# Quantitative Security Analysis for Programs with Low Input and Noisy Output

Tri Minh Ngo and Marieke Huisman

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

**Abstract.** Classical quantitative information flow analysis often considers a system as an information-theoretic channel, where private data are the only inputs and public data are the outputs. However, for systems where an attacker is able to influence the initial values of public data, these should also be considered as inputs of the channel. This paper adapts the classical view of information-theoretic channels in order to quantify information flow of programs that contain both private *and* public inputs.

Additionally, we show that our measure also can be used to reason about the case where a system operator on purpose adds noise to the output, instead of always producing the correct output. The noisy outcome is used to reduce the correlation between the output and the input, and thus to increase the remaining uncertainty. However, even though adding noise to the output enhances the security, it reduces the reliability of the program. We show how given a certain noisy output policy, the increase in security and the decrease in reliability can be quantified.

## 1 Introduction

Qualitative security properties, such as *noninterference* [12] and *observational determinism* [27,14], are essential for applications where private data need strict protection, such as Internet banking, e-commerce, and medical information systems, since they prohibit any information flow from a high security level to a low security level[1]. 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 typical example is a *password checker* ($PWC$) where an attacker (user) tries a string to guess the password [11,3,24]. Even when the attacker makes a wrong guess, secret information has been leaked, i.e., it reveals information about what the real password is *not*. Thus, despite the correct functioning, $PWC$ is rejected.

Therefore, an alternative approach for such applications is to relax the absolute properties by quantifying the information flow and determining how much

[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.

J. Jürjens, F. Piessens, and N. Bielova (Eds.): ESSoS 2014, LNCS 8364, pp. 77–94, 2014.

secret information has been leaked. This information can be used to decide whether we can tolerate *minor* leakages. A quantitative security theory can be seen as a generalization of an absolute one.

*Classical quantitative security analysis.* Classical quantitative theory sees a program as a channel in the information-theoretic sense, where the secret $S$ is the *only* input and the observable final outcomes $O$ are the output [3]. An attacker, by observing $O$, might be able to derive information about $S$. The quantitative analysis of information flow then concerns the amount of private data that an attacker might learn. The analysis is based on the notion of *entropy*. The entropy of a random private 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 typically defined as the difference between the secret's initial uncertainty, i.e., the uncertainty of the attacker about the 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.,

$$\text{Information leakage} = \text{Initial uncertainty - Remaining uncertainty.}$$

*Programs that contain low input.* This paper considers programs where an attacker is able to influence the initial values of low variables. This is a popular kind of programs with many real-world applications, e.g., login systems, the *PWC*, or banking system. For such programs, in addition to the secret, the initial low values are another input to the channel. Therefore, the traditional form of channel becomes *invalid* for such programs.

To apply the traditional channel where the only input is the secret to this situation, we consider the initial low values as *parameters* of the channel. In particular, we consider a collection of sets of initial low values, and for each set, we construct a channel corresponding to these low values. Each channel is seen as a *test*, i.e., the attacker sets up the low parameters to test the system. Since the attacker knows the program code, then he knows which test would help him to gain the most information. Therefore, the leakage of the program with low input is defined as the maximum leakage over all possible tests.

*A new measure for the remaining uncertainty.* The classical approaches of the one-try attack model often base the analysis on the Smith's definition of conditional min-entropy [24]. However, the literature also admits that there might be different measures for different situations [5]. This paper argues that in some cases, Cachin's version of conditional min-entropy [7] might be a more reasonable measure, i.e., it gives more intuitive-matching results than Smith's version. Thus, we propose to consider Cachin's version as a valid measure for the remaining uncertainty. We believe that this measure has not previously been used in the theory of quantitative information flow.

The literature argues that observable outcomes would reduce the initial uncertainty of the attacker on the secret; and thus, the value of leakage cannot be negative. However, we show that this non-negativeness property does not always

hold, for example in case the output of the program contains noise. The idea is that to enhance the security, the system operator might add noise to the output, i.e., instead of always producing the exact outcomes, the program might sometimes report a noisy one. The noisy-output policy makes the outcomes of program more random, and thus, it reduces the correlation between the output and the input. As a consequence, the noisy-output policy increases the remaining uncertainty, and the value of leakage might become negative. This property might open the door for a new understanding of how the measure of uncertainty should be.

*To design a noisy-output policy.* Adding noise enhances the security, but reduces the program's reliability, i.e., the probability that a program produces the correct outcomes. The totally random output might achieve the best confidentiality, but these outcomes are practically useless. Thus, it is clear that a noisy-output policy should consider the balance between confidentiality and reliability.

This paper discusses how to construct an efficient noisy-output policy such that the attacker cannot derive secret information from the public outcomes, while a certain level of reliability is still preserved. Since the policy is kept in secret, i.e., we do not want the attacker to find out that the system has been modified, the policy needs to satisfy some properties of the system. In this way, the noisy-output policy would help to protect the system effectively, while it still preserves the program's function at the same time.

***Contributions.*** We propose a model of quantitative security analysis for programs that contain low input. This kind of programs has a vast application, i.e., any system with user interface. Examples of such systems include login systems, web-based applications, or online banking systems, to name a few. We also propose to consider Cachin's version of conditional min-entropy as a new measure for the notion of remaining uncertainty in the model of one-try attack. Besides, we also discuss an important property of the information flow, i.e., the quantity of information flow might be negative. This observation might change the classical view of how to define the quantity of leakage. Finally, we give an algorithm to generate noisy outcomes, while still preserving a certain level of the system's reliability. This idea can be implemented as a policy to enhance the security for applications.

**Organization of the Paper.** Section 2 presents the preliminaries. Then, Section 3 discusses the classical analysis, and presents our quantitative security analysis model for programs that contain low input. We also show the application of our measure. Section 4 discusses when negative information flow is expected, and how to construct and evaluate a noisy-output policy. Section 5 discusses related work, while Section 6 concludes, and discusses future work.

## 2 Preliminaries

### 2.1 Probabilistic Distribution

Let $X$ be a discrete random variable with the carrier $\mathbf{X} = \{x_1, \ldots, x_n\}$. A *probability distribution* $\pi$ over a set $\mathbf{X}$ is a function $\pi : \mathbf{X} \to [0, 1]$, such that the sum of the probabilities of all elements over $\mathbf{X}$ is 1, i.e., $\sum_{x_i \in \mathbf{X}} \pi(x_i) = 1$. If $\mathbf{X}$ is uncountable, then $\sum_{x_i \in \mathbf{X}} \pi(x_i) = 1$ implies that $\pi(x_i) > 0$ for countably many $x_i \in \mathbf{X}$. The probabilistic behavior of $X$ is then simply given by probabilities $p(X = x_i) = \pi(x_i)$.

When $X$ is clear from the context, we use the notation $\pi = \{p(x_1), p(x_2), \ldots, p(x_n)\}$ to denote the probabilities of elements in $\mathbf{X}$, i.e., $p(X = x_i)$.

### 2.2 Min-entropy

Let $X$ and $Y$ denote two discrete random variables. Let $p(X = x)$ denote the probability that $X = x$, and let $p(X = x|Y = y)$ denote the *conditional* probability that $X = x$ when $Y = y$.

**Definition 1.** *The Rényi's min-entropy of a random variable $X$ is defined as [24]:* $\mathcal{H}_{Rényi}(X) = -\log \max_{x \in \mathbf{X}} \; p(X = x)$.

Rényi did not define the notion of conditional min-entropy, and there are different definitions of this notion.

**Definition 2 (Smith's version of conditional min-entropy [24]).** *The conditional min-entropy of a random variable $X$ given $Y$ is,*

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

**Definition 3 (Cachin's version of conditional min-entropy [7]).** *The conditional min-entropy of a random variable $X$ given $Y$ is,*

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

### 2.3 Information-Theoretic Channel

The quantitative security analysis in the information-theoretic sense models the system as a channel with the secret as the input and the observables as the output. Formally, an information-theoretic channel is a triple $(\mathbf{X}, \mathbf{Y}, M)$, where $\mathbf{X}$ represents a finite set of secret inputs, $\mathbf{Y}$ represents a finite set of observable outputs, and $M$ is a $|\mathbf{X}| \times |\mathbf{Y}|$ channel matrix which contains the conditional probabilities $p(y|x)$ for each $x \in \mathbf{X}$ and $y \in \mathbf{Y}$. Thus, each entry of $M$ is a real number between 0 and 1, and each row sums to 1.

### 2.4 Basic Settings for the Analysis

To argue why a program is considered more dangerous than another, we need to set up some basic settings for the discussion. First, we assume that programs always terminate, and the attacker knows its source code. To aim for simplicity and clarity, rather than full generality, following [24], we restrict to programs with just a single high security input $S$ and a single low security input $L$. Since the high security output is irrelevant, programs only give a low security outcome $O$. Our goal is to quantify how much information about $S$ is deduced by the attacker who can influence $L$, and observe the execution traces of $O$, i.e., a sequence of values of $O$ obtained from the program's execution. We also assume that the sets of possible values of data are finite, as in the traditional approaches.

Secondly, we assume that there is a *priori, publicly-known* probability distribution on the high values. We also assume that data at the same security level are indistinguishable in the *security meaning*. Thus, a system that leaks the last 9 bits of private data is considered to be just as dangerous as a system that leaks the first 9 bits. Finally, we consider the *one-try guessing model*, i.e., observing the public outcomes, the attacker is allowed to guess the value of $S$ by only one try. This model of attack is suitable to many security situations where systems trigger an alarm if an attacker makes a wrong guess. For the password checker, this one-try guessing model can be understood as that an attacker is only allowed to try once. If the entered string is not the correct password, the system will block the account.

Notice that these restrictions aim to demonstrate our core idea. However, the analysis might be adapted to more complex situations easily after some trivial modifications.

## 3 Quantitative Security Analysis for Programs with Low Input

Before introducing our model of analysis for programs that contain low input, we present the classical models, and discuss briefly their shortcomings (see [22] for a detailed discussion).

### 3.1 Classical Models of Quantitative Security Analysis

Classical works [21,9,8,20,19,28,24,6] use information theory to analyze information flow quantitatively. A program is seen as a standard channel with $S$ as the input and $O$ as the output. Let $\mathcal{H}(S)$ denote the uncertainty of the attacker on the secret before executing the program, and $\mathcal{H}(S|O)$ the uncertainty after the program has been executed and public outcomes are observed. The leakage of the program is defined as $\mathcal{L}(P) = \mathcal{H}(S) - \mathcal{H}(S|O)$, where $\mathcal{L}(P)$ denotes the leakage of $P$; $\mathcal{H}$ might be either Shannon entropy or min-entropy with Smith's version of conditional min-entropy.

*Classical Measures might be Counter-Intuitive.* Many researchers [21,9,8,20,19,28] quantify information flow with Shannon entropy [23]. However, Smith [24] shows that in context of the one-try threat model, the Shannon-entropy measure does not always result in a very good operational security guarantee. In particular, Smith [24] shows that this measure might be counter-intuitive, i.e., an intuitively more secure program leaks more information according to the measure.

For this reason, Smith develops a new measure based on min-entropy [24]. He defines uncertainty in terms of *vulnerability* of the secret to be guessed correctly in one try. The vulnerability of a random variable $X$ is the maximum of the probabilities of the values of $X$. This approach seems to match the intuitive idea of the one-try threat model, i.e., the attacker always chooses the value with the maximum probability. However, in [22], we show that the measure with Smith's version of conditional min-entropy still results in counter-intuitive values of leakage. Therefore, we agree with Alvim et al. [5]: no single leakage measure is likely to suit all cases.

*Leakage in Intermediate States.* Classical analysis often considers only leakages in the final states of the execution. However, for programs that contain parallel operators, the leakages in intermediate states should also be taken into account [27,14,25]. Consider the following example,

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

For notational convenience, let $C_1$ and $C_2$ denote the left and right operands of the parallel composition operator $\parallel$. Executing this program, we obtain the following traces $T_{|O}$ of $O$, depending on which thread is picked first, i.e., $T_{|O} = [0, 1, 1]$ if executing $C_1$ first, or $T_{|O} = [0, 1, S, 1]$ if executing $C_2$ first.

This program does not leak information in the final states, since the final values of $O$ are independent of the initial value of $S$. However, when $C_2$ is executed first, the attacker is able to access $S$ via an intermediate state.

Thus, to obtain a suitable model of quantitative security analysis, we need to consider the leakage given by a sequence of publicly observable data obtained during the execution of the program.

### 3.2 Leakage of Programs with Low Input

The only input of the information-theoretic channel is the secret. For programs where an attacker might influence the initial value of the low variable, the initial low value is also an input of the channel modeling the program. To use the traditional channel, we model such a program by a set of channels. Each channel corresponds to the case where the low input is assigned a specific value. Thus, in our approach, the initial low value is considered as a parameter. Since we assume that the low value set is finite, the set of channels is also finite.

We see a channel as a *test*. We run the analysis on the set of tests. Since the attacker knows the program code, and is also able to influence the initial

low value, he knows which test would give him more secret information. Thus, the leakage of the program that contains low input is defined as the maximum leakages over all tests.

Given a program $P$ that contains a low input $L$. Let $\pi$ denote the priori distribution on the possible values of the private data, and *LVal* denote the value set of $L$. Let $T_{|_O}$ denote a trace of $O$ obtained from the execution of $P$, i.e., the sequence of $O$ that occurs during the execution. To define the leakage of $P$, we carry out the following steps.

**Leakage of Programs with Low Input**

**1**: Set up a test $(P, \pi, L)$:
- **1.1**: Choose a value for $L$.
- **1.2**: Construct a channel where $S$ is the input, $L$ is the parameter of the channel, and the traces $T_{|_O}$ are the output.

**2**: Compute the leakage of the test $(P, \pi, L)$:

$$\mathcal{L}(P, \pi, L) = \mathcal{H}_{Rényi}(S) - \mathcal{H}_{Cachin}(S|L, T_{|_O}),$$

where $\mathcal{H}_{Rényi}(S)$ is the min-entropy of $S$ corresponding to $\pi$.

**3**: Define the leakage of $P$ as: $\mathcal{L}(P, \pi) = \max_{L \in LVal} \ \mathcal{L}(P, \pi, L)$.

Notice that Step **1** and **2** repeat for all values of $L$.

*Measures of Uncertainty.* Since we follow the one-try attack model, the initial uncertainty is computed as Rényi's min-entropy of $S$ with distribution $\pi$. In this work, we propose to use Cachin's conditional min-entropy as a measure for the remaining uncertainty. Notice that in the remainder of this paper, to denote our measure, we use the notation $\mathcal{L}_{Cachin}$, instead of $\mathcal{L}$, to distinguish between our measure and Smith's measure, i.e., $\mathcal{L}_{Smith}$.

### 3.3 Case Studies

Below, we analyze some case studies, and compare Smith's measure with our measure. We show that our measure agrees more with the intuition.

***Password Checker.*** Consider the following *PWC*. Let $S$ denote the password, $L$ the string entered by the attacker (low input), and $O$ the public answer,

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

Assume that $S$ might be $A_1$, $A_2$, or $A_3$, with $\pi = \{p(A_1) = 0.98, p(A_2) = 0.01, p(A_3) = 0.01\}$. Since the attacker tests $L$ based on the value of $S$, there are 3 corresponding tests, i.e., $L = A_1$, $L = A_2$, or $L = A_3$. The leakages of the tests $L = A_2$ and $L = A_3$ are the same. Hence, we only analyze $L = A_1$ and $L = A_2$.

Before interacting with the *PWC*, the attacker believes that the password is $A_1$, since $p(A_1)$ dominates the other cases. Thus, in both tests, the attacker's initial uncertainty about $S$ is $\mathcal{H}_{Rényi}(S) = -\log 0.98 = 0.02915$.

When $L = A_1$, the $PWC$ is modeled by the following channel $M$,

| $M$ | $O = 1$ | $O = 0$ |
|---|---|---|
| $S = A_1$ | 1 | 0 |
| $S = A_2$ | 0 | 1 |
| $S = A_3$ | 0 | 1 |

The channel $M$ and the distribution $\pi$ and determine the joint probability matrix $J$, where $J[s, o] = \pi(s) \cdot M[s, o]$.

| $J$ | $O = 1$ | $O = 0$ |
|---|---|---|
| $S = A_1$ | 0.98 | 0 |
| $S = A_2$ | 0 | 0.01 |
| $S = A_3$ | 0 | 0.01 |

The joint probability matrix $J$ determines a marginal distribution of $O$, i.e., $p(o) = \sum_{\forall s} J[s, o]$. Thus, $p(O = 1) = 0.98$ and $p(O = 0) = 0.02$. Since $p(S = s|O = o) = \frac{J[s,o]}{p(o)}$, then $p(S = A_1|O = 1) = 1$, $p(S = A_2|O = 1) = p(S = A_3|O = 1) = 0$, and $p(S = A_1|O = 0) = 0$, $p(S = A_2|O = 0) = p(S = A_3|O = 0) = 0.5$. Thus, $\mathcal{L}_{Smith}(P, \pi, A_1) = 0.01465$, while $\mathcal{L}_{Cachin}(P, \pi, A_1) = 0.00915$.

When $L = A_2$, we obtain the following channel,

| $M$ | $O = 1$ | $O = 0$ |
|---|---|---|
| $S = A_1$ | 0 | 1 |
| $S = A_2$ | 1 | 0 |
| $S = A_3$ | 0 | 1 |

Thus, $p(O = 1) = 0.01$ and $p(O = 0) = 0.99$, and $p(S = A_1|O = 1) = p(S = A_3|O = 1) = 0$, $p(S = A_2|O = 1) = 1$, and $p(S = A_1|O = 0) = 0.9899$, $p(S = A_2|O = 0) = 0$, $p(S = A_3|O = 0) = 0.0101$. Therefore, $\mathcal{L}_{Cachin}(P, \pi, A_2) = 0.01465$, while $\mathcal{L}_{Smith}(P, \pi, A_2) = 0.01465$. The measure proposed by Smith judges that the leakages of the two tests where $L = A_1$ and $L = A_2$ are the same. However, this contradicts the intuition. In the test $L = A_1$, if the $PWC$ answers yes, it only helps the attacker to confirm something that he already believed to be certainly true. However, if the answer is $O = 0$, it does not help the attacker at all, i.e., he still does not know whether either $A_2$ or $A_3$ is more likely to be the password, since the posteriori probability $p(S = A_2|O = 0)$ is still equal to $p(S = A_3|O = 0)$.

Intuitively, the test $L = A_2$ helps the attacker gain more secret information. If $O = 1$, it completely changes the attacker's priori belief, i.e., the password is not $A_1$, and it also confirms a very rare case, i.e., the password is $A_2$. If $O = 0$, this even strengthens what the attacker's belief about the secret, since the posteriori probability $p(S = A_1|O = 0) = 0.9899$ increases. The analysis should indicate that the test $L = A_2$ leaks more information than the test $L = A_1$.

Thus, in this example, our measure gives results that match more the intuition. The leakage of this $PWC$ is defined as the leakage of the test $L = A_2$. This example also shows that the test in which the attacker sets the low input based on the value that he believes to be the private data is not always the "*best test*". Since the attacker knows the source code of the program and the priori distribution of the private data, he knows which test would give him the most information. This is the reason that we define the leakage of a program with low input as the maximum leakage over all tests.

In the general case, given $\pi = \{p(A_1) = a, p(A_2) = b, p(A_3) = c\}$, whenever $a > c$ and $b > c$, Smith's measure cannot distinguish between the test $L = A_1$ and $L = A_2$, while our measure can and also agrees more with the intuition about what the leakage should be.

***A Multi-threaded Program.*** Consider the following example,

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

where $S$ is a 3-bit unsigned integer with the priori uniform distribution. The execution of this program results in the following traces of $O$, depending on whether $C_1$ or $C_2$ is picked first:

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

Consider a uniform scheduler, i.e., a scheduler that picks threads with equal probability. It is clear that the last command $O := S \mod 4$ always reveals the last 2 bits of $S$. The first bit might be leaked with probability $\frac{1}{2}$, depending on whether the scheduler picks thread $C_2$ first or not. Thus, with the uniform scheduler, intuitively, the real leakage of this program is 2.5 bits.

By observing the traces of $O$, the attacker is able to derive secret information. For example, if the 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 chance that $S$ is 5 is $\frac{2}{3}$. Therefore, $\mathcal{L}_{Cachin}(P, \pi) = 3 - (-(\frac{6}{16} \cdot \log 1 + \frac{4}{16} \cdot \log \frac{1}{2} + \frac{6}{16} \cdot \log \frac{2}{3})) = 2.53$, while $\mathcal{L}_{Smith}(P, \pi) = 3 - (-\log(\frac{6}{16} \cdot 1 + \frac{4}{16} \cdot \frac{1}{2} + \frac{6}{16} \cdot \frac{2}{3})) = 2.58$.

Consider a scheduler that picks thread $C_2$ first with probability $\frac{3}{4}$. With this scheduler, the real leakage of this program is 2.75. Our measure gives $\mathcal{L}_{Cachin}(P, \pi) = 2.774$, while $\mathcal{L}_{Smith}(P, \pi) = 2.807$. If the scheduler picks thread $C_2$ first with probability $\frac{1}{4}$, $\mathcal{L}_{Cachin}(P, \pi) = 2.271$, while $\mathcal{L}_{Smith}(P, \pi) = 2.321$. Of course in this case, the real leakage is 2.25. These results show that our measures are closer to the real leakage values.

These case studies show that our measure is more precise than the classical measure given by Smith's conditional min-entropy. The main difference between the two measures is the position of log in the expression of the remaining entropy. The idea of using logarithm is to express the notion of uncertainty in bits. Thus, the log should apply only to the probability of the guess, which represents the uncertainty of the attacker, as in our approach. Our measure distinguishes between the probabilities of the observable and the probabilities of the guess based on the observable. In Smith's measure, the logarithm applies to the

combination of the two probabilities, and does not distinguish between them, which might cause imprecise results.

However, as a side remark, we emphasize that no unique measure is likely to be suitable for all cases. We believe that for some examples, measures based on Shannon entropy or Smith's version of conditional min-entropy might match better the real values of leakage.

## 4 Adding Noise to the Output

### 4.1 Negative Information Flow

In relation to defining an appropriate measure for information flow quantification, this paper also discusses a claim of the existing theory of quantitative information flow, i.e., a quantitative measure of information leakage should return a non-negative value. The common idea of the classical analysis is that the observation of the program's public outcomes would enhance the attacker's knowledge about the private data, and consequently reduce the attacker's initial uncertainty.

However, we think that this non-negativeness property does not always hold. For some applications, to enhance the confidentiality, the system operator adds noise secretly to the output, i.e., via some output perturbation mechanism based on randomization. The noisy outcomes might mislead the attacker's belief about the secret, i.e., they increase the final uncertainty. As a consequence, the value of leakage might be negative. This idea is illustrated more as follows.

*Password checker with noisy outcomes.* Consider the $PWC$ in Section 3.3. We assume that the system operator *secretly* has changed its behavior, i.e., the real $PWC$ is a probabilistic $PWC$ where the system operator introduced some perturbation mechanism to the output (We assume that the attacker does not know about the security policy applied to the system.),

$$\texttt{if}\ (S = L)\ \texttt{then}\ \{O := 1\ \ {}_{0.9}[\!]\ O := 0\}\ \texttt{else}\ \{O := 0\ \ {}_{0.9}[\!]\ O := 1\};$$

In this version, the exact answers are reported with probability 0.9, i.e., when $S = L$, $O = 1$ is reported with probability 0.9, and $O = 0$ with probability 0.1. Consider the test $L = A_2$, the real channel $M'$ is as follows,

| $M'$ | $O = 1$ | $O = 0$ |
|---|---|---|
| $S = A_1$ | 0.1 | 0.9 |
| $S = A_2$ | 0.9 | 0.1 |
| $S = A_3$ | 0.1 | 0.9 |

Notice that the attacker still thinks that the system is $M$, but in fact, the real system is $M'$. Based on $\pi$ and $M'$, the computation gives the real distribution $p(O = 1) = 0.108$ and $p(O = 0) = 0.892$, and the real posteriori probabilities $p(S = A_2|O = 1) = 0.083$ and $p(S = A_1|O = 0) = 0.9887$.

Before observing the outcome, the guess, i.e., the secret is $A_1$, has 98% chance of being correct. If the outcome is $O = 0$, the real posteriori probability gives the attacker's guess, i.e., $S = A_1$, a 98.87% chance of being correct. This is almost the same as the guess without the outcome. When $O = 1$, the attackers guesses $S = A_2$, since his database tells that this guess has the highest chance to be correct. However, the real posteriori probability ensures that his guess only has a 8.3% chance of being correct. Therefore, the outcomes of the program not only reveal *no* secret information, but also cause him to decide wrongly. Therefore, intuitively, this is a negative information flow.

As we expected, our measure indicates a negative leakage: $\mathcal{L}_{Cachin}(P, \pi, A_2) = -\log 0.98 + (0.108 \log 0.083 + 0.892 \log 0.9887) = -0.37$, while $\mathcal{L}_{Smith}(P, \pi, A_2) = -0.137$. Notice that the value of the leakage is determined by the real probability of success, not by the probability in the attacker's database.

We believe that this observation of negative information flow has not been reported in the literature. We think that this property would change the classical view of how the measure of uncertainty should be, i.e., we do not need to avoid measures that do not guarantee the non-negativeness property.

### 4.2 Noisy-Output Policy

The noisy outcomes change the behavior of the system, i.e., they change the channel $M$ that models the system (the public channel that the attacker also knows) to $M'$ (the real channel in secret). The noisy outcomes should be added in such a way that they change the original channel, but still preserve a certain level of reliability, e.g., the above probabilistic $PWC$ works properly in 90% of the time. Totally random outcomes might achieve the best confidentiality, but these outcomes are practically useless. Besides, the noisy-output policy also needs to satisfy some *general* requirements that, on one hand, help to mislead the attacker, i.e., the attacker does not know that the system has been changed by the policy; thus, he still uses the posteriori distributions based on $M$ and $\pi$ to make a guess, and on the other hand, reduce the leakage. This section discusses how to design such an efficient noisy-output policy.

**Design a Policy.** Given a system $P$ that is described by a channel matrix $M$ of size $n \times m$, e.g., the set of secret input values is $\{A_1, \cdots, A_n\}$, and the set of observable outcomes is $\{Z_1, \cdots, Z_m\}$.

*General Requirements.* Since the attacker knows $\pi$ and $M$, he is able to compute the marginal distribution of the output. Thus, firstly, the *distribution of the output* has to be preserved by the channel $M'$, where the noise has been added. If the policy does not preserve this distribution, the attacker might find out that the channel $M$ has been changed, and then he will try to study the system before making a guess, i.e., trying to get the *real* program code of the system.

Secondly, for each outcome $Z_i$, assume that $p(S = A_j|Z_i)$ is the maximum posteriori probability, then $p'(S = A_j|Z_i)$ is also the maximum posteriori probability, i.e., the *maximum property* of the posteriori distributions has to be preserved.

For example, if $M$ gives a posteriori distribution where $p(S = A_j|Z_i) = 0.8$, then the real posteriori probability given by $M'$ might be $p(S = A_j|Z_i) = 0.6$. Thus, if the outcome is $Z_i$, the attacker thinks that the guess $S = A_j$ has a 80% chance to be correct. However, in reality, this guess only has a 60% chance of success. Notice that $p(S = A_j|Z_i)$ does not need to be equal to $p'(S = A_j|Z_i)$. The preservation of the maximum property of the posteriori distribution is necessary. Consider a uniform posteriori distribution $\{p(S = A_1|Z_i) = p(S = A_2|Z_i) = p(S = A_3|Z_i) = \frac{1}{3}\}$ in the attacker's database. Following the requirement, the posteriori distribution given by $M'$ has to be also uniform. If we do not require this, then the real distribution might possibly be $\{p'(S = A_1|Z_i) = 0.2, p'(S = A_2|Z_i) = 0.7, p'(S = A_3|Z_i) = 0.1\}$. According to his database, the attacker might guess $S = A_2$, since all three guesses have the same chance of being correct. In this case, the real probability would increase the chance of success, and thus, increase the leakage.

*Reliability.* Reliability of a system is the probability that a system will perform its intended function during a specified period of observation time. Let $\mathcal{R}_i$ ($\mathcal{R}_i \leq 1$) denote the reliability corresponding to the secret value $A_i$, i.e., the probability that the system will produce correct outcomes when the secret is $A_i$. Thus, the overall reliability of the system $P$ is $\mathcal{R}_P = \sum_i p(A_i) \cdot \mathcal{R}_i$. The noisy-output policy produces noise, and thus it reduces the reliability of the system. Therefore, we require that a noisy-output should guarantee at least a certain level of reliability.

*Noisy-output policy.* We propose a simple policy that might reduce the unwanted information flow, while still preserving a certain level of reliability. The following policy only aims to demonstrate the core idea of what a noisy-output policy should be. The practical policy might be customized due to the requirement of the application. Given a channel $M$ that models a system $P$. A noisy-output policy changes $M$ to $M'$ by choosing an appropriate set of $\{\mathcal{R}_1, \cdots, \mathcal{R}_n\}$.

Noisy-Output Policy

**1**: For each row $i$ of $M$, multiply each entry of the row by the reliability *variable* $\mathcal{R}_i$. Choose randomly one of the smallest entries, and add the value $1 - \mathcal{R}_i$ to it. Denote this modified matrix by $M'$.
**2**: Choose an overall reliability value that the policy has to guarantee, e.g., $\mathcal{R}_{min}$. Establish an inequality: $\sum_i p(A_i) \cdot \mathcal{R}_i \geq \mathcal{R}_{min}$.
**3**: For any outcome $Z_i$, let $p(O = Z_i)$ denote the probability determined by $\pi$ and $M$, and $p'(O = Z_i)$ determined by $\pi$ and $M'$, establish an equation: $p(O = Z_i) = p'(O = Z_i)$.
**4**: For each outcome $Z_i$, if $\forall k.\, p(S = A_j|Z_i) \geq p(S = A_k|Z_i)$, then establish the following condition: $\forall k.\, p'(S = A_j|Z_i) \geq p'(S = A_k|Z_i)$.
**5**: Solve these equations and inequalities. The set $\{\mathcal{R}_1, \cdots, \mathcal{R}_n\}$, are chosen in such a way that the leakage given by $M'$ is close to zero, and the reliability of the system $\mathcal{R}_P$ is as high as possible.

Notice that in Step 1, the sum of all entries of a row has to be 1; thus we have to add the value $1-\mathcal{R}_i$ to one of its entries. Step 3 establishes a set of equations, i.e., $m-1$ *independent* equations that correspond to $m-1$ observable outcomes, that preserve the output distribution. We also obtain $(n-1)\cdot m$ inequalities in Step 4, that preserve the maximum property of the posteriori distributions.

There always exists a trivial solution $\mathcal{R}_1 = \cdots = \mathcal{R}_n = 1$, i.e., $M$ and $M'$ are identical. When there are multiple solutions, we choose one that gives a low leakage, but a high overall reliability. However, this does not always happen. A solution that guarantees a very low leakage might also give a low reliability. In fact, a negative leakage, i.e., when the attacker decides wrongly based on the observable outcomes, is not always necessary. The goal of the policy is to ensure that the attacker cannot gain knowledge from the observable outcomes. Thus, $\mathcal{R}_1, \cdots, \mathcal{R}_n$ are chosen such that the leakage is close to zero and the overall reliability gets a high value. Next, we show an important property of our policy.

**Theorem 1.** *Given a priori distribution $\pi$ and a channel matrix $M$, the channel matrix $M'$ modified from $M$ by the noisy-output policy always gives a leakage quantity that is not greater than the one given by $M$.*

*Proof.* For any outcome $Z_i$, assume that the maximum likely secret is $A_j$. Since $p'(Z_i) = p(Z_i)$ and $p'(S = A_j|Z_i) = \mathcal{R}_j \cdot p(S = A_j|Z_i)$, thus $-p'(Z_i)\log p'(S = A_j|Z_i) \geq -p(Z_i)\log p(S = A_j|Z_i)$. Therefore, the value of remaining uncertainty given by $\pi$ and $M'$ is greater than or equal to the one given by $\pi$ and $M$. As a consequence, the corresponding leakage quantity is reduced.

*Example.* Consider a deterministic program $P$ where the secret might be $A_1$, $A_2$, or $A_3$ with a uniform $\pi = \{p(A_1) = p(A_2) = p(A_3) = \frac{1}{3}\}$. The system $P$ might produce three low outcomes $Z_1$, $Z_2$, and $Z_3$ as described by $M$,

| $M$ | $Z_1$ | $Z_2$ | $Z_3$ |
|---|---|---|---|
| $S = A_1$ | 1 | 0 | 0 |
| $S = A_2$ | 0 | 1 | 0 |
| $S = A_3$ | 0 | 0 | 1 |

Since the attacker knows the program code, he is able to construct $M$ in his database. Since the public outcomes are totally dependent on the secret, the attacker can derive the private data entirely from the outcomes, e.g., if the outcome is $Z_i$, the attacker knows for sure that $S = A_i$.

To protect the secret, the system operator might mislead the attacker by adding noise to the output, i.e., the real system is $M'$,

| $M'$ | $Z_1$ | $Z_2$ | $Z_3$ |
|---|---|---|---|
| $S = A_1$ | $\mathcal{R}_1$ | $1-\mathcal{R}_1$ | 0 |
| $S = A_2$ | 0 | $\mathcal{R}_2$ | $1-\mathcal{R}_2$ |
| $S = A_3$ | $1-\mathcal{R}_3$ | 0 | $\mathcal{R}_3$ |

Based on $\pi$ and $M$, the attacker knows that $p(O = Z_1) = p(O = Z_2) = p(O = Z_3) = \frac{1}{3}$. To satisfy this output distribution, $\mathcal{R}_1 = \mathcal{R}_2 = \mathcal{R}_3$. Besides, the maximum property of the posteriori distributions determines that

$\frac{1}{2} \leq \mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_2 \leq 1$. Thus, the reliability of the system is $\mathcal{R}_P = \mathcal{R}_1$, and for this example, $\mathcal{L}_{Cachin}(P, \pi) = \mathcal{L}_{Smith}(P, \pi) = \log 3\mathcal{R}_1$.

Thus, a high value of $\mathcal{R}_1$, which guarantees a high overall reliability, also gives a high leakage. If the goal is to reduce the leakage, we might choose $\mathcal{R}_1 = \mathcal{R}_2 = \mathcal{R}_3 = \frac{1}{2}$, which gives the smallest value of leakage, i.e., $\log \frac{3}{2}$, but also a very low reliability. If a high reliability is required, $\mathcal{R}_1 = \mathcal{R}_2 = \mathcal{R}_3 = \frac{2}{3}$ might be a good choice.

Consider the *PWC* example, for the test $L = A_2$, following the policy, we can choose $\mathcal{R}_1 = 0.995, \mathcal{R}_2 = 0.5, \mathcal{R}_3 = 0.99$ to have $\mathcal{L}_{Cachin}(P, \pi, A_2) = -0.00275$ with the reliability $\mathcal{R} = 0.99$. However, if we consider both tests, i.e., $L = A_1$ and $L = A_2$, $\mathcal{R}_1 = \mathcal{R}_2 = \mathcal{R}_3 = 1$.

As mentioned above, a noisy-output policy enhances the security, but reduces the reliability of a system, i.e., the system does not always work in a proper way. However, the drawback of the reduced reliability can be overcome. Consider a situation of the *PWC* in which an user or an attacker provides a *correct* password, but the system rejects it, and then blocks his account (the one-try model). If this context is for the attacker, it would be very nice, since the attacker does not have a chance to use the account again. If this context is for the real user; however, the situation would be different: the user is still allowed to reactivate the account by contacting the company/website administrators and proving that he is the real owner of the account, while the attacker cannot do the same.

The other way around, i.e., when the system accepts a wrong password, is not nice for the security. This is the reason that the policy should guarantee a high reliability. Notice that in this situation, the system accepts the login, but no private information is leaked, since the attacker still does not know the correct password. Thus, in the next login, there is a high chance that the system will reject this wrong password. Moreover, to avoid this situation, the system might also implement two-factor authentication, i.e., in addition to asking for something that only the user knows (e.g., user-name, password, PIN), the system also requires something that only the user has (e.g., ATM card, smart card). The ATM scenario illustrates the basic concept of most two-factor authentication systems, i.e., without the combination of both ATM card and PIN verification, authentication does not succeed.

Finally, it would be stressed that we only sketch the main idea of a noisy-output policy. However, for practical applications, depending on the real security requirements, the above policy might be customized, e.g., in Step **1**, instead of choosing randomly one of the smallest entries, and adding $1-\mathcal{R}_i$ to it, the policy can add to each of the smallest entries a value such that the sum of all these values is equal to $1 - \mathcal{R}_i$.

## 5 Related Work

Our proposal for programs with low input borrows ideas from Malacaria et al. [20] and Yasuoka et al. [26]. However, in these works, they [20,26] do not analyze the systems with low input sufficiently. They define the leakage of a program

as the leakage of a single test, while we define it as the maximum leakage of all tests. All examples in [20] are without low input, and their measure is based on Shannon entropy. Yasuoka et al. only consider the leakages in the final states.

Clarkson et al. [11] argue that the classical uncertainty-based analysis is not adequate to measure information flow of programs that contain low input. To define the leakage of the *PWC*, Clarkson et al. fix the value of the secret, i.e., the password, and the low input, i.e., by always assigning it the value that the attacker believes to be the password, then run the analysis under that specific circumstance. In case the value the attacker believes is not the real password, the uncertainty-based analysis might return a negative leakage value. The authors argue that this result flatly contradicts the intuition, i.e., from interacting with *PWC*, the attacker gains more knowledge by learning that the password is not the one that he has just entered. Based on this claim, they propose a different approach named accuracy-based information flow analysis. This trend of research has been expanded in [10,15,16,13]. However, the accuracy-based analysis often results in a quantity that is *inconsistent* with the size of the flow, i.e., the quantity of the secret information flow exceeds the size needed to store the secret [10].

We believe that there is a flaw in the way Clarkson et al. model the system. Clarkson et al. fix the value of the secret. Thus, this does not capture precisely the idea of information-theoretic channel. The information-theoretic channel has the secret as the input, and the entropy of the input quantifies the uncertainty involved in predicting the value of the secret. Thus, if the value of the secret is fixed, it implies that the priori distribution on the possible values of the secret is not valid anymore, i.e., the secret is now a certain value with the absolute probability 1. As a consequence, the entropy of the input does not reflect the true meaning of the initial uncertainty. Therefore, in these approaches, a wrong channel model has led to misleading results, i.e., a negative uncertainty-based result, or a size-inconsistent accuracy-based result.

Alvim et al. discuss limitations of the classical information-theoretic channel, i.e., showing that it is not a valid model for interactive systems where secrets and observables can alternate during the computation and influence each other [4]. In [3], Alvim et al. also discuss the example of the password checker. They fix the password by assigning it a specific value, and then consider the initial low values as the only input to the channel. As discussed before, this idea does not reflect the true idea of the information-theoretic channel.

Köpf et al. also consider systems with low input, i.e., cryptosystems where the attacker can control the set of input messages [17]. However, their proposal is only for deterministic systems, i.e., for each input, the system produces only one output, while in our proposal, the output might be nondeterministic and probabilistic. Besides, Köpf et al. consider a different threat model, i.e., the *multiple-try* guessing model, and they put a restriction on the priori distribution of the secret, requiring it to be uniform.

The idea of adding noise to the output comes from the *differential privacy* control, i.e., the problem of protecting the privacy of database's participants when performing *statistical* queries [3,1,2]. The differential privacy control also

uses some output perturbation mechanism to report a noisy answer among the correct ones for the queries. Thus, while the attacker is still able to learn properties of the population as a whole, he cannot learn the value of an individual. To construct an efficient noisy-output policy for a statistical database, it is necessary to consider the balance between *privacy*, i.e., how difficult to guess the value of an individual, and *utility*, i.e., the capacity to retrieve accurate answers from the reported ones. In [18], Köpf et al. also explore a similar idea to cope with timing attacks for cryptosystems, i.e., randomizing each cipher-text before decryption. As a consequence, the strength of the security guarantee is enhanced, while the efficiency of the cryptosystem is decreased, since the execution time of the cryptographic algorithm is increased.

In this work, we assume that the attacker cannot choose schedulers. The idea is to make our measure valid for both sequential and multi-threaded programs. Since sequential programs contain no parallel operator, the scheduler is not necessary for such programs. In [22], we propose a model of analysis for multi-threaded programs where the attacker is able to select an appropriate scheduler to control the set of program traces. In this current work, if the attacker can choose schedulers, i.e., if observations in our channel are not only traces, but also include scheduling decisions at each step, our measure and the proposed measure in [22] coincide. Notice that we did not consider the low input in [22].

## 6 Conclusions and Future Work

This paper discusses how to analyze quantitatively information flow of a program that contains low input. For such programs, we adapt the traditional information-theoretic channel by considering the initial low values as parameters of the channel. Besides, we also show that the value of information flow might be negative in case the system operator adds noise to the outcomes, i.e., the noise misleads the attacker's belief about the secret, and thus, it increases the final uncertainty. We believe that this property would change the way people often think about the measure of uncertainty. Since there is a growing appreciation that no unique measure is suitable for all cases, we suggest to measure the remaining uncertainty by Cachin's conditional min-entropy. This new measure matches the intuition in many cases. Finally, this paper discusses how to design an efficient noisy-output policy, which generates noisy outcomes, while still guarantees a high overall reliability.

**Future Work.** The classical approaches of the one-try attack model only base the analysis on the information of the value that the attacker believes to be the secret. Thus, the analysis ignores the extra leakage that might be derived from the values that the attacker disbelieves to be the secret. In the future work, we propose to include this extra information to the analysis. We also consider to define a measure for the *multiple-try* attack model.

Since there are many measures proposed for quantitative information flow analysis, and no unique measure is likely to suit all contexts, it might be interesting to evaluate each measure to determine under which circumstances, a certain measure might give the *best* answer.

**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. Alvim, M.S., Andrés, M.E., Chatzikokolakis, K., Palamidessi, C.: On the relation between differential privacy and quantitative information flow. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part II. LNCS, vol. 6756, pp. 60–76. Springer, Heidelberg (2011)
2. Alvim, M.S., Andrés, M.E., Chatzikokolakis, K., Degano, P., Palamidessi, C.: Differential privacy: on the trade-off between utility and information leakage. CoRR, abs/1103.5188 (2011)
3. Alvim, M.S., Andrés, M.E., Chatzikokolakis, K., Palamidessi, C.: Quantitative information flow and applications to differential privacy. In: Aldini, A., Gorrieri, R. (eds.) FOSAD VI 2011. LNCS, vol. 6858, pp. 211–230. Springer, Heidelberg (2011)
4. Alvim, M.S., Andrés, M.E., Palamidessi, C.: Information flow in interactive systems. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010. LNCS, vol. 6269, pp. 102–116. Springer, Heidelberg (2010)
5. Alvim, M.S., Chatzikokolakis, K., Palamidessi, C., Smith, G.: Measuring information leakage using generalized gain functions. In: Proceedings of the IEEE 25th Computer Security Foundations Symposium, CSF 2012, pp. 265–279. IEEE Computer Society (2012)
6. Andres, M.E., Palamidessi, C., Rossum, P., Sokolova, A.: Information hiding in probabilistic concurrent systems. In: Proceedings of the 2010 Seventh International Conference on the Quantitative Evaluation of Systems, QEST 2010, pp. 17–26. IEEE Computer Society (2010)
7. Cachin, C.: Entropy Measures and Unconditional Security in Cryptography. PhD thesis (1997)
8. Chatzikokolakis, K., Palamidessi, C., Panangaden, P.: Anonymity protocols as noisy channels. In: Montanari, U., Sannella, D., Bruni, R. (eds.) TGC 2006. LNCS, vol. 4661, pp. 281–300. Springer, Heidelberg (2007)
9. Clark, D., Hunt, S., Malacaria, P.: Quantitative information flow, relations and polymorphic types. J. Log. and Comput. 15, 181–199 (2005)
10. Clarkson, M.R., Myers, A.C., Schneider, F.B.: Quantifying information flow with beliefs. J. Comput. Secur. (2009)
11. Clarkson, M.R., Myers, A.C., Schneider, F.B.: Belief in information flow. In: In Proc. 18th IEEE Computer Security Foundations Workshop, pp. 31–45 (2005)
12. Goguen, J.A., Meseguer, J.: Security policies and security models. In: IEEE Symposium on Security and Privacy, pp. 11–20 (1982)
13. Hamadou, S., Sassone, V., Palamidessi, C.: Reconciling belief and vulnerability in information flow. In: Proceedings of the 2010 IEEE Symposium on Security and Privacy, SP 2010, pp. 79–92. IEEE Computer Society (2010)

14. Huisman, M., Ngo, T.M.: Scheduler-specific confidentiality for multi-threaded programs and its logic-based verification. In: Beckert, B., Damiani, F., Gurov, D. (eds.) FoVeOOS 2011. LNCS, vol. 7421, pp. 178–195. Springer, Heidelberg (2012)
15. Hussein, S.H.: A precise information flow measure from imprecise probabilities. In: Proceedings of the 2012 IEEE Sixth International Conference on Software Security and Reliability, SERE 2012, pp. 128–137. IEEE Computer Society (2012)
16. Hussein, S.H.: Refining a quantitative information flow metric. CoRR, abs/1206.0886 (2012)
17. Köpf, B., Basin, D.: An information-theoretic model for adaptive side-channel attacks. In: Proceedings of the 14th ACM Conference on Computer and Communications Security, CCS 2007, pp. 286–296. ACM (2007)
18. Köpf, B., Dürmuth, M.: A provably secure and efficient countermeasure against timing attacks. In: Proceedings of the 2009 22nd IEEE Computer Security Foundations Symposium, CSF 2009, pp. 324–335. IEEE Computer Society (2009)
19. Malacaria, P.: Risk assessment of security threats for looping constructs. J. Comput. Secur. 18, 191–228 (2010)
20. Malacaria, P., Chen, H.: 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 2008, pp. 135–146. ACM (2008)
21. Moskowitz, I.S., Newman, R.E., Crepeau, D.P., Miller, A.R.: Covert channels and anonymizing networks. In: Proceedings of the 2003 ACM Workshop on Privacy in the Electronic Society, WPES 2003, pp. 79–88. ACM (2003)
22. Ngo, T.M., Huisman, M.: Quantitative security analysis for multi-threaded programs. CoRR, abs/1306.2693 (2013)
23. Shannon, C.E., Weaver, W.: A Mathematical Theory of Communication. University of Illinois Press (1963)
24. Smith, G.: On the foundations of quantitative information flow. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 288–302. Springer, Heidelberg (2009)
25. Volpano, D., Smith, G.: Probabilistic noninterference in a concurrent language. J. Comput. Secur. 7, 231–253 (1999)
26. Yasuoka, H., Terauchi, T.: On bounding problems of quantitative information flow. In: Gritzalis, D., Preneel, B., Theoharidou, M. (eds.) ESORICS 2010. LNCS, vol. 6345, pp. 357–372. Springer, Heidelberg (2010)
27. Zdancewic, S., Myers, A.C.: Observational determinism for concurrent program security. In: Proceedings of 16th IEEE Computer Security Foundations Workshop, CSFW 2003, pp. 29–43. IEEE Computer Society (2000)
28. Zhu, Y., Bettati, R.: Anonymity vs. information leakage in anonymity systems. In: Proceedings of the 25th IEEE International Conference on Distributed Computing Systems, ICDCS 2005, pp. 514–524. IEEE Computer Society (2005)