Quantitative Security Analysis for Multi-threaded Programs

Tri Minh Ngo
University of Twente, Netherlands
triminhngo@gmail.com

Marieke Huisman
University of Twente, Netherlands
Marieke.Huisman@ewi.utwente.nl

Quantitative theories of information flow give us an approach to relax the absolute confidentiality properties that are difficult to satisfy for many practical programs. The classical information-theoretic approaches for sequential programs, where the program is modeled as a communication channel with only input and output, and the measure of leakage is based on the notions of *initial* uncertainty and *remaining* uncertainty after observing the final outcomes, are not suitable to multi-threaded programs. Besides, the information-theoretic approaches have been also shown to conflict with each other when comparing programs. Reasoning about the exposed information flow of multi-threaded programs is more complicated, since the outcomes of such programs depend on the scheduler policy, and the leakages in intermediate states also contribute to the overall leakage of the program.

This paper proposes a novel model of quantitative analysis for multi-threaded programs that also takes into account the effect of observables in intermediate states along the trace. We define a notion of the leakage of a program trace. Given the fact that the execution of a multi-threaded program is typically described by a set of traces, the leakage of a program under a specific scheduler is computed as the expected value of the leakages of all possible traces. Examples are given to compare our approach with the existing approaches.

1 Introduction

Qualitative and Quantitative Security Properties. Qualitative security properties, such as *noninterference* [9] and *observational determinism* [24, 11], are appropriate for applications like Internet banking, e-commerce, and medical information systems, where private data, like credit card details, and medical records, need strict protection. The key idea of such absolutely confidential properties is that secret information should not be derivable from public data. These properties prohibit any information flow from a high security level to a low level[1]. For example, the program $\texttt{if}\ (S > 0)\ \texttt{then}\ O := 0\ \texttt{else}\ O := 1$, where S is a private variable and O is a public variable, is rejected by qualitative security properties, since we can learn information about S from the value of O.

However, for many applications in which we want or need to reveal information that depends on private data, these absolutely confidential properties are not appropriate. A program that contains a leakage is rejected even when the leakage is *unavoidable*. An typical example is a *password checker*, where an attacker tries a string to guess the password. Even when the attacker makes a wrong guess, a small amount of secret information is still leaked, i.e., the attacker is able to derive that the password is not the string that he has just entered. In this example, the leakage is unavoidable.

Therefore, an alternative approach is to relax the definitions of absolute confidentiality by quantifying information flow and determining how much secret information is being leaked, i.e., expressing the amount of leakage in quantitative terms. A quantitative security policy offers a method to compute *bounds* on how much information is leaked. This information can be used to decide whether we can tolerate *minor* leakages. Quantifying information flow also provides a way to judge whether an application

[1]For simplicity, throughout this paper, we consider a simple two-point security lattice, where the data is divided into two disjoint subsets, of private (high) and public (low) security levels, respectively.

L. Bortolussi and H. Wiklicky (Eds.): QAPL 2013
EPTCS 117, 2013, pp. 34–48, doi:10.4204/EPTCS.117.3

leaks more information than another, although they may both be insecure. A quantitative theory thus can be seen a generalization of a theory of absolute confidentiality.

Quantitative Security Analysis for Sequential Programs. The foundation of quantitative analysis of information flow for sequential programs is the scenario that a program is considered as a channel in the information-theoretic sense, where the secret S is the input and the observables O are the output. An attacker, by observing O, might be able to derive information about S.

Quantitative security analysis studies how much information about S an attacker might learn from the output O. The analysis uses the notion of entropy. The entropy of a random variable expresses the *uncertainty* of an attacker about its value, i.e., how *difficult* it is for an attacker to discover its value. The leakage of a program is then defined as the difference between the secret's initial uncertainty, i.e., the uncertainty of the attacker about private data before executing the program, and the secret's remaining uncertainty, i.e., the uncertainty of the attacker after observing the program's public outcomes, i.e.,

Information leakage = Initial uncertainty - Remaining uncertainty.

In general, by observing the outcomes of the program, the attacker gains more knowledge about the initial private data. Thus, this reduces the initial uncertainty and causes the information leakage.

However, the existing approaches for sequential programs do not agree on a unique measure to quantify information flow. Past works have proposed several entropy measures to compute the program's leakage, i.e., *Shannon entropy*, *Rényi's min-entropy* and *Guessing entropy* (see [1] for more details). Since we expect that the right notion of entropy should make the value of the computed leakage non-negative, Guessing entropy is not suitable to quantify information flow of our interested programs in which the output is non-deterministic and probabilistic. Guessing entropy only guarantees the non-negativeness property of leakage for deterministic programs, i.e., the programs such that for each input, a unique output is produced. Therefore, for probabilistic systems, Shannon-entropy and Rényi's min-entropy are more appropriate. However, these measures might be in conflict, i.e., measures based on Shannon-entropy judge one program more dangerous than the other while min-entropy measures give an opposite result. Thus, it seems that no single measure is likely to be appropriate in all cases [2].

Quantitative Security Analysis for Multi-threaded Programs. With the trend of parallel systems, and the emergence of multi-core computing, multi-threading is becoming more and more standard. However, the question which measure, i.e., Shannon entropy or min entropy, is appropriate to quantify information flow of multi-threaded programs still needs an answer. Besides, the existing models of quantitative analysis are not appropriate for multi-threaded programs, since they consider only the input and the final outcomes of a program. For multi-threaded programs, due to the interactions between threads and the exchange of intermediate results, intermediate states should also be taken into account (see [24, 11] for more details). Thus, new methods have to be developed for an observational model where an attacker can access the full code of the program, and observe the traces of public data.

To obtain a suitable model of analyzing quantitatively information flow for multi-threaded programs, we need to (i) consider how the public values in the intermediate states also help an attacker to reduce his initial uncertainty about private data. Consequently, we need to define the leakage of an execution trace, i.e., the leakage given by a sequence of publicly observable data obtained during the execution of the program, and (ii) take into account the effect of schedulers on the overall leakage of the programs, since the outcomes of a multi-threaded program depend on the scheduling policy. Thus, we need to consider how the distribution of traces affects the overall leakage, i.e., the execution of a multi-threaded program always results in a set of traces.

This paper proposes a novel approach of quantitative analysis for multi-threaded programs. We model the execution of a multi-threaded program under the control of a probabilistic scheduler by a probabilistic state transition system. The probabilities of the transitions are decided by the scheduler that is used to deploy the program. States denote the probability distributions of private data S. The distribution of S changes from state to state along a trace, depending on the public values in the states and the program commands that result in such observables. This idea is sketched by the following example. Initially, given that an attacker knows that the true value of S is in the set $\{0,1,2,3\}$, his initial knowledge might be expressed by a uniform distribution of S, i.e., $\{0 \mapsto \frac{1}{4}, 1 \mapsto \frac{1}{4}, 2 \mapsto \frac{1}{4}, 3 \mapsto \frac{1}{4}\}$. Two program commands $O := S/2$ and $O := S \mod 2$ might result in the same O, e.g., $O = 1$, but if the result is from $O := S/2$, the updated distribution is $\{0 \mapsto 0, 1 \mapsto 0, 2 \mapsto \frac{1}{2}, 3 \mapsto \frac{1}{2}\}$, otherwise, it is $\{0 \mapsto 0, 1 \mapsto \frac{1}{2}, 2 \mapsto 0, 3 \mapsto \frac{1}{2}\}$.

The program can be seen a distribution transformer. During the program execution, the distribution of private data transforms from the initial distribution to the final ones in traces. The distributions of private data at the initial and final states of a trace can be used to present the *initial* uncertainty of the attacker about secret information, and his *final* uncertainty, after observing the sequence of public data along the trace, respectively. Based on these notions, we define the leakage of an execution trace as the difference between the *initial uncertainty* and the *final uncertainty*.

The quantitative term to denote uncertainty relates much to the model of attacker [1], i.e., an attacker can guess secret information by multiple tries, or just by one try. In our approach, we follow the *one-try threat* model, which is more suitable to many security situations i.e., the system will trigger an alarm when an attacker makes a wrong guess. Based on this threat model, we denote the initial and final uncertainty of the attacker by Rényi's min-entropies of the initial and final distributions of private data, respectively. Notice that in our approach, instead of the notion of *remaining* uncertainty as in classical approaches, we use the notion of *final* uncertainty. While remaining uncertainty depends only on the outcomes of the program, our notion of final uncertainty depends on the observables along the trace, and also on the program commands that result in such observables.

Given a multi-threaded program, a scheduler, the execution of the program under the control of the scheduler always results in a set of traces. The leakage of the program is then computed as the *expected* value of the leakages of traces. Via a case study (Section 4.5), we demonstrate how the leakage of a multi-threaded programs is measured. We also compare our approach with the existing quantitative analysis models. We believe that our approach gives a more accurate way to study quantitatively the security property of multi-threaded programs.

Organization of the Paper. Section 2 presents the preliminaries. Then, Section 3 discusses why the classical approaches are not suitable to quantify information flow of multi-threaded programs. Section 4 presents our quantitative analysis model, a case study, and also the comparison with the existing approaches. Section 5 discusses related work. Section 6 concludes, and discusses future work.

2 Preliminaries

2.1 Probabilistic Distribution

Let X and Y denote two discrete random variables with carriers $\mathbf{X} = \{x_1, \ldots, x_n\}$, $\mathbf{Y} = \{y_1, \ldots, y_m\}$, respectively. A *probability distribution* $p_X(\cdot)$ over a set $\mathbf{X}$ is a function $p_X : \mathbf{X} \to [0,1]$, such that the sum of the probabilities of all elements over $\mathbf{X}$ is 1, i.e., $\sum_{x \in \mathbf{X}} p_X(x) = 1$. If $\mathbf{X}$ is uncountable, then $\sum_{x \in \mathbf{X}} p_X(x) = 1$ implies that $p_X(x) > 0$ for countably many $x \in \mathbf{X}$.

We use $p_{X|Y}(x|y)$ to denote the *conditional* probability that $X = x$ when $Y = y$. We call $p_X(\cdot)$ a *priori* distribution of X, and $p_{X|Y}(\cdot|\cdot)$ a *posteriori* distribution. Thus, $p(x)$ and $p(x|y)$ are a *priori* and a *posteriori* probabilities of x.

2.2 Shannon Entropy

Definition 1 *The Shannon entropy of a random variable X is defined as [1],*

$$\mathscr{H}_{Shan}(X) = -\sum_{x\in\mathbf{X}} p(x)\log p(x),$$

where the base of the logarithm is set to 2*.*

Definition 2 *The conditional Shannon entropy of a random variable X given Y is [1],*

$$\mathscr{H}_{Shan}(X|Y) = \sum_{y\in\mathbf{Y}} p(y)\mathscr{H}_{Shan}(X|Y=y),$$

where $\mathscr{H}_{Shan}(X|Y=y) = -\sum_{x\in\mathbf{X}} p(x|y)\log p(x|y)$.

It is possible to prove that $0 \le \mathscr{H}_{Shan}(X|Y) \le \mathscr{H}_{Shan}(X)$. The minimum value of $\mathscr{H}_{Shan}(X|Y)$ is 0, if X is completely determined by Y. The maximum value of $\mathscr{H}_{Shan}(X|Y)$ is $\mathscr{H}_{Shan}(X)$, when X and Y are independent.

2.3 Min-Entropy

Definition 3 *The Rényi's min-entropy of a random variable X is defined as [21],*

$$\mathscr{H}_{Rényi}(X) = -\log \max_{x\in\mathbf{X}} p(x),$$

Notice that Shannon entropy and Rényi's min-entropy coincide on uniform distributions.

Rényi did not define the notion of conditional min-entropy, and there is no unique definition of this notion. Smith [21] considers the following definition.

Definition 4 *The conditional min-entropy of a random variable X given Y is,*

$$\mathscr{H}_{Min}(X|Y) = -\log \sum_{y\in\mathbf{Y}} p(y)\cdot \max_{x\in\mathbf{X}} p(x|y).$$

3 Classical Models of Quantitative Information Flow

This section presents classical models of quantitative security analysis, and discusses why they are not suitable to quantify information leakage of multi-threaded programs.

3.1 Information Leakage in the Classical Approaches

Existing works [17, 8, 5, 15, 14, 27, 21, 3] propose to use information theory as a setting to model information flow. A program is seen a standard input-output model with the secret S as the input and the public outcomes O as the output. Let $\mathscr{H}(S)$ denote the initial uncertainty of an attacker about secret information, and $\mathscr{H}(S|O)$ denote the uncertainty after the programs has been executed and the public outcomes are observed. The leakage of the program is given by,

$$\mathscr{L}(P) = \mathscr{H}(S) - \mathscr{H}(S|O).$$

where $\mathscr{L}(P)$ denotes the leakage of P; $\mathscr{H}$ might be either Shannon entropy or min-entropy.

3.2 Basic Settings for Leakage

To argue why a program is considered more dangerous than the another, we need to setup some basic settings for the discussion.

First, we consider the *one-try threat model*, i.e., the attacker is allowed to guess the value of S by only one try. Secondly, it is often the case that in practice, the attacker only knows the possible values of private data, but he does not know which value is likely to be the correct one. Thus, the attacker's initial knowledge about the secret is denoted by the uniform distribution on its possible values. In other words, we assume that initially, the attacker knows nothing but the uniform prior distribution of private data.

Finally, we also assume that systems are two-point security lattice, and data in the same category, i.e., high or low security level, are indistinguishable in *security meaning*. Thus, a system that leaks the last 9 bits of private data is considered just as dangerous as a system that leaks the first 9 bits.

3.3 Classical Measures might be Counter-intuitive and Conflict

Many authors [17, 8, 5, 15, 14, 27] measure leakage using Shannon entropy. However, Smith [21] shows that in the context of the one-try threat model, measures based on Shannon entropy do not always result in a very good operational security guarantee. In particular, Smith [21] shows that Shannon-entropy measures might be counter-intuitive. He shows two programs P and P' such that by intuitive understanding, P leaks more information than P', but Shannon-entropy measures yield a counter-intuitive result, i.e., $\mathscr{L}_{Shan}(P) < \mathscr{L}_{Shan}(P')$, where $\mathscr{L}_{Shan}()$ denotes the leakage given by Shannon-entropy measure. For example, consider the two following programs P_1 and P_2, from [21].

Example 1 (Program P_1)

$$\texttt{if}\,(S \bmod 8 = 0)\ \texttt{then}\ O := S\ \texttt{else}\ O := 1;$$

Basically, P_1 copies S to O when S is a multiple of 8, otherwise, it sets O to 1. Assume that S is a 64-bit unsigned integer, $0 \leq S < 2^{64}$. According to Shannon-entropy measure [21], the leakage of P_1 is $\mathscr{L}_{Shan}(P_1) = 8.17$.

Example 2 (Program P_2)

$$O := S \,\&\, (0\ldots0.111.111.111)_{\mathtt{b}};$$

where $(0\ldots0.111.111.111)_{\mathtt{b}}$ *is the 64-bit binary number such that the first 55 bits are* 0 *and the last 9 bits are* 1.

This program simply copies the last 9 bits of S into O. Shannon-entropy measure gives $\mathscr{L}_{Shan}(P_2) = 9$.

In P_1, whenever the public outcome $O \neq 1$, the attacker obtains the secret S completely. Thus, the expected probability of guessing S by one try is greater than $\frac{1}{8}$. In P_2, for any value of O, the probability of guessing S by one try is 2^{-55}, since the first 55 bits of S are still unknown. It means that, in the one-try threat model, intuitively, P_1 is considered more dangerous than P_2. However, the measures based on Shannon entropy judge P_2 worse than P_1.

For this reason, Smith develops an alternative theory of quantitative information flow based on min-entropy [21]. Smith defines uncertainty in terms of the *vulnerability* of S to be guessed in one try. The vulnerability of a random variable X is the maximum of the probabilities of the values of X. This approach seems match the intuitive idea of the one-try threat model, i.e., the attacker always chooses the value with the maximum probability.

However, min-entropy measures might still result in counter-intuitive values of leakage. Consider the two following programs P_3 and P_4, from [21, 26].

Example 3 (Program P_3: Password Checker)

$$\texttt{if}\,(S = L)\,\texttt{then}\,O := 1\,\texttt{else}\,O := 0;$$

where S denotes the password, L the string entered by the attacker, and O the observable answer, i.e., right or wrong.

Example 4 (Program P_4: Binary Search)

$$\texttt{if}\,(S \geq L)\,\texttt{then}\,O := 1\,\texttt{else}\,O := 0;$$

where $L = |S|/2$ is a program parameter.

Assume that S is a uniformly-distributed unsigned integer. The measures based on min-entropy do not distinguish between P_3 and P_4 by always judging that their leakages are the same, i.e., $\mathscr{L}_{Min}(P_3) = \mathscr{L}_{Min}(P_4) = 1$. However, if $|S|$ is large, the probability that $S = L$ becomes so low in P_3, i.e., $p(O = 1) = \frac{1}{|S|} \approx 0$. Thus, intuitively, P_3 leaks almost nothing, since the probability of one-try guessing S after observing the outcome is $\frac{1}{|S|-1} \approx \frac{1}{|S|}$. Program P_4 always leaks 1 bit of information, since the probability of guessing S after observing the outcome is $\frac{2}{|S|}$. Thus, program P_4 should be judged more dangerous than P_3. In his paper [21], Smith also admits that P_3 and P_4 should be treated differently. It is trivial that the uncertainty of the password guessing decreases slowly, while in binary search, the uncertainty of the secret decreases very rapidly.

Notice that for this example, Shannon-entropy measures give $\mathscr{L}_{Shan}(P_3) = 0$ and $\mathscr{L}_{Shan}(P_4) = 1$, that match the intuition. Therefore, there is growing appreciation that no single leakage measure is likely to be appropriate in all cases. This puts a question which measure is more suitable to quantify information leakage of multi-threaded programs.

3.4 Leakages in Intermediate States

Classical approaches of quantitative security analysis were based on the input-output model. However, for multi-threaded programs, due to the interactions between threads and the exchange of intermediate results, one should also consider the leakages in intermediate states along the traces. Consider the following example where S is a 3-bit binary number.

Example 5 *Given that $(100)_b$ and $(011)_b$ are the binary forms of 4 and 3, respectively.*

$$\begin{array}{l} O := 0; \\ O := S \,\&\, (100)_b; \\ O := S \,\&\, (011)_b; \end{array}$$

Let $(s_3 s_2 s_1)_b$ denote the binary form of S. The existing input-output models judge that this program leaks 2 bits of private data in the output, i.e., $O = (s_3 s_2 s_1)_b \,\&\, (011)_b = (0 s_2 s_1)_b$. However, due to the leakage in the intermediate state, i.e., $O = (s_3 s_2 s_1)_b \,\&\, (100)_b = (s_3 00)_b$, the attacker obtains the full secret value.

However, notice that leakages in intermediate states do not always contribute to the overall leakage of a trace, as in the following example,

Example 6

$$O := 0;$$
$$O := S \,\&\, (001)_b;$$
$$O := S \,\&\, (011)_b;$$

The overall leakage of this program trace is only 2 bits, since the leakage in the intermediate state, i.e., the last bit s_1, is included in the leakage in the output. This example shows that leakage of a program trace is not simply the sum of leakages of transition steps along the trace, as in the approach of Chen et al. [7] (We discuss this approach more in the Related Work).

3.5 Scheduler's Effect

The classical quantitative analysis of information flow do not take into account the effect of schedulers. Since the set of possible traces of a multi-threaded program depends on the scheduler that is used to execute the program, schedulers cannot be ignored when quantifying information flow. Consider the following example,

Example 7 *Given that* $\|$ *is the parallel operator, and S is a 2-bit secret information,*

$$O := S/2 \,\|\, O := S \mod 2.$$

Execute this program with a uniform scheduler, i.e., a scheduler that picks threads with the same probability in a time slot. Since S is a uniform 2-bit data, there are 4 possible traces with the same probability of occurrence, i.e., $\{00, 01, 10, 11\}$. If the obtained trace is either 00 or 11, the attacker can conclude for sure that the value of S is 0, or 3, respectively. However, if the trace is 01 or 10, he is only able to derive that S is either 1 or 2. Thus, with this scheduler, the secret is not leaked totally.

However, when the attacker chooses a scheduler which always executes $O := S/2$ first, he also obtains 4 possible traces $\{00, 01, 10, 11\}$ with the same probability of occurrence, but in this case, he can always derive the value of S correctly.

4 Quantitative Security Analysis for Multi-threaded Programs

4.1 Probabilistic Kripke Structures

We consider probabilistic Kripke structures (PKS) that can be used to model the semantics of probabilistic programs in a standard way [10, 19]. PKSs are like standard Kripke structures [13], except that each state transition $c \to \mu$ leads to a probability distribution μ over the next states, i.e., the probability to end up in state c' is $\mu(c')$ ($c, c' \in$ a set of states $Conf$). Each state may enable several probabilistic transitions, modeling different execution orders to be determined by a scheduler.

We assume that the set of variables of the program Var is partitioned into a set of low variables O and a set of high variables S, i.e., $Var = O \cup S$, with $O \cap S = \emptyset$. To aim for simplicity and clarity, rather than full generality, following [21], our development therefore restricts to programs with only one high and one low variables. Our goal is to quantify how much information about S is deduced by an attacker who can observe the set of execution traces of O. This restriction aims to demonstrate the core idea of the analysis process. However, the analysis can adapt to more complex multi-threaded programs easily after some trivial modifications.

Our PKSs label states with the distributions of S. Thus, each state c is labeled by a labeling function $V : Conf \to \mathscr{D}(S)$ that assigns a distribution $p_S \in \mathscr{D}(S)$ to each $c \in Conf$.

Definition 5 (Probabilistic Kripke structure) *A* probabilistic Kripke structure $\mathscr{A}$ *is a tuple* $\langle Conf, I, Var, V, \to \rangle$ *consisting of (i) a set Conf of* states*, (ii) an initial state* $I \in Conf$*, (iii) a finite set of variables* $Var = O \cup S$*, (iv) a* labeling function $V : Conf \to \mathscr{D}(S)$*, (v) a* transition relation $\to \subseteq Conf \times \mathscr{D}(Conf)$.

A PKS is *fully probabilistic* if each state has at most one outgoing transition, i.e., if $c \to \mu$ and $c \to \mu'$ implies $\mu = \mu'$. We assume that programs always terminate.

Traces. A *trace* T in $\mathscr{A}$ is a sequence $T = c_0 c_1 c_2 \dots c_n$ such that (i) $c_i \in Conf, c_0 = I$, and (ii) for all $0 \le i < n$, there exists a transition $c_i \to \mu$ with $\mu(c_{i+1}) > 0$. Let $Trace(\mathscr{A})$ denote the set of traces of $\mathscr{A}$.

4.2 Probabilistic Schedulers

A probabilistic scheduler is a function that implements a scheduling policy [20], i.e., that decides with which probabilities the threads are selected. To be general, we allow a scheduler to use the full history of computation to make decision: given a path ending in some state c, a scheduler chooses which of the enabled transitions in c to execute [19]. Since each transition results in a distribution of states, a probabilistic scheduler returns a distribution of distributions of states[2].

Definition 6 *A scheduler* δ *for* $\mathscr{A} = \langle Conf, I, Var, V, \to \rangle$ *is a function* $\delta : Trace(\mathscr{A}) \to \mathscr{D}(\mathscr{D}(Conf))$*, such that, for all traces* $T \in Trace(\mathscr{A})$*,* $\delta(T)(\mu) > 0$ *implies* $last(T) \to \mu$*, where* $last(T)$ *denotes the final state on* T.

The effect of a scheduler δ on a PKS $\mathscr{A}$ can be described by a PKS $\mathscr{A}_\delta$, i.e., the unreachable states of $\mathscr{A}$ under the scheduler δ are removed by the transition relation $\to_\delta$. Since all nondeterministic choices in $\mathscr{A}$ have been resolved by δ, $\mathscr{A}_\delta$ is fully probabilistic. The probability $p(T)$ given to a trace $T = c_0 c_1 \dots c_n$ is determined by $\delta(c_0)(c_1) \cdot \delta(c_0 c_1)(c_2) \cdots \delta(c_0 c_1 \dots c_{n-1})(c_n)$.

To summarize, our approach assumes that the prior distribution of private data is uniform. This distribution represents the attacker's initial uncertainty about the private data. The program is considered a distribution transformer, from the initial distribution at the input to the final ones at the outputs.

4.3 Leakage of a Program Trace

Example 6 shows that the leakage of a program trace is not simply the sum of leakages of transition steps along the trace. This section addresses how we can compute the leakage of a program trace. Consider again Example 6,

$$\begin{aligned} &O := 0;\\ &O := S \,\&\, (001)_b;\\ &O := S \,\&\, (011)_b; \end{aligned}$$

Let $(s_3 s_2 s_1)_b$ denote the binary form of S. The execution of this program results in only one trace, i.e., $\langle (000)_b \rangle \longrightarrow \langle (00 s_1)_b \rangle \longrightarrow \langle (0 s_2 s_1)_b \rangle$, where a state $\langle\rangle$ is represented by the value of O in that state.

Assume that $s_2 = 1$ and $s_1 = 1$, the obtained trace is $\langle (000)_b \rangle \longrightarrow \langle (001)_b \rangle \longrightarrow \langle (011)_b \rangle$.

[2]Thus, we assume a discrete probability distribution over the uncountable set $\mathscr{D}(Conf)$; only the countably many transitions occurring in $\mathscr{A}$ can be scheduled with a positive probability.

At the initial state $\langle(000)_b\rangle$, following our framework setting, i.e., the attacker only knows the possible values of private data, but he does not know which value is likely to be the true one, the attacker's initial uncertainty is represented by the uniform distribution of S, i.e., $\{0 \mapsto \frac{1}{8}, 1 \mapsto \frac{1}{8}, 2 \mapsto \frac{1}{8}, 3 \mapsto \frac{1}{8}, 4 \mapsto \frac{1}{8}, 5 \mapsto \frac{1}{8}, 6 \mapsto \frac{1}{8}, 7 \mapsto \frac{1}{8}\}$. At the state $\langle(001)_b\rangle$, the attacker learns that the last bit of S is 1. Thus, the distribution of S changes, for example, S cannot be 0. Hence, the updated distribution at state $\langle(001)_b\rangle$ is $\{0 \mapsto 0, 1 \mapsto \frac{1}{4}, 2 \mapsto 0, 3 \mapsto \frac{1}{4}, 4 \mapsto 0, 5 \mapsto \frac{1}{4}, 6 \mapsto 0, 7 \mapsto \frac{1}{4}\}$. Similarly, at final state $\langle(011)_b\rangle$, the updated distribution is $\{3 \mapsto \frac{1}{2}, 7 \mapsto \frac{1}{2}\}$[3].

Based on the final distribution of S, the attacker derives that the value of S is either 3 or 7. His uncertainty on secret information is reduced by the knowledge gained from the observation of the trace.

Since the program is a distribution transformer, the distribution of private data at the initial state of a trace can present the *initial* uncertainty of the attacker about the secret, and the distribution of private data at the final state can present his *final* uncertainty, after the trace has been observed. Thus, we can define the leakage of a program trace as,

Leakage of a program trace = Initial uncertainty - Final uncertainty.

The question arising is that how we represent the notion of uncertainty by a quantitative term.

Given a distribution of private data, the *best* strategy of the one-try threat model is to choose the value with the maximum probability. 'Best' means that this strategy induces the smallest probability of guessing wrong private data. Let S be private data with carrier $\mathbf{S}$, the value that affects the notion of uncertainty is $\max_{s\in\mathbf{S}} p(s)$. If $\max_{s\in\mathbf{S}} p(s) = 1$, the uncertainty must be 0, i.e., the attacker already knows the value of S. Thus, the notion of uncertainty is computed as the negation of logarithm of $\max_{s\in\mathbf{S}} p(s)$, i.e., uncertainty = $-\log\ \max_{s\in\mathbf{S}} p(s)$, where the negation is used to ensure the nonnegativeness property.

This measure coincides with the notion of Rényi's min-entropy. Thus, given a distribution of S, the uncertainty of the attacker about the secret in our approach is: Uncertainty = $\mathscr{H}_{Rényi}(S)$.

Therefore, leakage of a program trace T is,

$$\mathscr{L}(T) = \mathscr{H}_{Rényi}(S^{\mathbf{i}}_T) - \mathscr{H}_{Rényi}(S^{\mathbf{f}}_T),$$

where $\mathscr{H}_{Rényi}(S^{\mathbf{i}}_T)$ is Rényi's min-entropy of S with the initial distribution, and $\mathscr{H}_{Rényi}(S^{\mathbf{f}}_T)$ is Rényi's min-entropy of S with the final distribution, i.e., the distribution of the secret at the final state in T.

Following our measure, in Example 6, $\mathscr{L}(T) = -\log\frac{1}{8} - (-\log\frac{1}{2}) = 2$. This value matches the intuitive understanding that Example 6 leaks 2 bits of the private data.

Consider again Example 5. Assume that $s_1 = s_2 = s_3 = 1$, the execution of this program results in only one trace, i.e., $\langle(000)_b\rangle \longrightarrow \langle(100)_b\rangle \longrightarrow \langle(011)_b\rangle$.

At the state $\langle(100)_b\rangle$, the attacker learns that the first bit of S is 1. Thus, the distribution of S is $\{4 \mapsto \frac{1}{4}, 5 \mapsto \frac{1}{4}, 6 \mapsto \frac{1}{4}, 7 \mapsto \frac{1}{4}\}$. At final state $\langle(011)_b\rangle$, the distribution is updated to $\{7 \mapsto 1\}$, which is different from the final distribution in Example 6. Hence, $\mathscr{L}(T) = -\log\frac{1}{8} - (-\log 1) = 3$. This result also matches the intuition that the attacker is able to derive the value of S precisely by observing the execution trace of Example 5.

Notice that in our approach, instead of the notion of *remaining* uncertainty, we use the notion of *final* uncertainty. Both notions of initial and final uncertainty are denoted by the *same* notion of entropy, i.e., Rényi's min-entropy. The notion of remaining uncertainty depends only on the outcomes of the trace, while final uncertainty in our approach is based on the distribution computed for the final state, which takes into account the public values in the intermediate states along the trace, and also the program commands that result in such observables. This idea makes the following expression of program's leakage different from the one proposed by Smith [21].

[3]We leave out the elements that have probability 0.

4.4 Leakage of a Multi-threaded Program

The execution of a multi-threaded program P under the control of a scheduler δ often results in a set of traces, i.e., $Trace(P)$. Therefore, the leakage of P is computed as the *expected* value of the leakages of its traces, i.e.,

$$\begin{aligned}\mathscr{L}(P) &= \sum_{T\in Trace(\mathscr{A}_\delta)} p(T)\cdot\mathscr{L}(T)\\ &= \sum_{T\in Trace(\mathscr{A}_\delta)} p(T)(\mathscr{H}_{Rényi}(S^{\mathbf{i}}_T) - \mathscr{H}_{Rényi}(S^{\mathbf{f}}_T)).\end{aligned}$$

Since $\mathscr{H}_{Rényi}(S^{\mathbf{i}}_T)$ is the same for any $T \in Trace(\mathscr{A}_\delta)$, thus for notational convenience, we simply write it as $\mathscr{H}_{Rényi}(S^{\mathbf{i}})$. We rephrase the above expression as follows,

$$\mathscr{L}(P) = \mathscr{H}_{Rényi}(S^{\mathbf{i}}) - \sum_{T\in Trace(\mathscr{A}_\delta)} p(T)\cdot\mathscr{H}_{Rényi}(S^{\mathbf{f}}_T).$$

Since the initial distribution of S is uniform, $\mathscr{H}_{Rényi}(S^{\mathbf{i}})$ has the maximum value. Thus, $\mathscr{H}_{Rényi}(S^{\mathbf{i}}) \geq \mathscr{H}_{Rényi}(S^{\mathbf{f}}_T)$; then $\mathscr{L}(P) \geq \mathscr{H}_{Rényi}(S^{\mathbf{i}}) - \sum_{T\in Trace(\mathscr{A}_\delta)} p(T)\cdot\mathscr{H}_{Rényi}(S^{\mathbf{i}}) = 0$.

Hence, following our approach, the computed leakage of any program is always non-negative, i.e., $\mathscr{L}(P) \geq 0$ for all P. When $\mathscr{L}(P) = 0$, the program is totally secure.

4.5 A Case Study

This part illustrates how the leakage of a multi-threaded program is computed. The following case study also shows that our measure is more precise than the measures given by the classical approaches. Consider the following example,

Example 8 (Program P_8)

$$\begin{array}{l} O := 0;\\ \{\texttt{if}\,(O=1)\ \texttt{then}\ O := S/4\ \texttt{else}\ O := S \bmod 2\} \,\|\, O := 1;\\ O := S \bmod 4;\end{array}$$

where S is a 3-bit unsigned integer.

The execution of this program under the control of a uniform scheduler is illustrated by a PKS $\mathscr{A}$ in Figure 1. The PKS consists of 20 states that are numbered from **0** (the initial state) to **19**. The content of each state is the value of O in that state, e.g., in the initial state, the value of O is 0, which corresponds with the first command of the program $O := 0$.

Let C_1 and C_2 denote the left and right threads of the parallel composition operator. Since we consider the uniform scheduler, either thread C_1 or C_2 can be picked next with the same probability $\frac{1}{2}$. If the scheduler picks C_2 before C_1, $\mathscr{A}$ evolves from state **0** to state **1**, where $O = 1$. If C_1 is picked first, $\mathscr{A}$ might evolve from state **0** to either state **2** or state **3** with the same probability $\frac{1}{4}$. Since the current value of O is 0, i.e., the value of O in state **0**, the command $O := S \bmod 2$ is executed. Since the possible values of S are $\{0,\ldots,7\}$, the outcome O might be 0 (state **2**) if $S \in \{0,2,4,6\}$, or 1 (state **3**) if $S \in \{1,3,5,7\}$.

At state **1**, $\mathscr{A}$ might evolve to either state **4** or state **5** with the same probability. Since currently, O is 1, the command $O := S/4$ is executed. Thus, O might be 0 if $S \in \{0,1,2,3\}$, or 1 if $S \in \{4,5,6,7\}$.

The PKS $\mathscr{A}$ evolves from one state to another until the execution terminates, i.e., when the last command $O := S \bmod 4$ is executed.

The initial uncertainty of the attacker about S is denoted by the uniform distribution, i.e., $\{0 \mapsto \frac{1}{8}, 1 \mapsto \frac{1}{8}, 2 \mapsto \frac{1}{8}, 3 \mapsto \frac{1}{8}, 4 \mapsto \frac{1}{8}, 5 \mapsto \frac{1}{8}, 6 \mapsto \frac{1}{8}, 7 \mapsto \frac{1}{8}\}$. At state **1**, the distribution of S is still uniform since the

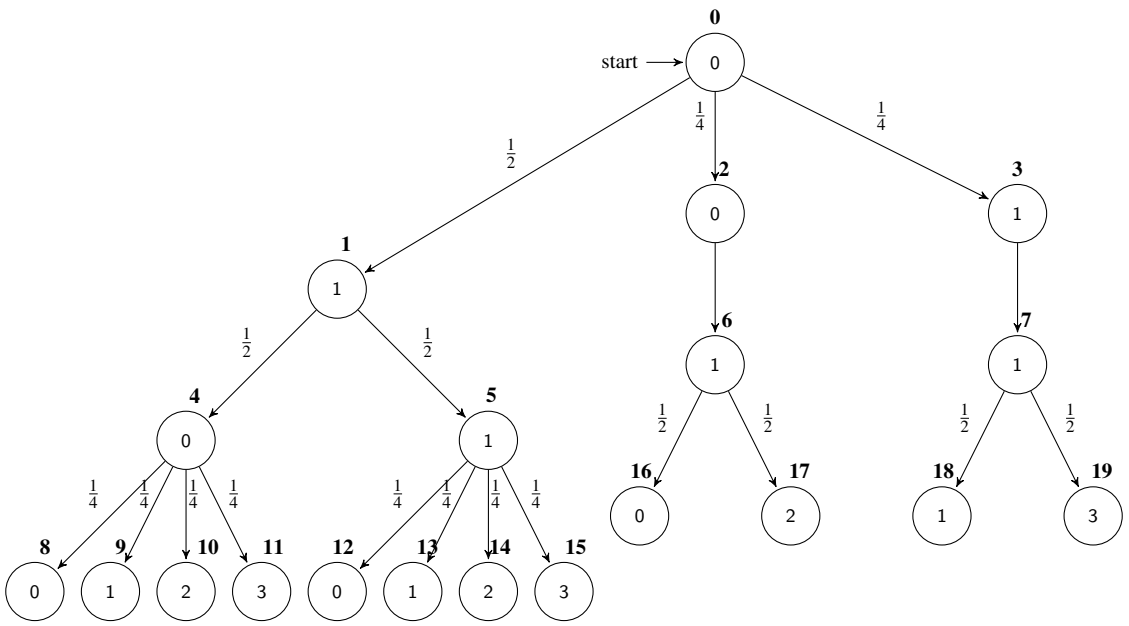

Figure 1: Model of the Program Execution

attacker learns nothing from the command $O := 1$. At state **4**, since the execution of $O := S/4$ results in 0, the attacker learns that the true value of O must be in the set $\{0,1,2,3\}$. Thus, the updated distribution of S at this state is $\{0 \mapsto \frac{1}{4}, 1 \mapsto \frac{1}{4}, 2 \mapsto \frac{1}{4}, 3 \mapsto \frac{1}{4}\}$. In the next step, the outcome of $O := S \bmod 4$ helps the attacker derive S precisely, e.g., at state **8**, since $O = 0$, the distribution of S is $\{0 \mapsto 1\}$. Similarly, the attacker is also able to derive S precisely, basing on the final distributions at states **9**, ..., **15**.

At state **2**, the execution of $O := S \bmod 2$ results in 0. Thus, the distribution of S at this state is $\{0 \mapsto \frac{1}{4}, 2 \mapsto \frac{1}{4}, 4 \mapsto \frac{1}{4}, 6 \mapsto \frac{1}{4}\}$. This distribution remains unchanged at state **6**, since no information is gained from the execution of $O := 1$. At state **16**, the update distribution of S is $\{0 \mapsto \frac{1}{2}, 4 \mapsto \frac{1}{2}\}$ since the execution of $O := S \bmod 4$ results in 0. The same form of distributions is obtained at states **17**, **18**, **19**.

Among the 12 possible traces, 8 traces have the final uncertainty 0, i.e., $\mathscr{H}_{Rényi}(S_T^{\mathbf{f}}) = -\log 1 = 0$, and the other 4 traces have the final uncertainty 1, i.e., $\mathscr{H}_{Rényi}(S_T^{\mathbf{f}}) = -\log \frac{1}{2} = 1$. The probability of traces with the final uncertainty 0, i.e., traces end in state **8**, ..., **15**, is equal to the probability of traces with the final uncertainty 1. Thus, according to our approach,

$$\mathscr{L}(P_8) = 3 - (\frac{1}{2} \cdot 0 + \frac{1}{2} \cdot 1) = 2.5.$$

This value coincides with the real leakage of the program. As we see, the last command $O := S \bmod 4$ always reveals the last 2 bits of S. The first bit of S might be leaked with the probability $\frac{1}{2}$, depending on whether the scheduler picks thread C_2 first or not. Thus, the real leakage of this program is 2.5.

4.6 Comparison

To reason about the exposed information flow of multi-threaded programs, Malacaria et al. [15] and Andrés et al. [3] propose to reuse the classical information-theoretic approach but the observables are traces of public variables, instead of only the final outcomes. Consider the above case study. Since the possible values of S are $\{0,\ldots,7\}$, the execution of P_8 results in the following traces,

S	0		1		2		3		4		5		6		7	
$T_{\mid O}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1	1
	1	0	1	0	1	0	1	0	1	1	1	1	1	1	1	1
	0	0	1	1	2	2	3	3	0	0	1	1	2	2	3	3

where $T_{\mid O}$ denotes the trace of O.

By observing the traces of O, the attacker is able to derive information about S. For example, if the obtained trace is 0100, the attacker can derive S precisely, since this trace is produced only when $S = 0$. If the trace is 0010, the attacker can conclude that S is either 0 or 4 with the same probability, i.e., $\frac{1}{2}$. If the trace is 0111, the possible value of S is either 1 or 5, but with different probabilities, i.e., the probability of S to be 5 is $\frac{2}{3}$.

There are 6 traces such that the attacker is able to derive the value of S precisely from them. There are 4 traces such that the attacker is able to guess S correctly with the probability $\frac{1}{2}$, and 6 traces with the probability $\frac{2}{3}$. Therefore,

$$\mathscr{L}_{Min}(P_8) = 3 - (-\log(\frac{6}{16} \cdot 1 + \frac{4}{16} \cdot \frac{1}{2} + \frac{6}{16} \cdot \frac{2}{3})) = 2.585.$$

This computed leakage does not match the intuitive understanding, as we have just argued above. Notice that the classical approaches with the input-output model judge that this program leaks only 2 bits of information, i.e., for each final outcome, the attacker might derive the secret with the probability $\frac{1}{2}$.

While the information-theoretic approach uses the notion of conditional min-entropy proposed by Smith [21] to denote the remaining uncertainty, we use the same notion of entropy to denote the initial and final uncertainty. The main difference is the position of the log in the expression of leakage. The idea of using logarithm is to express the notion of uncertainty in bits. Thus, the log should apply only to the distribution of private data which represents the uncertainty of the attacker, as in our approach. For multi-threaded programs, the distribution of traces depends strongly on schedulers, and thus we should distinguish between the distribution of traces and the distribution of private data. The notion of conditional min-entropy proposed by Smith [21], in which the logarithm applies to the combination of two distributions, do not distinguish between these two kinds of distribution, and thus it might cause imprecise results.

Notice that the expression $\sum_{T \in Trace(\mathscr{A}_\delta)} p(T) \cdot \mathscr{H}_{Rényi}(S^{\mathbf{f}}_T)$ is similar to the notion of conditional min-entropy defined by Cachin [4]. However, in our approach, we also take into account the effect of the scheduler, i.e., the transformation of distributions of private data depends also on which program commands producing the public values in the states.

4.7 Technique for Computing Leakage

Another aspect is to develop a technique for computing the leakage of a multi-threaded program. The analysis consists of two steps. First, the execution of a multi-threaded program P under the control of a given scheduler is modeled as a PKS $\mathscr{A}$ in a standard way: The states of $\mathscr{A}$ are tuples $\langle P, V \rangle$, consisting of a program fragment P and a valuation $V : Conf \to \mathscr{D}(S)$. The state transition relation $\to$ follows the small-step semantics of C. Based on the executed command and the obtained public result, the distribution of private data at each state is derived. We apply Kozen's probabilistic semantics [12] to present the transformation of probability distribution of S during the program execution. The intuition behind a state transition is that it transforms an *input* distribution of S to an *output* distribution so that the execution of a multi-threaded program results in a set of traces of probability distribution of S.

The computed leakage then follows trivially as the difference between the initial uncertainty and the expected value of the final uncertainties of all traces.

5 Related Work

Several authors propose to use the concept from information theory to define leakage quantitatively. Most of the approaches are based on Shannon entropy and Rényi's min-entropy [17, 8, 5, 15, 14, 27, 21, 3]. As illustrated above, these approaches might be counter-intuitive in some situations and also prone to conflicts when comparing between programs, i.e., Shannon-entropy measures judge this program more dangerous than the other one, but min-entropy measures give a counter-result. Thus, as argued by Alvim et al. [2], it seems that there is no unique measure that is likely to be appropriate in all cases.

To avoid the conflicts between classical approaches, Zhu et al. [25] propose to view program execution as a probabilistic state transition from one state to another, where states denote probability distributions of the private data. This approach is close to our model in spirit, but their approach only aims to compare between programs. Their approach constructs probability distribution functions over the residual uncertainty about private data and then the comparison between programs is done by the means and the variances of the distributions.

The quantitative security analysis proposed by Chen et al. [6] for multi-threaded programs defines the amount of leakage for each interleaving of the scheduler. The leakage of a program is then the expected value over all interleavings. The basis idea of this approach is that, for each interleaving of the scheduler, a set of possible traces is obtained. Based on the observables of the final states on these traces, the leakage of this interleaving is determined. This approach is imprecise since it does not consider leakages in intermediate states and the distribution of traces in a interleaving.

Chen et al. [7] also define the leakage of a program trace as the sum of the products of the leakage generated by each transition step and the probability of the transition. Examples in Section 4.3 show that this idea is not precise. This approach is only suitable to estimate the coarse maximum and minimum leakages of a program, i.e., the maximal and minimal values of leakages of traces. However, these values are not very helpful to judge programs, or to compare between different programs, because the gap between the maximum and minimum values is often large.

In another attempt to define the quantitative leakage for multi-threaded programs, Malacaria et al. [15] and Andrés et al. [3] propose to reuse the classical information-theoretic approach but the observables are traces of public variables, instead of only the final outcomes. These approaches have been discussed in Section 4.6. Mu et al. [18] also model the execution of a probabilistic program by a probabilistic state transition system where states represent distributions of private data. Their paper proposes an automatic analyzer for measuring leakage. However, the analysis is only for sequential programs.

6 Conclusions and Future Work

We propose a novel approach for estimating the leakage of a multi-threaded program. The notion of the leakage of a program trace is defined. The leakage of a program is then given as the expected value of the leakages of traces. Our approach takes into account the observables in intermediate states, and also the effect of the scheduler. The reasonableness of our approach should be further studied, in both theoretical properties and experimental case studies. However, we believe that our approach gives a more accurate way to study the quantitative security of multi-threaded programs, i.e., it agrees with the intuition about

what the leakage should be. Thus, we consider this work as an important contribution in the field of quantitative security analysis for multi-threaded programs.

In the next step, the implementation of the analysis is planned. Since programs are modeled as PKSs, we are also considering some methods to avoid the state space explosion problem.

As mentioned in Section 4, the expression of the expected final uncertainty is very similar to the notion of conditional min-entropy defined by Cachin [4]. Thus, we believe that our approach would coincide with the classical metric if we consider a channel where the observations are execution traces including scheduling decisions at steps along traces, and apply the Cachin's version of conditional min-entropy to the resulting channel. However, a formal proof needs to be done.

We also plan to study whether our approach also obtains good results for sequential programs. In this standard input-output model, if the initial distribution is non-uniform, the computed leakage might become negative. However, we believe that this negativeness property is only in the input-output setting. For multi-threaded programs, the value of leakage is always positive even when the priori distribution is non-uniform. Thus, it is also valuable to study this property more.

Acknowledgments. The authors would like to thank Catuscia Palamidessi and Kostas Chatzikokolakis for many fruitful discussions. Our work is supported by NWO as part of the SlaLoM project.

References

[1] M.S. Alvim, M.E. Andrés, K. Chatzikokolakis & C. Palamidessi (2011): *Quantitative information flow and applications to differential privacy*. In A. Aldini & R. Gorrieri, editors: *Foundations of security analysis and design VI*, Springer-Verlag, pp. 211–230, doi:10.1007/978-3-642-23082-0-8.

[2] M.S. Alvim, K. Chatzikokolakis, C. Palamidessi & G. Smith (2012): *Measuring Information Leakage Using Generalized Gain Functions*. In: *CSF '12: Proceedings of the 2012 IEEE 25th Computer Security Foundations Symposium*, IEEE Computer Society, pp. 265–279, doi:10.1109/CSF.2012.26.

[3] M.E. Andres, C. Palamidessi, P. Rossum & A. Sokolova (2010): *Information Hiding in Probabilistic Concurrent Systems*. In: *Proceedings of the 2010 Seventh International Conference on the Quantitative Evaluation of Systems*, QEST '10, IEEE Computer Society, pp. 17–26, doi:10.1109/QEST.2010.11.

[4] C. Cachin (1997): *Entropy Measures and Unconditional Security in Cryptography*. Ph.D. thesis.

[5] K. Chatzikokolakis, C. Palamidessi & P. Panangaden (2007): *Anonymity protocols as noisy channels*. In: *Proceedings of the 2nd international conference on Trustworthy global computing*, TGC'06, Springer-Verlag, pp. 281–300, doi:10.1007/978-3-540-75336-0-18.

[6] H. Chen & P. Malacaria (2007): *Quantitative analysis of leakage for multi-threaded programs*. In: *Proceedings of the 2007 workshop on Programming languages and analysis for security*, PLAS '07, ACM, pp. 31–40, doi:10.1145/1255329.1255335.

[7] H. Chen & P. Malacaria (2010): *The optimum leakage principle for analyzing multi-threaded programs*. In: *Proceedings of the 4th international conference on Information theoretic security*, ICITS'09, Springer-Verlag, pp. 177–193, doi:10.1007/978-3-642-14496-7-15.

[8] D. Clark, S. Hunt & P. Malacaria (2005): *Quantitative Information Flow, Relations and Polymorphic Types*. *J. Log. and Comput.* 15, pp. 181–199, doi:10.1093/logcom/exi009.

[9] J.A. Goguen & J. Meseguer (1982): *Security Policies and Security Models*. In: *IEEE Symposium on Security and Privacy*, pp. 11–20.

[10] A. Gurfinkel & M. Chechik (2006): *Why Waste a Perfectly Good Abstraction*. In: *In TACAS'06*, Springer, p. 3920.

[11] M. Huisman & T.M. Ngo (2012): *Scheduler-specific confidentiality for multi-threaded programs and its logic-based verification*. In: *FoVeOOS'11*, Springer-Verlag, pp. 178–195, doi:10.1007/978-3-642-31762-0-12.

[12] D. Kozen: *Semantics of probabilistic programs*. In: *Proceedings of the 20th Annual Symposium on Foundations of Computer Science*, SFCS '79, IEEE Computer Society, pp. 101–114, doi:10.1109/SFCS.1979.38.

[13] S.A. Kripke (1963): *Semantical Considerations on Modal Logic*. *Acta Philosophica Fennica* 16, pp. 83–94.

[14] P. Malacaria (2010): *Risk assessment of security threats for looping constructs*. *J. Comput. Secur.* 18, pp. 191–228.

[15] P. Malacaria & H. Chen (2008): *Lagrange multipliers and maximum information leakage in different observational models*. In: *Proceedings of the third ACM SIGPLAN workshop on Programming languages and analysis for security*, PLAS '08, ACM, pp. 135–146, doi:10.1145/1375696.1375713.

[16] J.L. Massey (1994): *Guessing and Entropy*. In: *Proceedings of the 1994 IEEE International Symposium on Information Theory*, p. 204, doi:10.1109/ISIT.1994.394764.

[17] I.S. Moskowitz, R.E. Newman, D.P. Crepeau & A.R. Miller (2003): *Covert Channels and Anonymizing Networks*. In: *In Workshop on Privacy in the Electronic Society (WPES 2003)*, ACM, pp. 79–88, doi:10.1145/1005140.1005153.

[18] C. Mu & D. Clark (2009): *Quantitative Analysis of Secure Information Flow via Probabilistic Semantics*. In: *Proceedings of the The Forth International Conference on Availability, Reliability and Security, ARES 2009, 2009, Japan*, IEEE Computer Society, pp. 49–57, doi:10.1109/ARES.2009.88.

[19] T.M. Ngo, M. Stoelinga & M. Huisman (2013): *Confidentiality for probabilistic multi-threaded programs and its verification*. In: *Proceedings of the 5th international conference on Engineering Secure Software and Systems*, ESSoS'13, Springer-Verlag, Berlin, Heidelberg, pp. 107–122.

[20] A. Sabelfeld & D. Sands (1999): *Probabilistic Noninterference for Multi-threaded Programs*. In: *In Proc. IEEE Computer Security Foundations Workshop*, IEEE Computer Society Press, pp. 200–214.

[21] G. Smith (2009): *On the Foundations of Quantitative Information Flow*. In: *FOSSACS '09*, Springer-Verlag, pp. 288–302, doi:10.1007/978-3-642-00596-1-21.

[22] G. Smith (2011): *Quantifying Information Flow Using Min-Entropy*. In: *QEST*, pp. 159–167. Available at `http://doi.ieeecomputersociety.org/10.1109/QEST.2011.31`.

[23] M.I.A. Stoelinga (2002): *Alea jacta est: verification of probabilistic, real-time and parametric systems*. Ph.D. thesis, University of Nijmegen, the Netherlands.

[24] S. Zdancewic & A.C. Myers (2003): *Observational Determinism for Concurrent Program Security*. In: *In Proc. 16th IEEE Computer Security Foundations Workshop*, pp. 29–43.

[25] J. Zhu & M. Srivatsa (2011): *Poster: on quantitative information flow metrics*. In: *Proceedings of the 18th ACM conference on Computer and communications security*, CCS '11, ACM, pp. 877–880, doi:10.1145/2046707.2093516.

[26] J. Zhu & M. Srivatsa (2011): *Quantifying Information Leakage in Finite Order Deterministic Programs*. In: *ICC*, pp. 1–6, doi:10.1109/icc.2011.5963509.

[27] Y. Zhu & R. Bettati (2005): *Anonymity vs. Information Leakage in Anonymity Systems*. In: *In Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS 2005)*, pp. 514–524, doi:10.1109/ICDCS.2005.13.