(\infty)}_{R_i}$ to obtain $\bar{\xi}_r$ for each $r \in R_i$

Algorithm 3: Compute the expected residence time rate for all the states of a closed communicating class of a Markov chain.

Generally both entropy and leakage rate could be infinite, for instance for a program that leaks 1 bit at time 1, 2 bits at time 2, and so on, the leakage rate would be infinite. Since we postulated that there exists a very large but finite maximum size M for the variables declared in the system, it is impossible to declare an unbounded amount of secret or observable bits on each step of the program execution. We do not think that this restriction limits significantly the programs that can be analyzed, while guaranteeing that the entropy and leakage rate do not diverge to positive infinity is a significantly useful result. Both entropy rate and leakage rate may still oscillate, even though since they are defined in terms of Cesàro limits this happens only in pathological cases. We do not expect these cases to be common in normal secret-dependent systems, and leave finding a meaningful measure of leakage for this cases an open problem. This reflects similar issues in related definitions of leakage rate [32, 60].

A case in which both entropy and leakage rate exist is when the marginal processes modeling the behavior of the observable and secret variables are both Markovian. Intuitively, this happens when the secret gets periodically replaced with a new one, and thus the information the attacker has on it is reset to the prior information. We will show this with an example in Section 5.3.1.

When any of the marginal is a Markov chain it is possible to compute its entropy more efficiently. We provide an algorithm to compute the entropy rate of a Markov Chain $\mathcal{C} = (S, s_0, P)$. Note that the algorithm terminates only on finite state Markov chains, so it cannot be applied to non-Markovian processes, like non-Markovian marginals. The

Data: A Markov Chain $\mathcal{C} = (S, s_0, P)$, its closed communicating classes $R_i, ..., R_k$, the expected residence time rate $\bar{\xi}_s$ of each state $s \in S$.
Result: The entropy rate $\bar{H}(\mathcal{C})$

$\bar{H}(\mathcal{C}) \leftarrow 0$;
foreach *closed communicating class* R_i **do**
  foreach $s \in R_i$ **do**
    $L(s) \leftarrow -\sum_{t \in R_i} P_{s,t} \log P_{s,t}$;
    $\bar{H}(\mathcal{C}) \leftarrow \bar{H}(\mathcal{C}) + \bar{\xi}(s) L(s)$;
  end
end

Algorithm 4: Compute the entropy rate of a MC.

algorithm uses the concept of closed communicating classes (CCC) of a MC, introduced in Definition 2.2.1. The entropy rate of a Markov chain with a given initial probability distribution exists and is unique [36].

Let $R_1, ..., R_k$ be the distinct closed communicating classes of $\mathcal{C}$ and T the set of states not in any closed communicating class. For each state $t \in S$ let $\pi_t^{(0)}$ be its starting probability. To compute entropy rate we will also define the expected residence time rate in a state of the Markov chain:

Definition 5.3.2. *Let $\mathcal{C} = (S, s_0, P)$ be a Markov chain. Then for each state $s \in S$ we define the* expected residence time rate $\bar{\xi}_s$ *of s as*

$$\bar{\xi}_s = \lim_{k \to \infty} \frac{\sum_{i=1}^{k} P_{s_0,s}^i}{k}$$

Algorithm 3 computes the expected residence time rate of each state of a closed communicating class of a Markov chain. Its correctness and worst-case time complexity are established in Theorem 5.3.3. It can be ran for each CCC to obtain $\bar{\xi}_s$ for each state $s \in S \setminus T$, which is sufficient since $\bar{\xi}_t$ is zero for each state $t \in T$.

Theorem 5.3.3. *Algorithm 3 terminates in polynomial time in $|S|$ and when it does it returns the expected residence time rate $\bar{\xi}_r$ for each $r \in R_i$.*

Proof. For each state $s \in S$ let

$$Q_{s,R_i} = \lim_{n \to \infty} P(X_n \in R_i | X_0 = s)$$

Then it holds that

$$\pi_{R_i}^{(\infty)} = \lim_{n \to \infty} P(X_n \in R_i) = \sum_{s \in S} \pi_s^{(0)} Q_{s,R_i}$$

We start by calculating Q_{s,R_i} for each state $s \in S$. Recall that $\forall n.\ P(X_n \notin R_i | X_{n-1} \in R_i) = 0$ by Def. 2.2.1. Then clearly $\forall r \in R_i.\ Q_{r,R_i} = 1$ and $\forall s \in S \setminus T, R_i.\ Q_{s,R_i} = 0$. It remains to calculate Q_{t,R_i} for the transient states $t \in T$. This

is accomplished by noting that

$$\begin{aligned}
Q_{t,R_i} &= \lim_{n\to\infty} P(X_n \in R_i | X_0 = s) && \text{(by def. of Q)} \\
&= \lim_{n\to\infty} \sum_{r\in R_i} P^n_{s,r} && \text{(by Kolmogorov-Chapman equations)} \\
&= \sum_{u\in S} P_{t,u} \lim_{n\to\infty} \sum_{r\in R_i} P^{n-1}_{u,r} && \text{(extruding the first element of the sum)} \\
&= \sum_{u\in S} P_{t,u} Q_{u,R_i} && \text{(by def. of Q)}
\end{aligned}$$

producing a system of $O(|S|)$ linear equation. Solving it allows to compute Q_{t,R_i} for the transient states $t \in T$, giving us all Q_{s,R_i} and thus allowing us to compute $\pi^{(\infty)}_{R_i}$. Now consider the stationary distribution μ of the chain. Since the chain is not necessarily ergodic then μ depends on $\pi^{(0)}$, but it is unique given $\pi^{(0)}$. Also since each R_i is irreducible we can compute the stationary distribution of each of them independently under the condition that $\sum_{r\in R_i} \mu_r = \pi^{(\infty)}_{R_i}$ by solving the system of equations

$$\forall r \in R_i.\ \bar{\xi}_r = \sum_{r'\in R_i} \bar{\xi}_{r'} P_{r',r}$$

Note that since $\mu_s = \lim_{k\to\infty} P^k_{s_0,s}$ then

$$\bar{\xi}_s = \lim_{k\to\infty} \frac{\sum_{i=1}^{k} P^i_{s_0,s}}{k} \to \mu_s$$

by Cesàro limit theorem, thus $\forall s \in S.\ \mu_s = \bar{\xi}_s$ completing the proof of correctness.

For the time complexity, we have to solve $O(|S|)$ systems of $O(|S|)$ equations twice. Karmarkar's algorithm solves linear programming problems in time $\sim O(n^{3.5})$, thus the final complexity is $\sim O(|S|^{4.5})$.

□

Algorithm 4 uses the expected residence time rates to compute the entropy rate of $\mathcal{C}$, according to the formula $\bar{H}(\mathcal{C}) = \sum_{s\in S} L(s)\bar{\xi}_s$. We start by proving that the formula is correct.

Lemma 5.3.4. *For an MC $\mathcal{C} = (S, s_0, P)$ it holds that $\bar{H}(\mathcal{C}) = \sum_{s\in S} L(s)\bar{\xi}_s$.*

Proof. Consider the chain rule for conditional entropy:

$$\begin{aligned}
&H(X_k | X_1, X_2, ..., X_{k-1}) = \\
&= -\sum_{t\in S} \sum_{s_0\in S} \cdots \sum_{s_{n-1}\in S} P(X_k = t, X_0 = s_0, ..., X_{k-1} = s_{k-1}) \cdot \\
&\cdot \log(P(X_k = t | X_1 = s_1, ..., X_{k-1} = s_{k-1})) \\
&= \sum_{s\in S} \pi^{(k-1)}_s H(X_k | X_{k-1} = s) = \sum_{s\in S} \pi^{(k-1)}_s L(s)
\end{aligned}$$

Where $\pi^{(0)}_s$ is 1 if $s = s_0$ and 0 otherwise. We can see that the entropy at time k is the sum on each state of the local entropy of the state multiplied by the probability to be in

that state at time $k-1$. Applying this to the chain rule for joint entropy over an infinite time we have that

$$
\begin{aligned}
\bar{H}(\mathcal{C}) &= \lim_{k\to\infty} \frac{1}{k} H(X_0, X_1, ..., X_k) && \text{(by def. 2.3.7)} \\
&= \lim_{k\to\infty} \frac{1}{k} \left(H(X_0) + \sum_{i=1}^{k} H(X_k | X_0, X_1, ..., X_{k-1}) \right) \\
& && \text{(by chain rule of entropy)} \\
&= \lim_{k\to\infty} \frac{1}{k} \left(0 + \sum_{i=1}^{k} H(X_k | X_0, X_1, ..., X_{k-1}) \right) && \text{(since } X_0 \text{ is Dirac)} \\
&= \lim_{k\to\infty} \frac{1}{k} \left(\sum_{i=1}^{k} \sum_{s\in S} \pi_s^{(k-1)} L(s) \right) && \text{(as shown above)} \\
&= \lim_{k\to\infty} \sum_{s\in S} \frac{1}{k} \left(\sum_{i=1}^{k} \pi_s^{(k-1)} L(s) \right) && \text{(since } s \text{ does not depend on } k) \\
&= \sum_{s\in S} \lim_{k\to\infty} \frac{1}{k} \left(\sum_{i=1}^{k} \pi_s^{(k-1)} L(s) \right) && \text{(since limit of sum is sum of limits)} \\
&= \sum_{s\in S} L(s) \lim_{k\to\infty} \frac{1}{k} \sum_{i=1}^{k} \pi_s^{(k)} && \text{(since } L(s) \text{ does not depend on } k) \\
&= \sum_{s\in S} L(s) \lim_{k\to\infty} \frac{\sum_{i=1}^{k} P^i_{s_0,s}}{k} && \text{(by def. of } \pi_s^{(k)}) \\
&= \sum_{s\in S} L(s) \bar{\xi}_s && \text{(by def. 5.3.2)}
\end{aligned}
$$

□

The correctness and worst-case time complexity of Algorithm 4 are established in Theorem 5.3.5. Note that Algorithm 4 is a particular case of an algorithm to compute the expected infinite-horizon reward rate of a reward function defined on the transitions of the Markov chain.

Theorem 5.3.5. *Algorithm 4 terminates in time $O(|S|^2)$ and when it does it outputs the entropy rate $\bar{H}(\mathcal{C})$ of $\mathcal{C}$.*

Proof. Let $L(s) = -\sum_{t\in S} P_{s,t} \log_2 P_{s,t}$ be the local entropy of state s. The algorithm computes the entropy rate of $\mathcal{C}$ as $\bar{H}(\mathcal{C}) = \sum_{s\in S} L(s)\bar{\xi}_s$. The correctness of the formula is shown in Lemma 5.3.4. Computing the local entropy of each state in time $O(|S|^2)$ is trivial. □

Having computed the entropy rates of the joint, secret and attacker's marginals we can apply Definition 5.3.1 to obtain the leakage rate of the system.

5.3.1 Solved Example: Leaking Mix Node Implementation

We show an example of a program leaking an infinite amount of information and we compute its leakage rate using the method described above.

```
1 secret int3 inorder;
2 public int3 rand;
3 observable int3 outorder;
4 while (0==0) do
5   assign inorder := [0,5];
6   random rand := random(0,5);
7   assign outorder := (inorder ^ rand)%6;
8 od
9 return;
```

Figure 5.5: A leaking implementation of a mix node.

A mix node [37] is a program meant to scramble the order in which packages are routed through a network, to increase the anonymity of the sender. Even if the packages are encrypted, some information about the sender could be inferred by observing the order in which they are forwarded. A mix node changes this order to a random one, thus making it harder for an attacker to connect each package to its sender.

A mix node waits until it has accumulated a fixed amount of packages, and then forwards them in a random order. If the exit order of the packages is independent from the entrance order, then no information about the latter can be inferred by observing the former. We will present an implementation of a mix node where the entrance and exit order are not independent and compute the rate of the information leakage.

The implementation of the mix node is shown in Fig. 5.5. This particular node waits until it has accumulated 3 packages and then sends them in a random order. Naming the packages A, B and C there are 6 possible entrance orders: ABC, ACB, BAC, BCA, CAB, and CBA. We will number them from 0 to 5.

		rand					
		0	**1**	**2**	**3**	**4**	**5**
inorder	**0**	0	1	2	3	4	5
	1	1	0	3	2	5	4
	2	2	3	0	1	0	1
	3	3	2	1	0	1	4
	4	4	5	0	1	0	1
	5	5	4	1	4	1	0

Figure 5.6: Exit order as a function of entrance order and random variable.

In line 5 of the code a random number from 0 to 5 is assigned to the secret variable `inorder`, modeling the secret entrance order. Then in line 6 a random value uniformly distributed from 0 to 5 is assigned to the variable `rand`. Finally, in line 7 the bitwise exclusive OR modulo 6 of the variables `inorder` and `rand` is assigned to the observable variable `outorder`, which represents the order in which the packages exit from the mix node and is observable to the attacker. After producing an exit order the mix node receives three more packages in a new entrance order, scrambles them the same way and forwards them, and so on forever.

Assume that the prior distribution over the entrance order is uniform. The resulting probability distribution on the exit order is $P(\texttt{outorder}) = \{0 \mapsto {}^{4}/_{18}, 1 \mapsto {}^{5}/_{18}, 2 \mapsto {}^{2}/_{18}, 3 \mapsto {}^{2}/_{18}, 4 \mapsto {}^{3}/_{18}, 5 \mapsto {}^{2}/_{18}\}$. This depends on the fact that bitwise OR and modulo operations do not preserve distribution uniformity. The value assigned to the exit order in line 7 as a function of the entrance order and random variable is shown in the table in Fig. 5.6.

The Markov chain semantics of the system has more than 300 states, so we do

not reproduce it here. It can be computed and analyzed automatically in less than a second. Assuming that each line of code is executed in one time step, the entropy rates of the marginals are $\bar{H}(\texttt{inorder}) = 0.86165...$ bits, $\bar{H}(\texttt{outorder}) = 0.82766...$ and $\bar{H}(\texttt{inorder}, \texttt{outorder}) = 1.59367...$, giving a leakage rate of $\bar{I}(\texttt{inorder}; \texttt{outorder}) = \bar{H}(\texttt{inorder}) + \bar{H}(\texttt{outorder}) - \bar{H}(\texttt{inorder}, \texttt{outorder}) = 0.86165... + 0.82766... - 1.59367... \approx 0.09564$ bits.

The leakage rate for each time unit is ≈ 0.09564 bits. Since the entropy rate of the secret is ≈ 0.86165 bits, we can conclude that this implementation of a mix node has a rate of leakage $0.09564/0.86165 \approx 11.1\%$ of each of its infinite secrets. Note that the exact value of the leakage rate and of the entropy rate of the secret depend on how many computation step are considered in the loop, but the percentage of leaked secret does not, as changing the size of the loop would only change both values by the same multiplicative factor.

5.4 Bounded Time/Leakage Analysis

We consider two similar bounded approaches to the leakage problem: computing the leakage of a Markov chain within a given time frame, or computing how long it takes for the Markov chain to leak a given amount of information. These approaches are valid when the marginal processes are Markov chains; limit-based techniques have to be developed otherwise.

5.4.1 Bounded Time

We want to compute the leakage for an attacker that is able to observe the behavior of the program for $t < \infty$ time units. We abstract time by considering each time unit as a step in the evolution of the Markov chain modeling the system. The definition of mutual information from a time t_1 to a time $t_2 > t_1$ is as follows:

Definition 5.4.1. *Let X_i and Y_i be two time-indexed probabilistic processes. Then the mutual information between X_i and Y_i from time t_1 to time t_2 steps is*

$$I(X_{t_1}, ..., X_{t_2}; Y_{t_1}, ..., Y_{t_2}) = H(X_{t_1}, ..., X_{t_2}) + H(Y_{t_1}, ..., Y_{t_2}) - \\ - H(X_{t_1}, ..., X_{t_2}, Y_{t_1}, ..., Y_{t_2})$$

We will refer to it as $I^{(t_1,t_2)}(X_i, Y_i)$ for simplicity.

Consider as usual the Markov chain $\mathcal{C}_{O,h}$ of size n encoding the behavior of the system relevant to the secret and the observation. We show an iterated algorithm to compute $I^{(t_1,t_2)}(\mathcal{C}_O, \mathcal{C}_h)$ in time $O(t_2 n^2)$.

The following theorem proves that Algorithm 5 computes $I^{(t_1,t_2)}$:

Theorem 5.4.2. *Algorithm 5 terminates in time $O(t_2|S|^2)$ and when it does it outputs $I^{(t_1,t_2)}$.*

Proof. The algorithm uses a chain rule for mutual information. Let X_i and Y_i be two time-indexed Markovian probabilistic processes. Then

$$I^{(t_1,t_2)}(X_i, Y_i) = \sum_{i=t_1}^{t_2} H(X_i|X_{i-1}, ..., X_{t_1}) + H(Y_i|Y_{i-1}, ..., Y_{t_1}) - \\ - H(X_i, Y_i|X_{i-1}, Y_{i-1}, ..., X_{t_1}, Y_{t_1})$$

Data: A Markov Chain $\mathcal{C}_{O,h} = (S, s_0, P)$ in which each state is labeled with values for variables $\mathtt{o}$ and $\mathtt{h}$, two integers t_1 and t_2 satisfying $t_1 \leq t_2$.
Result: The leakage from time t_1 to time t_2 $I^{(t_1,t_2)}(\mathtt{o}, \mathtt{h})$

Compute the marginals $\mathcal{C}_{|\mathtt{o}}$ and $\mathcal{C}_{|\mathtt{h}}$;
Compute the probability distribution over the states of $\mathcal{C}_{|\mathtt{o},\mathtt{h}}$ at time t_1; let it be $\pi^{(t_1)}_{|\mathtt{o},\mathtt{h}}$;
Similarly compute $\pi^{(t_1)}_{|\mathtt{o}}$ for $\mathcal{C}_{|\mathtt{o}}$ and $\pi^{(t_1)}_{|\mathtt{h}}$ for $\mathcal{C}_{|\mathtt{h}}$;
Compute $H^{(t_1)}(\mathcal{C}_{|\mathtt{o},\mathtt{h}}) = H(\pi^{(t_1)}_{|\mathtt{o},\mathtt{h}})$, $H^{(t_1)}(\mathcal{C}_{|\mathtt{o}}) = H(\pi^{(t_1)}_{|\mathtt{o}})$ and $H^{(t_1)}(\mathcal{C}_{|\mathtt{h}}) = H(\pi^{(t_1)}_{|\mathtt{h}})$;
Compute $I^{(t_1,t_1)} = H^{(t_1)}(\mathcal{C}_{|\mathtt{o}}) + H^{(t_1)}(\mathcal{C}_{|\mathtt{h}}) - H^{(t_1)}(\mathcal{C}_{|\mathtt{o},\mathtt{h}})$;
for $i = t_1 + 1$ *to* t_2 **do**
  $H^{(i)}(\mathcal{C}_{|\mathtt{o},\mathtt{h}}) \leftarrow H^{(i-1)}(\mathcal{C}_{|\mathtt{o},\mathtt{h}}) + \sum_{s \in S_{|\mathtt{o},\mathtt{h}}} \pi^{(i-1)}_{|\mathtt{o},\mathtt{h}}(s) L_{|\mathtt{o},\mathtt{h}}(s)$;
  $H^{(i)}(\mathcal{C}_{|\mathtt{o}}) \leftarrow H^{(i-1)}(\mathcal{C}_{|\mathtt{o}}) + \sum_{s \in S_{|\mathtt{o}}} \pi^{(i-1)}_{|\mathtt{o}}(s) L_{|\mathtt{o}}(s)$;
  $H^{(i)}(\mathcal{C}_{|\mathtt{h}}) \leftarrow H^{(i-1)}(\mathcal{C}_{|\mathtt{h}}) + \sum_{s \in S_{|\mathtt{h}}} \pi^{(i-1)}_{|\mathtt{h}}(s) L_{|\mathtt{h}}(s)$;
  $I^{(t_1,i)} \leftarrow H^{(i)}(\mathcal{C}_{|\mathtt{o}}) + H^{(i)}(\mathcal{C}_{|\mathtt{h}}) - H^{(i)}(\mathcal{C}_{|\mathtt{o},\mathtt{h}})$;
  $\pi^{(i)}_{|\mathtt{o},\mathtt{h}} \leftarrow \pi^{(i-1)}_{|\mathtt{o},\mathtt{h}} P^{(i-1,i)}_{|\mathtt{o},\mathtt{h}}$;
  $\pi^{(i)}_{|\mathtt{o}} \leftarrow \pi^{(i-1)}_{|\mathtt{o}} P^{(i-1,i)}_{|\mathtt{o}}$;
  $\pi^{(i)}_{|\mathtt{h}} \leftarrow \pi^{(i-1)}_{|\mathtt{h}} P^{(i-1,i)}_{|\mathtt{h}}$;
end
return $I^{(t_1,t_2)}$;

Algorithm 5: Compute the leakage of a MC from a time t_1 to a time t_2.

This can be proved as follows:

$$
\begin{aligned}
I^{(t_1,t_2)}(X_i, Y_i) &= H^{(t_1,t_2)}(X_i) + H^{(t_1,t_2)}(Y_i) - H^{(t_1,t_2)}(X_i, Y_i) = \\
&= \sum_{i=t_1}^{t_2} H(X_i | X_{i-1}, ..., X_{t_1}) + \sum_{i=t_1}^{t_2} H(Y_i | Y_{i-1}, ..., Y_{t_1}) - \\
&\quad - \left(\sum_{i=t_1}^{t_2} H(X_i | X_{i-1}, ..., X_{t_1}, Y_{i-1}, ..., Y_{t_1}) + H(Y_i | X_i, ..., X_{t_1}, Y_{i-1}, ..., Y_{t_1}) \right) \\
&= \sum_{i=t_1}^{t_2} H(X_i | X_{i-1}, ..., X_{t_1}) + \sum_{i=t_1}^{t_2} H(Y_i | Y_{i-1}, ..., Y_{t_1}) - \\
&\quad - \sum_{i=t_1}^{t_2} H(X_i, Y_i | X_{i-1}, Y_{i-1}, ..., X_{t_1}, Y_{t_1}) \\
&= \sum_{i=t_1}^{t_2} H(X_i | X_{i-1}, ..., X_{t_1}) + H(Y_i | Y_{i-1}, ..., Y_{t_1}) - \\
&\quad - H(X_i, Y_i | X_{i-1}, Y_{i-1}, ..., X_{t_1}, Y_{t_1})
\end{aligned}
$$

The algorithm starts by computing the mutual information at time t_1, then updates it step by step until time t_2. The correctness of an update step comes from considering

that due to the Markov property it holds that

$$H(X_i|X_{i-1},...,X_0) = H(X_i|X_{i-1}) = \sum_{s\in S} \pi_s^{(i-1)} L(s)$$

The algorithm clearly terminates for a finite t_2 and $|S|$. The computation of the marginalss can be solved in $O(|S|^2)$, while the probability distributions at time t_1 require time $O(t_1|S|^2)$, dominating the cost of the other operations before the `for` cycle. Inside the cycle the operations have cost $O(|S|^2)$ and the cycle gets repeated $t_2 - t_1$ times, bringing the total cost to $O(t_2|S|^2)$ □

Note that Algorithm 5 is pseudopolynomial, as it depends not only on the size of the chain but also on the parameter t_2. Also note that due to the Markov property it holds that $I^{(t_1,t_2)} = I^{(t_1',t_2')}$ whenever $\pi_{|\mathtt{o},\mathtt{h}}^{(t_1)} = \pi_{|\mathtt{o},\mathtt{h}}^{(t_1')}$ and $t_2 - t_1 = t_2' - t_1'$.

5.4.2 Bounded Leakage

We want to determine how many time units it takes for the system to leak a given amount c of bits of information. The problem is more complex than the similar one analyzed in the previous section, since leakage is a complex function of the behavior of the system in time and finding a way to bound or reverse it is not obvious.

We start by considering the qualitative version of the problem: does there exist a time t such that $I_t(\mathcal{C}_O, \mathcal{C}_h) \geq c$? To answer we note that the leakage is a non-decreasing function, so if the answer is yes then it will remain greater than c for each time $t' \geq t$. This allows us to answer the qualitative question by computing the leakage on the infinite time horizon as shown in Section 5.2; let it be l^∞. Then we have three cases:

$l^\infty < c$ Then there is no time t such that the leakage is c

$l^\infty > c$ Then there is a finite time t such that the leakage is c

$l^\infty = c$ Then the system leaks c bits on the infinite time horizon but we have no guarantee that this amount will be reached in finite time.

In the case in which $l^\infty \geq c$ we can ask the quantitative question, i.e. at what time t the system will have leaked at least c bits. If $l^\infty > c$ we know that such time t exists, while if $l^\infty = c$ it may not. We will define the *bounded leakage problem* as follows: given a Markov chain $\mathcal{C} = (S, s_0, P)$ labeled with secrets and observations and a positive real number c, determine if there exists a finite time t such that the information leakage of the chain at time t is exactly c.

The problem is harder than it seems. For deterministic programs, it has been shown by Terauchi that it is not a k-safety property for any k [89]. The problem has also been addressed computationally by Heusser and Malacaria [45]. For randomized programs, we will show that the problem can be reduced from Skolem's problem [73]. While smaller instances have been shown to be decidable, the full decidability of Skolem's problem is still an open question [42, 74].

Akshay et al. [3] show that Skolem's problem is equivalent to the following: given a Markov chain $\mathcal{C} = (S, s_0, P)$, a state s and a probability r determine whether there is a time t such that $\pi_s^{(t)} = r$. We will call this *Skolem's Markov chain reachability problem*. Intuitively, information leakage is a harder problem than reachability, as formally stated by the following theorem:

Theorem 5.4.3. *Let $\mathbb{A}$ be an algorithm deciding the bounded leakage problem. Then $\mathbb{A}$ decides Skolem's Markov chain reachability problem.*

Proof. We show a reduction to the bounded leakage problem from Skolem's Markov chain reachability problem.

Consider an instance of Skolem's Markov chain reachability problem, i.e. a Markov chain $\mathcal{C} = (S, s_0, P)$, a state s and a probability r. Label state s with the information $\mathtt{h} = 0, \mathtt{o} = 0$ and all other states with the information $\mathtt{h} = [0, 1], \mathtt{o} = 1$, for a secret variable $\mathtt{h}$ and an observable variable $\mathtt{o}$. Then being in state s gives us exactly 1 bit of information over $\mathtt{h}$, while no other state gives any information.

Let $I(\mathtt{o};\mathtt{h})^k$ be the mutual information between $\mathtt{o}$ and $\mathtt{h}$ at time $k \in \mathbb{N}$. Let p_k be a path in $\mathcal{C}$ of length k, $p_k(s)$ a path in $\mathcal{C}$ of length k ending in state s, and $p_k(\neg s)$ a path in $\mathcal{C}$ of length k not ending in state s. Then it holds that

$$
\begin{aligned}
I(\mathtt{o};\mathtt{h})^k &= H(\mathtt{h}) - H(\mathtt{h}|\mathtt{o}_1, ..., \mathtt{o}_k) && \text{(by def. of } I(\mathtt{o};\mathtt{h})^k\text{)} \\
&= 1 - H(\mathtt{h}|\mathtt{o}_1, ..., \mathtt{o}_k) && \text{(since } H(\mathtt{h}) = 1 \text{ by construction)} \\
&= 1 - \sum_{p_k \in S^k} P(p_k) H(\mathtt{h}|\mathtt{o}_1, ..., \mathtt{o}_k = p_k) \\
&= 1 - \sum_{p_k(\neg s) \in S} P(p_k(\neg s)) H(\mathtt{h}|\mathtt{o}_1, ..., \mathtt{o}_k = p_k(\neg s)) \\
& && \text{(since } H(\mathtt{h}|\mathtt{o}_1, ..., \mathtt{o}_k = p_k(s)) = 0\text{)} \\
&= 1 - \sum_{p_k(\neg s) \in S} P(p_k(\neg s)) && \text{(since } H(\mathtt{h}|\mathtt{o}_1, ..., \mathtt{o}_k = p_k(t)) = 1 \text{ for } t \neq s\text{)} \\
&= 1 - (1 - \sum_{p_k(s) \in S} P(p_k(s)) \\
& && \text{(since the events } X_k = s \text{ and } X_k \neq s \text{ are complementary)} \\
&= 1 - (1 - P^{(k)}_{s_0,s}) && \text{(by definition of } P^{(k)}_{s_0,s}\text{)} \\
&= P^{(k)}_{s_0,s}
\end{aligned}
$$

proving that the leakage at time k is equivalent to the reachability probability of state s at time k. Thus if we had a way to determine whether the leakage of the system at some time t is r bits we would conclude that $\pi^{(t)}_s = r$, deciding Skolem's Markov chain reachability problem. Thus we conclude that determining whether there is a time t such that the leakage of the system is a given value is at least as hard as deciding Skolem's problem. □

Chapter 6

The QUAIL Quantitative Analyzer

> *Civilization advances by extending the number of important operations which we can perform without thinking of them.*

Alfred North Whitehead, *An Introduction to Mathematics*, 1911

The QUAIL tool implements the analysis techniques we introduced in Chapter 4. In particular, it is able to compute the information leakage of a randomized program written in an imperative language when observed by an ignorant attacker. In models the case in which the attacker can observe the values of the observable variables after the termination of the program. If the program does not necessarily terminate, it is assumed that the non-termination of the program is one of the possible observable outcomes.

6.1 Preliminaries

6.1.1 QUAIL Imperative Language

QUAIL supports a simple but powerful imperative WHILE language. The language includes constants and array declarations, but not function declarations.

Variable declarations

All variables in QUAIL are fixed sized integers. We force them to be declared at the beginning of the program to ensure that the analysis terminates and to simplify scoping.

Variables are declared in one of the four following types: *public, private, secret* and *observable*.

Public and observable variables have an explicit value during the computation. They represent variables whose value is known at computation time. The only difference between public and observable variables is that the attacker can discriminate states that have different values of the latter but not of the former. Public and observable variables can be manipulated using standard arithmetic operators.

Public and observable variables are declared as follows:

```
public int4 v;
```

or

```
observable int4 v;
```

declares a 4 bits integer variable whose name is v, either public or observable.

```
public int4 v := 5;
```

declares v and initializes it to value 5. Any expression can be used to initialize a variable, provided that the variables used in the expression are public or constants and have been previously declared. Since QUAIL does not support function declaration there is a single variable scope. Declaring multiple variables with the same name is consequently forbidden.

Private and secret variables represent variables that do not have a known fixed value, but instead a uniform probability distribution over a set of values. The set of values is represented as a sequence of integer intervals, thus for instance the set $\{0, 1, 2, 3, 5, 6, 8, 9, 10, 13\}$ would be represented as $[0, 3][5, 6][8, 10][13, 13]$.

The difference between private and secret variables is that the information leakage is only computed on the latter, while knowing the exact value of the former is considered to be information that is not interesting to the attacker.

Private and secret variables cannot be used in assignment and expressions. They can only appear to the left of the operator in a guard. This is to restrict the leakage analysis to indirect flow of information, but direct flow can still be modeled if necessary.

Private and secret variables are declared as follows:

```
private int4 v;
```

or

```
secret int4 v;
```

declares a 4 bits integer variable whose name is v, either private or secret.

```
private int4 v := [0,1][2,5];
```

declares `var` and restricts its range to the two intervals [0,1] and [2,5]. Again any expression can be used in the bounds of the intervals.

QUAIL allows for the declaration of integer constants. Constants are declared as follows:

```
const N := 4;
```

They are replaced by their value during the preprocessing step.

Arrays

Variables can also be arrays of integers and multi-dimensional arrays. Arrays are declared before the integer type of a variable.

```
public array[7] of int4 tab;
```

declares a public variable `tab` that is an array of 4 bits integer of size 7 whose indexes range from 0 to 6, while

```
public array[1..7] of int4 tab;
```

declares `tab` as an array of size 4 whose indexes range from 1 to 7. The size of an array can be any expression that evaluates to an integer.

An array may be initialized with a set of initial values:

```
public array[1..4] of int4 tab := {1,1,2,2};
```

initializes `tab` such that `tab[1]` and `tab[2]` are equal to 1, while `tab[3]` and `tab[4]` are equal to 2. Private arrays can be initialized like any private variable, with a set of intervals:

```
private array[1..4] of int4 tab := [0,1];
```

In that case all the variables in the array are initialized to the same range of integers.

Expressions

Expressions are used in guards, assignments, variables initialization and arrays indexes. Binary operators (`||`,`&&`,`^`,`+`,`-`,`*`,`/` and `%`) and unary operators (`-`,`!`) can used. Classical operators precedence is assumed. For Boolean operations integer variables are considered as a true value if non null, and false if null. Only public and observable variables, constants and integers can be used in expressions.

Guards

Guards are limited to a single comparison between a variable on the left side (either public, or private, or constant, or an integer value) and an expression on the right side. Any comparison operator among `<`, `>`, `<=`, `>=`, `==` and `!=` can be used.

Assignments

An assignment statement is written in the following manner:

```
assign v := expr;
```

where `v` is a public or observable variable (possibly with indexes) and *`expr`* is an expression containing no private or secret variables.

Random assignments

The program can use two types of random primitives to assign values to a variable.

```
random v := random(expr_min,expr_max);
```

assigns to a public variable `v` a random value, chosen between the values of *`expr_min`* and *`expr_max`*, with a uniform probability distribution.

```
random v := randombit(p);
```

where `p` is a float value lower than 1, assigns to a public variable `v` a random bit value, that is 0 with probability `p`, and 1 with probability $1 - \texttt{p}$.

IF statements

IF conditional statements starts with the keyword `if`, possibly followed by `elif` and `else`, and ends with `fi`. The consequent statements are listed after the keyword `then`. For example the following structures are allowed:

```
if (h <= l) then assign v:=1;
fi


if (h <= l) then assign v:=1;
else assign v:=2;
fi


if (h <= l) then assign v:=1;
elif (h==l) then assign v:=2;
fi


if (h <= l) then assign v:=1;
elif (h==l) then assign v:=2;
elif (h==l+1) then assign v:=3;
else assign v:=4;
fi
```

WHILE statements

Conditional WHILE loop starts with the keyword `while`, followed by a guard, and the statements included in the loop are listed between the keywords `do` and `od`. For example the following structure is allowed:

```
while (h <= l) do
assign l := 1;
assign v := 2;
od
```

FOR statements

A FOR loop can be used to iterate over all the elements of an array. The syntax is:

```
for (v in tab) do
assign v := v+1;
od
```

The variable `v` is a local variable that must only be used inside the loop. It will take successively each value in the array tab. Note that if `tab` is a multi-dimensional array `v` is also an array.

Return statements

The program ends when a return statement is reached. Its syntax is simply:

```
return;
```

6.1.2 Attacker Encoding

QUAIL assumes that the attacker is ignorant, i.e. does not have any information about the value of the secret except in which range it is, e.g. from 0 to 7 for a 3-bit secret. Like all information-theoretical analysis we assume that the attacker has access to the source code of the system: assuming otherwise would invalidate the analysis in case the attacker was able to obtain or infer information about such code.

The assumption allows QUAIL to build directly the Markov chain model of the scenario, since whenever a prior distribution on the secret is encountered it can be assumed to be uniform.

The attacker is assumed to be able to start the program and observe the values of the output variables after the program's termination. For this reason, all and only the internal states of the Markov chain are hidden during the hiding part of the modeling (see Chapter 4). If the program does not terminate, we assume that the attacker is able to recognize this, e.g. via a timeout. In this case the attacker knows that the program did not terminate but is not able to read any variable's value.

For the discrimination relation, as we said the attacker is assumed to be able to observe only a given subset of the variables, that we call *observable* variables. The attacker can thus discriminate two states if and only if they differ in the value of any of the observable variables, while different states that assign the same values to the observable variables are impossible to discriminate for him.

To encode an attacker, the QUAIL user only has to specify which variables are the secret and which variables are observable to the attacker. The rest of the encoding is automatically handled by the tool.

6.2 Procedure

QUAIL's analysis proceeds as explained in Chapter 4, with some improvements to make the process more streamlined and implementable.

Step 1: Preprocessing In this step the imperative code gets rewritten in a simplified `if-goto` language, following common compiler practice. All `if-elif-else-fi` conditional statements and `while` loop statements are rewritten as follows:

- if-elif-else-fi statement

```
1 if CONDITION1 then STATEMENT1;
2 elif CONDITION2 then STATEMENT2;
3 else STATEMENT3;
4 fi
```

becomes

```
1 if CONDITION1
2 then goto 4;
3 else goto 6;
4 STATEMENT1;
5 goto 12;
6 if CONDITION2
7 then goto 9;
8 else goto 11;
9 STATEMENT2;
```

```
10 goto 12;
11 STATEMENT3;
12 ...
```

- `while` statement

```
1 while CONDITION do
2 STATEMENT
3 od
```

becomes

```
1 if CONDITION
2 then goto 4
3 else goto 6
4 STATEMENT
5 goto 1
6 ...
```

Also, array calls are substituted with single indexed variables and `for` statements rewritten to a sequence of commands on such variables. Finally, constants are substituted with their value. In this step we also add automatically a `free` command to signal when a variable is not used anymore and can be collected.

Step 2: Probabilistic Symbolic Execution QUAIL symbolically executes the preprocessed code and builds an annotated Markov chain semantics of the program execution, as explained in Chapter 4. This step includes building the MDP model of the system and applying the prior information of the attacker to it: since we assume that the prior probability distribution over the values of the secret is uniform, encoding the ignorant attacker, we can build the Markov chain directly. Whenever a conditional guard is found QUAIL has sufficient information to compute the probability that the guard will be satisfied, and constructs two successor states, one if the guard is true and one if it is false, with appropriate transition probabilities. This is the most time-consuming step of the computation, since it requires building a full control flow graph of the system's behavior and assigning probabilities to it. In the worst case the graph has exponential size in the size of the variables, but for most academic cases it has a reasonable size of thousands or tens of thousands of nodes. We apply on-the-fly reduction techniques like avoiding producing states that we know will be removed by the next step, but these do not significantly reduce the computation time of this step.

Step 3: State Hiding and Model Reduction QUAIL assumes that the attacker can only observe the values of some variables at the end of the computation, thus all internal states of the system are to be hidden. We apply Algorithm 1 iteratively on all internal states until only the initial state and the output states remain. In this step we also detect nonterminating behavior and, if needed, we construct an output state modeling non-termination. This operation removes more than 90% of the states of the Markov chain model, in fact producing a Markov chain with a single probability distribution from the initial state to the output states. To make this operation as quick as possible, states are equipped with a list of their predecessors and successors.

Step 4: Quotienting In this step we construct three quotients of the Markov chain, as explained in Chapter 4. A quotient is obtained by merging together the states that correspond to the same equivalence class in a given equivalence relation, as explained in Chapter 2. The quotients are as follows:

- The *attacker's quotient* represents the view that the attacker has of the system. It is obtained by merging together states that assign the same values to all and only the observable variables.
- The *secret's quotient* represents the system as it depends on the secret. Its entropy is a measure of how much of the secret is actually used in the execution of the program. It is obtained by merging together states that assign the same sets of values to all and only the secret variables.
- The *joint quotient* represents the joint behavior of the observable variables and secret variables. It is obtained by merging together states that assign the same values and sets of values to the observable and secret variables.

To speed up the process, QUAIL drops the information about the variable assignments in the states of the quotients. Such information is not needed to perform the rest of the analysis.

Step 5: Entropy and Leakage Computation Finally, QUAIL computes the entropy of the three quotients using Theorem 4.2.2 in linear time in the size of the quotients. The three computations are independent and can be parallelized. The information leakage is then computed as the sum of the entropies of the attacker's and secret's quotients minus the entropy of the joint quotient, as explained in Chapter 4. QUAIL outputs the result with the requested amount of significant digits. If requested, the tool also prints any of the Markovian models it has produced during the analysis.

6.3 Case Studies

We show how to analyze a number of protocols and academic examples with QUAIL. For each example we show the commented source code we used to encode it and comment on the results and variants.

6.3.1 Simple Authentication

In this basic example the attacker is trying to infer the password of a system by trying to provide as a password a given number in the password domain. The model is shown in Fig. 6.1 on the left.

The length of the secret is the size of the variable `password` on line 8. Since the variable is not explicitly initialized, QUAIL assumes that it is in the interval $[0, 2^{size} - 1]$. On line 5 the variable `input`, representing what is input by the attacker, is initialized with an arbitrary value. The value chosen is irrelevant as long as it is in the range of the possible values for the password.

Increasing the size of the password changes the amount of leakage, since the attacker learns less information about the password from a single attack attempt. The information leakage values for some password sizes are shown in Fig. 6.1 on the right.

It is worth noting that QUAIL solves this example with any password size in milliseconds because it uses the Markovian process encoding that we presented in this

```
1  // this bit is observable by the
       user; it is 0 for REJECT
     and 1 for ACCEPT
2  observable int1 o;
3
4  // this represents the password
     inserted by the user
5  public int2 input:=2;
6
7  // this is the secret
8  secret int2 password;
9
10 //
11 if (password==input) then
12   assign o:=1;
13 else
14   assign o:=0;
15 fi
16
17 // terminate
18 return;
```

Password length	Leakage
1	1
2	$8.11 \cdot 10^{-1}$
32	$7.78 \cdot 10^{-9}$
64	$3.54 \cdot 10^{-18}$
512	$3.81 \cdot 10^{-152}$

Figure 6.1: Simple authentication example: model (on the left) and resulting leakage according to password length (on the right).

thesis. Any analysis based on channel matrices would have to build the matrix for this examples, having a number of rows exponential in the size of the secret. This operation alone would require from days to millennia, according to the chosen size of the password.

We finally remark that the results are presented with 2 decimal digits, but QUAIL can work with any precision, as requested by the user.

6.3.2 Bit XOR

This is one of the simplest examples of randomized programs depending on a secret. In this case the secret is a bit. The system produces a random bit with a probability distribution known to the attacker, computes the exclusive OR of the secret and the random bit, and outputs the result to the attacker. The question is how much of the secret bit can the attacker infer by knowing the result of the exclusive OR. The model is shown in Fig. 6.2 on the left.

On line 9 we assign to the random bit `r` the value 0 with the given probability and 1 otherwise. We remind that the attacker possesses the source code, so he knows the probability distribution over `r`, but not the value that gets assigned to `r` during a given execution.

Note that we cannot directly calculate the result of the exclusive or by writing `assign o := s XOR r` because QUAIL does not allow private or secret variables to appear in an assignment statement. This is a limit of the representation of the variables in the tool, not in the theory. Work is under way to allow for a more natural encoding of this kind of operations.

6.3.3 Conditional Non-Termination

This example shown QUAIL's treatment of non-terminating programs. We have a 2-bit secret variable `s`, i.e. `s` is either 0, 1, 2 or 3. There is also an observable variable `o` that is initialized to 0 and never changes its value. Then the program terminates if `s` is 1 or 2, and loops forever otherwise. The model for the example is shown in Fig. 6.3 on the left.

```
1  observable int1 o; // this bit
       represents the output
2
3  public int1 r; // this bit is
       randomly generated
4
5  // this bit is the secret
6  secret int1 s;
7
8  // randomize the random bit with
       a given probability
9  random r:=randombit(0.5);
10
11 // calculate the XOR
12 if (s==r) then
13 assign o:=0;
14 else
15 assign o:=1;
16 fi
17
18 // terminate
19 return;
```

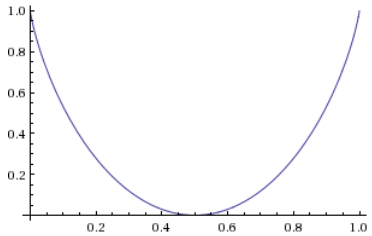


Figure 6.2: Bit XOR example: model (on the left) and graph of the information leakage over the probability of the random bit on line 9 (on the right).

In this example QUAIL reports a leakage of 1 bit. The reason is that the attacker, being able to distinguish whether the program terminates or not via a timeout, can infer whether the secret is 1 or 2 or whether is 0 or 3. In both cases its ignorance goes down from 4 possible cases to 2 possible cases, thus being quantified in 1 bit of gained information.

In Fig. 6.3 on the right we can see the last part of the Markov chain produced by QUAIL, in particular showing how QUAIL detects non-terminating loops.

6.4 Bibliographic Note

The content of this chapter is mostly based on Paper C and its appendix.

```
1  // a variable to
       loop on
2  observable int2 o
       :=0;
3
4  // this is the
       secret, either
       0, 1, 2 or 3
5  secret int2 s
       :=[0,3];
6
7  // if the secret is
       0 then loop
       forever
8  if (s==0) then
9    while (o==0) do
10   skip;
11   od
12 fi
13
14 // if the secret is
       3 also loop
       forever
15 if (s==3) then
16   while (o==0) do
17   skip;
18   od
19 fi
20
21 // terminate
22 return;
```

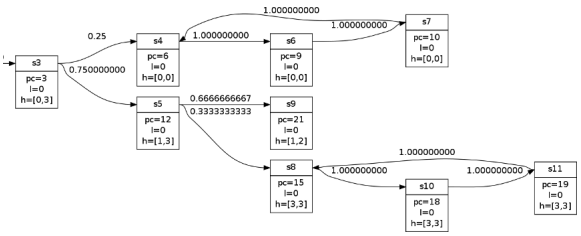


Figure 6.3: Conditional non-termination example: model (on the left) and snippet of the Markov chain model (on the right).

Chapter 7

Solved Cases

> *The worthwhile problems are the ones you can really solve or help solve, the ones you can really contribute something to. No problem is too small or too trivial if we can really do something about it.*
>
> Richard Feynman (1966), published in *Perfectly Reasonable Deviations from the Beaten Track: The Letters of Richard P. Feynman*, 2005

In this Chapter we present some secret-dependent protocols and explain how we examined them using QUAIL.

7.1 The Dining Cryptographers Protocol

The Dining Cryptographers protocol is an anonymity protocol in which a number of agents collaborate to a shared computation depending on each agent's secret [26].

A group of n cryptographers is dining around a round table. At the end of the dinner, the waiter informs them that the bill has already been settled by someone who would prefer to remain anonymous. The cryptographers respect the payer's wish for anonymity, but would like to know whether the benefactor is one of them or an external party. To determine this, each pair of adjacent cryptographers toss a coin hidden from everybody else, so that each cryptographers knows the value of the coin to his left and to his right. Then each cryptographers declares aloud the exclusive OR of the two coins he sees, i.e. 0 if they have the same value and 1 otherwise. If one of the cryptographers is the payer, he declares the opposite. In the end, if an even number of ones is declared then someone else paid the bill, while if an odd number of ones is declared one of the cryptographers is the benefactor.

Figure 7.1 on the left shows the QUAIL model of the Dining Cryptographers protocol. The leakage of the protocol depends on the randomness of the coin that the cryptographers toss; as the coin become more deterministic, so the probability of getting a head gets closer to 0 or 1, the attacker is more able to determine the identity of the payer. Some results are shown in Fig. 7.1 on the right; for different numbers of cryptographers we show that as the probability of the coin toss approaches 0 or 1 the

```
// N is the number of cryptographers
     at the table
const N:=3;

// this bit represents the output
observable int1 output;

// these bits represent the coin
     tosses
public array [N] of int1 coin;

// these are the observable coins
observable array[2] of int1 obscoin;

// this is just a counter
public int32 i:=0;

// these bits represent the bits
     declared by the three
     cryptographers
observable array [N] of int1 decl;

// the secret has N+1 possible
     values:
// 0 if someone else paid
// 1 if Cryptographer A paid
// 2 if Cryptographer B paid
// 3 if Cryptographer C paid
// ... and so on
secret int32 h := [0,N];

// tossing the coins
for (c in coin) do
  random c:=randombit(0.5);
od

// if the attacker is one of the
     cryptographers, he can observe
     two of the coins.
// To encode an external attacker
     comment the next two lines.
assign obscoin[0]:=coin[0];
assign obscoin[1]:=coin[1];

// producing the declarations
     according to the secret value
while (i<N) do
  assign decl[i]:=coin[i]^coin[(i+1)
      %N];
  if (h==i+1) then
    assign decl[i]:=!decl[i];
  fi
  assign i:=i+1;
od

//producing the output bit and
     terminating
for (d in decl) do
  assign output := output^d;
od

return;
```

		Cryptographers			
		3	**4**	**5**	**6**
Coin probability	**0**	2	2.32	2.59	2.81
	0.1	1.76	1.90	2.03	2.15
	0.3	1.56	1.49	1.45	1.41
	0.5	1.50	1.37	1.25	1.15
	0.7	1.56	1.49	1.45	1.41
	0.9	1.76	1.90	2.03	2.15
	1.0	2	2.32	2.59	2.81

Figure 7.1: Dining Cryptographers example: model for the Dining Cryptographers (on the left) and leakage table as a function of the number of cryptographers and of the probability of the random coin toss (on the right).

leakage increases. When it is 0 or 1 the leakage is equivalent to the bit size of the secret, i.e. the logarithm in base 2 of $n + 1$, proving that the whole secret gets leaked, and thus the attacker learns the identity of the payer, whoever he is.

In this encoding we are assuming that the attacker is one of the cryptographers, namely Cryptographer A. This is encoded in lines 34 and 35, where we copy the result of the two coins that are observable to Cryptographer A to observable variables. We are accepting the case in which Cryptographer A is the payer; in this case the leakage is slightly higher because A could learn by observing his coins and declarations that *he* is the secret payer. The case in which A knows that he is not the secret payer can be encoded by changing the declaration of the secret on line 25 to `secret int32 h := [0,0][2,N];` thus excluding that the secret can take value 1, corresponding to the fact that A paid.

The zero-leakage scenario in this case corresponds to the case in which the attacker is an external observer and knows that one of the philosophers paid the bill but not who. This can be encoded by changing the declaration of the secret in line 25 to `secret int32 h := [1,N];`, thus excluding the case in which the payer is an external agent, and commenting lines 34 and 35. In this case QUAIL correctly computes the leakage of the protocol as 0, independently from the number of cryptographers at the table.

7.2 Majority and Preferential Voting

In this example we want to compare two different voting protocols according to how effective they are in preserving the anonymity of the votes.

In both protocols we have a constant number n of voters and a constant number c of candidates, with $c \leq n$. The difference of the protocols is in how the voters express their preference.

In the Majority Voting protocol each voter expresses one vote for one of the candidates. The votes for each candidate are summed up and only the results are published, thus hiding information about which voter voted for which candidate. Each voter has a secret of size $\log_2 c$ representing its vote. The model for this example is shown on Fig. 7.2 on the left. The scenario is simple to formalize, and it can be shown that the information leakage for this protocol corresponds to

$$\sum_{k_1+k_2+\ldots+k_c=n} \frac{1}{c^n} \binom{n}{k_1+k_2+\ldots+k_c} \left(\log_2 c^n - \log_2 \binom{n}{k_1+k_2+\ldots+k_c} \right)$$

In the Preferential Voting protocol the voter does not express a single vote, but rather an order of preference of the candidates; thus if the candidates are A, B, C and D the voter could express the fact that he prefers B, then D, then C and finally A. Then each candidate gets c points for each time he appears as first choice, $c - 1$ points for each time he appears as second choice, and so on. The points of each candidate are summed up and the results are published. The model for this example is shown on Fig. 7.2 on the right. The complexity of the code depends on the fact that each voter can express $c!$ different votes, corresponding to the possible complete orderings of the c candidates. The secret vote of each voter is encoded as a number from 0 to $c! - 1$, and then transformed in a preferential order with the algorithm in lines 39-66.

The leakage comparison for the two voting protocols is shown in Table 7.1. The small examples we considered are sufficient to conclude that the Preferential Voting

```
// N is the number of voters
const N:=4;

// C is the number of candidates
const C:=2;

// the result is the number of
      votes of each candidate
observable array [C] of int32
      result;

// The secret is the preference of
      each voter
secret array [N] of int32 vote
      :=[0,C-1];

// these bits represent the votes
      received by the voting
      machine
public array [N] of int32 decl;

// this is just a counter
public int32 i:=0;
public int32 j:=0;
// voting
while (i<N) do
  while (j<C) do
    if (vote[i]==j) then
      assign decl[i]:=j;
    fi
    assign j:=j+1;
  od
  assign j:=0;
  assign i:=i+1;
od

// count the number of votes for
      one of the candidates
assign i:=0;
assign j:=0;
while (i<N) do
  while (j<C) do
    if (decl[i]==j) then
      assign result[j]:=result[j
            ]+1;
    fi
    assign j:=j+1;
  od
  assign j:=0;
  assign i:=i+1;
od

return;
```

```
// N is the number of voters
const N:=4;

// C is the number of candidates
const C:=2;

// the result is the number of votes of each
      candidate
observable array [C] of int32 result;

// these bits represent the preferences of each
      voter, from 0 to C!-1
secret array [N] of int32 vote:=[0,C!-1];

// these bits represent the votes received by the
      voting machine
public array [N] of int32 decl;
public array [N] of array [C] of int32 prefs;

public array [C] of int32 temparray;
public int32 pos;
// this is just a counter

public int32 voter:=0;
public int32 candidate:=0;
public int32 k:=0;
public int32 y:=0;

// voting
while (voter<N) do
  while (candidate<C) do
    if (vote[voter]==candidate) then
      assign decl[voter]:=candidate;
    fi
    assign candidate:=candidate+1;
  od
  assign candidate:=0;
  assign voter:=voter+1;
od

// transform the secret of each voter into the
      order of the preferences
assign voter:=0;
while (voter<N) do
   // build the initial array
  assign candidate:=0;
  while (candidate<C) do
    assign temparray[candidate]:=candidate;
    assign candidate:=candidate+1;
  od
  assign k:=C;
  // find a position
  while (k>0) do
    assign pos := decl[voter]%k;
    assign candidate:=C-k;
    // update the vote of the candidate
    assign prefs[voter][candidate]:=temparray[pos];
    // remove the element from the array
    assign y:=pos;
    while (y<C-1) do
      assign temparray[y]:=temparray[y+1];
      assign y:=y+1;
    od
    // update the vote of the voter
    assign decl[voter]:=decl[voter]/k;
    // decrease the counter
    assign k:=k-1;
  od
  assign voter:=voter+1;
od
// calculate the results
assign candidate:=0;
while (candidate<C) do
  assign voter:=0;
  while (voter<N) do
    assign result[candidate]:=result[candidate]+
          prefs[voter][candidate];
    assign voter:=voter+1;
  od
  assign candidate := candidate+1;
od

return;
```

Figure 7.2: Voting example: model for the Majority Voting (on the left) and for the Preferential Voting (on the right).

Table 7.1: Voting protocols: leakage tables for Majority Voting (on the left) and Preferential Voting (on the right)

Maj		**Voters**		
		2	**3**	**4**
Cands	**2**	1.50	1.81	2.03
	3	/	3.12	3.57
	4	/	/	4.81

Pref		**Voters**		
		2	**3**	**4**
Cands	**2**	1.50	1.81	2.03
	3	/	2.54	2.96
	4	/	/	timeout

scheme protects the anonymity of the votes more efficiently than the Majority Voting scheme.

Regrettably, QUAIL takes some hours to analyze the cases with 4 voters and candidates. This is particularly true for the Preferential Voting scheme, being the most complex of the two. Since the Markov chain built by QUAIL contains many copies of the same trees, we expect to be able to deploy compositional techniques in the future to reduce the computation time significantly.

7.3 Topological Protocols

We use QUAIL to analyze the security of anonymity protocols that depend on the topology of a given network. In particular we analyze the Onion Routing protocol [79], its implementation Tor [40], and the Crowds protocol [80]. The Onion Routing and Tor scenarios we analyze are similar to the one presented in Paper B and by Chen and Malacaria [28], while the Crowds scenario follows similar analyses [24, 25, 28]. The analysis method includes producing the channel matrix modeling of the protocols and translating them in QUAIL. In all cases we abstract away details about encryption by assuming it to be perfect. This method can be used to analyze with QUAIL any protocol encoded as a channel matrix.

The goal of all three protocols is the same: protect the anonymity of the identity of the sender of a message that gets delivered over the network. The techniques used are different.

In the Onion Routing protocol, the message is passed from the sender to an *onion proxy*. The proxy encrypts the message in layers of encryption, with each layer designed to be decrypted by one of the nodes of the system and containing the information about which subsequent node to forward the message. This way the onion proxy determines a path through the network that the message will follow, terminating in delivery to the intended recipient of the message. The restriction on the path structure is that no node can appear in it twice. It is assumed that the onion proxy chooses one of the available paths with uniform probability.

Tor is an implementation of Onion Routing with the important difference that the path is always exactly three nodes long, passing through an *entry node*, a *relay node* and an *exit node*. Also the implementation makes use of symmetric cryptography whenever possible. Both changes to the original protocol are due to efficiency reasons.

Finally, in the Crowds protocol the message gets sent by the sender in the network. Then the node in the network receiving the message delivers it directly to the recipient with probability p, or forwards it to another node in the network otherwise. The successor

```
1  public int1 r;
2  if (h == x) then
3    random r := randombit(0.5);
4    if (r == 0) then
5        assign o := 1;
6    else
7        random r := randombit(0.66666666666666666);
8        if (r == 0) then
9            assign o := 2;
10     else
11       assign o := 3;
12     fi
13   fi
14 fi
```

Figure 7.3: Fragment of QUAIL code encoding one row of a channel matrix.

node is chosen with uniform probability among the ones connected with the current holder of the message, including itself. The subsequent node receiving the message delivers it with probability p or forwards it to another node otherwise, and so on until the message is delivered.

We also assume that some of the nodes in the system are compromised by the attacker. When the message passes through a compromised node the attacker observes which node forwarded the message to the compromised node. In the case of the Onion Routing and Tor protocols the compromised node forwards the message further in the system anyway, while in the Crowds protocol the compromised node discards the message. This is due to the fact that forwarding the message may produce further useful observations for the attacker in Onion Routing and Tor but not in Crowds.

In all three cases, we assume that the nodes are connected by a given network topology, so any note can forward messages only to the nodes connected to itself. We consider the case in which the network is a clique and the case in which the network is composed of two cliques of the same size connected by a bottleneck of variable size. In the latter case we analyze what happens if the corrupted node are in one or both of the cliques or in the bottleneck.

We encode the protocols in QUAIL. The secret is the identity of the originator of the message. In the Crowds protocol the observation is from which node the message originated before passing in a compromised node. In the Onion Routing and Tor protocols the observation is the predecessor and successor of each compromised node in each path. The probabilities for each observation are computed and encoded in a QUAIL program. This is a generic method for directly encoding and analyzing a channel matrix with a QUAIL program. Let X be the set of possible values for the secret ranged over by x, Y the set of possible values for the observation ranged over by y, and $p(y|x)$ the probability of observing $y \in Y$ if the secret is $x \in X$.

We produce a fragment of QUAIL code that assigns to an observable variable `o` the values of the observation according to the value of a secret variable `h` such that the conditional probabilities are coherent with the given channel matrix, using the `randombit` command. Consider a given secret value $x \in X$. Assume that $Y = \{1, 2, 3\}$ and $P(Y|X = x) = \{1 \mapsto 1/2, 2 \mapsto 1/3, 3 \mapsto 1/6\}$. Then the channel matrix row corresponding to x is encoded by the code fragment depicted in Fig. 7.3.

The probability of the bit in line 3 is $P(Y = 1|X = x)$ and the probability in line 7

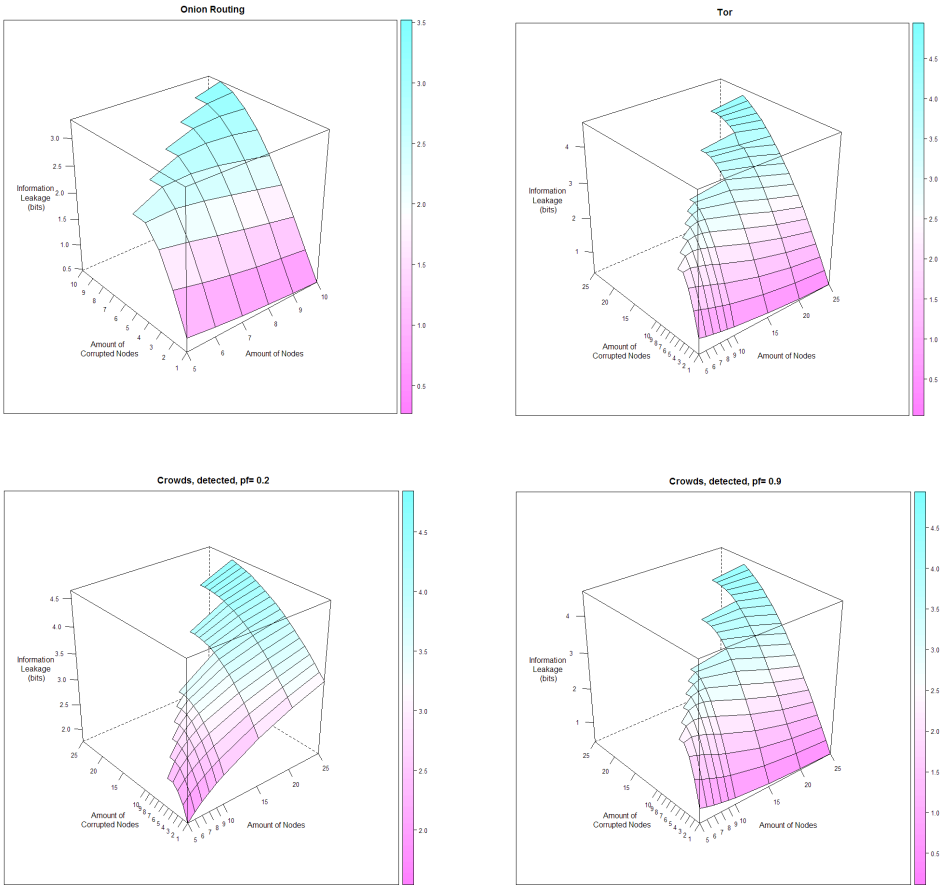


Figure 7.4: Information leakage for various protocols, network sizes and number of compromised nodes. Top left: Onion Routing protocol. Top right: Tor protocol. Bottom left: Crowds protocol with a probability of forwarding of 0.2. Bottom right: Crowds protocol with a probability of forwarding of 0.9.

is $P(Y = 2|X = x, Y \neq 1)$. The remaining case would have probability 1, so we can directly assign 3 to the observable variable in line 11. Any probability distribution can be modeled this way as a chain of binary distributions. The obtained distribution over the possible values of the observable variable corresponds to the distribution over Y in the channel matrix.

The resulting code is then analyzed with QUAIL to quantify how much information about `h` is inferred by an attacker able to observe `o`. The results are consistent with previous analyses of the protocols [24, 25, 28].

Some of the leakage results for the clique cases are summarized in Fig. 7.4. The analysis also provides insight about the effectiveness of the different protocols against correupted nodes positioned in strategical points of the topology. We refer to the full thesis [75] for the full discussion, results on the bottleneck topologies, and comparison of QUAIL and LeakiEst [31] for performing this case analysis. The models and strategies we used to compute the channel matrix representation of the protocols can also be found in the reference [75].

7.4 Bibliographic Note

Section 7.1 is based on the appendix of Paper C, Section 7.2 is original, and Section 7.3 has been developed in collaboration with Mads Tingaard Pedersen for his master thesis work [75].

Chapter 8

Conclusions and Future Work

> *He who moves not forward goes backward! A capital saying!*
>
> Johann Wolfgang von Goethe, *Herman and Dorothea*, Canto III, line 66

In this chapter we draw the conclusions for the work presented in this thesis and trace the path for future work in the field.

8.1 Concluding Remarks

We review the thesis statement from Chapter 1 and discuss how the material presented supports it.

Thesis Statement *Using Markovian models instead of channel matrices for modeling system-attacker scenarios for information leakage has the following advantages:*

1. *The Markovian model can be directly obtained from the source code of the system*
2. *The model is more compact and consequently the computation of the leakage is faster*
3. *The system and attacker are modeled separately, allowing for model reuse*
4. *It is possible to model and analyze systems with non-terminating behavior*

Statement 1: *The Markovian model can be directly obtained from the source code of the system.*

We have provided in Section 4.1.2 a Markov Decision Process semantics automatically obtained from the source code of the system under analysis. The semantics can be combined with the prior distribution of the attacker over the secret to obtain a Markov chain model of the system-attacker scenario, and the Markov chain model itself can be analyzed to compute the information leakage of the system to the attacker.

In Chapter 6 we have presented QUAIL, an implementation of the analysis method for terminating processes exposed in Chapter 4. QUAIL takes as input the source code of

the system and performs the complete analysis without needing any further intervention from the user.

In Chapter 7 we have used QUAIL to analyze interesting cases of security protocols. In the Dining Cryptographers and Majority and Preferential Voting cases in particular we have shown how the models of the system we analyze correspond to the source code of the system implementing the protocols, respectively in Figures 7.1 and 7.2.

Directly analyzing source code opens the possibility of performing precise quantitative information leakage analysis on the source code of off-the-shelf security products.

Statement 2: *The model is more compact and consequently the computation of the leakage is faster*

In Fig. 1.1 we show how we can produce a Markov chain with three states to model an example that would require a channel matrix with more than 10^{19} cells. The example we show is for a 64-bit password. While increasing the size of the password would increase the size of the channel matrix exponentially, the Markov chain model would still have the same three states. This small example is explicative of the compactness of the Markov chain model compared to the channel matrix model for encoding terminating systems.

We remark that in the worst case the size of the Markovian model may be not significantly smaller or may be even bigger than channel matrix. Such worst case would realize when all rows of the matrix are completely different, and it is therefore impossible to compress it by collapsing them together. Still, we argue that in such worst case there is *no* way to completely represent the system that is more compact than the channel matrix, thus no improvement on that is to be expected.

Non-terminating systems can be modeled by channel matrices only if infinite-size matrices are allowed. For instance, the non-terminating program in Fig. 5.2 can be modeled as a channel matrix with two rows and a countably infinite number of columns, one for each string of bits of infinite size. The Markov chain model for the same example, shown in Fig. 5.3a, has 5 states. We will discuss this further in Statement 4.

Statement 3: *The system and attacker are modeled separately, allowing for model reuse*

Like a channel matrix model, the MDP semantics does not model the prior information of the attacker on the secret. Unlike the channel matrix, the MDP semantics *also* does not contain information about which parts of the system the attacker is able to observe and discriminate. The MDP semantics only models information about the system; the information about the attacker is considered later to transform it in the Markov chain modeling the system-attacker scenario.

This separation of concerns allows us to analyze the MDP semantics by itself, and to consider attackers that have different views of the system without having to recompute the semantics itself. The QUAIL tool spends most of the computation time on building the model for the system, thus being able to reuse previously computed models for cases with different attackers would reduce computation time by more than 80%.

The channel matrix ordering allows for the comparison of attackers with different discriminatory power over the system's output, but the different matrices have to be computed from scratch. Reusing the same MDP semantics and modifying it separately for different attackers decreases the time required to compare attackers significantly.

Statement 4: *It is possible to model and analyze systems with non-terminating behavior*

This thesis opens the way for the modeling and analysis of non-terminating systems. The Markovian modeling allows us to express the behavior of a non-terminating system in a finite space, as shown by the model in Fig. 5.3a, instead of the infinite space that a channel matrix modeling of the same system would require.

Markovian models can express succinctly both terminating and non-terminating system-attacker scenarios, and the two can automatically distinguished - as explained in Sections 4.2.1 and 5.1 - by analyzing whether the projections of the behavior of the system on the observable and secret variables have finite entropy. This allows for analysis technique and tools that handle indifferently terminating and non-terminating scenarios.

The MDP modeling also considers the behavior of the system in time, allowing us to analyze how the information leakage evolves in relation with the structure of the system. This allows us to ask questions about how the leakage behaves in time, e.g. "How much information is leaked in a given time frame?" in Section 5.4.1 and "How much time does it take to leak a given amount of information?" in Section 5.4.2. These problems cannot be examined with the input-output encoding of the system behavior that the channel matrix model provides.

8.2 Future Work

We discuss some of the future directions for further work in the topic, both theoretical and tool-related.

8.2.1 Developing a Unified Leakage Algorithm

As we explained in Section 5.2, the projection of the system's joint behavior on the secret and observable variables is not necessarily a Markov chain, even if the system's behavior itself is. This makes it harder to provide an algorithm to compute entropy in such cases. Nonetheless, we believe that progress can be made.

Even if the projections themselves are not Markovian, they are produced as a weighted sum of residence probabilities of different states in a Markov chain that is available to us, giving us the idea that they can still be expressed in a finite space and analyzed accordingly. This is not obvious, as in general a non-Markovian probabilistic process over an infinite time may have a description of infinite size.

We show how to solve one such case in Section 5.2, but we feel that further research will allow us to develop a more elegant, unified leakage computation algorithm that can be used in any case in which the leakage is finite.

8.2.2 Representing Large Probability Distributions

One of the challenges in leakage computation for real systems is that the secret is not 7 or 8 bits long but most likely stored in a 64-bit, 128-bit or even 2056-bit variable. Since the number of possible values for the variable is exponential in the bit size, this makes even storing a single probability distribution over such values an computationally very hard problem.

If a probability distribution has unrelated values for each of its outcomes, there is no efficient way to represent it: all values have to be listed. Fortunately, most distributions

we handle in leakage computation do not suffer from this problem, as they usually contain multiple repetitions of similar patterns, like the one produced by multiplying a uniform distribution by a constant.

It would be interesting to understand which properties of a distribution make it easier ti represent and handle. Also, it is unclear whether more efficient ways of computing expected values and probability of events can be obtained. Progress in this field would impact all fields working with large probability distributions, including Artificial Intelligence, Operational Research and Probabilistic Model Checking.

8.2.3 Improving QUAIL

The QUAIL tool is an excellent proof of the implementability of the technique described in Chapter 4, but it has limitations. For instance it assumes that the attacker has no prior information about the secret. This is due to the fact that no prior information translates in a uniform distribution, that can be easily handled by the tool regardless of its size. Representation of arbitrary large probability distributions is still inefficient, as explained in the previous point.

Also, QUAIL spends most of its time building the system model. This time could be avoided, at the expense of some precision, by instead implementing a simulation technique similar to the ones proposed by Chothia for channel matrices [23, 30] and implemented in his tools LeakWatch and LeakiEst [31, 32].

Also, it would be interesting to extend QUAIL to analyze different imperative languages, specifically C, thus allowing it to analyze current security programs and libraries.

Finally, the algorithms described in Chapter 5 can be also implemented in QUAIL, allowing it to analyze non-terminating cases.

Bibliography

[1] *Proceedings of the 23rd IEEE Computer Security Foundations Symposium, CSF 2010, Edinburgh, United Kingdom, July 17-19, 2010*. IEEE Computer Society, 2010.

[2] *Eighth International Conference on Quantitative Evaluation of Systems, QEST 2011, Aachen, Germany, 5-8 September, 2011*. IEEE Computer Society, 2011.

[3] S. Akshay, Joël Ouaknine, Timos Antonopoulos, and James Worrell. Reachability problems for Markov chains. Personal communication, November 2013.

[4] Mário S. Alvim, Miguel E. Andrés, and Catuscia Palamidessi. Information flow in interactive systems. In Paul Gastin and François Laroussinie, editors, *CONCUR*, volume 6269 of *Lecture Notes in Computer Science*, pages 102–116. Springer, 2010.

[5] Mário S. Alvim, Miguel E. Andrés, and Catuscia Palamidessi. Quantitative information flow in interactive systems. *Journal of Computer Security*, 20(1):3–50, 2012.

[6] Mário S. Alvim, Konstantinos Chatzikokolakis, Catuscia Palamidessi, and Geoffrey Smith. Measuring information leakage using generalized gain functions. In Stephen Chong, editor, *CSF*, pages 265–279. IEEE, 2012.

[7] Dieter-Michael Arnold, Hans-Andrea Loeliger, Pascal O. Vontobel, Aleksandar Kavcic, and Wei Zeng. Simulation-based computation of information rates for channels with memory. *IEEE Transactions on Information Theory*, 52(8):3498–3508, 2006.

[8] Michael Backes, Boris Köpf, and Andrey Rybalchenko. Automatic discovery and quantification of information leaks. In *IEEE Symposium on Security and Privacy*, 2009.

[9] Murilo S. Baptista, Rero M. Rubinger, Emilson R. Viana, José C. Sartorelli, Ulrich Parlitz, and Celso Grebogi. Mutual information rate and bounds for it. *PLoS ONE*, 7(10):e46745, 10 2012.

[10] Gilles Barthe, Pedro R. D'Argenio, and Tamara Rezk. Secure information flow by self-composition. In *CSFW*, pages 100–114. IEEE Computer Society, 2004.

[11] D.P. Bertsekas and J.N. Tsitsiklis. *Introduction To Probability*. Athena Scientific books. Athena Scientific, 2002.

[12] Fabrizio Biondi, Axel Legay, Pasquale Malacaria, and Andrzej Wasowski. Quantifying information leakage of randomized protocols. In Roberto Giacobazzi, Josh Berdine, and Isabella Mastroeni, editors, *VMCAI*, volume 7737 of *Lecture Notes in Computer Science*, pages 68–87. Springer, 2013.

[13] Fabrizio Biondi, Axel Legay, Bo Friis Nielsen, and Andrzej Wasowski. Maximizing entropy over Markov processes. In *24nd Nordic Workshop on Programming Theory - NWPT*, 2012.

[14] Fabrizio Biondi, Axel Legay, Bo Friis Nielsen, and Andrzej Wasowski. Maximizing entropy over Markov processes. In Adrian Horia Dediu, Carlos Martín-Vide, and Bianca Truthe, editors, *LATA*, volume 7810 of *Lecture Notes in Computer Science*, pages 128–140. Springer, 2013.

[15] Fabrizio Biondi, Axel Legay, Bo Friis Nielsen, and Andrzej Wasowski. Maximizing entropy over Markov processes. *Journal of Logic and Algebraic Programming*, under revision.

[16] Fabrizio Biondi, Axel Legay, Louis-Marie Traonouez, and Andrzej Wasowski. QUAIL: A quantitative security analyzer for imperative code. In Sharygina and Veith [83], pages 702–707.

[17] Michele Boreale. Quantifying information leakage in process calculi. *Inf. Comput.*, 207(6):699–725, 2009.

[18] Michele Boreale and Francesca Pampaloni. Quantitative multirun security under active adversaries. In *QEST*, pages 158–167. IEEE Computer Society, 2012.

[19] Michele Boreale, Francesca Pampaloni, and Michela Paolini. Asymptotic information leakage under one-try attacks. In Martin Hofmann, editor, *FOSSACS*, volume 6604 of *Lecture Notes in Computer Science*, pages 396–410. Springer, 2011.

[20] Christelle Braun, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. Quantitative notions of leakage for one-try attacks. *Electr. Notes Theor. Comput. Sci.*, 249:75–91, 2009.

[21] David Brumley and Dan Boneh. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716, 2005.

[22] Krishnendu Chatterjee, Koushik Sen, and Thomas A. Henzinger. Model-checking omega-regular properties of interval Markov chains. In Roberto M. Amadio, editor, *FoSSaCS*, volume 4962 of *Lecture Notes in Computer Science*, pages 302–317. Springer, 2008.

[23] Konstantinos Chatzikokolakis, Tom Chothia, and Apratim Guha. Statistical measurement of information leakage. In Javier Esparza and Rupak Majumdar, editors, *TACAS*, volume 6015 of *Lecture Notes in Computer Science*, pages 390–404. Springer, 2010.

[24] Konstantinos Chatzikokolakis, Catuscia Palamidessi, and Prakash Panangaden. Anonymity protocols as noisy channels. *Inf. Comput.*, 206(2-4):378–401, 2008.

[25] Konstantinos Chatzikokolakis, Catuscia Palamidessi, and Prakash Panangaden. On the bayes risk in information-hiding protocols. *Journal of Computer Security*, 16(5):531–571, 2008.

[26] David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1:65–75, 1988.

[27] Han Chen and Pasquale Malacaria. The optimum leakage principle for analyzing multi-threaded programs. In Kaoru Kurosawa, editor, *ICITS*, volume 5973 of *Lecture Notes in Computer Science*, pages 177–193. Springer, 2009.

[28] Han Chen and Pasquale Malacaria. Quantifying maximal loss of anonymity in protocols. In Wanqing Li, Willy Susilo, Udaya Kiran Tupakula, Reihaneh Safavi-Naini, and Vijay Varadharajan, editors, *ASIACCS*, pages 206–217. ACM, 2009.

[29] Han Chen and Pasquale Malacaria. Studying maximum information leakage using karush-kuhn-tucker conditions. In Michele Boreale and Steve Kremer, editors, *SECCO*, volume 7 of *EPTCS*, pages 1–15, 2009.

[30] Tom Chothia and Apratim Guha. A statistical test for information leaks using continuous mutual information. In *CSF*, pages 177–190. IEEE Computer Society, 2011.

[31] Tom Chothia, Yusuke Kawamoto, and Chris Novakovic. A tool for estimating information leakage. In Sharygina and Veith [83], pages 690–695.

[32] Tom Chothia, Yusuke Kawamoto, Chris Novakovic, and David Parker. Probabilistic point-to-point information leakage. In *CSF*, pages 193–205. IEEE, 2013.

[33] David Clark, Sebastian Hunt, and Pasquale Malacaria. Quantitative analysis of the leakage of confidential data. *Electr. Notes Theor. Comput. Sci.*, 59(3):238–251, 2001.

[34] David Clark, Sebastian Hunt, and Pasquale Malacaria. A static analysis for quantifying information flow in a simple imperative language. *Journal of Computer Security*, 15(3):321–371, 2007.

[35] Michael R. Clarkson, Andrew C. Myers, and Fred B. Schneider. Quantifying information flow with beliefs. *Journal of Computer Security*, 17(5):655–701, 2009.

[36] T.M. Cover and J.A. Thomas. *Elements of Information Theory*. Wiley, 2012.

[37] George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a type iii anonymous remailer protocol. In *IEEE Symposium on Security and Privacy*, pages 2–15. IEEE Computer Society, 2003.

[38] Dorothy E. Denning. A lattice model of secure information flow. *Commun. ACM*, 19(5):236–243, 1976.

[39] Dorothy E. Denning. *Cryptography and Data Security*. Addison-Wesley, 1982.

[40] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In *USENIX Security Symposium*, pages 303–320. USENIX, 2004.

[41] Barbara Espinoza and Geoffrey Smith. Min-entropy as a resource. *Inf. Comput.*, 226:57–75, 2013.

[42] Vesa Halava, Tero Harju, Mika Hirvensalo, and Juhani Karhumäki. Skolem's problem - on the border between decidability and undecidability. Technical Report 683, 2005.

[43] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *SIGKDD Explorations*, 11(1):10–18, 2009.

[44] Martin E. Hellman and Josef Raviv. Probability of error, equivocation, and the chernoff bound. *IEEE Transactions on Information Theory*, 16(4):368–372, 1970.

[45] Jonathan Heusser and Pasquale Malacaria. Quantifying information leaks in software. In Carrie Gates, Michael Franz, and John P. McDermott, editors, *ACSAC*, pages 261–269. ACM, 2010.

[46] Andrew K. Hirsch and Michael R. Clarkson. Belief semantics of authorization logic. *CoRR*, abs/1302.2123, 2013.

[47] James W. Gray III. Toward a mathematical foundation for information flow security. In *IEEE Symposium on Security and Privacy*, pages 21–35, 1991.

[48] Edwin Thompson Jaynes. *Probability Theory: The Logic of Science*. Cambridge, 2003.

[49] S. Kiefer, A. S. Murawski, J. Ouaknine, B. Wachter, and J. Worrell. Apex: An analyzer for open probabilistic programs. In *Proc. CAV'12*, LNCS. Springer, 2012.

[50] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In Neal Koblitz, editor, *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[51] Paul C. Kocher, Joshua Jaffe, Benjamin Jun, and Pankaj Rohatgi. Introduction to differential power analysis. *J. Cryptographic Engineering*, 1(1):5–27, 2011.

[52] Boris Köpf and David A. Basin. Automatically deriving information-theoretic bounds for adaptive side-channel attacks. *Journal of Computer Security*, 19(1):1–31, 2011.

[53] Boris Köpf, Laurent Mauborgne, and Martín Ochoa. Automatic quantification of cache side-channels. In P. Madhusudan and Sanjit A. Seshia, editors, *CAV*, volume 7358 of *Lecture Notes in Computer Science*, pages 564–580. Springer, 2012.

[54] Boris Köpf and Geoffrey Smith. Vulnerability bounds and leakage resilience of blinded cryptography under timing attacks. In *CSF* [1], pages 44–56.

[55] Igor Kozine and Lev V. Utkin. Interval-valued finite Markov chains. *Reliable Computing*, 8(2):97–113, 2002.

[56] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Proc. CAV'11*, volume 6806 of *LNCS*. Springer, 2011.

[57] Jaisook Landauer and Timothy Redmond. A lattice of information. In *CSFW*, pages 65–70. IEEE Computer Society, 1993.

[58] Bing-Rong Lin and Daniel Kifer. A framework for extracting semantic guarantees from privacy. *CoRR*, abs/1208.5443, 2012.

[59] Bing-Rong Lin and Daniel Kifer. Information preservation in statistical privacy and bayesian estimation of unattributed histograms. In Kenneth A. Ross, Divesh Srivastava, and Dimitris Papadias, editors, *SIGMOD Conference*, pages 677–688. ACM, 2013.

[60] Pasquale Malacaria. Assessing security threats of looping constructs. In Martin Hofmann and Matthias Felleisen, editors, *POPL*, pages 225–235. ACM, 2007.

[61] Pasquale Malacaria. Algebraic foundations for information theoretical, probabilistic and guessability measures of information flow. *CoRR*, abs/1101.3453, 2011.

[62] Pasquale Malacaria and Han Chen. Lagrange multipliers and maximum information leakage in different observational models. In Úlfar Erlingsson and Marco Pistoia, editors, *PLAS*, pages 135–146. ACM, 2008.

[63] Pasquale Malacaria and Jonathan Heusser. Information theory and security: Quantitative information flow. In Alessandro Aldini, Marco Bernardo, Alessandra Di Pierro, and Herbert Wiklicky, editors, *SFM*, volume 6154 of *Lecture Notes in Computer Science*, pages 87–134. Springer, 2010.

[64] Ueli M. Maurer. Authentication theory and hypothesis testing. *IEEE Transactions on Information Theory*, 46(4):1350–1356, 2000.

[65] Stephen McCamant and Michael D. Ernst. Quantitative information flow as network flow capacity. In Rajiv Gupta and Saman P. Amarasinghe, editors, *PLDI*. ACM, 2008.

[66] John McLean. Security models and information flow. In *IEEE Symposium on Security and Privacy*, pages 180–189. IEEE Computer Society, 1990.

[67] S.P. Meyn and R.L. Tweedie. *Markov Chains and Stochastic Stability*. Cambridge Mathematical Library. Cambridge University Press, 2009.

[68] Silvio Micali and Leonid Reyzin. Physically observable cryptography (extended abstract). In Moni Naor, editor, *TCC*, volume 2951 of *Lecture Notes in Computer Science*, pages 278–296. Springer, 2004.

[69] Jonathan K. Millen. Covert channel capacity. In *IEEE Symposium on Security and Privacy*, pages 60–66. IEEE Computer Society, 1987.

[70] Chunyan Mu and David Clark. A tool: Quantitative analyser for programs. In *QEST* [2].

[71] James Newsome, Stephen McCamant, and Dawn Song. Measuring channel capacity to distinguish undue influence. In S. Chong and D. A. Naumann, editors, *PLAS*. ACM, 2009.

[72] Kevin R. O'Neill, Michael R. Clarkson, and Stephen Chong. Information-flow security for interactive programs. In *CSFW*, pages 190–201. IEEE Computer Society, 2006.

[73] Joël Ouaknine. Decision problems for linear recurrence sequences. In Leszek Gasieniec and Frank Wolter, editors, *FCT*, volume 8070 of *Lecture Notes in Computer Science*, page 2. Springer, 2013.

[74] Joël Ouaknine and James Worrell. Positivity problems for low-order linear recurrence sequences. In Chandra Chekuri, editor, *SODA*, pages 366–379. SIAM, 2014.

[75] Mads Tinggaard Pedersen. Automated information-theoretical evaluation of different anonymity protocols on coherent topologies. Master's thesis, IT University of Copenhagen, 2013.

[76] Quoc-Sang Phan, Pasquale Malacaria, Oksana Tkachuk, and Corina S. Pasareanu. Symbolic quantitative information flow. *ACM SIGSOFT Software Engineering Notes*, 37(6):1–5, 2012.

[77] Alessandra Di Pierro, Chris Hankin, and Herbert Wiklicky. Approximate non-interference. *Journal of Computer Security*, 12(1):37–82, 2004.

[78] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience, April 1994.

[79] Michael G. Reed, Paul F. Syverson, and David M. Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communications*, 16(4):482–494, 1998.

[80] Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for web transactions. *ACM Trans. Inf. Syst. Secur.*, 1(1):66–92, 1998.

[81] Nandakishore Santhi and Alexander Vardy. Analog codes on graphs. *CoRR*, abs/cs/0608086, 2006.

[82] Claude E. Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27:379–423, July 1948.

[83] Natasha Sharygina and Helmut Veith, editors. *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*. Springer, 2013.

[84] Geoffrey Smith. On the foundations of quantitative information flow. In Luca de Alfaro, editor, *FOSSACS*, volume 5504 of *Lecture Notes in Computer Science*, pages 288–302. Springer, 2009.

[85] Geoffrey Smith. Quantifying information flow using min-entropy. In *QEST* [2], pages 159–167.

[86] Geoffrey Smith and Rafael Alpízar. Non-termination and secure information flow. *Mathematical Structures in Computer Science*, 21(6):1183–1205, 2011.

[87] Dennis M. Volpano, Cynthia E. Irvine, and Geoffrey Smith. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(2/3):167–188, 1996.

[88] Hirotoshi Yasuoka and Tachio Terauchi. Quantitative information flow - verification hardness and possibilities. In *CSF* [1], pages 15–27.

[89] Hirotoshi Yasuoka and Tachio Terauchi. On bounding problems of quantitative information flow. *Journal of Computer Security*, 19(6):1029–1082, 2011.

Main Contributions

Hereafter we attach the following papers, in their most complete versions, which constitute the main contributions of this thesis:

Paper A

Fabrizio Biondi, Axel Legay, Bo Friis Nielsen, and Andrzej Wasowski. Maximizing entropy over Markov processes. *Journal of Logic and Algebraic Programming*, under revision

Paper B

Fabrizio Biondi, Axel Legay, Pasquale Malacaria, and Andrzej Wasowski. Quantifying information leakage of randomized protocols. In Roberto Giacobazzi, Josh Berdine, and Isabella Mastroeni, editors, *VMCAI*, volume 7737 of *Lecture Notes in Computer Science*, pages 68–87. Springer, 2013

Paper C

Fabrizio Biondi, Axel Legay, Louis-Marie Traonouez, and Andrzej Wasowski. QUAIL: A quantitative security analyzer for imperative code. In Sharygina and Veith [83], pages 702–707

Maximizing Entropy over Markov Processes

Fabrizio Biondi[a], Axel Legay[b], Bo Friis Nielsen[c], Andrzej Wąsowski[a]

[a] *IT University of Copenhagen, 7 Rued Langgaards Vej, 2300 Copenhagen-S, Denmark*
[b] *IRISA/INRIA Rennes, Campus de Beaulieu, 263 Avenue du Général Leclerc, 35042 Rennes, France*
[c] *Technical University of Denmark, Kongens Lyngby, Denmark*

Abstract

The channel capacity of a deterministic system with confidential data is an upper bound on the amount of bits of data an attacker can learn from the system. We encode all possible attacks to a system using a probabilistic specification, an Interval Markov Chain. Then the channel capacity computation reduces to finding a model of a specification with highest entropy.

Entropy maximization for probabilistic process specifications has not been studied before, even though it is well known in Bayesian inference for discrete distributions. We give a characterization of global entropy of a process as a reward function, a polynomial algorithm to verify the existence of a system maximizing entropy among those respecting a specification, a procedure for the maximization of reward functions over Interval Markov Chains and its application to synthesize an implementation maximizing entropy.

We show how to use Interval Markov Chains to model abstractions of deterministic systems with confidential data, and use the above results to compute their channel capacity. These results are a foundation for ongoing work on computing channel capacity for abstractions of programs derived from code.

1. Introduction

Context. Consider a common authentication protocol: a user is asked to enter a password. She is granted access to the system if the password corresponds to the one stored in the system. The goal of the protocol is to refuse access to users that do not know the password (the secret). The protocol should also prevent an attacker from guessing or learning the secret, by interacting with the system. We want to decide whether the protocol is secure or not against an attacker that wants to access it, but does not know the password. But what does it mean for a system to be secure?

☆Earlier versions of this paper appeared in the 24th Nordic Workshop on Programming Theory (2012), and in the 7th International Conference on Language and Automata Theory and Applications (2013).

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

Email addresses: fbio@itu.dk (Fabrizio Biondi), axel.legay@inria.fr (Axel Legay), bfn@imm.dtu.dk (Bo Friis Nielsen), wasowski@itu.dk (Andrzej Wąsowski)

One possible answer is that a system is secure if the attacker cannot authenticate in the system. This, however, cannot be guaranteed. There is a non-zero probability that an attacker is lucky and simply guesses the password. We could require that the attacker cannot learn the password by interacting with the system. However, a password is not an atomic object: would we consider the system secure if the attacker could learn 90% of the password? Learning 90% of the password significantly increases the attacker's chance of guessing the whole password. Finally, we could require that in a safe authentication protocol no attacker can learn anything about the password through interaction, the so called *non-interference* definition [17]. Unfortunately, this definition usually breaks in practice: if the attacker inserts a random password and gets rejected he will learn that the password is not the one he tried, breaking non-interference [30].

We believe that it is not possible to give a qualitative, yes-no definition of security that is not overly strict (rejects any authentication protocol) or overly permissive (accepts protocols compromising a significant amount of the password). The core problem is that even a secure protocol will allow the attacker to learn a small amount of information about the password [24].

Quantitative Information Flow techniques can be employed to precisely quantify the number of bits of information an attacker would gain about the confidential data of a system by interacting with the system and observing its behavior [11], leaving to the analyst to decide whether this amount is acceptable for the protocol analyzed.

The difference between the information, in Shannon's information-theoretical sense [31], that a given attacker possesses about the secret before and after a single attack is called *leakage* [11]. Different attackers would infer different amount of information. A possible quantitative definition of security of a system is the maximum leakage of information (so leakage towards the attacker that can infer the most information). Maximum leakage is known in security theory as *channel capacity*, ranging over all attackers with the same observational power but different prior information on the secret. As said, no single attack can leak an amount of information higher than the system's channel capacity [28]. For a deterministic system, the leakage is the entropy of the observable behavior of the system conditioned by the attacker's behavior [25], and thus computing the channel capacity reduces to computing the behavior of the system that maximizes entropy. This security guarantee is only valid for one process; any perturbation of the process' behavior would invalidate it.

Problem. The quantification of information leakage evaluates only the security of a completely defined system, since it needs to evaluate the likelihood and estimate the loss of confidential data of each possible user behavior and system's reaction to it. Alas, when a security protocol is designed the designer needs the freedom to leave some parts underspecified, for instance the size of a password or the number of usernames in the system. An underspecified system can be seen as the infinite union of all its implementations. Providing a security guarantee for it is challenging, as it requires computing the maximum leakage over all attacks by all possible attackers on all possible implementations.

Contribution. We will present a method to obtain a safety guarantee for a protocol specification when it is possible and signal that the protocol is unsafe otherwise. To the

best of our knowledge, no other approach to channel capacity computation for infinite classes of processes exists.

To execute a formal analysis of a process specification we need models for the processes and specifications and procedures to analyze them. Following the approach we introduced in [6] we use Markov chains (MCs) as process models, with the states of the MC representing the observable components of the system and the transition probabilities raising from the attacker's uncertainty about the secret. A single MC represents an attack scenario, in which a given attacker interacts with and observes a given deterministic system. The entropy of the MC for the scenario is the leakage from the system to the attacker.

In order to encode the infinite number of possible attackers and implementations, we need the continuity of real-valued transition probability intervals, so we use Interval Markov Chains (Interval MCs) [19] as specification models. We also show how to employ Markov Decision Processes (MDPs) [29] as specification models. We present algorithms that synthesize the process with maximum entropy among all those respecting a given probabilistic specification, allowing us to give a security guarantee valid for all the infinite processes respecting the specification, encoded as an Interval MC or as an MDP.

Specification models, and Interval MCs in particular, allow us to generalize the system-attacker scenario to all possible attackers interacting with all possible implementations of a specification. The Maximum Entropy resolution of the nondeterminism in the Interval MC is thus the scenario with the greatest leakage, and its entropy is the channel capacity of the specification and the maximum amount of information that can be leaked by any single attack to any given implementation of the specification.

Given a deterministic protocol specified as an Interval MC, we show how to:

1. *Compute the entropy of a given implementation.* We provide a polynomial-time procedure to compute entropy for a Markov chain, by reduction to computation of the Expected Total Reward [29, Chpt. 5] of a local non-negative reward function associated with states over the infinite horizon.

2. *Check if a protocol allows for insecure implementations.* We provide a polynomial-time procedure for deciding finiteness and boundedness of the entropy of all implementations of an Interval MC. In general a Maximum Entropy implementation might not exist. Implementations might be non-terminating and accumulate infinite entropy, or they may have arbitrarily high, i.e. unbounded, finite entropy, so standard optimization techniques would diverge. In such case it is not possible to give a security guarantee for the implementations. We provide a polynomial-time algorithm to distinguish the two cases. If a protocol allows implementations with infinite or unbounded entropy then no matter the size n of the secret, it will have an implementation leaking all n bits of it. We detect this so that the designer can strengthen the protocol design appropriately.

3. *Compute channel capacity of a protocol.* This is a multidimensional nonlinear maximization problem on convex sets [33]. We use a numerical procedure for synthesizing with arbitrary approximation an implementation maximizing a reward function over Interval MCs. An Interval MC can be considered as an infinite set of

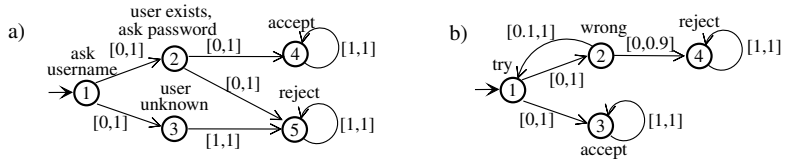


Figure 1: a) Two-step Authentication b) Repeated Authentication

processes, and since entropy is a nonlinear function of all possible behaviors of a system, finding the one with highest entropy is not trivial. We apply this procedure to synthesize a Maximum Entropy process implementing an Interval MC; the entropy of such a process is the channel capacity of all processes implementing the Interval MC.

On the theoretical level, this paper extends the well know Maximum Entropy Principle of Jaynes [18], from constraints on probability distributions to interval constraints on discrete probabilistic processes. We consider Interval MCs as constraints over probabilistic processes and resolve them for maximum entropy. As a result we validate the intuition that channel capacity computation as known in security research corresponds to obtaining least biased solutions in Bayesian inference for processes.

Examples. Consider two examples of models of deterministic authentication processes. Figure 1a presents an Interval MC for a two-step authentication protocol with username and password. The actual transition probabilities will depend on how many usernames exist in the system, on the respective passwords, on their length and on the attacker's knowledge about all of these; but staying at the specification level allows us to consider the worst case of all these possible combinations, and thus gives an upper bound on the leakage. The Maximum Entropy implementation for the Two-step Authentication is given in Fig. 2; its entropy is the channel capacity of the system over all possible prior informations and behaviors of the attacker and design choices of the implementer.

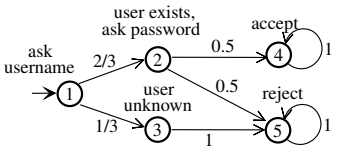


Figure 2: Maximum Entropy implementation for the Two-step Authentication

Consider another example. Figure 1b presents an Interval MC specification of the Repeated Authentication protocol, in which the user inserts a password to authenticate, and is allowed access if the password is correct. If not, the system verifies if the password entered is in a known black list of common passwords, in which case it rejects the user, considering it a malicious attacker. If the password is wrong but not black listed the user is allowed to try again. The black list cannot cover more than 90% of the possible passwords, and we assume that the attacker has no information about it.

Note that the transition probabilities from state 2 is a design choice left to the implementer of the system, while the transition probabilities from state 1 depend on a given attacker's knowledge about the password, and thus cannot be controlled even by the implementer. By abstracting these two different sources of nondeterminism at the

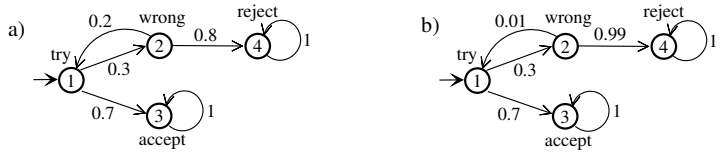


Figure 3: a) Correct implementation of the Repeated Authentication. b) Incorrect implementation of the Repeated Authentication.

same time we maximize entropy over all possible combinations of design choices and attackers, effectively finding the channel capacity of the specification.

The implementation maximizing entropy is the one revealing most information about the system's secret. This is consistent with our intuition. For instance, if the black list is empty the user can continue guessing the password indefinitely: the probability of eventually reaching state 3 is 1. In this implementation sooner or later the attacker will discover the password, and thus the system's secret will be completely revealed.

Figure 3 shows two implementations of the Repeated Authentication presented in Fig. 1a. None of them maximizes entropy. In fact, it is not possible to give a Maximum Entropy implementation for such protocol; this means that whatever the length n of the secret, it is possible to give an implementation that leaks all n bits of information. We will discuss the significance of this in Sect. 7. The Maximum Entropy implementation for the Two-step Authentication is given in Fig. 2. We explain how it has been synthesized in Sect. 6.

The paper is structured as follows. We introduce terminology and notation in Sect. 3. Section 4 recalls the definition of entropy for MCs, and relates it to abstract specifications. In Sect. 5 we introduce algorithms computing entropy of a given MC. In Sect. 6 we find the implementation of an Interval MC maximizing the entropy. In Sect. 7 we deal with Interval MCs that allow for implementations with infinite or unbounded entropy. In Sect. 8 we discuss the use of MDPs as specification models. We conclude in Sect. 9.

2. State of the Art and Related Work

This paper appeared previously as an extended abstract in the 24th Nordic Workshop on Programming Theory (2012) and subsequently in the conference version in the 7th International Conference on Language and Automata Theory and Applications (2013). This considerably extended version features extended introduction and related work sections, a section on the application of the technique described to Markov Decision Processes and a conclusion section. We also added detailed proof to all lemmata and theorems, some definition and many explainations and intuitions to clarify the ideas behind the paper.

The information-theoretical approach to system security that we consider here follows Clark et al. [11, 13] in using the difference between the Shannon entropy of the probability distribution over the secret before and after the attack as a measure

for insecurity. Among the other measures vulnerability has been shown by Smith to have interesting qualities when considering a single attack by an attacker with no prior information about the secret [32], and Köpf et al. show how to use probabilistic measures to quantify the security of side channels [20, 21]. Providentially, for deterministic protocols the orderings induced by all measures of insecurity coincide [25, 35], showing the robustness of the approach. Recently, Alvim et al. proposed to generalize the approach to generic gain functions depending on properties of the protocol, and also introduced an ordering based on channel factorization that is conjectured to extend the ordering to the probabilistic case [4].

The ordering of the attackers according to their potential for inferring information about the secret forms a lattice, known as the Lattice of Information [23]. The states of the Markov chains we use to model protocols represent observables, i.e. classes of equivalence of the possible models according to the attacker's observational power; we refer to [6] for the details. This use of equivalence relation is closely related to the application of abstract interpretation to security [1, 15, 27], even though they have been applied to obtain or approximate non-interference. The union of abstract interpretation and information theory would be a natural and promising development in the field of quantitative information flow.

Channel capacity as a security guarantee [28] has been studied for many different models of computation. Chatzikokolakis, Palamidessi and Panangaden use it to give a formula for anonymity analysis of protocols described by weakly symmetric matrices [9], based on the probabilistic anonymity approach by Bhargava and Palamidessi [5]. Chen and Malacaria generalize this result to asymmetric protocols [10] and also study the channel capacity of deterministic systems under different observation models [26]. Our method, unlike theirs, handles infinite classes of models instead of single models, and uses different states of a Markov chain to represent the different logical states of the system instead of considering the system as a function from inputs to outputs.

3. Background on Probabilistic Processes

We often refer to deterministic, stochastic and nondeterministic behavior. We use the adjective *deterministic* for a completely predictable behavior, *stochastic* for a behavior that follows a probability distribution over some possible choices, and *nondeterministic* for a choice where no probability distribution is given.

3.1. Markov Chains

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

We slightly misuse notation, interpreting states as natural numbers and indexing matrices with state names. This is reflected in figures by labeling of states both with textual descriptions and numbers.

A state is *deterministic* if it has exactly one outgoing transition with probability 1, *stochastic* otherwise. It is known [12] that 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 matrix P^k. 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 t is *reachable* from a state s if $\exists k. P_{s,t}^k > 0$. We assume that all states are reachable from s_0 in the MCs considered. A subset $R \subseteq S$ is *strongly connected* if for each pair of states $s, t \in R$, t is reachable from s.

For the purpose of defining entropy, it is useful to consider the alternative, more probabilistic view of an MC. An MC can be seen as an infinite sequence of discrete random variables $(X_n, n \in \mathbb{N})$, where $\mathbf{P}(X_k = s) = \pi_s^{(k)}$ represents the probability that the chain will be visiting state $s \in S$ at time k. The sequence of random variables must respect the *Markov property*: $P(X_n = s_n \mid X_{n-1} = s_{n-1}, \ldots, X_0 = s_0)$=$P(X_n = s_n \mid X_{n-1} = s_{n-1})$, $\forall s_0, s_1, ..., s_n \in S, n \in \mathbb{N}$, meaning that the probability distribution over the states at a given time n depends only on the distribution at time $n-1$ and is independent from the distributions at any previous time.

For a state s of a Markov chain, the *expected residence time* denotes the expected amount of time that the Markov chain will spend in state s, while the *first hitting probability* of state t after state s denotes the probability that state t will ever be visited under the condition that state s is being visited.

Definition 2. The *expected residence time* in a state s is defined as

$$\xi_s = \sum_{n=0}^{\infty} P_{s_0,s}^n$$

Definition 3. The *first hitting probability* of state t after state s is defined as

$$\rho(s,t) = \sum_{n=1}^{\infty} \mathbf{P}(X_n = t, X_{n-1} \neq t, X_{n-2} \neq t, ..., X_1 \neq t \mid X_0 = s)$$

We will sometime write ρ_s for $\rho(s_0, s)$. Note that $\xi_s = \rho_s \sum_{n=0}^{\infty} P_{s,s}^n$. A state s is *recurrent* iff $\xi_s = \infty$, *transient* otherwise.

We show how the expected residence time of all states in a Markov chain can be computed in polynomial time by reducing the problem to solving a system of linear equations. By definition expected residence time is infinite for each recurrent state, so we first check which states are recurrent, and then we calculate the expected residence time for the remaining (transient) states. We say that two states $s, t \in S$ *communicate*, written $s \leftrightarrow t$, if s is reachable from t and vice versa. Note that $s \leftrightarrow s$. Communication is an equivalence relation and as such it induces *communicating classes* $C(s) = \{t \in s \mid s \leftrightarrow t\}$. A communicating class $C(s)$ is said to be *closed* if $\mathbf{P}(X_{k+1} \in C(s) \mid X_k \in C(s)) = 1$. Whether a communicating class is closed can be decided in polynomial time.

A Markov chain can be divided in its communicating classes by finding its Strongly Connected Components with e.g. Tarjan's algorithm [34]. It is known [12] that all states in a closed communicating class of a finite Markov Chain are recurrent, while all other states are transient, so we can divide the states in recurrent and transient states in polynomial time. Let R be the set of the recurrent states; then $\forall s \in R. \xi_s = \infty$.

The expected residence time of the transient states can be found by solving the system of equations $\xi_t = \pi_t^{(0)} + \sum_{u \in S \setminus R} P_{u,t} \xi_u$ for all $t \in S \setminus R$ (in polynomial time).

Use of Markov chains to model generic secret-dependent processes has been previously introduced in [6], including ways to automatically generate them from imperative program code. Each state of the MC represents a reachable combination of values of the public variables of the system and levels of knowledge about the private variables. We refer to [6] for the full discussion.

3.2. Interval Markov Chains

Definition 4. *[8]* A closed-interval *Interval Markov Chain* (Interval MC) is a tuple $\mathcal{I} = (S, s_0, \check{P}, \hat{P})$ where S is a finite set of states containing the initial state s_0, $\check{P}$ is an $|S| \times |S|$ bottom transition probability matrix, $\hat{P}$ is a $|S| \times |S|$ top transition probability matrix, such that for each pair of states $s, t \in S$ we have $0 \leq \check{P}_{s,t} \leq \hat{P}_{s,t} \leq 1$.

The following defines when an MC implements an Interval MC in the Uncertain Markov Chain (UMC) semantics [8]:

Definition 5. A Markov chain $\mathcal{C} = (S, s_0, P)$ *implements* an Interval Markov Chain $\mathcal{I} = (S, s_0, \check{P}, \hat{P})$, written $\mathcal{C} \vDash \mathcal{I}$, if $\forall s, t \in S.\ \check{P}_{s,t} \leq P_{s,t} \leq \hat{P}_{s,t}$.

An example of an Interval MC is the Repeated Authentication of Fig. 1b. The MC in Fig. 3a uses a black list covering 80% of the passwords and is thus an implementation of the Interval MC, while the Markov chain in Figure 3b uses a black list covering 99% of the passwords and it is not an implementation of the Interval MC.

We assume without loss of generality that our Interval MCs are *coherent*, meaning that every value for each transition interval is attained by some implementation. Coherence can be established by checking that both following conditions hold [22]:

1. $\forall s, t \in S. \check{P}_{s,t} \geq (1 - \sum_{u \neq t} \hat{P}_{s,u})$
2. $\forall s, t \in S. \hat{P}_{s,t} \leq (1 - \sum_{u \neq t} \check{P}_{s,u})$

Any Interval MC $\mathcal{I} = (S, s_0, \check{P}, \hat{P})$ can be transformed into a coherent Interval MC $\mathcal{I}' = (S, s_0, \check{P}', \hat{P}')$ by changing the top and bottom transition probability matrices to the following:

1. $\check{P}'_{s,t} = \max(\check{P}_{s,t}, 1 - \sum_{u \neq t} \hat{P}_{s,u})$
2. $\hat{P}'_{s,t} = \min(\hat{P}_{s,t}, 1 - \sum_{u \neq t} \check{P}_{s,u})$

The resulting coherent Interval MC $\mathcal{I}'$ is unique and has the same implementations as the original incoherent Interval MC $\mathcal{I}$ [22], so in particular it has an implementation iff $\mathcal{I}$ has at least one implementation. It is also interesting to note that if an Interval MC is coherent then it is possible to give an implementation in which a positive probability is assigned to all transitions for which this is allowed by the Interval MC, as stated by the following lemma.

Lemma 1. *Let $\mathcal{I} = (S, s_0, \check{P}, \hat{P})$ be a coherent Interval MC. Then there exists an implementation $\mathcal{C} = (S, s_0, P)$ of $\mathcal{I}$ in which $\forall s, t \in S. \hat{P}_{s,t} > 0 \Rightarrow P_{s,t} > 0$.*

Proof. [1] We recall the coherence assumptions:

1. $\forall s,t \in S. \check{P}_{s,t} \geq (1 - \sum_{u \neq t} \hat{P}_{s,u})$
2. $\forall s,t \in S. \hat{P}_{s,t} \leq (1 - \sum_{u \neq t} \check{P}_{s,u})$

Assume with no loss of generality that $\forall s,t \in S. \hat{P}_{s,t} > 0 \Rightarrow \check{P}_{s,t} < \hat{P}_{s,t}$.

We will proceed ad absurdum. Let's consider a state $s \in S$ and assume there is no implementation of $\mathcal{I}$ in which $\forall t \in S. \hat{P}_{s,t} > 0 \Rightarrow P_{s,t} > 0$, meaning in all implementations there exists at least one state t' such that $\hat{P}_{s,t'} > 0 \wedge P_{s,t'} = 0$. Let $\mathcal{C} = (S, s_0, P)$ be one such implementation. Clearly it must be $\check{P}_{s,t'} = 0$. Since the implementation is a Markov chain it holds that $\sum_{t \in S} P_{s,t} = 1$ thus $\sum_{\substack{t \in S \\ t \neq t'}} P_{s,t} = 1$. Let's assume that $\forall u \neq t'. P_{s,u} = \check{P}_{s,u}$; we have that $1 - \sum_{u \neq t} \check{P}_{s,u} = 1 - \sum_{u \neq t} P_{s,u} = 0$, but this is not possible since $\hat{P}_{s,t'} > 0$ by assumption and $\hat{P}_{s,t'} \leq (1 - \sum_{u \neq t'} \check{P}_{s,u})$ by coherence. Thus there must be a state $t'' \in S$ such that $t'' \neq t'$ and $P_{s,t''} > \check{P}_{s,t''}$.

But then consider a positive constant ε such that $\varepsilon < \hat{P}_{s,t'}$ and $\varepsilon < P_{s,t''} - \check{P}_{s,t''}$; the implementation with transition probabilities P' defined as

$$P'_{s,t} = \begin{cases} \varepsilon & \text{if } t = t' \\ P_{s,t''} - \varepsilon & \text{if } t = t'' \\ P'_{s,t} & \text{otherwise} \end{cases}$$

is an implementation of $\mathcal{I}$ in which $\forall t \in S. \hat{P}_{s,t} > 0 \Rightarrow P_{s,t} > 0$, contradicting the assumption that no such implementation exists. □

We will use this result later in the proof of Theorem 2.

A state s of an Interval MC is *deterministic* if $\exists t. \check{P}_{s,t} = 1$, *stochastic* otherwise. We say that a state t is *reachable* from a state s if $\exists s_1, s_2, ..., s_n \in S. s_1 = s \wedge s_n = t \wedge \hat{P}_{s_i, s_{i+1}} > 0$ for $1 \leq i < n$. We say that a subset $R \subseteq S$ is *strongly connected* if $\forall s,t \in R$. t is reachable from s.

Note that if there is an implementation in which a subset of states $R \subseteq S$ is strongly connected, then R must be strongly connected in the Interval MC.

3.3. Markov Decision Processes

Definition 6. A *Markov Decision Process* (MDP) [29] 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 write $P_{(s,a),t} = x$ for $P(s,a,t) = x$.

[1] We extend our thanks to Lei Song for the proof of this lemma.

An MDP makes a nondeterministic choice of an action in each state. A selection of an action determines a probability distribution over the successor states. A *probabilistic strategy* is a function $\sigma : S \to \delta(\Lambda_s)$ assigning to a state s a probability distribution over the actions in Λ_s. A probabilistic strategy is *positional* if its probability distributions assign probability 1 to some action. The (infinite) set of strategies of the process $\mathcal{P}$ is denoted $\Sigma_{\mathcal{P}}$.

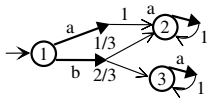


Figure 4: A simple MDP

In the MDP depicted in Fig. 4 there are two possible positional strategies, namely the one choosing action *a* in state **1** and transitioning to state **2** with probability 1, and the one choosing action *b* and transitioning to state **2** with probability $\frac{1}{3}$ and to state **3** with probability $\frac{2}{3}$. There is also one probabilistic strategy for each probability distribution over the two actions.

A strategy resolves an MDP into an MC, and in that sense MDPs, similarly to Interval MCs, can also be used as specifications for MCs; every strategy defines a different MC.

4. Entropy of Processes and Specifications

The entropy of a discrete probability distribution quantifies lack of information about the events involved. This idea can be extended to quantify nondeterminism, understood as degree of unpredictability of an MC.

Definition 7. Let X be a random variable on the discrete domain $(x_1, ..., x_n)$ and $\mathbf{P}(X = x_k)$ its probability distribution. Then the *entropy* of the random variable X is defined as

$$H(X) = -\sum_{i=1}^{n} \mathbf{P}(x_i) \log_2(\mathbf{P}(x_i))$$

Entropy is always non-negative. It is maximal for the uniform distribution, in which case its value is $\log_2 n$ [12], representing complete ignorance about the value assumed by the random variable. On the other extreme, $H(X)$ is 0 if and only if the probability distribution on X assigns a given value to it with probability 1. For MCs, entropy is maximum for the process in which all possible paths in the chain have the same probability.

To define the entropy of a Markov chain $\mathcal{C}$ we need to introduce the concepts of *conditional entropy* and *joint entropy* [12, 31]. Conditional entropy quantifies the remaining entropy of a variable Y given that the value of other random variables (X_i here) is known.

$$H(Y \mid X_1, \ldots, X_n) = -\sum_{t \in S} \sum_{s_1 \in S} \cdots \sum_{s_n \in S} \mathbf{P}(Y = t, X_1 = s_1, \ldots, X_n = s_n) \cdot \\ \cdot \log_2 \mathbf{P}(Y = t \mid X_1 = s_1, \ldots, X_n = s_n) \ ,$$

where $\mathbf{P}(Y = t, X_1 = s_1, \ldots, X_n = s_n)$ denotes the joint probability of the events $Y = t$, $X_1 = s_1, \ldots, X_n = s_n$.

Joint entropy is simply the entropy of several random variables computed jointly, i.e. the combined uncertainty due to the ignorance of n random variables. It turns out [12] that joint entropy can be calculated in the following way, using conditional entropy, which will be instrumental in our developments for MCs.

$$H(X_0, X_1, \ldots, X_n) = -\sum_{s_0 \in S} \sum_{s_1 \in S} \cdots \sum_{s_n \in S} \mathbf{P}(X_0 = s_0, X_1 = s_1, \ldots, X_n = s_n) \cdot \\ \cdot \log_2(\mathbf{P}(X_0 = s_0, X_1 = s_1, \ldots, X_n = s_n)) = \\ = H(X_0) + H(X_1 \mid X_0) + \cdots + H(X_n \mid X_0, X_1, \ldots, X_{n-1})$$

Now, the definition of entropy of an MC is unsurprising, if we take the view of the processes as a series of random variables; it is the joint entropy of these variables (recall that due to the Markov property, the automata-view, and the probabilistic view of MCs are interchangeable):

Definition 8. We define the entropy of a Markov chain $\mathcal{C} = (X_n, n \in \mathbb{N})$ as the joint entropy over all X_n: $H(\mathcal{C}) = H(X_0, X_1, X_2, \ldots) = \sum_{i=0}^{\infty} H(X_i \,|X_{i-1} \ldots X_0)$.

Note that since we have assumed a single starting state in each MC, it is always the case that $H(X_0) = 0$. Also the above series always converges to a real number, or to infinity, since it is a sum of non-negative real numbers.

In leakage analysis, entropy corresponds to the information leakage of the system only when the system is deterministic and the attacker cannot interact with it [25]. Using probability intervals we can lift the latter restriction, as different distributions on the attacker's input would only lead to different transition probabilities, and the intervals already consider all possible transition probabilities.

The entropy of an MC is in general infinite. In the next Section we will show that the entropy of an MC is finite if and only if the chain is absorbing. Considering only absorbing MCs avoids the problem of the entropy of an MC being in general infinite. We always consider terminating protocols, and they can be encoded as absorbing MCs where the absorbing states represent the termination of the protocol; consequently the entropy of a Markov chain encoding a terminating process is always finite.

We stress that it is common [16, 31] to compute the average entropy of each step of the MC and to call it the entropy of the MC. In our opinion, technically speaking this notion should be more precisely named an entropy rate [12]. Here we want to compute the actual entropy since the entropy, not its rate, represents the information leakage in a security scenario where the states of the MC are the observables of a deterministic protocol.

An alternative characterization of entropy of a process depends explicitly on which states get visited during the lifetime of the process. Since every state s has a probability distribution over the next state we can compute the entropy of that distribution, which we will call *local entropy* $L(s)$ of s: $L(s) = H(X_{k+1} \mid X_k = s) = -\sum_{t \in S} P_{s,t} \log_2 P_{s,t}$. Note that $L(s) \leq \log_2(|S|)$. Also, in general, the value of entropy of an MC is not equal to the sum of the local entropy values for each state. Such sum will have to be weighted against the expected residence time of each state to characterize the "global" entropy.

Now consider the Interval MC specification of the Repeated Authentication protocol in Fig. 1b. Different implementations of it, like the ones in Fig. 3ab, will have different

entropy values. We define a *Maximum Entropy implementation* for an Interval MC, as an implementation MC, which has entropy not smaller than entropy of any other implementation (if such exists):

Definition 9. Let $\mathcal{I}$ be an Interval Markov Chain. Then a Markov chain $\mathcal{C}$ is a *Maximum Entropy implementation* of $\mathcal{I}$ if the following holds:

1. $\mathcal{C} \models \mathcal{I}$
2. $H(\mathcal{C})$ is finite
3. $\forall \mathcal{C}'.\mathcal{C}' \models \mathcal{I} \Rightarrow H(\mathcal{C}') \leq H(\mathcal{C})$

The boundedness and synthesis of a Maximum Entropy implementation of an Interval MC will be treated in Sect. 6. It may be that the maximum entropy is actually infinite, or that the set of attainable entropies is unbounded; we discuss these cases in Sect. 7.

5. Computing Entropy of Markov Chains

We now provide an algorithm for computing entropy of a given MC. We cast entropy as a non-negative reward function on an MC, and then apply standard techniques to compute it. We also provide a simple decision procedure for deciding whether entropy of an MC is finite.

A *non-negative reward function* over the transitions of an MC is a function $R : S \times S \to \mathbb{R}^+$ assigning a non-negative real value, called reward, to each transition. Given a reward function R we can compute the value of the reward for a concrete execution of an MC by summing reward values for the transitions exercised in the execution. More interestingly, we can compute the expected reward of each state $s \in S$ as $R(s) = \sum_{t \in S} P_{s,t} R_{s,t}$, and then the expected reward over the infinite behavior of an MC $\mathcal{C}$ is $R(\mathcal{C}) = \sum_{s \in S} R(s)\xi_s$ [29, Chpt. 5].

Each $R(s)$ can be computed in time linear in the number of states, so calculation of expected rewards for all states can be done in quadratic time. Since computing the expected residence time for a state s is in PTIME as shown in Sect. 3, we can also compute $R(\mathcal{C})$ in polynomial time.

Let the reward function be $R(s,t) = -\log_2 P_{s,t}$. Then the expected reward for each state is its local entropy, or $R(s) = -\sum_{t \in S} P_{s,t} \log_2 P_{s,t} = L(s)$. Note that this is an unorthodox non-negative reward function, since the reward for choosing a transition in a given state is not a constant but depends on the probability distribution over the successors of the state, as a function of the form $R : S \times S \times P \to \mathbb{R}^+$. It turns out that the (global) entropy of the MC is the expected reward with this reward:

Theorem 1. *For an MC* $\mathcal{C} = (S, s_0, P)$ *we have that* $H(\mathcal{C}) = \sum_{s \in S} L(s)\xi_s$.

Proof. Consider the chain rule for conditional entropy:

$$\begin{aligned}
&H(X_n|X_1, X_2, ..., X_{n-1}) = \\
&= -\sum_{t\in S}^{N}\sum_{s_0\in S}^{N}\cdots\sum_{s_{n-1}\in S}^{N} P(X_n = t, X_0 = s_0, ..., X_{n-1} = s_{n-1})\cdot \\
&\cdot \log(P(X_n = t|X_1 = s_1, ..., X_{n-1} = s_{n-1})) \\
&= \sum_{s\in S} \pi_s^{(n-1)} H(X_n|X_{n-1} = s) = \sum_{s\in S} \pi_s^{(n-1)} L(s)
\end{aligned}$$

Where $\pi_s^{(0)}$ is 1 if $s = s_0$ and 0 otherwise. We can see that the entropy at time n is the sum on each state of the local entropy of the state multiplied by the probability to be in that state at time $n-1$. Applying this to the chain rule for joint entropy over an infinite time we have that

$$\begin{aligned}
H(X_0, X_1, X_2, ...) &= H(X_0) + \sum_{n=1}^{\infty} H(X_n|X_0, X_1, ..., X_{n-1}) \\
&= H(X_0) + \sum_{n=1}^{\infty}\sum_{s\in S} \pi_s^{(n-1)} L(s) \\
&= H(X_0) + \sum_{s\in S} L(s) \sum_{n=1}^{\infty} \pi_s^{(n)} \\
&= H(X_0) + \sum_{s\in S} L(s) \sum_{n=1}^{\infty} P_{s_0,s}^{n} \\
&= \sum_{s\in S} L(s) \sum_{n=0}^{\infty} P_{s_0,s}^{n} = \sum_{s\in S} L(s)\xi_s
\end{aligned}$$

□

As any other reward of this kind, the entropy of an MC can be infinite. Intuitively, the entropy is finite if it almost surely stops increasing. This happens if the execution is eventually confined to a set of states with zero local entropy (deterministic). Since the recurrent states of a chain are exactly the ones that are visited infinitely often, we obtain the following characterization:

Corollary 1. *The entropy $H(\mathcal{C})$ of a chain $\mathcal{C}$ is finite iff the local entropy of all its recurrent states is zero.*

The above observation gives us an algorithmic characterization of finiteness of entropy for MCs: the entropy of a chain is finite if and only if the chain has one or more absorbing states or absorbs into closed deterministic cycles. Entropy can only be infinite for infinite behaviors; for the first n execution steps the entropy is always bounded by $n \log_2 |S|$.

We can classify the processes in two categories: those which eventually terminate the stochastic behavior, and those which do not. Many processes become deterministic

(or even terminate) after some time. This is the case for a terminating algorithm like a randomized primality test, or for randomized IP negotiation protocols like Zeroconf, which stops behaving randomly as soon as an IP number is assigned. Such processes have finite entropy. On the other hand, the processes that take probabilistic choices forever and never become deterministic have infinite entropy. Using Corollary 1 we characterize the processes having finite entropy as terminating and the processes having infinite entropy as non-terminating. In this sense we consider a process as terminated when its behavior will be forever deterministic, as it will not reveal any more information about the secret.

6. Maximum Entropy Implementation of an Interval MC

Interval MCs describe infinite sets of MCs. We now show how to find an implementation that maximizes entropy, in the sense of Definition 9. Since our Markov chains represent the behavior of deterministic processes, the Maximum Entropy implementation we synthesize is also the one with maximum leakage, and its leakage is thus the channel capacity of all implementations.

In Fig. 2 we show the Maximum Entropy implementation of the Two-step Authentication specification in Fig. 1a. Its entropy is $\log_2 3 \approx 1.58496$ bits. This allows us to guarantee that none of the infinite possible implementations of the Two-step Authentication will leak more than $\log_2 3$ bits of information to any possible attacker.

Obtaining a Maximum Entropy implementation is a challenging problem. In the first place, such an implementation may not exist, as the set of attainable entropies may be unbounded, so we need an algorithm to verify its existence. Secondly, even if it exists finding it consists of solving a nonlinear optimization problem with constraints over an infinite domain.

We propose a numerical approach to the general problem of solving Interval MCs for non-negative reward functions, and apply it to finding a Maximum Entropy implementation. We first check that such an implementation exists, and then proceed to synthesize it. Remember that an implementation maximizing the expected total reward for the reward function $R(s,t) = -\log_2(P_{s,t})$ is a Maximum Entropy implementation.

The expected reward of a reward function may be infinite. An Interval MC admits implementations with infinite entropy if it has a state that can be recurrent and stochastic in the same implementation. We call this the *infinite* case.

If an Interval MC has a state that is recurrent and deterministic in some implementations and transient and stochastic in some others, but never both recurrent and stochastic in the same implementation, the set of entropies of its implementations is unbounded, despite all the individual implementations having finite entropy; an example is the Repeated Authentication in Fig. 1b. We call this the *unbounded* case. This happens because the reward assigned to a transition is not a constant, but a logarithmic function of the actual transition probability. With such reward it is possible that the total reward value can be unbounded across possible implementations (not just finite or infinite as for classical non-negative rewards). Note that this does not happen with constant rewards, and is specific to our problem.

6.1. Existence of a Maximum Entropy Implementation

We now show an algorithm for determining whether an Interval MC has a Maximum Entropy implementation with finite entropy. To do this we first give a definition of end components [2, 3] for Interval MCs. Then we show the algorithm for deciding the existence of a Maximum Entropy implementation.

An end component is a set of states of the Interval MC for which there exists an implementation such that once the behavior enters the end component it will stay inside it forever and choose all transitions inside it an infinite number of times with probability 1. We refer to [2] for further discussion.

Definition 10. Given an Interval MC $\mathcal{I} = (S, s_0, \check{P}, \hat{P})$, a set of states $R \subseteq S$ is an *end component of* $\mathcal{I}$ if the following holds:

1. R is strongly connected;
2. $\forall s \in R, t \in S \backslash R. \check{P}_{s,t} = 0$;
3. $\forall s \in R. \sum_{u \in R} \hat{P}_{s,u} \geq 1$.

Corollary 2. *For an Interval MC $\mathcal{I} = (S, s_0, \check{P}, \hat{P})$, if $R \subseteq S$ is an end component of $\mathcal{I}$ then there is an implementation of $\mathcal{I}$ in which $\boldsymbol{P}(X_{n+1} \notin R \mid X_n \in R) = 0$.*

An end component is maximal if no other end component contains it. In the Interval MC pictured in Fig. 5 we have that $\{1, 2\}$ is an end component, $\{1, 3\}$ is an end component, and $\{1, 2, 3\}$ and $\{4\}$ are maximal end components.

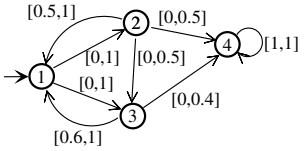


Figure 5: Interval MC with multiple end components

Algorithm 1 finds all maximal end components of an Interval MC. It first identifies all candidate end-components and their complement—the obviously transient states; then it propagates transient states backwards to their predecessors who cannot avoid reaching them. The predecessors are pruned from the candidate end-components and the procedure is iterated until a fixed point is reached.

Lemma 2. *Algorithm 1 runs in polynomial time, and upon termination the states of the Interval MC are tagged as* ENDCOMPONENTSTATE *if they are part of any maximal end component, and tagged as* TRANSIENT *otherwise.*

Proof. Clearly the algorithm always terminates, and when it does no state is tagged UNCHECKED.

We show that a state t is tagged as TRANSIENT iff there is no maximal end component containing it. If t is not in any strongly connected component, it is obvious. Note that if C is a SCC then there is no end component that contains both states inside C and outside C. If t is in a SCC C at the beginning of the loop, but not after the end of the algorithm, it was removed in some iteration of the loop; remember that the two checks that are performed for each state check that there is at least one implementation such that $\mathbf{P}(X_{n+1} \notin R | X_n = t)$. Let's consider the iteration in which t was removed. If it was the first state to be removed from C, it means that for any implementation there is

Tag all states of S as UNCHECKED;
Find the strongly connected components (SCCs) of the Interval MC (e.g. with Tarjan's algorithm) and tag any state not in any SCC as TRANSIENT;
repeat
  foreach *SCC* C **do**
    Select a state $s \in C$ tagged UNCHECKED;
    Check that $\forall t \in S \backslash C . \hat{P}_{s,t} = 0$. If not, remove s from C, tag it TRANSIENT, tag UNCHECKED all states in C with a transition to s and select another state;
    Check that $\sum_{u \in C} . \hat{P}_{s,u} \geq 1$. If not, remove s from C, tag it TRANSIENT, tag UNCHECKED all states in C with a transition to s and select another state;
    Tag s as ENDCOMPONENTSTATE;
  end
until *all states in any non-empty SCC are tagged as* ENDCOMPONENTSTATE;

Algorithm 1: Find all maximal end components of an Interval MC.

a positive probability of transitioning from t to a state outside C, so t must be transient in any implementation. Thus we can assume by inductive hypothesis that any state removed from C at a certain iteration k is necessarily transient; then if t gets removed after iteration k it means that it has a positive probability to go to a state $u \notin C$ or to a state that has been removed from C in a previous step; in both cases t has necessarily positive probability to go to a transitive state, and is thus transitive.

Since we proved that no states are tagged UNCHECKED after running the algorithm and that states are tagged TRANSITIVE iff they are not in any maximal end component, necessarily states are tagged ENDCOMPONENTSTATE iff they are in some maximal end component.

COMPLEXITY: Let $n = |S|$. Tarjan's algorithm runs in time quadratic in n. Each time a state is selected in the loop it is tested in time linear in n, and each state gets visited at most $O(n^2)$ times, so the total running time of the algorithm is $O(n^3)$. □

Algorithm 2 establishes finiteness of Maximum Entropy across all implementations of an Interval MC. After finding the maximal end components we check whether each end component state in $\mathcal{I}$ is deterministic. Because the Interval MC is coherent, this check simply amounts to verifying that for each state in each end component there is a successor state with lower bound on transition probability being 1. If this is the case, then there exists a Maximum Entropy implementation for $\mathcal{I}$ with a finite entropy value.

Algorithm 2 for deciding existence of finite maximum entropy implementation is sound and complete:

Theorem 2. *Let $\mathcal{I}$ be an Interval MC and S_ω the union of all its end components. Then $\mathcal{I}$ has no Maximum Entropy implementation iff a state $s \in S_\omega$ is stochastic.*

Before proving Theorem 2 we need an additional lemma showing a connection between recurrent states and end components:

Make the Interval MC coherent (see Sect.3);
Find the maximal end components of the Interval MC with Algorithm 1 and call their union S_ω;
if *there is a stochastic state in* S_ω **then**
| signal that no Maximum Entropy implementation exists
else
| signal that a Maximum Entropy implementation exists
end

Algorithm 2: Verify whether an IMC has a Maximum Entropy implementation

Lemma 3. *Let $\mathcal{I} = (S, s_0, \check{P}, \hat{P})$ be an Interval MC and S_ω the union of all states of all its end components. Then $s \in S_\omega$ iff there exists an implementation of $\mathcal{I}$ in which s is recurrent.*

Proof. Let R be an end component of $\mathcal{I}$ and $s \in R$.

IF: We want to show that there exists an implementation in which s is recurrent. Consider an Interval MC $\mathcal{I}' = (S, s_0, \check{P}, \hat{P}')$ identical to $\mathcal{I}$ except that $\hat{P}'$ is defined as follows:

$$\forall s, t \in S. \hat{P}'_{s,t} = \begin{cases} 0 & \text{if } s \in R \wedge t \notin R \\ \hat{P}_{s,t} & \text{otherwise} \end{cases}$$

Note the following:

- $\mathcal{I}'$ is coherent: since $\mathcal{I}$ is coherent we need to check $\mathcal{I}'$ only on the intervals that differ from $\mathcal{I}$, meaning the $\hat{P}_{s,t}$ where $s \in R \wedge t \notin R$; we need to show that $\hat{P}_{s,t} = 0 \leq 1 - \sum_{u \neq t} \check{P}_{s,u}$, but this is obvious since $\forall s. \sum_{t \in S} \check{P}_{s,t} \leq 1$ by consistency assumption of $\mathcal{I}$.
- R is an end component in $\mathcal{I}'$: this is obvious since $\check{P}$ is not modified and $\hat{P}$ for transitions to state inside R is also unmodified.
- Any implementation of $\mathcal{I}'$ is also an implementation of $\mathcal{I}$, since $\mathcal{I}'$ refines $\mathcal{I}$.

We know by Definition 10 that R is a strongly connected component in $\mathcal{I}'$, and by Lemma 1 this means that there is an implementation in which R is strongly connected. By the definition of $\mathcal{I}'$ clearly for any implementation of $\mathcal{I}'$, $\forall n \in \mathbb{N}. \mathbf{P}(X_{n+1} \notin R | X_n \in R) = 0$. This means that there is an implementation of $\mathcal{I}'$ in which R is a closed communicating class, meaning all states in it are recurrent. Since every implementation of $\mathcal{I}'$ is also an implementation of $\mathcal{I}$, this is also true for $\mathcal{I}$.

ONLY IF: We want to show that if there exists an implementation $\mathcal{C}$ of $\mathcal{I}$ in which s is recurrent, then there is an end component R of $\mathcal{I}$ such that $s \in R$. By definition of recurrent states we know that in the implementation $\mathcal{C}$ we have $\rho(s, s) = 1$. Let's consider the set of the successors of s in any number of steps: $Succ^*(s) = \{t \in S | \exists k \in \mathbb{N}. P^k_{s,t} > 0\} \subseteq S$.

Clearly $s \in Succ^*(s)$. Let's consider a state $t \in Succ^*(s)$: since t is a successor of s obviously $\rho(s, t) > 0$, but then since $\rho(s, s) = 1$ it must be $\rho(t, s) = 1$, meaning that $s \in Succ^*(t)$. Since $\forall t, u \in Succ^*(s). \exists k, k'. P^k_{t,s} > 0 \wedge P^{k'}_{s,u} > 0$ then $\forall t, u \in$

$Succ^*(s).P_{t,u}^{k+k'} > 0$, meaning $Succ^*(s)$ is strongly connected in $\mathcal{C}$ and consequently in $\mathcal{I}$.

Let's again consider a state $t \in Succ^*(s)$: we proved that all states in $Succ^*(s)$ are reachable from it, now we prove ad absurdum that no state outside $Succ^*(s)$ is reachable from t. Consider $u \in S \backslash Succ^*(s)$ such that $P_{t,u} > 0$. Then since t is reachable from s and u is reachable from t then u is reachable from s, contradicting $u \in S \backslash Succ^*(s)$. Thus in $\mathcal{C}$ there is no transition with probability greater than 0 from a state in $Succ^*(s)$ to a state outside $Succ^*(s)$; since $\mathcal{C}$ implements $\mathcal{I}$ it must be $\forall t \in Succ^*(s), u \in S \backslash Succ^*(s).\hat{P}_{t,u} = 0$ and since $\mathcal{C}$ is a Markov chain it must be that $\forall t \in Succ^*(s), \sum_{u \in Succ^*(s)}.\hat{P}_{t,u} \geq 1$. We have shown that $Succ^*(s)$ respects all conditions in Definition 10, and thus it is an end component. □

Now we can proceed with the proof of Theorem 2:

Proof of Theorem 2. Let R be an end component of $\mathcal{I}$ and $s \in R$.

IF: Let $\mathtt{N} = \{\mathcal{C} \vDash \mathcal{I} | s \text{ is stochastic in } \mathcal{C}\}$ and $\mathtt{R} = \{\mathcal{C} \vDash \mathcal{I} | s \text{ is recurrent in } \mathcal{C}\}$. Since $s \in S_\omega$ then by Lemma 3 $\mathtt{R} \neq \emptyset$, while $\mathtt{N} \neq \emptyset$ by assumption. We can distinguish two cases:

- $\mathtt{N} \cap \mathtt{R} \neq \emptyset$: let $\mathcal{C} \in \mathtt{N} \cap \mathtt{R}$; then in the implementation $\mathcal{C}$ state s is both recurrent and stochastic, and consequently $H(\mathcal{C}) = \infty$ by Theorem 1, thus no Maximum Entropy implementation exists. In this case we say that the entropy of $\mathcal{I}$ is *infinite*.

- $\mathtt{N} \cap \mathtt{R} = \emptyset$: let $\mathcal{C} \in \mathtt{R}$ be an implementation in which R is strongly connected; it exists by Lemma 1. We argue that $\exists s \in R.\exists t \notin R.\hat{P}_{s,t} > 0$ ad absurdum: let's assume no state in R has an allowed transition outside R, then since $\mathtt{N} \neq \emptyset$ there is an implementation in which a state $s \in R$ is stochastic and recurrent, contradicting $\mathtt{N} \cap \mathtt{R} = \emptyset$. Then $\exists s \in R.\exists t \notin R.\hat{P}_{s,t} > 0$. Consider $\mathcal{C}' \in \mathtt{N}$ which is identical to $\mathcal{C}$ except that we assign an arbitrarily small positive probability ε to the transition from s to t and the remaining $1 - \varepsilon$ to states inside R. Note that $\mathcal{C}' \notin \mathtt{R}$. Since we did not modify any other state in R it is still true that $\forall u \in R \backslash \{s\}.\rho(u, s) = 1$ while $\rho(s, s) = 1 - \varepsilon$. Consider an arbitrarily small $1 > \delta \geq \varepsilon$. If there exists a path of finite length l from s to itself with probability $1 - \varepsilon$, let $\delta = \varepsilon$; if such a finite path does not exist it means that $\sum_{n=0}^{\infty} P_{s,s}^n = 1 - \varepsilon$, in which case take $\delta = 1 - \sum_{n=0}^{l} P_{s,s}^n$ for an arbitrarily large fixed $l \in \mathbb{N}$. In any case we will have $\sum_{n=0}^{l} P_{s,s}^l = 1 - \delta$. Note that δ gets arbitrarily close to ε as $l \to \infty$.

  Let's study ξ_s. Consider the set $\mathtt{K}=\{n \mid P_{s_0,s}^n > 0\}$; it is nonempty by reachability

of s. Let $k = \inf(\mathbb{K})$ and $\rho = P^k_{s_0,s}$. Since $\rho \leq \rho_s$ we can write

$$\begin{aligned}
\xi_s &= \sum_{n=0}^{\infty} P^n_{s_0,s} \geq \rho \sum_{n=0}^{\infty} P^n_{s,s} \\
&= \rho \left(\sum_{n=0}^{l-1} P^n_{s,s} + \sum_{n=l}^{2l-1} P^n_{s,s} + \sum_{n=2l}^{3l-1} P^n_{s,s} + ... \right) \\
&\geq \rho \left((1-\delta) + (1-\delta) \sum_{n=1}^{l} P^n_{s,s} + ... \right) \\
&= \rho \left((1-\delta) + (1-\delta)^2 + (1-\delta)^3 + ... \right) \\
&= \rho \sum_{m=1}^{\infty} (1-\delta)^m = \frac{\rho}{\delta} - 1 \approx \frac{\rho}{\varepsilon}
\end{aligned}$$

since δ can get arbitrarily close to ε. Note that ρ does not depend on ε. We proved that ξ_s is bounded from below by $\frac{\rho}{\varepsilon}$, which can grow as much as we want since ρ is a positive constant and ε can be chosen arbitrarily small.

Now let's study the local entropy of s. By the properties of entropy we have that the smallest case[2] is when it has two outgoing transitions, one with probability ε to a state outside R and the other with probability $1-\varepsilon$ to a state inside R. Thus we say that $L(s) \geq -((\varepsilon \log_2 \varepsilon) + ((1-\varepsilon)\log_2(1-\varepsilon)))$, and consequently

$$L(s)\xi_s \geq -\frac{\rho((\varepsilon \log_2 \varepsilon) + ((1-\varepsilon)\log_2(1-\varepsilon)))}{\varepsilon}$$

Note that

$$\lim_{\varepsilon \to 0} -\frac{\rho((\varepsilon \log_2 \varepsilon) + ((1-\varepsilon)\log_2(1-\varepsilon)))}{\varepsilon} = \infty$$

meaning $L(s)\xi_s$ is an increasing function as ε tends to zero, and since $H(\mathcal{C}) = \sum_{t \in S} L(t)\xi_t$ clearly for each implementation it is possible to give another with a smaller ε and finite but greater entropy, meaning no maximum entropy implementation exists. In this case we say that the entropy of $\mathcal{I}$ is *unbounded*.

ONLY IF: We will assume we can give implementations of the Interval MC with any entropy value and show that such a state s must exist. We will proceed ad absurdum: let's assume for any implementation it is possible to give another implementation with higher entropy for the Interval MC but for any implementation any state $s \in S_\omega$ is deterministic. Since deterministic states contribute no entropy, only states not in the end component have positive entropy; let $t \notin S_\omega$ be any one of them. From Lemma 3 we know that there is no implementation in which t is recurrent, meaning that in each implementation ξ_t is finite and $\rho(t,t) < 1$. This means that there cannot be a cycle in any implementation from t to itself with probability 1, and since we allow

[2]Since ε can be arbitrarily small, the cases in which the transitions have probability ε' and $1-\varepsilon'$ with $\varepsilon' < \varepsilon$ are included

only closed intervals this also means that there exists a small positive δ such that in no implementation t has a cycle to itself with probability greater than $1-\delta$. ξ_t can be written as the expected residence time in t from itself multiplied by the first hitting probability ρ_t of t:

$$\xi_t = \rho_t \sum_{n=0}^{\infty} P_{t,t}^n$$

Note that:

- since ρ_t is a probability, then it is bounded from above by 1;
- since t has no path to itself with probability greater than $1-\delta$, then $\forall l \in \mathbb{N}. P_{t,t}^l \leq 1-\delta$;

thus we can write

$$\xi_t = \rho_t \sum_{n=0}^{\infty} P_{t,t}^n \leq \sum_{n=0}^{\infty} (1-\delta)^n = \frac{1}{\delta}$$

Since local entropy is bounded from above by $\log_2 |S|$ then for each state t it holds that $L(t)\xi_t \leq \frac{\log_2 |S|}{\delta}$; then $H(\mathcal{C}) = \sum_{t \in S} L(t)\xi_t \leq \frac{|S| \log_2 |S|}{\delta}$, contradicting the assumption that for any implementation it is possible to give one with higher entropy. □

Theorem 2 proves that Alg. 2 is sound and complete to verify the existence of a Maximum Entropy implementation as defined in Def. 9 for a given Interval MC $\mathcal{I}$. If such an implementation exists, its entropy is the channel capacity for all implementations of the system and can be used as an upper bound on the information leakage of all such implementations, providing a security guarantee for the protocol represented by $\mathcal{I}$. We will now show a numerical method to synthesize a Maximum Entropy implementation with arbitrary precision. In Sect. 7 we discuss the case in which such an implementation does not exist.

6.2. Synthesis of a Maximum Entropy Implementation

We have been characterizing the existence of a Maximum Entropy implementation with finite entropy, now we propose a numerical technique to synthesize it with an arbitrary precision [33]; the Maximum Entropy implementation of the Two-step Authentication in Fig. 2 has been obtained this way. We reduce the problem to solving a multidimensional maximization on convex sets by considering each of the $|S|^2$ transition probabilities $P_{s,t}$ in the chain as different dimensions, each of which can take values in the interval $[\check{P}_{s,t}, \hat{P}_{s,t}]$, generating a convex polytope.

Due to coherence of the Interval MC there exists at least one Markov chain implementing it, so the polytope will be nonempty. We need to add to the system the constraints $\forall s \in S. \sum_{t \in S} P_{s,t} = 1$ to ensure every solution can be interpreted as an MC. Since these constraints are linear, the domain is still a convex polytope. A point in the polytope thus defines a Markov chain. The objective function to maximize is the entropy of such Markov chain, which can be calculated in PTIME as shown in Sect. 5.

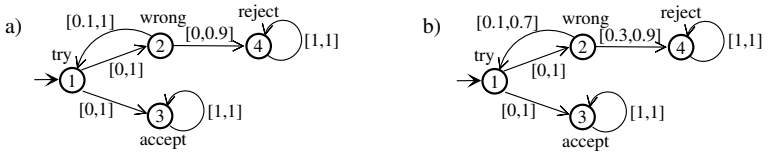


Figure 6: a) Specification for the Repeated Authentication with unbounded entropy. b) Specification for the Repeated Authentication with bounded entropy.

This optimization problem for an everywhere differentiable function can be solved using numerical methods. Once the global maximum is found with a numerical algorithm, the parameters $P_{s,t}$ interpreted as a MC give a Maximum Entropy implementation.

Example. Consider the Two-step Authentication in Fig. 1a. The entropy of the system is $H = (-(P_{1,2}\log_2(P_{1,2})) - ((1-P_{1,2})\log_2(1-P_{1,2})))+P_{1,2}(-(P_{2,4}\log_2(P_{2,4})) - ((1-P_{2,4})\log_2(1-P_{2,4})))$ under constraints $0 \leq P_{1,2} \leq 1 \wedge 0 \leq P_{2,4} \leq 1$. It is maximal for $P_{1,2} = 2/3$, $P_{2,4} = 0.5$. The Maximum Entropy implementation is shown in Fig. 2.

7. Infinite vs Unbounded Entropy for Interval MCs

We now give insight about the difference between unbounded and infinite entropy for an Interval MC and give a decision procedure to distinguish the two cases.

The infinite case means that it is possible to give implementations with infinite entropy, and by Corollary 1 an implementation with infinite entropy must be non terminating. Thus this case means that the specification allows for non-terminating implementations, during which an arbitrary amount of information will be leaked.

The unbounded case means that all implementations terminate, but whatever the size of the secret, it is possible to give an implementation that leaks all of it, and thus we cannot give security guarantees for their behavior.

Consider the Repeated Authentication in Fig. 6a; since state 1 can be both recurrent and stochastic but never both, we are in the unbounded case, and in fact it is possible to give implementations with arbitrary entropy. Since the Repeated Authentication is a security scenario, this means that it is possible to give implementations that leak any amount of information about the confidential data, and thus this should be considered an insecure authentication protocol, as it is not possible to give any security guarantee for it.

In this particular case this depends on the fact that we allow the black list to be empty; in this implementation the attacker can try all possible passwords, and thus will eventually leak all of the confidential data. In Figure 6b we show a modified version in which the black list covers at least 30% of the passwords; for this case the Interval MC has a Maximum Entropy implementation, and is thus possible to give a security guarantee.

Algorithm 3 discriminates these two cases. The idea is to build an implementation that maximizes the end components (in which all states that can be stochastic are stochastic). If this implementation has stochastic states in a strongly connected component,

Find all maximal end components of the Interval MC;
Modify the transition probabilities so that all end components are closed: for each end component $\mathcal{R}$, set $\hat{P}_{s,t} = 0$ for all $s \in \mathcal{R}$, $t \notin \mathcal{R}$;
Make the Interval MC coherent again with the coherence algorithm;
if *the states of all end components are deterministic* **then**
  signal that no infinite entropy implementation exists
else
  signal that an infinite entropy implementation exists
end

Algorithm 3: Verify whether an Interval MC allows for infinite implementations.

then it will be possible to generate an infinite amount of entropy, otherwise the entropy of any implementation is always finite.

After step 2 the Interval MC will still have implementations, since by the definition of end components it is possible to give an implementation that has probability 0 of leaving the end component; we are just forcing it to happen for all our end components and checking if this makes them necessarily deterministic or not.

8. Discussion about Maximum Entropy Strategy of an MDP

We briefly discuss the usage of Markov Decision Processes as specification models, to show that the approach can be generalized to different specification theories.

As for Interval MCs, reward functions over transitions can be defined for MDPs. In an MDP $\mathcal{P} = (S, s_0, P, \Lambda)$ a non-negative reward function is a function $R : S \times \Lambda \times S \to \mathbb{R}^+$ assigning a non-negative reward value to each triple (s, a, t) such that $s, t \in S$, $a \in \Lambda_s$. The expected reward for a state-action pair (s, a) is a linear combination $R(s, a) = \sum_{t \in S} P_{(s,a),t} R_{(s,a),t}$. Finding a strategy maximizing a non-negative reward function is known as the Positive-bounded Expected Total Reward problem in Puterman's book [29].

As for Interval MCs, we want to cast entropy as a non-negative reward function over transitions of MDPs and we want to find the strategy that maximizes entropy. Alas, the reward function for entropy would have to sum up all the probabilities of going from a state s to a state t through all the actions in Λ_s, and thus cannot be cast in this form. Consider Fig. 4; the probability of transitioning from state **1** to state **2** depends on both $P_{(1,a),2}$ and $P_{(1,b),2}$ and thus we cannot compute it by assigning rewards to a transition without considering every other transition from the same state. The reward function would have to be defined on states, becoming $R(s) = -\sum_{t \in S} \left(\left(\sum_{a \in \Lambda_s} P_{(s,a),t} \right) \log_2 \left(\sum_{a \in \Lambda_s} P_{(s,a),t} \right) \right)$, which contains a non-linear dependency on the choice of strategy ($P_{(s,a),t}$), so standard reward optimization methods do not apply.

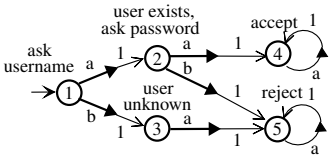


Figure 7: MDP model for the Two-step Authentication

More importantly, the Maximum Entropy strategy would not be positional. Consider the MDP model of the Two-step Authentication protocol in Fig. 7. Each strategy for

assigning probabilities to actions a and b in states **1** and **2** resolves the MDP in a Markov chain, allowing us to use the MDP as a specification model. The Maximum Entropy strategy is the strategy assigning probability $\frac{2}{3}$ to a and $\frac{1}{3}$ to b in state **1** and probability $\frac{1}{2}$ to a and $\frac{1}{2}$ to b in state **2**, so this is an example of a case in which the Maximum Entropy strategy is not positional. This prevents us from using common algorithms like value iteration, policy iteration and reduction to linear programming [29, Chpt. 7] to find the optimal solution. A numerical approximation approach needs to be applied, showing the inherent hardness of the problem and validating our approximation approach in the case of Interval MCs.

9. Conclusion and Future Work

It can be possible to give an upper bound to the leakage of a protocol even if some of its parameters are unspecified, computing the entropy of the Maximum Entropy implementation of the specification when it exists. If such implementation does not exist, the protocol is inherently insecure and should be modified. Existence of the Maximum Entropy implementation can be determined in polynomial time in the size of the model, while its synthesis requires the solution of a nonlinear multidimensional maximization problem.

The intention is to allow implementation of intelligent tools that can automatically concretize underspecified models and give security guarantees to specifications, and eventually optimize the parameters of such models to make them more resistant to attacks. It would be interesting to extend our results (and the tool) to specifications with open intervals, and to other abstractions such as non-trivial subclasses of Constraint MCs [7] or Abstract Probabilistic Automata [14].

References

[1] A. Aldini, A.D. Pierro, Estimating the maximum information leakage, Int. J. Inf. Sec. 7 (2008) 219–242.

[2] L. de Alfaro, Formal Verification of Probabilistic Systems, Ph.D. thesis, Stanford, 1997.

[3] L. de Alfaro, Temporal logics for the specification of performance and reliability, in: R. Reischuk, M. Morvan (Eds.), STACS, volume 1200 of *Lecture Notes in Computer Science*, Springer, 1997, pp. 165–176.

[4] M.S. Alvim, K. Chatzikokolakis, C. Palamidessi, G. Smith, Measuring information leakage using generalized gain functions, in: S. Chong (Ed.), CSF, IEEE, 2012, pp. 265–279.

[5] M. Bhargava, C. Palamidessi, Probabilistic anonymity, in: M. Abadi, L. de Alfaro (Eds.), CONCUR, volume 3653 of *LNCS*, Springer, 2005, pp. 171–185.

[6] F. Biondi, A. Legay, P. Malacaria, A. Wasowski, Quantifying information leakage of randomized protocols, in: R. Giacobazzi, J. Berdine, I. Mastroeni (Eds.), VMCAI, volume 7737 of *Lecture Notes in Computer Science*, Springer, 2013, pp. 68–87.

[7] B. Caillaud, B. Delahaye, K.G. Larsen, A. Legay, M.L. Pedersen, A. Wasowski, Constraint markov chains, Theor. Comput. Sci. 412 (2011) 4373–4404.

[8] K. Chatterjee, K. Sen, T. Henzinger, Model-checking ω-regular properties of interval markov chains, in: FoSSaCS.

[9] K. Chatzikokolakis, C. Palamidessi, P. Panangaden, Anonymity protocols as noisy channels, in: Information and Computation, Springer, 2006.

[10] H. Chen, P. Malacaria, Quantifying maximal loss of anonymity in protocols, in: W. Li, W. Susilo, U.K. Tupakula, R. Safavi-Naini (Eds.), ASIACCS, ACM, 2009.

[11] D. Clark, S. Hunt, P. Malacaria, A static analysis for quantifying information flow in a simple imperative language, Journal of Computer Security 15 (2007).

[12] T. Cover, J. Thomas, Elements of information theory, Wiley, New York, 1991.

[13] S.H..P.M. D. Clark, Quantitative information flow, relations and polymorphic types, Journal of Logic and Computation, Special Issue on Lambda-calculus, type theory and natural language 18 (2005) 181–199.

[14] B. Delahaye, J.P. Katoen, K.G. Larsen, A. Legay, M.L. Pedersen, F. Sher, A. Wasowski, Abstract probabilistic automata, in: VMCAI.

[15] R. Giacobazzi, I. Mastroeni, Abstract non-interference: parameterizing non-interference by abstract interpretation, in: N.D. Jones, X. Leroy (Eds.), POPL, ACM, 2004, pp. 186–197.

[16] V. Girardin, Entropy maximization for markov and semi-markov processes, Methodology and Computing in Applied Probability 6 (2004) 109–127.

[17] J.A. Goguen, J. Meseguer, Security policies and security models, in: IEEE Symposium on Security and Privacy, pp. 11–20.

[18] E.T. Jaynes, Information Theory and Statistical Mechanics, Physical Review Online Archive (Prola) 106 (1957) 620–630. doi:10.1103/PhysRev.106.620.

[19] B. Jonsson, K.G. Larsen, Specification and refinement of probabilistic processes, in: LICS, IEEE Computer Society, 1991, pp. 266–277.

[20] B. Köpf, D.A. Basin, An information-theoretic model for adaptive side-channel attacks, in: P. Ning, S.D.C. di Vimercati, P.F. Syverson (Eds.), ACM Conference on Computer and Communications Security, ACM, 2007, pp. 286–296.

[21] B. Köpf, L. Mauborgne, M. Ochoa, Automatic quantification of cache side-channels, in: P. Madhusudan, S.A. Seshia (Eds.), CAV, volume 7358 of *Lecture Notes in Computer Science*, Springer, 2012, pp. 564–580.

[22] I. Kozine, L.V. Utkin, Interval-valued finite markov chains, Reliable Computing 8 (2002) 97–113.

[23] J. Landauer, T. Redmond, A lattice of information, in: CSFW, pp. 65–70.

[24] P. Malacaria, Assessing security threats of looping constructs, in: Proceedings of the 34th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages, POPL '07, ACM, New York, NY, USA, 2007, pp. 225–235. doi:10.1145/1190216.1190251.

[25] P. Malacaria, Algebraic foundations for information theoretical, probabilistic and guessability measures of information flow, CoRR abs/1101.3453 (2011).

[26] P. Malacaria, H. Chen, Lagrange multipliers and maximum information leakage in different observational models, PLAS '08, ACM, New York, USA, 2008, pp. 135–146. doi:10.1145/1375696.1375713.

[27] I. Mastroeni, R. Giacobazzi, An abstract interpretation-based model for safety semantics, Int. J. Comput. Math. 88 (2011) 665–694.

[28] J.K. Millen, Covert channel capacity, in: IEEE Symposium on Security and Privacy, pp. 60–66.

[29] M.L. Puterman, Markov Decision Processes: Discrete Stochastic Dynamic Programming, Wiley-Interscience, 1994.

[30] P. Ryan, J. McLean, J. Millen, V. Gligor, Non-interference, who needs it?, in: Computer Security Foundations Workshop, 2001. Proceedings. 14th IEEE, pp. 237 –238. doi:10.1109/CSFW.2001.930149.

[31] C.E. Shannon, A mathematical theory of communication, The Bell system technical journal 27 (1948) 379–423.

[32] G. Smith, On the foundations of quantitative information flow, in: L. de Alfaro (Ed.), FOSSACS, volume 5504 of *Lecture Notes in Computer Science*, Springer, 2009, pp. 288–302.

[33] J. Stoer, R. Bulirsch, R. Bartels, W. Gautschi, C. Witzgall, Introduction to Numerical Analysis, Texts in Applied Mathematics, Springer, 2010.

[34] R. Tarjan, Depth-First Search and Linear Graph Algorithms, SIAM Journal on Computing 1 (1972) 146–160.

[35] H. Yasuoka, T. Terauchi, Quantitative information flow - verification hardness and possibilities, in: CSF, IEEE Computer Society, 2010, pp. 15–27.

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.

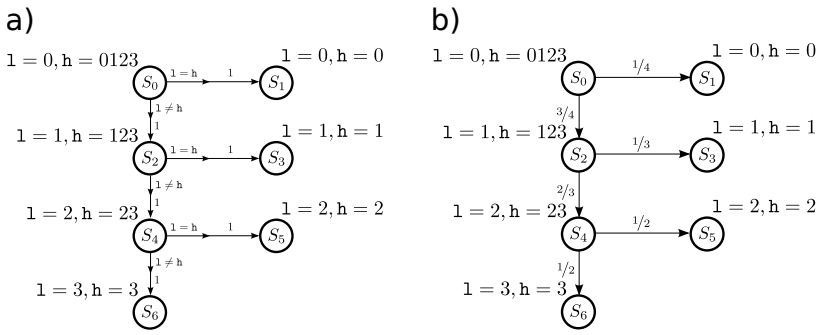


Fig. 1. Simple loop example a) MDP semantics b) MC model

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*

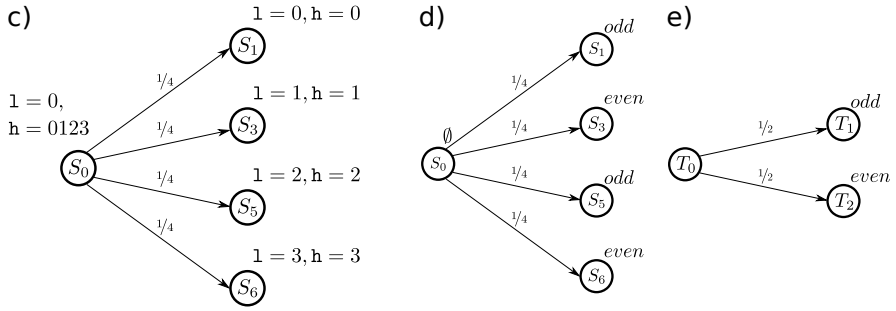


Fig. 2. Simple loop example c) Observable reduction d) Relabeling e) Quotient

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 `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.

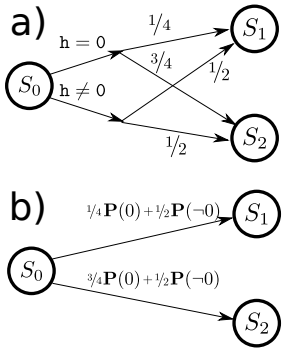


Fig. 5. Reduction from MDP to parameterized MC

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

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

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.

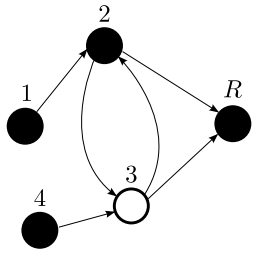


Fig. 6. Network topology for Onion Routing

h	Path	o	P(O|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.

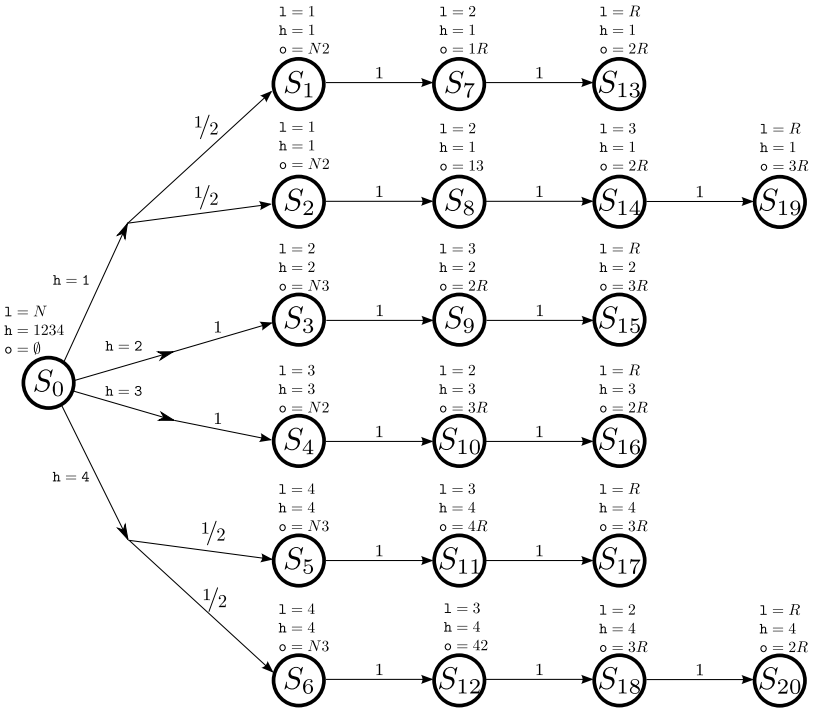


Fig. 8. Markov Decision Process for Onion Routing

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 $\mathbb{C}/\mathcal{R}_h \cap \mathcal{R}_\mathcal{A} = \mathbb{C}$.

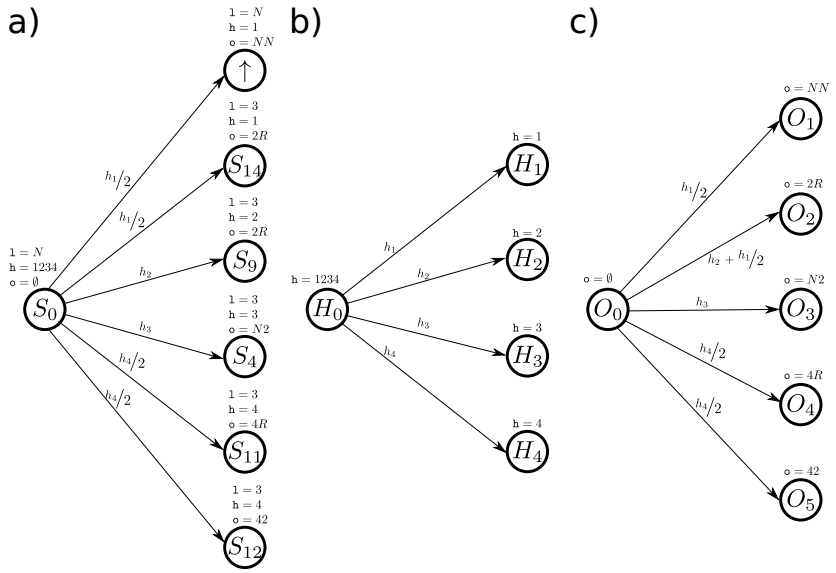


Fig. 9. Markov chains for Onion Routing: a) Observable reduction $\mathbb{C}$ b) $\mathbb{C}/\mathcal{R}_h$ c) $\mathbb{C}/\mathcal{R}_{\mathcal{A}}$

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.

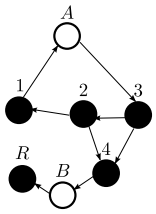


Fig. 10. Network topology for Timed Onion Routing

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

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

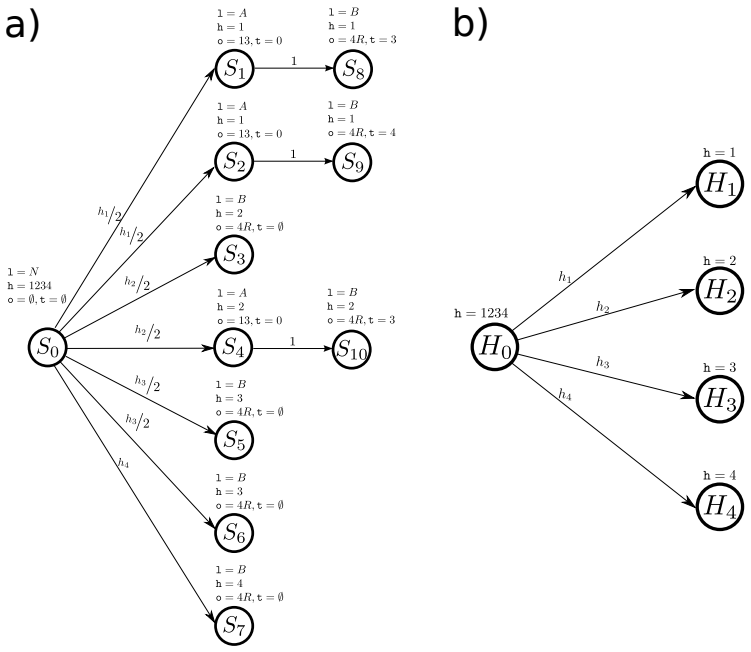


Fig. 12. Markov chains for Timed Onion Routing: a) Observable reduction $\mathbb{C}$ b) $\mathbb{C}/\mathcal{R}_h$

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

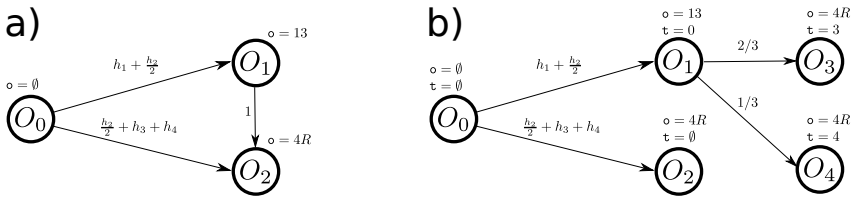


Fig. 13. Markov chains for Timed Onion Routing: a) $\mathbb{C}/\mathcal{R}_{\mathcal{A}_\mathcal{N}}$ b) $\mathbb{C}/\mathcal{R}_{\mathcal{A}_\mathcal{T}}$

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

Appendix

Corollary 2. *Let $\mathcal{P}$ be a program and o_i its outputs. For each o_i, the total probability of reaching absorbing states labelled with o_i in the MDP semantics of $\mathcal{P}$ is equal to $\mathbf{P}([[\mathcal{P}]] = o_i)$ where $[[\mathcal{P}]]$ is the r.v. derived from the denotational semantics of the program (i.e. LoI($\mathcal{P}$)).*

Proof (of Theorem 2). Since all internal states are hidden, the MC will look like a 1-step probability distribution from the starting state to the output states. By Corollary 2 the probability of observing the observations o_i is consistent with the probability of observing the same observations in $\mathcal{P}$, thus $H(\mathcal{C}/\mathcal{R}_{\mathcal{A}_{\mathcal{I}/\mathcal{O}}}) = H([[\mathcal{P}]]) = H(LoI(\mathcal{P})) = Leakage(\mathcal{P})$.

The proof of Theorem 3 follows immediately from the following Lemma:

Lemma 1. *Let $\mathcal{C}_1 = (S_1, s_0^1, P_1)$, $\mathcal{C}_2 = (S_2, s_0^2, P_2)$ be Markov chains. Let $\mathcal{C}(S, s_0, P)$ be their synchronous parallel composition, i.e. $S = S_1 \times S_2$, $s_0 = s_0^1 \times s_0^2$ and*

$$P_{s_1 \times s_2, t_1 \times t_2} = P_{s_1,t_1} P_{s_2,t_2}.$$

Let R^+ and R^- be reward functions on the transitions of $\mathcal{C}$ such that

$$R^+(s_1 \times s_2, t_1 \times t_2) = \log_2 \left(P_{s_1,t_1} P_{s_2,t_2}\right).$$

$$R^-(s_1 \times s_2, t_1 \times t_2) = \log_2 \left(\frac{P_{s_1,t_1}}{P_{s_2,t_2}}\right).$$

Then the expected total infinite time reward of $\mathcal{C}$ with the reward function R^+ is equivalent to $H(\mathcal{C}_1) + H(\mathcal{C}_2)$ and the expected total infinite time reward of $\mathcal{C}$ with the reward function R^- is equivalent to $H(\mathcal{C}_1) - H(\mathcal{C}_2)$.

Proof. We will prove the result for $\mathcal{R}^-$; the proof for $\mathcal{R}^+$ is symmetrical. Consider a state $s = s_1 \times s_2$ of $\mathcal{C}$. The expected reward of s is

$$\begin{aligned}
R^-(s) &= \sum_{t \in S} P_{s,t} R^-(s,t) \\
&= \sum_{t_1 \in S_1} \sum_{t_2 \in S_2} P_{s_1,t_1} P_{s_2,t_2} \log_2 \frac{P_{s_1,t_1}}{P_{s_2,t_2}} \\
&= \sum_{t_1 \in S_1} \sum_{t_2 \in S_2} P_{s_1,t_1} P_{s_2,t_2} (\log_2 P_{s_1,t_1} - log_2 P_{s_2,t_2}) \\
&= \sum_{t_1 \in S_1} \sum_{t_2 \in S_2} P_{s_1,t_1} P_{s_2,t_2} \log_2 P_{s_1,t_1} - \sum_{t_1 \in S_1} \sum_{t_2 \in S_2} P_{s_1,t_1} P_{s_2,t_2} \log_2 P_{s_2,t_2} \\
&= \sum_{t_2 \in S_2} P_{s_2,t_2} \sum_{t_1 \in S_1} P_{s_1,t_1} \log_2 P_{s_1,t_1} - \sum_{t_1 \in S_1} P_{s_1,t_1} \sum_{t_2 \in S_2} P_{s_2,t_2} \log_2 P_{s_2,t_2} \\
&= 1 \cdot \sum_{t_1 \in S_1} P_{s_1,t_1} \log_2 P_{s_1,t_1} - 1 \cdot \sum_{t_2 \in S_2} P_{s_2,t_2} \log_2 P_{s_2,t_2} \\
&= L(s_1) - L(s_2)
\end{aligned}$$

thus the expected total reward of $\mathcal{C}$ is

$$\begin{aligned}
R^-(C) &= \sum_{s\in S} R^-(s)\xi_s \\
&= \sum_{s\in S} R^-(s) \sum_{n=0}^{\infty} P^n_{s_0,s} \\
&= \sum_{s_1\in S_1}\sum_{s_2\in S_2} (L(s_1)-L(s_2)) \sum_{n=0}^{\infty} P^n_{s_0^1,s_1} P^n_{s_0^2,s_2} \\
&= \sum_{s_1\in S_1} L(s_1)\left(\sum_{n=0}^{\infty} P^n_{s_0^1,s_1} \sum_{s_2\in S_2} P^n_{s_0^2,s_2}\right) - \sum_{s_2\in S_2} L(s_2)\left(\sum_{n=0}^{\infty}(P^n_{s_0^2,s_2} \sum_{s_1\in S_1} P^n_{s_0^1,s_1}\right) \\
&= \sum_{s_1\in S_1} L(s_1) \sum_{n=0}^{\infty}\left(P^n_{s_0^1,s_1}\cdot 1\right) - \sum_{s_2\in S_2} L(s_2) \sum_{n=0}^{\infty}\left(P^n_{s_0^2,s_2}\cdot 1\right) \\
&= \sum_{s_1\in S_1} L(s_1)\xi_{s_1} - \sum_{s_2\in S_2} L(s_2)\xi_{s_2} \\
&= H(\mathcal{C}_1) - H(\mathcal{C}_2).
\end{aligned}$$

QUAIL: A Quantitative Security Analyzer for Imperative Code*

Fabrizio Biondi[1], Axel Legay[2], Louis-Marie Traonouez[2], and Andrzej Wąsowski[1]

[1] IT University of Copenhagen, Denmark
[2] INRIA Rennes, France

Abstract. Quantitative security analysis evaluates and compares how effectively a system protects its secret data. We introduce QUAIL, the first tool able to perform an arbitrary-precision quantitative analysis of the security of a system depending on private information. QUAIL builds a Markov Chain model of the system's behavior as observed by an attacker, and computes the correlation between the system's observable output and the behavior depending on the private information, obtaining the expected amount of bits of the secret that the attacker will infer by observing the system. QUAIL is able to evaluate the safety of randomized protocols depending on secret data, allowing to verify a security protocol's effectiveness. We experiment with a few examples and show that QUAIL's security analysis is more accurate and revealing than results of other tools.

1 Introduction

The Challenge. Qualitative analysis tools can verify the complete security of a protocol, i.e. that an attacker is unable to get any information on a secret by observing the system—a property known as *non-interference*. Non-interference holds when the system's output is independent from the value of the secret, so no information about the latter can be inferred from the former [20]. However, when non-interference does not hold, qualitative analysis cannot rank the security of a system: all unsafe systems are the same.

Quantitative analysis can be used to decide which of two alternative protocols is more secure. It can also asses security of systems that are insecure, but nevertheless useful, in the qualitative sense, such as a password authentication protocol, for which there is always a positive probability that an attacker will randomly guess the password. A quantitative analysis is challenging because it is not sufficient to find a counterexample to a specification to terminate. We need to analyze all possible behaviors of the system and quantify for each one the probability that it will happen and how much of the protocol's secret will be revealed. So far no tool was able to perform this analysis precisely.

Quantitative analysis with QUAIL. We use Quantified Information Flow to reduce the comparison of security of two systems to a computation of expected amount of information, in the information-theoretical sense, that an attacker would learn about the secret by observing a system's behavior. This expected amount of information is known as *information leakage* [9,15,8,12,21] of a system. It amounts to zero iff the system is

* Partially supported by MT-LAB — VKR Centre of Excellence on Modeling of IT

non-interfering [15], else it represents the expected number of bits of the secret that the attacker is able to infer. The analysis generalizes naturally to more than two systems, hence allowing to decide which of them is less of a threat to the secrecy of the data.

To compute information leakage we use a stochastic model of a system as observed by the attacker. The model is obtained by resolving non-determinism in the system code, using the prior probability distribution over the secret values known to the attacker before an attack. Existing techniques represent this with a channel matrix from secret values to outputs [5]. They build a row of the channel matrix for each possible value of the secret, even if the system would behave in the same way for most of them. In contrast, we have proposed an automata based technique [3], using Markovian models. One state of a model represents an interval of values of the secret for which the system behaves in the same way, allowing for a much more compact and tractable representation.

We build a Markov chain representing the behavior observed by the attacker, then we hide the states that are not observable by the attacker, obtaining a smaller Markov chain—an *observable reduction*. Then we calculate the correlation between the output the attacker can observe and the behavior dependent on the secret, as it corresponds to the leakage. Since leakage in this case is mutual information, it can be computed by adding the entropy of the observable and secret-dependent views of the system and subtracting the entropy of the behavior depending on both. See [3] for details.

QUAIL (QUantitative Analyzer for Imperative Languages) implements this method. It is the first tool supporting arbitrary-precision quantitative evaluation of information leakage for randomized systems or protocols with secret data, including anonymity and authentication protocols. Systems are specified in a simple imperative modeling language further described on QUAILS website.

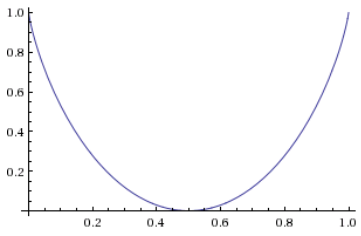


Fig. 1: Bit XOR leakage as a function of $\Pr(r = 1)$

QUAIL performs a white-box leakage analysis assuming that the attacker has knowledge of the system's code but no knowledge of the secret's value, and outputs the result and eventually information about the computation, including the Markov chains computed during the process.

Example. Consider a simple XOR operation example. Variable h stores a 1-bit secret. The protocol generates a random bit r, where $r = 1$ with probability p. It outputs the result of exclusive-or between values of h and r. The attacker knows p and can observe the output, so if $h = r$, but not the values of r or h.

If $p = 0.5$ the attacker cannot infer any information about h, the leakage is zero bits (non-interference). If $p = 0$ or $p = 1$ then she can determine precisely the value of h, and thus the leakage is 1 bit. This can be verified efficiently with language-based tools like APEX [10]. However, QUAIL is the only tool able to precisely compute the leakage for all possible values of p with arbitrary precision. Figure 1 shows that XOR protocol leaks more information as the value of r becomes more deterministic. For instance $p = 0.4$ is safer than $p = 0.8$.

2 QUAIL Implementation

The input model is specifed in QUAIL's imperative language designed to facilitate succinct and direct modeling of protocols, providing features such as arbitrary-size integer variables and arrays, random assignments, `while` and `for` loops, named constants and conditional statements. Figure 2 presents the input code for the bit XOR example.

For a given input code QUAIL builds an annotated Markov chain representing all possible executions of the protocol, then modifies it to encode the protocol when observed by the attacker whose aim is to discover the protocol's secret data. Finally, QUAIL extracts a model of the observable and secret-dependent behavior of the system, and computes the correlation between them, which is equivalent to the amount of bits of the secret that the attacker can infer by observing the system. We now discuss QUAIL implementation following the five steps of the method proposed in [3]:

Step 1: Preprocessing. QUAIL translates the input code into a simplified internal language. It rewrites conditional statements and loops (`if`, `for` and `while`) to conditional jumps (`if-goto`) and substitutes values for named constant references.

Step 2: Probabilistic symbolic execution. QUAIL performs a symbolic forward execution of the input program constructing its semantics as a finite Markov chain (a fully probabilistic transition system) with a single starting state. To this end, QUAIL needs to know the attacker's probability distributions over the secret variables. For each conditional branch, we compute the conditional probability of the guard being satisfied given the values of the public variables and the probability distributions over the secret variables. Then QUAIL generates two successor states, one for the case in which the guard is satisfied and one when not satisfied. This is the most time-consuming step, so QUAIL uses an on-the-fly optimization to avoid building internal states that would be removed in the next step. For instance, it does not generate new states for assignments to a non-observable public variable. Instead it changes the value of the variable in the current state.

Step 3: State hiding and model reduction. To represent what the attacker can examine, QUAIL reduces the Markov chain model by iteratively hiding all unobservable states. For the standard attacker, these are all the internal states, i.e. all the states except the initial and the output states. A state is hidden by creating transitions from its predecessors to its successors and removing it from the model. This operation normally eliminates more than 90% of the states of the Markov chain model, building its *observable reduction*. This operation also detects non-terminating loops and collapses then in a single non-termination state. States are equipped with a list of their predecessors and successors to quicken this step. An observable reduction looks like a probability distribution from the starting states to the output states, since all other states are hidden.

Step 4: Quotienting. Recall from Sect.1 that we have to quantify the correlation between the observable and secret-dependent views of the system. QUAIL relies on the notion of quotients to represent different views of the system and compute their correlation. A quotient is a Markov chain obtained by merging together states in the observable

```
1  observable int1 l; // bit l is the output
2  public int1 r; // bit r is random
3  secret int1 h; // bit h is the secret
4  random r:=randombit(0.5); // randomize r
5  if (h==r) then // calculate the XOR
6  assign l:=0;
7  else
8  assign l:=1;
9  fi
10 return; //terminate
```

```
1  observable int1 l;
2  public int1 r;
3  secret int1 h;
4  random (r):=randombit(0.5);
5  if ((h)==(r))
6  then goto 8;
7  else goto 10;
8  assign (l):=(0);
9  goto 11;
10 assign (l):=(1);
11 return;
```

Fig. 2: Bit XOR example: input code (on the left) and preprocessed code (on the right).

reduction that give the same value to some of the variables. QUAIL quotients the observable reduction separately three times to build three different views of the system. QUAIL uses the attacker model again to know which states are indistinguishable as they assign the same values to the observable variables. These states are merged in the *attacker's quotient*. Similarly, in the *secret's quotient* states are merged if they have the same possible values for the secret, while in the *joint quotient* states are merged if they both have the same values for the secret and cannot be discriminated by the attacker. Since information about the states' variables is not needed to compute entropy, quotients carry none, reducing time and memory required to compute them.

Step 5: Entropy and leakage computation. The *information leakage* can be computed as the sum of the entropies of the attacker's and secret's quotients minus the entropy of the joint quotient [3]. The three entropy computations are independent and can be parallelized. QUAIL outputs the leakage with the desired amount of significant digits and the running time in milliseconds. If requested, QUAIL plots the Markov chain models using Graphviz.

3 On Using QUAIL

QUAIL is freely available from https://project.inria.fr/quail, including source code, binaries and example files. We demonstrate usage of QUAIL to analyze the bit XOR example. Let bit_xor.quail be the file containing the input shown in Fig. 2. The command

```
quail bit_xor.quail -p 2 -v 0
```

executes QUAIL with precision limited to 2 digits (`-p 2`), suppressing all output except the leakage result (`-v 0`). In response QUAIL generates a file bit_xor.quail.pp with the preprocessed code shown in Fig. 2, analyzes it and finally answers `0.0` showing that in this case the protocol leaks no information (so non-interference). For different probability of the random bit r in line 4 QUAIL obtains a different leakage (cf. Fig. 1). For instance, for $p = 0.8$ the leakage is $\sim$0.27807 bits.

4 Comparison with Other Tools

QUAIL precisely evaluates the value of leakage of the input code. This not only allows proving non-interference (absence of leakage) but also enables comparing relative safety of similar protocols. This is particularly important for protocols that exhibit

Table 1: QUAIL analysis of the leakage in an authentication program

Password length	2	32	64	500
Leakage	$8.11 \cdot 10^{-1}$	$7.78 \cdot 10^{-9}$	$3.54 \cdot 10^{-18}$	$1.52 \cdot 10^{-148}$

inherent leakage, such as authentication protocols. For instance, with a simple password authentication, the user inputs a password and is granted access privilege if the password corresponds to the secret stored in the system. The chance of an attacker guessing a password is always positive (although it depends on the password's length). Also, even if the attacker gets rejected she learns something about the secret—the fact that the attempted value was not correct. QUAIL can quantify the precise leakage as a function of the bit length of the password, as shown in Table 1.

Existing *qualitative* tools can establish whether a protocol is completely secure or not, i.e. whether it respects non-interference. They cannot discriminate protocols that allow acceptable and unacceptable violations of non-interference. APEX [10] is an analyzer for probabilistic programs that can check programs equivalence, while PRISM [14] is a probabilistic model-checker. With these tools authentication protocols will always be flagged as unsafe, and a comparison between them is impossible.

QUAIL can be used also to analyze anonymity protocols, like the grade protocol and the dining cryptographers [6]. The interested reader can find discussion and input code for these examples on the QUAIL website. These protocols provide full anonymity on the condition that some random data is generated with a uniform probability distribution; their effectiveness in these cases can be efficiently verified with the qualitative tools above. If the probability distribution over the random data is not uniform some private data is leaked, and QUAIL is again the only tool that can quantify this leakage. Presently, the models for these protocols tend to grow exponentially, so the analysis becomes time-consuming already for about 6–7 agents.

Qualitative tools and technique are more closely related to QUAIL; we present some of them and discuss the main differences. It is worth noting that most of them either do not work for analyzing probabilistic programs [1,11,13,17] or are based on a channel matrix with an impractical number of lines [4,7].

JPF-QIF [19] is a tool that computes an upper bound of the leakage by estimating the number of possible outputs of the system. JPG-QIF is much less precise than QUAIL, and it is not able for instance to prove that the security of an authentication increases by increasing the password size.

McCamant and Ernst [16] and Newsome, McCamant and Song [18] propose quantitative extensions of taint analysis. This approach, while feasible even for large programs, still does not allow to analyze probabilistic programs, making it unsuitable for security protocols.

Bérard et al. propose a quantification of information leakage based on mutual information, though they name it restrictive probabilistic opacity [2] and do not refer to some of the core papers of the subject, like the works of Clark, Hunt and Malacaria [8,9]. The approach tries to quantify leakage on probabilistic models, and is thus phylosophically close to ours. They compute mutual information as the expected difference between

prior and posterior entropy, and since the latter depends on all possible values of the secret we expect that an eventual implementation would be in general very inefficient compared to the QUAIL quotient-based approach.

References

1. Michael Backes, Boris Köpf, and Andrey Rybalchenko. Automatic discovery and quantification of information leaks. In *IEEE Symposium on Security and Privacy*, 2009.
2. Béatrice Bérard, John Mullins, and Mathieu Sassolas. Quantifying opacity. In G. Ciardo and R. Segala, editors, *QEST'10*. IEEE Computer Society, September 2010.
3. Fabrizio Biondi, Axel Legay, Pasquale Malacaria, and Andrzej Wasowski. Quantifying information leakage of randomized protocols. In Roberto Giacobazzi, Josh Berdine, and Isabella Mastroeni, editors, *VMCAI*. Springer, 2013.
4. Konstantinos Chatzikokolakis, Tom Chothia, and Apratim Guha. Statistical measurement of information leakage. In Javier Esparza and Rupak Majumdar, editors, *TACAS*, 2010.
5. Konstantinos Chatzikokolakis, Catuscia Palamidessi, and Prakash Panangaden. Anonymity protocols as noisy channels. In *Information and Computation*. Springer, 2006.
6. David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1:65–75, 1988.
7. Tom Chothia and Apratim Guha. A statistical test for information leaks using continuous mutual information. In *CSF*, pages 177–190. IEEE Computer Society, 2011.
8. David Clark, Sebastian Hunt, and Pasquale Malacaria. Quantitative analysis of the leakage of confidential data. *Electr. Notes Theor. Comput. Sci.*, 59(3):238–251, 2001.
9. David Clark, Sebastian Hunt, and Pasquale Malacaria. A static analysis for quantifying information flow in a simple imperative language. *Journal of Computer Security*, 15, 2007.
10. S. Kiefer, A. S. Murawski, J. Ouaknine, B. Wachter, and J. Worrell. Apex: An analyzer for open probabilistic programs. In *Proc. CAV'12*, LNCS. Springer, 2012.
11. Vladimir Klebanov. Precise quantitative information flow analysis using symbolic model counting. In Fabio Martinelli and Flemming Nielson, editors, *QASA*, 2012.
12. Boris Köpf and David A. Basin. An information-theoretic model for adaptive side-channel attacks. In *ACM Conference on Computer and Communications Security*, 2007.
13. Boris Köpf, Laurent Mauborgne, and Martín Ochoa. Automatic quantification of cache side-channels. In P. Madhusudan and Sanjit A. Seshia, editors, *CAV*, 2012.
14. M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Proc. CAV'11*, volume 6806 of *LNCS*. Springer, 2011.
15. Pasquale Malacaria. Algebraic foundations for information theoretical, probabilistic and guessability measures of information flow. *CoRR*, abs/1101.3453, 2011.
16. Stephen McCamant and Michael D. Ernst. Quantitative information flow as network flow capacity. In Rajiv Gupta and Saman P. Amarasinghe, editors, *PLDI*. ACM, 2008.
17. Chunyan Mu and David Clark. A tool: Quantitative analyser for programs. In *QEST*. IEEE Computer Society, 2011.
18. James Newsome, Stephen McCamant, and Dawn Song. Measuring channel capacity to distinguish undue influence. In S. Chong and D. A. Naumann, editors, *PLAS*. ACM, 2009.
19. Quoc-Sang Phan, Pasquale Malacaria, Oksana Tkachuk, and Corina S. Pasareanu. Symbolic quantitative information flow. *ACM SIGSOFT Software Engineering Notes*, 37(6):1–5, 2012.
20. Andrei Sabelfeld and Andrew C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, 2003.
21. Geoffrey Smith. On the foundations of quantitative information flow. In Luca de Alfaro, editor, *FOSSACS*, volume 5504 of *LNCS*, pages 288–302. Springer, 2009.

A Running QUAIL through an example

We detail in this appendix the result of the computations performed by QUAIL when analyzing the bit XOR example presented in Fig. 2. We consider the value of 0.8 for the random bit probability on line 4. We launch QUAIL as explained in Section 3:

```
quail bit_xor.quail -p 3 -v 0 -mc 0 1 2 3 4 -Tpdf
```

This will compute the leakage of the program with a precision of 3 digits for the computations. Using the options `-mc 0 1 2 3 4 -Tpdf` it will also produce 5 Markov chains in a PDF format, corresponding to the different intermediate steps of the computation. As a result the tool outputs the leakage of the program which is:

```
0.28
```

To compute this result Step 1 converts the input program given on the left of Fig. 2 into a preprocessed program shown on the right. This program is written in a separate file `bit_xor.quail.pp`.

In Step 2, QUAIL parses the preprocessed code and builds the Markov chain model shown in Fig. **??** that corresponds to all the executions of the program. Note that public variables are labeled with a precise value in each state (e.g. `l=0`) while private variables have intervals of allowed values (e.g. `h=[0,1]`).

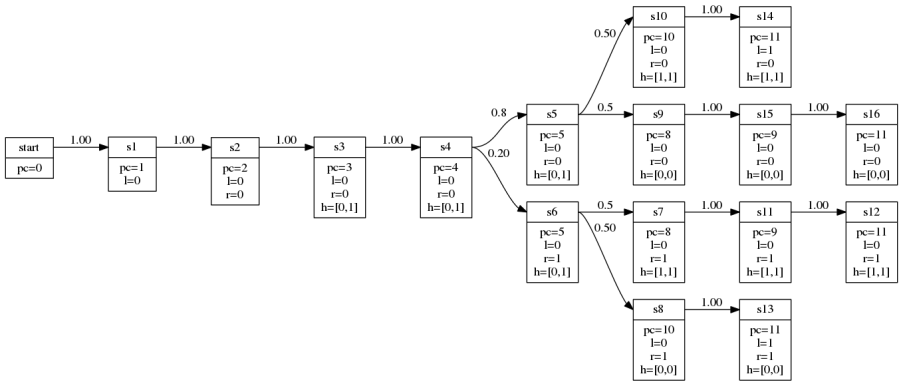


Fig. 3: Bit XOR example: Markov chain model of the program

In Step 3, QUAIL hides all non observable states, i.e. the internal states, and produces a Markov chain with only transitions between the initial states and the output states, as shown in Fig. **??**.

In Step 4, QUAIL computes the quotient Markov chains. In the attacker's quotient, since in both states s13 and s14 the observable variable l equals to 1, these two states are merged together, and similarly for the states s12 and s16 in which $l = 0$. The result is shown in Fig. **??**.

In the secret's quotient, the states s12 and s14 share the value $h = 0$ for the secret variable, whereas the states s13 and s16 share the value $h = 1$. These states are merged together as shown in Fig. **??**.

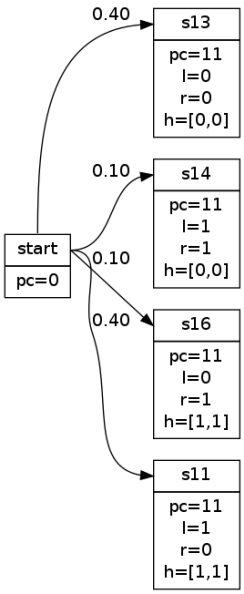


Fig. 4: Bit XOR example: observable reduction of the Markov chain model

Finally, in the joint quotient, we consider the intersection of the two previous discrimination relations, i.e. only states with the same value to both observable and secret variables can be merged together. This is not the case for any state in the observable reduction, and therefore the joint quotient Markov chain shown in Fig. **??** is similar to the original observable reduction of Fig. **??**.

In the final Step 5, QUAIL computes the entropy of the three quotients Markov chains. The entropy of the attacker's quotient is:

$$H_a = -0.5 * log(1/0.5) - 0.5 * log(1/0.5) = 1$$

The entropy of the secret's quotient is the same:

$$H_s = -0.5 * log(1/0.5) - 0.5 * log(1/0.5) = 1$$

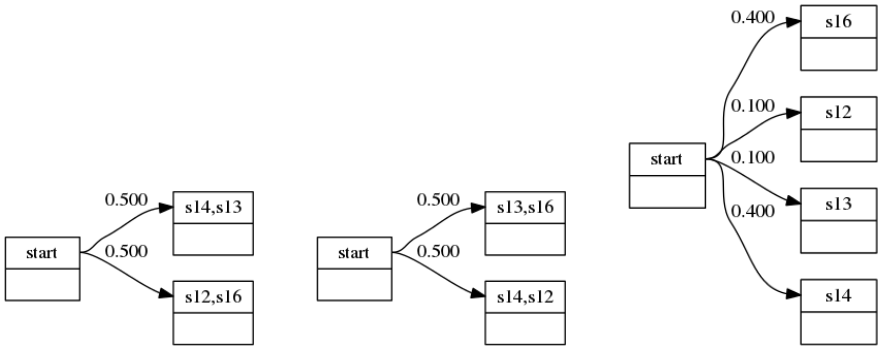
The entropy of the joint quotient is:

$$H_j = -0.4 * log(1/0.4) * 2 - 0.1 * log(1/0.1) * 2 = 1.058 + 0.664 = 1.722$$

The information leakage is the sum of the first two entropies minus the entropy of the joint quotient, therefore:

$$L = H_a + H_s - H_j = 2 - 1.722 = 0.278$$

This result is rounded to 0.28 in the final output.



(a) Attacker's quotient (b) Secret's quotient (c) Joint quotient

Fig. 5: Markov chains of the quotients

B QUAIL imperative language

B.1 Variable declarations

All variables in QUAIL are fixed sized integers. They are declared at the beginning of the program. Constants can be declared in the following manner:

```
const N := 4;
```

They are replaced by their value during the preprocessing step.

Public variables are either public or observable. In the latter case the attacker will be able to distinguish their value. They are declared in the following manner:

`public int4 var;` or `observable int4 var;`

declares a 4 bits integer variable whose name is var, either public or observable.

```
public int4 var := 5;
```

declares `var` and initializes it to value 5. Any expression can be used to initialized a variable, provided that the variables used in the expression are public or constants and have been previously declared. Variables not initialized are implicitly initialized to the value 0.

Private variables are either private or secret. The attacker will only infer knowledge on the latter. They are declared in the following manner:

`private int4 var;` or `secret int4 var;`

declares a 4 bits integer variable whose name is var, either private or secret.

```
private int4 var := [0,1][2,5];
```

declares `var` and restricts its range to the two intervals [0,1] and [2,5]. Again any expression can be used in the bounds of the intervals.

B.2 Arrays

Variables can also be arrays of integers and multi-dimensional arrays. Arrays are declared in addition to the integer type of a variable.

```
public array[4] of int4 tab;
```
declares a public variable `tab` that is an array of 4 bits integer of size 4 whose indices range from 0 to 3, while
```
public array[1..4] of int4 tab;
```
declares `tab` as an array of size 4 whose indices range from 1 to 4. The size of an array can be any expression that evaluates to an integer.

Arrays are replaced during the preprocessing. Therefore, an array variable named `tab`, whose indices range from 0 to 3, declares 4 variables, whose name are `tab[0]`, `tab[1]`, `tab[2]` and `tab[3]`. They have the same publicity and the same integer type as the array.

An array may be initialized with a set of initial values:
```
public array[1..4] of int4 tab := {1,1,2,2};
```
initializes `tab` such that `tab[1]` and `tab[2]` are equal to 1, while `tab[3]` and `tab[4]` are equal to 2. Private arrays can be initialized like any private variable, with a set of intervals:
```
private array[1..4] of int4 tab := [0,1];
```
In that case all the variables in the array are initialized to the same range of integers.

B.3 Expressions

Expressions are used in guards, assignments, variables initialization and arrays indices. Binary operators (`||`,`&&`,`^`,`+`,`-`,`*`,`/` and `%`) and unary operators (`-`,`!`) can used. Classical operators precedence is assumed. For boolean operations integer variables are considered as a true value if non null, and false if null. Only public variables, constants and integers can be used in expressions.

B.4 Guards

Guards are limited to a single comparison between a variable on the left side (either public, or private, or constant, or an integer value) and an expression on the right side. Any comparison operator among `<`, `>`, `<=`, `>=`, `==` and `!=` can be used.

B.5 Assignments

An assignment statement is written in the following manner:
```
assign var := expr;
```
where `var` is a public variable (possibly with indices) and *`expr`* is an expression containing no private variables.

B.6 Random assignments

The program can used two types of random primitives to assign values to a variable.
```
random var := random(expr_min,expr_max);
```
assigns to a public variable `var` a random value, chosen between the values of *`expr_min`* and *`expr_max`*, with a uniform probability distribution.

```
random var := randombit(p);
```
where p is a float value lower than 1, assigns to a public variable var a random bit value, that is 0 with probability p, and 1 with probability $1 - p$.

B.7 IF statements

IF conditional statements starts with the keyword if, possibly followed by elif and else, and ends with fi. The consequent statements are listed after the keyword then. For example the following structures are allowed:

```
if (h <= l) then assign var:=1;
fi
```

```
if (h <= l) then assign var:=1;
else assign var:=2;
assign var:=var+1;
fi
```

```
if (h <= l) then assign var:=1;
elif (h==l) then assign var:=2;
fi
```

```
if (h <= l) then assign var:=1;
elif (h==l) then assign var:=2;
elif (h==l+1) then assign var:=2;
else assign var:=2;
fi
```

B.8 WHILE statements

Conditional WHILE loop starts with the keyword while, followed by a guard, and the statements included in the loop are listed between the keywords do and od. For example the following structure is allowed:

```
while (h <= l) do
assign l := 1;
assign var := 2;
od
```

B.9 FOR statements

A FOR loop can be used to browse all the elements of an array. The syntax is:

```
for (var in tab) do
assign var := var+1;
od
```

The variable var is a local variable that must only be used inside the loop. It will take successively each value in the array tab. Note that if tab is a multi-dimensional array var is also an array.

B.10 Return statements

The program ends when a return statement is reached. Its syntax is simply:

```
return;
```

C Further examples

C.1 The Grade protocol

In the Grades protocol a group of k students $s_1, ..., s_k$ is given each a secret grade g_i between 0 and m-1. The students want to compute the sum of their grades without disclosing them. To this aim they produce k random numbers between 0 and $n = (m-1) * k + 1$ such that the number r_i is known only to the students s_i and $s_{(i+1)\%k}$. Then each student s_i outputs a number $d_i = g_i + r_i - r_{(i+1)\%k}$, and the sum of all grades is equivalent to the sum of the outputs modulo n. The input code for the Grade protocol is shown on Fig. **??** on the left.

To prove the security of the protocol, and thus the secrecy of the grades, we need to show that the information the attacker gains by knowing the declarations and the sum is the same as the information he would gain by knowing only the sum; the input code for the latter system is shown in Fig. **??** on the right. The leakage of the protocol for different numbers of students and grades is shown in Table **??**(a); the leakage of the protocol declaring only the final sum is shown in Table **??**(b). The tables are identical, demonstrating that when the attacker knows the students' declarations and the sum of the grades, she does not learn more information than the sum of the grades.

Table 2: Grades: leakage tables for attacker knowing a) the outputs and b) the sum only

(a)

Grades \ Students	**2**	**3**	**4**	**5**
2	1.500	1.811	2.030	2.198
3	2.197	2.525	2.745	2.910
4	2.655	2.984	3.201	3.365
5	2.999	3.325	3.541	timeout

(b)

Grades \ Students	**2**	**3**	**4**	**5**
2	1.500	1.811	2.030	2.198
3	2.197	2.525	2.745	2.910
4	2.655	2.984	3.201	3.365
5	2.999	3.325	3.541	timeout

C.2 The Dining Cryptographers protocol

The Dining Cryptographers protocol is an anonymity protocol in which a number of agents collaborate to a shared computation depending on each agent's secret [6]. A group of n cryptographers is dining around a round table. At the end of the dinner, the waiter informs them that the bill has already been settled by someone who would prefer to remain anonymous. The cryptographers respect the payer's wish for anonymity,

```
// S is the number of students
const S:=2;
// G is the number of grades (from 0 to G-1)
const G:=2;
// n is the number of possible random numbers generated
public int32 n;
// this is the sum that will be printed
observable int32 output;
// this is an internal counter for the sum
public int32 sum:=0;
// these are the random numbers; each one is shared between two students
public array [S] of int32 numbers;
// these are the public announcements of each student
observable array [S] of int32 announcements;
// there are S secret votes, each one with G possible values:
secret array [S] of int32 h := [0,G-1];
// these are just counters
public int32 i:=0;
public int32 j:=0;

// calculating n
assign n:=((G-1)*S)+1;

// generate the random numbers
for (num in numbers) do
  random num:=random(0,n-1);
od

// producing the declarations according to the secret value
while (i<S) do
  assign j:=0;
  while (j<G) do
    if (h[i]==j) then
      assign announcements[i]:=(j+numbers[i]-numbers[(i+1)%S])%n;
    fi
    assign j:=j+1;
  od
  assign i:=i+1;
od

//computing the sum, producing the output and terminating
for (a in announcements) do
  assign sum := sum+a;
od
assign output := sum%n;

return;
```

```
// S is the number of students
const S:=2;
// G is the number of grades (from 0 to G-1)
const G:=2;
// this is the sum that will be printed
observable int32 output;
// this is an internal counter for the sum
public int32 sum:=0;
// there are S secrets, each one with G possible values:
secret array [S] of int32 h := [0,G-1];
// these are just counters
public int32 i:=0;
public int32 j:=0;

// computing the sum of the secrets
while (i<S) do
  assign j:=0;
  while (j<G) do
    if (h[i]==j) then
      assign sum := sum + j;
    fi
    assign j:=j+1;
  od
  assign i:=i+1;
od

//producing the output and terminating
assign output := sum;

return;
```

Fig. 6: Grade example: input code for Grade (on the left) and for a protocol revealing only the sum of the grades (on the right).

but would like to know whether the benefactor is one of them or an external party. To determine this, each pair of adjacent cryptographers toss a coin hidden from everybody else, so that each cryptographers knows the value of the coin to its left and to its right. Then each cryptographers declares aloud the exclusive OR of the two coins he sees, i.e. 0 if they have the same value and 1 otherwise. If one of the cryptographers is the payer, he declares the opposite. In the end, if an even number of ones is declared then someone else paid the bill, while if an odd number of ones is declared one of the cryptographers is the benefactor.

Figure **??** on the left shows the QUAIL input code for the Dining Cryptographers protocol. The leakage of the protocol depends on the randomness of the coin that the cryptographers toss; as the coin become more deterministic, so the probability of getting a head gets closer to 0 or 1, the attacker is more able to determine the identity of the payer.

Some results are shown in Fig. **??** on the right; for different numbers of cryptographers we show that as the probability of the coin toss approaches 0 or 1 the leakage increases. When it is 0 or 1 the leakage is equivalent to the bit size of the secret, i.e. the logarithm in base 2 of $n + 1$, proving that the whole secret gets leaked, and thus the attacker learns the identity of the payer, whoever he is.

```
1  // N is the number of cryptographers at the
       table
2  const N:=3;
3
4  // this bit represents the output
5  observable int1 output;
6
7  // these bits represent the coin tosses
8  public array [N] of int1 coin;
9
10 // these are the observable coins
11 observable array[2] of int1 obscoin;
12
13 // this is just a counter
14 public int32 i:=0;
15
16 // these bits represent the bits declared
       by the three cryptographers
17 observable array [N] of int1 decl;
18
19 // the secret has N+1 possible values:
20 // 0 if someone else paid
21 // 1 if Cryptographer A paid
22 // 2 if Cryptographer B paid
23 // 3 if Cryptographer C paid
24 // ... and so on
25 secret int32 h := [0,N];
26
27 // tossing the coins
28 for (c in coin) do
29   random c:=randombit(0.5);
30 od
31
32 // if the attacker is one of the
       cryptographers, he can observe two of
       the coins.
33 // To encode an external attacker comment
       the next two lines.
34 assign obscoin[0]:=coin[0];
35 assign obscoin[1]:=coin[1];
36
37 // producing the declarations according to
       the secret value
38 while (i<N) do
39   assign decl[i]:=coin[i]^coin[(i+1)%N];
40   if (h==i+1) then
41     assign decl[i]:=!decl[i];
42   fi
43   assign i:=i+1;
44 od
45
46 //producing the output bit and terminating
47 for (d in decl) do
48   assign output := output^d;
49 od
50
51 return;
```

		Cryptographers			
		3	**4**	**5**	**6**
Coin probability	**0**	2	2.32	2.59	2.81
	0.1	1.76	1.90	2.03	2.15
	0.3	1.56	1.49	1.45	1.41
	0.5	1.50	1.37	1.25	1.15
	0.7	1.56	1.49	1.45	1.41
	0.9	1.76	1.90	2.03	2.15
	1.0	2	2.32	2.59	2.81

Fig. 7: Dining Cryptographers example: input code for the Dining Cryptographers (on the left) and leakage table as a function of the number of cryptographers and of the probability of the random coin toss (on the right).