Quantitative Information Flow for Security: A Survey

Technical Report: TR-08-06 (updated on 2010)

Chunyan Mu [1]

Department of Computer Science
King's College London
The Strand, London, UK

Abstract

Information-flow security enforces limits on the use of information that cover both access and propagate in programs. Quantifying and measuring information flow in software has recently become an active research topic in computer security community. There has been a significant activity around the question of how to measure the amount of information flow caused by interference between variables by using information theoretic quantities. This report presents a detailed survey of related work in this area. It covers several area that have been associated with security information flow, providing with an overall view on what has been done and which are available metrics that can help researcher to get a more comprehensive view of the direction in this area. Possible future works are also suggested.

Keywords: Survey, Security, Measurement, Quantity, Information Flow, Language.

1 Introduction

Quantifying and measuring information flow in software has recently become an active research topic. Access control systems are designed to restrict access to information, but cannot control internal information propagation once accessed. The goal of information flow security is to ensure that the information propagates throughout the execution environment without security violations such that only controlled secure information is leaked to public outputs. The traditional theory based reasoning and analysis of software systems has largely relied on logics and abstracted away from quantitative aspects. It would therefore be useful to have a mathematical quantitative study geared towards the tasks relevant to the computational environment in which we live.

Traditionally, the approach of information flow security is based on *interference* [30]. Informally, *interference* between program variables can be considered as

[1] Email: chunyan.mu@kcl.ac.uk

the capacity of variables to affect the values of other variables. Non-interference, *i.e.*, absence of interference, is often used in proving that a system is well behaved, whereas interference can lead to mis-behaviours. However, mis-behaviours in the presence of interference will generally happen only when there is *enough* interference. A concrete example is a software system with *access control*. To enter such a system the user has to pass an identification stage; whether subsequently authorized or failed, some information has been leaked, therefore these systems present interference. Otherwise, if the interference in such systems is *small* enough we can be confident in the security of the system.

The security community hence requires to determin *how much* information flows from *high* level to *low* level, which is known as *quantitative information flow* (QIF). It is concerned with measuring the *amount* of information flow caused by interference between variables by using information theoretic quantities. The precursor for this work was that of Dorothy Denning in the early 1980's. Denning [23] presented that the data manipulated by a program can be typed with security levels, which naturally assume the structure of a partially ordered set. Moreover, this partially ordered set is a lattice under certain conditions [22]. Given a simple program, the lattice only contains two elements, typed with *high* and *low*. The principle of non-interference dictates that low-security variables of the program cannot be affected by any high-security data [30]. Assuming that *high* means *secret* and *low* means *public*, then verifying that no information ever flows from higher to lower security levels is equivalent to verifying *confidentiality*. Later related work is that of McLean and Gray and McIver and Morgan in 1990's. McLean presented a very general Flow Model in [44]. McLean also gave a probabilistic definition of security with respect to flows of a system based on this flow model. Gray presented a less general and more detailed elaboration of McLean's flow model in [31], which made an explicit connection with information theory through his definition of the channel capacity of flows between confidential and non-confidential variables. Recently McIver and Morgan [42,43] devised a new information theoretic definition of information flow and channel capacity. They added demonic non-determinism as well as probabilistic choice to *while* program thus deriving a non-probabilistic characterization of the security property for a simple imperative language. There are some other attempts in the 2000s: Di Pierro, Hankin and Wiklicky [24] gave a definition of probabilistic measures on flows in a probabilistic concurrent constraint system where the interference came via probabilistic operators. Clarkson et al. [19,20] suggested a probabilistic beliefs-based approach to non-interference. Clark et al. [14,15,17] was concerned with measuring the amount of information flow caused by interference between variables by using information theoretic quantities. The main weakness of this work is that the bounds for loops are over pessimistic, if any leakage is possibly leaked in a loop then all the security information are leaked via the loop according to their rules. A more precise leakage analysis is needed. Malacaria [39] gave a more precise quantitative analysis of loop construct using partition property of entropy. However, the problem is that its application is hard to automate and there is no formal treatment.

There has been significant activity around the question of how to measure the amount of secure information flow caused by interference between variables in a

security context by using information theoretic quantities. This section gives a brief overview of the related work and provides a context for our work.

The precursor for this work was that of Denning in the early 1980's. Denning [23] showed that the data manipulated by a program can be typed with security levels which naturally assume the structure of a partially ordered set. Moreover, this partially ordered set is a lattice under certain conditions [22]. Denning was the first to explore the use of information theory as the basis for a quantitative analysis of information flow in programs [23]. However, she did not suggest how to automate the analysis or deal with loops. In 1987, Millen [45] first built a formal correspondence between non-interference and mutual information, and established a connection between Shannon's information theory and state-machine models of information flow in computer systems. Later related work is that of McLean and Gray in the 1990's. McLean presented a very general Flow Model in [44], and also gave a probabilistic definition of security with respect to flows of a system based on this flow model. The main weakness of this model is that it is too strong to distinguish between statistical correlation of values and causal relationships between high and low objects. It is also difficult to apply to real systems. Gray presented a less general, more formal and detailed elaboration of McLean's flow model in [31], making an explicit connection with information theory through his definition of the channel capacity of flows between confidential and non-confidential variables.

Clark, Hunt, and Malacaria's work [14,16,15,17] presented a more complete quantitative analysis but the bounds for loops are imprecise. They developed a system of syntax-directed analysis rules to give the analysis of information flow in a simple deterministic while language. The rules for each command provided lower and upper bounds on the amount of leakage of one or more variables in the command. However, bounds for loops based on the dependency analysis rules gave an overly conservative treatment for the while loops as with any pure static analysis. It pessimistically assumed all information that can be leaked from the variables on which the loop depends will be leaked. A more precise leakage analysis was needed. Malacaria [39] gave a more precise quantitative analysis of loop constructs by using the partition property of entropy. However, this analysis is hard to automate. Following this work, Chen and Malacaria [13] presented a mechanism for quantitative leakage analysis for multi-threaded programs. The idea was based on program transformation: a multi-threaded program with a probabilistic scheduler was transformed into a single-threaded program with probabilistic operators and the leakage of the single-threaded program was computed by applying the leakage formula introduced in [39]. Clark and Mu [49,52] automated the exact leakage analysis for programs in a simple while-lanague by using Kozen's probabilistic semantics. They also presented an approximation of the exact leakage analysis by using abstract interpretation [50,48].

There are also some achievements on probabilistic and concurrent systems in the 2000s. Di Pierro, Hankin and Wiklicky [24] gave a definition of probabilistic measures on flows in a probabilistic concurrent constraint language where the interference came via probabilistic operators. However, the approach of approximate non-interference presented in this paper is based on the specific probabilistic declarative language: Probabilistic Concurrent Constraint Programming (PCCP). Gavin

Lowe [37] measured information flow in CSP by counting refusals. He devised a formal definition for the information quantity transmitted from a high level user (High) to a low level user (Low) in a computing system. The definition was based on the number of different behaviours of High that can be distinguished from Low's point of view. According to Lowe, if there are n such distinguishable behaviours (under uniform distribution), then $\log_2 n$ bits information was leaked to Low. Like other quantitative definitions, Lowe's definition was based on Shannon's information theory. Unlike other models, this definition was sensitive to non-determinism. Aldini and Di Pierro [3] introduced a method of quantifying information flow on a probabilistic process system. They measured the process similarity relation with regard to an approximation of weak bisimulation of CCS. Backes [5] addressed quantifying information flow within the environment with reactive settings and allowing interactive behaviours. The author introduced a definition for measuring the quantity of information flow within interactive settings by measuring the distance between different behaviours of high user from low user's views. The key idea of the information flow quantity is to study the different behaviours of high confidential user that result in different *views* of the low confidential user. The maximum distance between these different views provides a sound measure of the information flow quantity in the worst case. However, the paper did not suggest how to actually measure the distance of user views or probability distributions due to any specific attack models. Boreale [9] studied quantitative models of information leakage in the process calculi by applying an information theoretic framework. The author presented two quantitative models of information leakage in the process calculus: one for measuring absolute leakage and the other for measuring the rate at which information was leaked. The *absolute leakage*, measured in bits, presented the absolute leakage of zero precisely when it satisfies secrecy. The *rate of leakage*, measured in bits per action, presented the maximum information extracted by repeated experiments coincided with the absolute leakage of the process. A weakness of the ratio formulation was that it was difficult to apply to recursive processes.

Recently, some new approaches and definitions have been presented. McCamant and Ernst [41] have investigated techniques to quantify information flow revealed by complex programs by building flow graphs and considering the weight of the maximum flow over it. The flow is measured by counting the tainted bits of the concerned public variable. This method provided an automatic dynamic quantitative analysis of flows but did not compute upper bounds on the information that a program can release. Köpf and Basin [34] developed a quantitative model to assess a system's vulnerability to adaptive side-channel attacks. The algorithms compute information-theoretic bounds according to the number of program executions. The method is hard to apply to large systems. More recently, Smith [60] considered a new foundation based on the concept of vulnerability, which measures uncertainty by applying Rényi's min-entropy rather than Shannon entropy. Backes, Köpf, and Rybalchenko [7] presented an automatic method for information flow analysis that discovered what information was leaked by considering equivalence relations on program variables. Heusser and Malacaria [32] developped a method to measure information leaks in C programs by calculating the partition of input states.

In summary, quantitative analysis has been studied in the following languages

and contexts: a core imperative language [14,16,15,17,39,34,7], real imperative language with refinements [41], probabilistic concurrent constraint language [24,28], process algebras [37,38,3,5,9], and general models [45,44,31,63,62,42,43]. The following measures have been investigated and applied: Shannon's information theory [23,14,16,15,9,17,39,34], min-entropy [60], guessing entropy [34], refusals counting [37,38], tainted bits counting [41], distance or similarity measures [5,24,28,3,47,51], and belief-based measures [19,20]. The foundational work in quantifying information flow for security within software systems has been developed as discussed above. With respect to the problem of generalisation and quality of the quantified analysis, some of the approaches discussed above can provide automatic and general verified analysis but their leakage bounds are not precise enough [14,16,17,41,7], while some of the approaches can give tight bounds on leakage but there are no automatic analyses [39]. Some automatic leakage analysis approach are developed but suffered from tractability problems [52,50,48,32,7].

All the work to date suffers one of two problems: either it is verified but does not give tight bounds or all the examples give tight bounds but there is no general verified analysis. The *quality* of the analysis for the measurement of information flow need to be improved. A both automatic and precise analysis system is required.

The purpose of this report is to summarize the related work in the area of quantifying information flow for information security and to present some open challenges in this area. It covers several areas that have been associated with security information flow: non-interference, declassification, non-deducibility on strategies, information flow quantity, approximate non-interference, static analysis for quantifying information flow in simple a imperative language, quantifying information flow in process calculi, probabilistic beliefs based approach to non-interference, and quantitative approaches to covert channel analysis etc.

The rest of this report is structured as follows. In the next section we give a brief introduction to information theory. Section 3 gives a detailed survey in the area of quantifying information flow for security. Finally, we sum up and present some open challenges in this area.

2 Background

2.1 Secure information flow and quantity

Information flow security is concerned with how secure information is allowed to flow through a computer system. Information is processed and flows between devices and agents. Intuitively, the flow is considered *secure* if it accepts a specified policy. A program is considered secure if all the flows in the program satisfy the policy. Information flow policies [30,44] are end-to-end security policies, which provide more precise control of information propagation than access control models.

The security policies can be categorized into two core principles: confidentiality, and integrity. The *confidentiality* policies require that high-security information does not affect the low-security observable behaviour of the system. Such policies constrain who can read the data and where the secret data will flow to in the future. Dually, information flow *integrity* policies require that untrustworthy information

does not affect results that must be correct. Such policies constrain who can write to the data and where the data is derived from in the history. In classical lattice security models (Bell-LaPadula and Biba), confidentiality and integrity are treated as duals of one another, where information may only flow up the confidentiality lattice, and information may only flow down the integrity lattice. In this report, we concentrate on confidentiality problems. Such policies are especially useful in protecting data confidentiality, where the goal is to ensure that secret data does not influence public data. An ideal confidentiality property called *non-interference* is a guarantee that no information about secret inputs can be obtained by observing a program's public outputs, for any choice of its public inputs.

Non-interference was first introduced by Goguen and Messeguer [30]. In its original form, a computer is modeled as a machine with inputs and outputs. Inputs and outputs are classified as two security levels: low (public) and high (confidential). A computer satisfies the non-interference property if and only if any sequence of low inputs will produce the same low outputs, regardless of what the high level inputs are: for any system with output function $out(u, I)$, whose value is the output generated by input history I to user u, $out(u, I) = out(u, I')$, where I' is I removed all inputs from users with security levels that are higher than u's. That is, if a low user is operating on the machine, it will produce the exactly same behaviours whether or not a high user is working with sensitive data. The low user will not be able to deduce any information about the behaviours of the high user. The basic concept of non-interference can be applied as a strict multilevel security policy model.

Next let us consider *interference* between program variables. Secure information flow of a terminating program can be considered as follows: given a terminating program P with high security variables $H = h_1, \ldots, h_n$ and low security variables $L = l_1, \ldots, l_n$, P is secure if and only if the values of L at the point that P terminates are independent of the initial values of H. Informally, interference happens if the values of variables affect the values of other variables. Formally, *non-interference* for confidentiality can be defined as the following:

Definition 2.1 [non-interference] Define the low equivalence relation as:

$$\sigma \equiv_L \sigma' \quad \textit{iff} \quad \forall x \in Ide_L,\ \sigma x = \sigma' x$$

where Ide_L denotes a set of low level identifiers, σ and σ' denote stores. Program P is Non-interfering (secure) if,

$$\sigma_1 \equiv_L \sigma_2,\ [\![P]\!]\sigma_1 = \sigma_1',\ \textit{and}\ [\![P]\!]\sigma_2 = \sigma_2'$$

then $\sigma_1' \equiv_L \sigma_2'$, where σ_1 and σ_2 are the stores before the execution of program $[\![P]\!]$, and σ_1' and σ_2' are the stores after the execution of program $[\![P]\!]$.

Non-interference, *i.e.* absence of interference, has been often used in proving that a system is well behaved, whereas interference can lead to misbehaviour. However, the presence of interference considered as misbehaviour will generally happen only when the interference is above a *threshold*. A concrete example is a software system with *access control*. To enter such a system the user has to pass an identification stage: whether subsequently authorised or failed, some information has been leaked so these systems present interference. If the interference in such systems is *small*

enough we can be confident in the security of the system. Security requires determining *how much* information flows from *high* level to *low* level, which is known as *Quantitative Information Flow* (QIF). Consider the following examples, which show that secure information flow is violated during the execution of the programs:

```
(1) l := h;
(2) if (h==0) then l := 0 else l := 1;
(3) l := h; while (l<>0) l := l*2;
```

where l is a low security variable, h is a high security variable. It is obvious that, the *assignment* command causes the entire information in h to flow to l, the *if statement* makes one bit of information in h flow to l, and l learns some information (whether h equals to zero or not) about h via the termination behaviours of the *while loop* command.

Note that executing a program reduces the uncertainty about secure information and causes the information leakage. Quantifying and measuring information flow provides a way to compute bounds on how much information is leaked, which can be used to compare with the thresholds, and thus suggesting whether the program is secure (the leakage is below the threshold) or not (the leakage is above the threshold) from a quantitative point of view. Most quantitative measurements of information flow are based on Shannon's *information theory*. Lower and upper bounds on such measure are given in units of information bits. In order to measure information flow, we treat the program as a communication channel. Shannon's mathematical theory of communication, information theory, is a suitable way to measure the quantity of the information flow over such communication channels.

2.2 Probability Theory

We first review the basic definitions and concepts of probability theory. A more detailed introduction can be found in [58]. Probability theory is concerned with the analysis of random experiments such as tossing a coin or rolling a die. A random experiment is an experiment which can be repeated numerous times under the same conditions. The outcome of each individual experiment depends on "chance": must be independent, identically distributed and can not be predicted with certainty. The relative frequency of an event for any random experiment is called the probability of the event. Consider the experiment of tossing a coin as an example. The experiment can yield two possible outcomes: heads and tails, both with probability of one half in any long series of trials. Probability theory normally treats discrete probability distributions and continuous distributions separately. The definition of a discrete probability distribution is presented as follows.

Definition 2.2 [Discrete probability distribution] Let D be a set. A discrete probability distribution on D is a function $f : D \to [0, 1]$ such that $\sum_{d \in D} f(d) = 1$.

2.3 Information theory

In order to measure the information flow, we treat the program as a communication channel. Information theory introduced the definition of *entropy*, $\mathcal{H}$, to measure the *average uncertainty* in random variables. Shannon's measures were based on a log-

arithmic measure of the unexpectedness in a probabilistic event (random variable). The unexpectedness of an event which occurred with some non-zero probability p was $\log_2 \frac{1}{p}$. Hence the total information carried by a set of n *events* was computed as the weighted sum of their unexpectedness:

$$\mathcal{H} = \sum_{i=1}^{n} p_i \log_2 \frac{1}{p_i}$$

where $p_i \log_2 \frac{1}{p_i} = 0$ if $p_i = 0$. This quantity is called the *entropy* of the set of events.

Considering a program as a state transformer, random variable X is a mapping between two states which are equipped with distributions. We generally use the language of random variables rather than talk directly about distributions, the definition of discrete random variable is given as follows.

Definition 2.3 [Discrete random variable] A discrete random variable X is a surjective function which maps events to values of a countable set (e.g., the integers), with each value in the range having probability greater than zero, *i.e.* $X : D \to \mathcal{R}(D)$, where D is a finite set with a specified probability distribution, and $\mathcal{R}$ is the finite range of X.

In what follows, we review the basic definitions in Shannon's information theory. For more details on the fundamental definitions of information theory, see [59,21].

Definition 2.4 [Shannon entropy] Let $p(x)$ denote the probability that X takes the value x, then the *entropy* $\mathcal{H}(X)$ of discrete random variable X is defined as:

$$\mathcal{H}(X) = \sum_{x \in X} p(x) \log_2 \frac{1}{p(x)} = -\sum_{x} p(x) \log_2 p(x)$$

The entropy is measured in bits and is a measure of the uncertainty of a random variable (which is never negative). Intuitively, it is the number of bits on the average required to describe the random variable.

The definition of the entropy of a single random variable can be extended to a pair of random variables, which is called their *joint entropy*.

Definition 2.5 [Joint entropy] The joint entropy $\mathcal{H}(X, Y)$ of a pair of discrete random variables (X, Y) with a joint distribution $p(x, y)$ is defined as

$$\mathcal{H}(X, Y) = \sum_{x \in X} \sum_{y \in Y} p(x, y) \log_2 p(x, y)$$

Furthermore, given two random variables X and Y, the notion of *conditional entropy* suggests possible dependencies between random variables, i.e., knowledge of one may change the information about the another.

Definition 2.6 [Conditional entropy] Assume (X, Y) follows the joint distribution $p(x, y)$, the conditional entropy $\mathcal{H}(Y|X) = \sum_x p(x)\mathcal{H}(Y|X = x)$ is defined as:

$$\begin{aligned} \mathcal{H}(Y|X) &= \sum_{x \in X} p(x)\mathcal{H}(Y|X = x) \\ &= -\sum_{x \in X} p(x) \sum_{y \in Y} p(y|x) \log_2 p(y|x) \end{aligned}$$

$$= -\sum_{x\in X}\sum_{y\in Y} p(x,y)\log_2 p(y|x)$$

The concept of *mutual information* is a measure of the amount of information that one random variable contains about another one, *i.e.* shared information. It implies the reduction in the uncertainty of one random variable due to the knowledge of the other.

Definition 2.7 [Mutual information] Let $p(x,y)$ denote the joint distribution of $x \in X$ and $y \in Y$, the notion of*mutual information* between X and Y, $\mathcal{I}(X;Y)$, is given by:

$$\begin{aligned}\mathcal{I}(X;Y) &= \sum_{x}\sum_{y} p(x,y)\log_2 \frac{p(x,y)}{p(x)p(y)}\\ &= \mathcal{H}(X) - \mathcal{H}(X|Y)\\ &= \mathcal{H}(Y) - \mathcal{H}(Y|X)\end{aligned}$$

There are also conditional versions of mutual information: $\mathcal{I}(X;Y|Z)$ denotes the mutual information between X and Y given knowledge of Z. The conditional mutual information can be considered as the reduction in the uncertainty of X due to knowledge of Y when Z is given.

Definition 2.8 [Conditional mutual information] The conditional mutual information of random variable X and Y given Z is defined as follows:

$$\begin{aligned}\mathcal{I}(X;Y|Z) &= \mathcal{H}(X|Z) + \mathcal{H}(Y|Z) - \mathcal{H}(X,Y|Z)\\ &= \mathcal{H}(X|Z) - \mathcal{H}(X|Y,Z)\\ &= \mathcal{H}(Y|Z) - \mathcal{H}(Y|X,Z)\end{aligned}$$

3 Survey of quantifying information flow

In this Section, a detailed survey of related work in quantitative information flow analysis from 1980's to date is provided. Figure 1 shows the development history of quantitative information flow.

3.1 First exploration: information theory and QIF

Denning [23] presented a definition of information flow quantity for programs based on information theory. Consider a terminating program P and two program variables h, l with two states σ, σ' in the execution of P. Denning suggested that there was a flow of information from h at state σ to l at state σ' if the uncertainty about the value of h at the initial state σ given knowledge of l at the final state σ' was less than the uncertainty about the value of h at σ given knowledge of l at σ.

Denning defined the following condition to indicate that there was a flow:

$$\mathcal{H}(h_\sigma|l_{\sigma'}) < \mathcal{H}(h_\sigma|l_\sigma)$$

Based on this definition, if there was a flow, the amount of information transferred was quantified by Denning as the difference in uncertainty between the two situations:

$$\mathcal{H}(h_\sigma|l_\sigma) - \mathcal{H}(h_\sigma|l_{\sigma'})$$

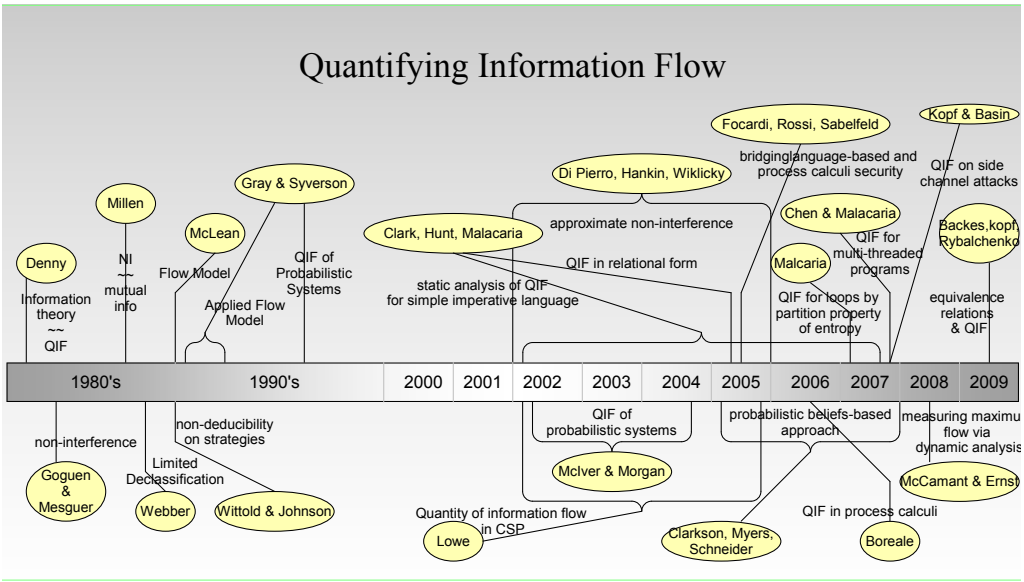

Fig. 1. History: Quantitative Information Flow

The first term $\mathcal{H}(h_\sigma|l_\sigma)$, represented the initial uncertainty in h *after observing* l, whereas the second term represented the final uncertainty in h, after observing final value of l. A problem lay in the second term: it accounted for the final observation of l but effectively assumed that the initial observation had been forgotten. This was a safe assumption only if we knew that the observer had no memory. However, we must assume that the observer knew *both* the initial *and* final values of l in general.

Denning first explored the use of information theory as the basis for a quantitative analysis of information flow in programs. However she did not suggest how to automate the quantitative analysis nor account for loops.

3.2 Secure information as non-interference

Information flow security is associated historically with *non-interference* [30]. Consider *interference* between variables of program: informally, the capacity of variables affects the values of other variables. Non-interference, i.e., absence of interference, is often used in proving that a system is well-behaved, while interference can lead to mis-behaviours. Suppose that the variables of a program are partitioned into two sets, *high* and *low*. High variables may contain confidential information when the program is running, but these variables cannot be examined by an attacker at any point before, during or after the program's execution. Suppose low variables do not contain confidential information before the program runs and can be freely examined by an attacker before and after (but not during) the program's execution. This raises the question of what an attacker may be able to learn about the confidential inputs by examining the low variable outputs. Non-interference looks for conditions under which the values of the high variables have no effect on (do not "interfere" with) the values of the low variables when the program is running. For the formal definition of non-interference in the current setting see Definition 2.1. If this condition holds, an attacker learns nothing about the confidential inputs by

examining the low outputs. Thus non-interference addresses the question whether a program leaks confidential information or not.

Following this section, we present a review of related work on non-interference from an information theory or probability theory perspective, which was quite extensively studied in [30,31,33,44,45] as an approach to confidentiality.

3.2.1 Millen's approach

Millen [45] established a connection between Shannon's theory of communication and state-machine models of information flow in computer systems. This work was the first to build a formal correspondence between non-interference and mutual information.

Using the state machine model, the author proved that the notion of non-interference is equivalent to the mutual information between random variables representing certain inputs and outputs being equal to zero. He then used this equivalence to measure interference in state machine systems, in particular to study the channel capacity for covert channels.

In this model, Millen described the computer system as a communication channel having one input and one output, which was relative to information flow from one given user to another given user. The information transmitted over the channel from X to Y in one trial was defined as the mutual information $\mathcal{I}(X;Y)$ in information theory. Here a *trial* was a sequence of inputs W and a single final output Y. The calculation of $\mathcal{I}(X;Y)$ was based on the formula:

$$\mathcal{I}(X;Y) = \mathcal{H}(Y) - \mathcal{H}(Y|X)$$

where $\mathcal{H}(Y)$ was the entropy of Y. The maximum mutual information over all possible distributions for X was defined as the channel capacity in information theory. The information flow over a covert channel then can be calculated using the standard formulas for channel capacity, with assumptions that inputs from X were independent of those from other users. It was necessary to point out that calculation of information flow depended on the distribution of inputs.

3.2.2 McLean's flow model

McLean developed a security Flow Model (FM) [44] based on Shannon-style information flow. McLean argued that Sutherland's non-deducibility model [61] was most self-consciously based on information theory. This model stated that there was no information flow from high-security-level objects to low-security-level objects if and only if

$$p(H) > 0 \ \wedge \ p(L) > 0 \ \Rightarrow \ p(H|L) > 0$$

and also

$$p(H) > 0 \ \wedge \ p(L) > 0 \ \Rightarrow \ p(L|H) > 0$$

based on the the fact that $p(H|L)p(L) = p(L|H)p(H)$, *i.e.* non-deducibility does not allow information flow from low-level objects to high-level objects as well. However, security policies normally allowed for information flow from low-level objects to high-level objects. Hence non-deducibility may be helpful to reason about independent subsystems of a system, it cannot capture the notion of non-interference as intended

in a security context.

To avoid this problem, McLean introduced *time* into his security model and viewed information as flowing from H to L only if H assigns values to objects in a state that preceded the state in which L makes its assignment. In this model, security did not require that $p(H_t|L_{t-1}) = p(H_t)$, *i.e.* low information cannot contribute to high information, while the requirement was that $p(L_t|H_{t-1}) = p(L_t)$ or $p(H_{t-1}|L_t) = p(H_{t-1})$ equivalently, *i.e.* high information cannot contribute to following low information.

However such a model did not rule out the possibility that users may learn something about H_{t-1} by taking the combined knowledge of L_t and L_{t-1}. To address this concern, the requirement $p(L_t|(H_{t-1}, L_{t-1})) = p(L_t|L_{t-1})$ was introduced.

The main weakness of this model is that it is too strong to distinguish between statistical correlation of values between high and low objects and causal relationships between high and low objects. For example, a security violation emerges not only when the value of L_t is statistically correlated with the value of H_{t-1}, but also when the value of H_{t-1} has exerted some causal influence on the value of L_t. The statistical correlation cannot not arise simply because the values of both are caused by some other low-level event.

Although this model considered only information flow between a state and its successor state rather than all security relevant causal relations, McLean gave a more inclusive information flow model called *Flow Model (FM)* next. It stated that a system was secure if $p(L_t|(H_s, L_s)) = p(L_t|L_s)$, where L_t described the values taken by the low-level objects at time t, L_s and H_s were the sequences of values by low-level objects and high-level objects, respectively, in every state preceding time t. His definition of security was given in the form of an equation involving conditional probabilities, where the probability space included user inputs. The problem here was that the probability of a given user input was unknown.

The author also examined *non-deducibility*, *non-interference*, and the *Bell and LaPadula* model by using this model as a standard. He argued that although *non-deducibility* and extensions to generalised non-interference helped to protect high-level output generated from low-level input, they cannot take necessary causal information into account. Finally, he showed the relationship between *non-interference* and the *Bell and LaPadula Model* and stated that they were equivalent. McLean gave a general treatment of information flow security models. However, it is difficult to apply to real systems since this work is intended to provide a method for evaluating security models rather than a means of evaluating real systems.

3.2.3 Gray's applied flow model

Gray gave a probabilistic state machine model [31,33] to model a more general non-deterministic computer systems. Different from McLean's flow model, Gray's model used probabilistic transitions based on the inputs and internal states over the channels rather than non-deterministic transitions between a set of high channels and a set of low channels.

Gray's finite state machine flow model consisted of a finite set of states including inputs, outputs, internal system states and initial states, a set of communication channels, and a probabilistic transition function T used to describe the transition

of internal probabilistic states. The transition function T was defined as:

$$S \times I[C] \times S \times O[C] \rightarrow [0,1]$$

It mapped a tuple consisting of a source state $s \in S$ at some point in time, and the input vector $a \in I[C]$ at that time on all communication channels; a target state $s' \in S$, and the output vector $b \in O[C]$ on all communication channels to the probability of the system producing this output vector b and transiting to this target state s'. Hence for the whole system $\Sigma = (C, I, O, S, T)$, for all $s \in S$ and for all $a \in I[C]$, the *probability measure function* $P_{s,a} : \mathcal{P}(S \times O[C]) \rightarrow [0,1]$ was defined as: $P_{s,a}(x) \equiv \Sigma_{(s',b)\in x} T(s, a, s', b)$.

Gray's information flow security condition was based on *non-interference*, which was given as:

$$P_t(\alpha_L, \beta_L, \alpha_H, \beta_H) > 0 \Rightarrow P_t(l_t|\alpha_L, \beta_L, \alpha_H, \beta_H) = P_t(l_t|\alpha_L, \beta_L)$$

where $\alpha_L(\alpha_H)$ was the input history on channel LOW(HIGH) up to and including time $t-1$, $\beta_L(\beta_H)$ was the output history on channel LOW(HIGH) up to and including time $t-1$, and l_t was the final output event at time t. This definition indicated that the probability of a low output might depend on previous low events, but not on previous high events. Hence low users could not get any information about secure behaviour by observing low outputs, written as $H \not\rightarrow L$.

Gray gave the definition of channel capacity with memory (i.e. internal state), outputs to the receiving end of the channel L, and inputs from the sending end of the channel H. Note that Shannon's original formulation of channel capacity was for discrete memoryless channels with no inputs from the receiver and no feedback to the sender. Gray defined the channel capacity from H to L as:

$$C \equiv \lim_{n\rightarrow\infty} C_n$$

where C_n was defined as,

$$C_n \equiv max_{H,L} \left(\frac{1}{n} \Sigma_{i=1}^{n} \mathcal{I} \left(\begin{array}{c} \textit{In-Seq-Event}_{H,i}, \\ \textit{out-Seq-Event}_{H,i}; \\ \textit{Final-Out-Event}_{L,i}| \\ \textit{In-Seq-Event}_{L,i}, \\ \textit{Out-Seq-Event}_{L,i} \end{array} \right) \right)$$

The intuition behind this definition was that the channel capacity was the least upper bound on the rate at which information was able to be reliably transmitted over the channel. Gray also established a relationship with information theory. He proved that if the flow security condition $H \not\rightarrow L$ holds, the channel capacity from H to L is zero.

3.3 n-Limited secure system

In [63], Weber defined a property of n-limited security, which took flow-security and specifically prevented downgrading unauthorised information flows. He also generalised it to formulate a policy that allowed limited downgrading of protections.

Weber modeled the property of *flow-security* by a specific class of state machines. A state machine consisted of a set of internal states, a set of input and output events, and a special function *lvl* which mapped events into the series possible security levels. A sequence of events γ caused the state transitions from the current state to a new state, written as, $s_{old} \xrightarrow{\gamma} s_{new}$. Each event including input event and output event was assigned a security level. The information flow from high-level inputs was then propagated through each transition of the state machine. Each state contained a high-level component and low-level component, and equivalence relation provided a way to make two states belong to the same equivalence class when they shared the same low-level component. Hence security was expressed as properties of the transition relation which prevent high-level events of the state from causing changes to low-level components of the state. A machine was flow-secure if it satisfied the defined security conditions at every (low) level.

Weber also defined a composed state machine of two interconnected state machines. Each event of the interconnected state machine was an output of one machine and an input of the other machine, and the common set of events of the two machines agreed on the security level of these events. A composable property for flow-security held for the interconnected state machines if it was agreed with both of the two state machines independently.

The definition of *n-limited* security property was given to limit downgrading the perfect security to a degree pointed by the positive integer n via storage channels. Given a positive integer n, a state machine was *n-limited* secure at level l if there existed two equivalence relations on states for *high-level* and *low-level* components and a *shared* component: $\equiv_H, \equiv_L, and \equiv_S$, such that information can flow from the high component to the low component only by passing through the shared component. The shared region refined each equivalence class of the low component into at most n new equivalence classes, *i.e.* a low-level user can distinguish n shared states. Note that the degree of the *n-limited* system is not measured, but just bounded. For example, a *n-limited* secure system at level l may leak more than $\log_2 n$ bits because it can leak $\log_2 n$ bits at each level. It would be better to provide a security policy which specifies the worst leakage that can actually be released by the system.

Furthermore, the *n-limited* property was composable such that if machine A was *n-limited* secure and B was *m-limited* secure, then the composed machine of A and B was *mn-limited* secure. Thus the damage of a channel for downgrading by a composite machine is limited by the maximum damage of channels on its component machines. The composed machine may leak more than the two original leaky machines, but not arbitrarily badly. Finally, Weber pointed out that the limiting bandwidth of composed machine would be $(2\log_2 nm)/t$ bits per time t, which was double the bandwidth that of each concurrent machine running independently. However, a more detailed analysis of the relation between equivalence classes of states and bandwidth is required.

3.4 Non-deducibility strategies in probabilistic systems

Wittbold and Johnson [64] gave an analysis of certain combinatorial theories of computer security from an information-theoretic perspective, and introduced the

security property non-deducibility on strategies. The combinatorial theories of security considered here share three properties: they characterise security in terms of legal and illegal information flows; they are theories of non-deducibility; they are applicable in non-deterministic environments. They compared three information-flow theories of security: non-deducibility on transmitter inputs, non-deducibility on strategies, and forward correctability.

Sutherland's non-deducibility [61] stated that low security levels cannot deduce anything about the behaviour of the high level processes or users with certainty. Formally, given the set of traces of the system T, he defined f_2 as a function giving the low view which returns the subsequence of low events from a given trace, and defined f_1 as a function that extracts the high behaviour which returned the subsequence of high input events. Thus the *security condition* translates as there is no information flow from f_1 to f_2. This general non-deducibility on transmitter inputs was based on the state machine mechanism, the traces were generated by a non-deterministic state machine. However, Wittbold and Johnson showed that non-deducibility on transmitter inputs was not sufficient to protect a system against transmission from high to low by giving an abstract *encryption* example which satisfied non-deducibility but permitted a noiseless communication channel from high input to low output. In other words, the receiver could deduce something about the high level inputs via the *feedback* from the system to the transmitter if the program can be run multiple times and feedback was permitted.

Wittbold and Johnson then introduced non-deducibility on strategies due to the problem of *feedback*. The new model consisted of a non-deterministic state machine controlled by two processes (high transmitter T and low receiver R) with both inputs and outputs. Given a sequence of trials and inputs from both high transmitter and the low receiver, the machine went into a new state which was non-deterministically chosen from a set of possible next states. Formally, a trace of the state machine was a finite sequence of transitions such as:

$$s_0(i_1, j_1, k_1, l_1)s_1 \dots s_{n-1}(i_n, j_n, k_n, l_n)s_n$$

where i_m was the input from receiver R, j_m was the input from transmitter T, k_m was the output to receiver R, l_m was the output to transmitter T, and s_m were the new states returned from the *next-state* function which mapped the current states, the inputs from R and inputs from T to a set of next states, and $1 \leq m \leq n$. They also defined the *transmitter strategy* π to choose an input to be used from a given sequence of its previous inputs and outputs. If we cannot find non-deducibility on strategies to exclude some receiver view λ, then the transmitter can use π to choose an input for the receiver. Otherwise the failure of non-deducibility on strategies provided a way to build noiseless channels between a transmitter and a receiver. In other words, noiseless communication between transmitter and receiver was removed if the system had the property of non-deducibility on strategies.

The authors then defined a class of contention systems which consisted of three process: transmitter, noise, and receiver. Such systems were used to build channels between the transmitter and receiver. They showed that the system contained noiseless channels from high input to low output if the associated state machine was not non-deducible on strategies and argued that the covert channel of such systems could be analyzed completely. On the one hand, they expressed that each contention

system was associated with a non-deterministic synchronised state machine, and thus provided a way to characterise the security of the state machine in terms of how the output of the receiver relied on the inputs of transmitter and noise. Secondly, they stated that each contention system could be transformed to discrete memoryless channels, and then it was possible to represent the channel capacity no matter what distribution applied in the noise process.

Their examples provided for the introduced resource contention systems showed the weakness of the combinatorial theories of security. The main weakness of their definitions is that its security properties applied only through possibilistic specifications. However, possibilistic view of security for computer systems is inadequate for addressing the main problems of computer security. A probabilistic view of security is required to process the problem of noiseless communication channels.

3.5 On the Foundations of Quantitative Information Flow

There has been much recent work that considered the foundational definitions of quantitative information flow, including the works of McIver and Morgan [42], Clarkson, Myers, and Schneider [19,20], Clark, Hunt, Malacaria, and Chen [14,16,15,17,39,13], Smith [60].

3.5.1 Probabilistic Approach for Information Flow Security

McIver and Morgan developed an information flow theory for a sequential programming language accommodated with probabilities in [42]. They devised a different information theoretic definition of information flow and channel capacity to Gray's [31], aiming at a different security property rather than the one embodied in the Flow Model (FM). They then used this definition to derive a non-probabilistic characterisation of the security property for a simple imperative language. The paper also established a number of equivalent conditions for a sequential language program to meet their security condition via looking for simpler formulations based on program refinement.

In traditional information theory, a channel was a model of data transmission, which was described by a probabilistic function from input values to output values. In this paper, the authors argued that channel can be viewed as the operation of a program fragment combined with channel-like observations of the *before* and *after* states via projection onto the LOW variables. Hence the information quantity was defined as the difference between what can be deduced about high security variables h before the program runs and what can be deduced after via observing low security variables l, which is also a notion of channel capacity of the program. Formally, let h and l be the random variables corresponding respectively to the high security and low security partitions of the store of the program, $\triangle$ and $\triangle'$ be the initial distribution and final distribution for a specific execution of the program. The information flow ("net information escape" as they called) from h to l was given by

$$\mathcal{H}_{\triangle}(h|l) - \mathcal{H}_{\triangle'}(h|l)$$

It can be interpreted that "uncertain before" minus "uncertain after", which indicated how much information will be leaked by the program. They then defined a notion of channel capacity based on this definition of flow quantity. The channel ca-

pacity of the program was defined as the least upper bound over all possible input distributions of the information flow quantity by considering the "worst case" of the program's behaviour over all possible h, l distributions. The system was secure when the channel capacity of the program was zero. This was similar to Gray, but McIver and Morgan's definition of flow quantity did not consider the history.

However, the calculation of exact channel capacities was very difficult for most realistic cases. They then developed some simpler alternative formulations based on program refinement. First they showed that a secure permutation can only permute low values, which guaranteed neither the initial nor the final high values can be deduced from the final values of program operations. Then they showed that this secure permutation condition was equivalent to previous information security condition given by channel capacity, and secure permutation also preserved maximum entropy of high values.

3.5.2 Probabilistic Beliefs-based Approach

In [19,20], Clarkson, Myers, and Schneider presented a novel perspective for quantitative information flow in programs by incorporating attacker beliefs into the analysis. They presented a model to describe the variation of attacker beliefs based on the attacker's observation of the program execution. *Belief* was interpreted as the judge about the state of the system with a measure of how certain about this judge, *i.e.* a probability distribution over high states. A mathematical structure was developed to represent the "beliefs reasoning" about the probabilistic programs. This belief was then revised using Bayesian techniques by repeatedly the running of the program.

An example proposed in [19,20] is a password checker: suppose there were three possible passwords A, B and C in a password checking program. The attacker believed that A was the most possible real password with 99% confidence and he thought B and C were equally likely with probability of 0.5% for each. If the attacker failed by guessing A then his belief had greatly changed: he was more confused than before about what the password was, and the outcome of this experiment revealed more information than the previous one. Thus the information conveyed by executing program depended on how strongly the attacker believed something to be true.

The model presented in this paper was based on an *experiment* (*i.e.* a single run of the program). The experiment described the interaction between attackers and the system, how the attacker reasoned about the secret data, and how the attacker updated beliefs by interacting and observing execution of programs. The attacker firstly chose a pre-belief about high state, which described his belief at the beginning of the experiment. Then the system chose a high projection of the initial state. Using the semantics of program along with the pre-belief as a distribution on high input, the attacker conducted a "thought experiment" to generate a prediction of the output distribution. The systems executed the program, which produced a state as output. Then the attacker inferred a post-belief based on his observation of the low projection of the output state.

Their approach of measuring the amount of information flow was defined as the difference between the accuracy of the attacker's pre-belief and post-belief. The

amount information flow was thus the improvement in the accuracy of the attacker's belief. In addition, the model of experiments also enabled compositional reasoning about information flow from attacks in sequences of interactions by performing a series of repeated experiment, where the post-belief from one experiment becomes the pre-belief to the next. Thus the total information flow was given by the sum of the information flow for all experiments.

This paper developed a metric for quantifying information flow to measure both accuracy and uncertainty of an attacker's beliefs, which could be treated as complementing traditional information flow metric which was based on the change of uncertainty. However, in general, the result depended on the probability distribution of the secret. This paper argued that the attacker's prior beliefs should also be considered, since an attacker could gain an unbounded amount of information from the result of a single yes-no query, if it allowed him to correct a previous erroneous belief. However, such distributions are not likely to be available in most situations. Different attackers have different pre-beliefs, it might not work for some situations, so worst-case bounds must be used instead.

3.5.3 Static Analysis and Equivalence Relations for QIF

In [17], Clark, Hunt, and Malacaria presented a more complete static quantitative information flow analysis for a simple imperative programming language with assignments, sequencing, conditionals and while loop structures. This paper revised and expanded their earlier work [14,16,15], and applied information theory to the static analysis. They established the relations between information theory and the classical programming language definition of non-interference. A set of syntax based analysis rules were presented. The rules for a command provided lower and upper bounds on the amount of leakage of one or more variables in the command.

The metric for information leakage was based on conditional mutual entropy. The definition of information leakage used here was that the high security information flowed to the low output of the program. Let X and Z be input observations, and Y be an output observation. Then the definition of the information flow from X to Y given knowledge of Z was given as:

$$\mathcal{F}_Z(X \to Y) \stackrel{\text{def}}{=} \mathcal{I}(X^{in}; Y^{out} | Z^{in})$$

The authors developed a system of syntax-directed analysis rules to give the analysis of information flow in the simple deterministic while language [14,15,17]. The rules were grouped into five categories: rules for expressions, logical combinations, dependency analysis, direct and indirect flow rules. Leakage inference rules for expression dealt with boolean and arithmetic expressions, which provided the leakage interval for the arithmetic and boolean expressions. They defined logical rules for the analysis of commands to be used to combine the results derived by overlapping rule sets thus constructing the leakage interval with larger domains and specifying smaller intervals. Rules for data processing provided the quantitative dependency analysis, which relies on the theorem which states that the entropy of a function of a random variable cannot exceed the entropy of the random variable itself. [Dependent] rules captured the variable dependencies for commands. However, bounds for loops based on this dependency analysis rules gave an overly conserva-

tive treatment for the while loops as with any pure static analysis. For instance, it pessimistically assumed all information that can be leaked from the variables on which the loop depends on will be leaked. A set of rules for direct flow were defined to track simple direct flows of information due to assignment between variables and sequential control flow. The indirect flow rules were then devised to provide bounds for information leakage rising from the influence of confidential data on the control flow through conditionals, i.e., *if statement*. Finally, they gave an improved leakage analysis for arithmetic expressions by exploiting algebraic knowledge of the operations including addition, subtraction, multiplication $(+, -, *)$ together with information about the operands acquired through supplementary analysis such as parity analysis, constant propagation analysis etc.

Shannon's information theory is used to give a quantitative definition of information flow in inputs/outputs systems. This definition is related to the classical security condition of non-interference and an equivalence is established between non-interference and independence of random variables. Additionally, in [16], the authors investigated the correspondence between equivalence relations and surjective maps. They used this correspondence to derive a relational presentation of information for deterministic transformational systems. They showed that the relational parametricity can be used to derive upper and lower bounds on information flows via families of functions defined in the second-order lambda calculus. Quantitative information flow for deterministic systems is then presented in relational form.

This was the first time information theory has been used to measure interference between variables in a simple imperative language with loops. The information flow to low security variables will be output given a simple imperative language program. It gave a systematic and reasonable quantitative analysis for a particular program in imperative languages. However, the main weakness of this work is that the bounds for loops are over pessimistic, *i.e.*, if any leakage is possibly leaked in a loop then all the security information are leaked via the loop. A more precise leakage analysis was needed.

3.5.4 QIF for Loops by Partition Properties

Malacaria [39] gave a more precise quantitative analysis of the loop construct via a different approach to Clark, Hunt, and Malacaria's static analysis. He defined an information theoretical formula for leakage of the command **while e M** by using the partition property of the entropy. Both the amount and the rate of leakage were described in this paper.

The *partition property* of entropy suggests that the entropy of the space with a partition can be computed by summing the entropy of each weighted partition. Formally, given a distribution μ over a set $S = \{s_{1,1}, \ldots, s_{n,m}\}$: $\mu(S_i) = \sum_{1 \leq j \leq m} \mu(s_{i,j})$, and a partition of S into sets $(S_i)_{1 \leq i \leq n}$: $S_i = \{s_{i,1}, \ldots, s_{i,m}\}$, the entropy of S can be computed by [57]:

$$\mathcal{H}(\mu(s_{1,1}), \ldots, \mu(s_{n,m})) = \mathcal{H}(\mu(S_1), \ldots, \mu(S_n)) + \sum_{i=1}^{n} \mu(S_i) \mathcal{H}\left(\frac{\mu(s_{i,1})}{\mu(S_i)}, \ldots, \frac{\mu(s_{i,m})}{\mu(S_i)}\right)$$

The semantics we are going to use is a state transformer which can be considered as a mapping, so let us consider a function $f : X \to Y$, X and Y being probability spaces. The function f is considered as the union of a family of functions $(f_i)_{i\in I}$ with disjoint domains $(f^{-1}(y_i))_{i\in I}$, *i.e.* for each i, $f^{-1}(y_i) \in X$ is the domain of f_i and $(f^{-1}(y_i))_{i\in I}$ is a partition of X. Specifically, f is called *collision free* if the family $(f_i)_{i\in I}$ has also disjoint domains. The entropy of f can be obtained as the entropy of its inverse images, i.e. $\mathcal{H}(\mu(f^{-1}(y_1)), \ldots, \mu(f^{-1}(y_n)))$, by using the partition property:

$$\begin{aligned}
&\mathcal{H}(\mu(f^{-1}(y_1)), \ldots, \mu(f^{-1}(y_n))) \\
=&\mathcal{H}(\mu(f_1^{-1}(Y)), \ldots, \mu(f_n^{-1}(Y))) + \\
&\sum_{i=1}^{n} \mu(f_i^{-1}(Y))\mathcal{H}\left(\frac{\mu(f^{-1}(y_{i,1}))}{\mu(f_i^{-1}(Y))}, \ldots, \frac{\mu(f^{-1}(y_{i,m}))}{\mu(f_i^{-1}(Y))}\right)
\end{aligned}$$

For the case of f with collisions, the range of the function Y is extended to Y' with new elements to eliminate collisions. Let $f' : X \to Y'$. The derived function with no collisions f'_i is defined as

$$f'_i(x) = \begin{cases} f_i(x) & \forall j \neq i, f_i(x) \neq f_j(x) \\ (f_i(x), i) & \textit{otherwise} \end{cases}$$

Let $C_f(Y)$ denote the set of collisions of f in Y, and $x_1^y, \ldots, x_m^y$ denote the elements of $f^{-1}(y)$. The entropy of f with collisions is given as:

$$\begin{aligned}
&\mathcal{H}(\mu(f^{-1}(y_1)), \ldots, \mu(f^{-1}(y_n))) \\
=&\mathcal{H}(\mu(f'^{-1}(y_1)), \ldots, \mu(f'^{-1}(y_{n'}))) - \\
&\sum_{y\in C_f(Y)} \mu(f^{-1}(y))\mathcal{H}\left(\frac{\mu(x_1^y)}{\mu(f^{-1}(y))}, \ldots, \frac{\mu(x_m^y)}{\mu(f^{-1}(y))}\right)
\end{aligned}$$

Next consider the leakage definition for loops. The basic idea was that it took partitions of the while loop based on the interactions, computed the entropy by summing the entropy of the partition with the weighted entropies of the partition sets. A terminating (terminating on all inputs) loop `while e M` was viewed as a map

$$F \stackrel{\text{def}}{=} \sum_{1\leq i\leq n} F_i$$

Here n was the maximum number of iterations of the loop. F_i was the map that iterated the body M i times with guard being true, until it quit the loop, *i.e.* the guard was false after the i^{th} iteration of M. Hence formally,

$$\texttt{while e M} \stackrel{\text{def}}{=} \sum_{\texttt{0}\leq\texttt{i}\leq\texttt{n}} (\texttt{M}^{\texttt{i}}|\texttt{e}^{<\texttt{i}>})$$

where

$$e^{<i>} = \begin{cases} e = \mathit{ff}, & \textit{if } i = 0 \\ e = tt \wedge e^2 = \mathit{ff}, & \textit{if } i = 1 \\ e = tt, \ldots, e^i = tt \wedge e^{i+1} = \mathit{ff}, & \textit{if } i > 1 \end{cases}$$

and $M^0 = skip$. The events $e^{<0>}, \ldots, e^{<n>}$ built the partition of the states for while loop and since the loop was going to terminate, the sum of the probability of $e^{<i>}$ was 1.

By applying the probability distribution to compute the entropy of functions, Malacaria gave the formal definition of the leakage for a collision free loop `while e M` up to n iterations

$$\begin{aligned}\mathcal{L}_{\texttt{while e M}} &= \lim_{n\to\infty} W(e,M)_n \\ &= \mathcal{H}(\mu(e^{<0>}), \ldots, \mu(e^{<n>}), 1 - \sum_{0\le i\le n} \mu(e^{<i>})) \\ &\quad + \sum_{1\le i\le n} \mu(e^{<i>})\mathcal{H}(M^i|e^{<i>})\end{aligned}$$

where $\mu(e^{<i>})$ presented the probability distribution of iteration i.

For the case of a loop with collisions, the leakage was defined as the leakage of a loop with no collisions minus the weighted sum of the entropies of the collisions. Formally, it was defined as:

$$\lim_{n\to\infty} W'(e,M)_n - \sum H(C(W'(e,M)))$$

The *rate* of leakage was thus given as

$$\lim_{n\to\infty, \mu(e^{<0>})\neq 0} \frac{W(e,M)_n}{n}$$

i.e. the rate of terminating loops was the absolute leakage divided by the number of iterations.

This work provided a new approach to measure security properties of programs via both absolute and rate of leakage. However, its application is difficult to automate. In comparison, Clark, Hunt, and Malacaria [14,16,17] presented a quantitative analysis based on the connection between the notion of leakage of information in a system based on program analysis and concepts from information theory with a given input distribution, which was overly conservative but was a more complete system and worked for all the situations.

Following this work by Malacaria, Chen and Malacaria [13] presented a mechanism for quantitative leakage analysis for multi-threaded programs. The idea was based on program transformation: the multi-threaded program with a probabilistic scheduler was transformed into a single-threaded program with probabilistic operators, and then leakage of the single-threaded program was computed by applying the formula discussed above.

3.5.5 Min-Entropy as the Measure of Information Flow

Smith [60] introduced a new leakage definition based on the concept of vulnerability, which measures uncertainty by applying Rényi's min-entropy rather than Shannon entropy. The threat model considered in this work is focused on the probabilities that the attacker can guess the value of confidential input (H) correctly *in one try*. Consider example programs discussed in the paper:

(i) `if (h mod 8 = 0) then l := h else l := 1`

(ii) `l := h & 0^{7k-1}1^{k+1}`

where h denotes a $8k$-bit high-level variable with a uniform distribution, and l denotes a low-level variable. According to the consensus definition based on conditional mutual information between high input and low output given low input, the leakage of the first program is computed by

$$\mathcal{L} = \mathcal{H}(l) = \frac{7}{8}\log_2\frac{8}{7} + 2^{8k-3}2^{-8k}\log_2 2^{8k} \doteq k + 0.169$$

and the leakage of the second program is computed by

$$\mathcal{L} = \mathcal{H}(l) = k + 1$$

According to Smith, such computation does a poor job of measuring the threat since the second one is actually worse than the first one. The author argued that the consensus definitions do not provide a good security guarantee about the probability that h could be guessed with respect this threat model. Due to these limitations, the author proposed an alternative definition based on the concept of *vulnerability* for quantitative information flow. The vulnerability denoted by $V(X)$ is simply defined as the maximum probability of the values of X: $V(X) = \mathtt{max}_{\mathtt{x}\in\mathtt{X}}\mathtt{P}[\mathtt{X} = \mathtt{x}]$. The vulnerability is thus the worst-case probability that the value of X can be guessed correctly in one try. By using the concept of vulnerability, a new measure based on *min-entropy* denoted by $\mathcal{H}_\infty$, and defined by $\mathcal{H}_\infty(X) = -\log_2 V(X)$, is introduced. The following definitions are proposed:

- initial uncertainty $= \mathcal{H}_\infty(H)$;
- remaining uncertainty $= \mathcal{H}_\infty(H|L)$;
- information leaked $= \mathcal{H}_\infty(H)\mathcal{H}_\infty(H|L)$

Such a definition can be applied to measure the maximum information flow that an attacker could correctly guess the value of X in one try with respect to the threat model considered above. Specifically, for the case that the program is deterministic and H is uniformly distributed, the information leaked is $\mathcal{H}_\infty(L) = \log_2 |L|$, where $|L|$ denotes the *number* of equivalent classes of random variable L. However, we argue that there are problems with this definition. It is known that min-entropy is always less than or equal to Shannon entropy. This disagrees with the discussion by the author: the leakage of the first program is more than the leakage computed by using Shannon's entropy.

3.6 QIF in Communicating Programs

Real world programs normally allow input/output, and behave as a reactive system. It is important to consider a quantitative analysis over reactive systems in the computational world. There have been several attempts on probabilistic and concurrent systems: Di Pierro, Hankin and Wiklicky [24] gave a definition of probabilistic measures on flows in a probabilistic concurrent constraint system where the interference came via probabilistic operators. Gavin Lowe [37] measured information flow in CSP by counting refusals. Aldini and Di Pierro [3] introduced a method of quantifying information flow on a probabilistic process system. The method is based on a process similarity relation with regard to an approximation of weak bisimulation of CCS. Backes [5] gave a definition for measuring the quantity

of information flow within interactive settings by measuring the distance between the different behaviours of a high user from a low user's views. Boreale [9] studied quantitative models of information leakage in a process calculus by applying an information theoretic framework.

3.6.1 Non-Interference for Interactive Programs

O'Neil, Clarkson, and Chong [53] studied the information flow security conditions in a simple imperative programming language including input-output interactive behaviours. A notion of non-interference in terms of strategies was introduced. The strategies were designed to model the behaviours of users, *i.e.* how users chose inputs due to their observations. Such strategies-based semantic security was based on Wittbold and Johonson's earlier ideas of non-deducibility on strategies. The basic idea is to guarantee that no confidential information flowed from high confidential users to low-confidential users.

The system model considered situations in which users interacted with programs with channels. Input-output events occurred on such channels. For the purpose of studying security properties, the channels and users were divided into H (high) and L (low) as usual. High users interacted with high channels and low users interacted with low channels. The interaction security conditions were as follows: low users can not observe high's input-outputs; high users can not input on low channels; high users may observe low input-outputs directly. The program was considered secure (satisfied non-interference) if low users can not deduce anything about high behaviour given low observations. Observations here can be considered as a sequence of input-output events, *i.e.* a trace of (input, output). Therefore, to guarantee the program satisfied the security condition, the users should not be able to tell which strategy is better or worse given the observations. Formally, a user strategy for a channel is a function from traces of events on this channel to inputs. The program execution was modelled as a tuple: a command c, a state σ (maps from program variables to integer values), a trace event t, and a joint strategy ω which specified a user strategy for each channel and mapped from channel names to user strategies. Non-interference was thus formalised due to the observations with trace restriction on low: if $t \upharpoonright L = t' \upharpoonright L$, traces t and t' had the same subsequence of low events. Furthermore, command c satisfied non-interference if for all executions $m = (c, \sigma, \epsilon, \omega)$ and $m' = (c, \sigma, \epsilon, \omega')$ such that $\omega(L) = \omega'(L)$, and for all traces t such that m produces trace t: there exists t' such that $t \upharpoonright L = t' \upharpoonright L$ and m' produces trace t'. The paper therefore introduced a formal semantic security condition for *interactive* imperative programs.

Clark and Hunt [18] considered a wider class of strategies than those of O'Neil, Clarkson, and Chong [53] discussed above. A notion of Input-Output Labelled Transition System (IOLTS) was introduced to model the user's input behaviours as strategies and to model interactive programming language. The IOLTS was a labelled transition system with a set of labels consisting of inputs, outputs, and internals. The security properties were thus expressed at the level of input-output traces and the strategies. The definition of non-interference in this paper was considered as a generalisation of the definition introduced in [53]: a state of an IOLTS was simply noninterfering for a set of strategies if it was non-interfering for the set

of all strategies. Such notions of non-interference were also shown to apply to a simple deterministic interactive programming language.

The problem of notions of security condition (non-interference) for deterministic interactive programs were considered in the papers [53,18] discussed above. However, they did not consider a method of measuring the information flow caused by interference. There is still a long way to go by way of quantified information flow for security.

3.6.2 Approximate Non-Interference by Abstract Interpretation in PCCP

In [24,27], Di Pierro, Hankin and Wiklicky defined probabilistic measures on flows in a probabilistic concurrent constraint setting where the interference came via probabilistic operators and more recently in distributed systems [28]. They also provided a probability sensitive program analysis that ensured precise and approximated non-interference for a probabilistic constraint programming calculus. They used this to derive a quantitative measure of the similarity between agents written in Probabilistic Concurrent Constraint Programming (PCCP) language. Finally, they presented an abstract interpretation to approximate the confinement of processes and showed the correctness of the abstract analysis with respect to the concrete semantics. However, they did not measure quantities of information.

This work used a particular process algebraic framework with probabilistic constructs, PCCP, to deal with probabilistic information flows, in which probability was introduced via a probabilistic choice made by *agents* and a form of probabilistic parallelism maintained by a probabilistic scheduling, and all processes shared a common *store*. The operational semantics of PCCP was given by a probabilistic transition system $(\langle A, d\rangle, \longrightarrow_p)$. The set of pairs $\langle A, d\rangle$ described the state of the system at a certain moment, where A was the *agent* to be executed, d was the *common store*, and p in the transition relation suggested the probability of the transition to happen. The observable $\mathcal{O}(A, d)$ of a PCCP agent A in store d was defined as a probability distribution on constraints, showing the probability established by the agent at each constraint.

The observable of a PCCP agent corresponded to a distribution on the constraint system $\mathcal{C}$, which was a vector in the space $\mathcal{V}(C)$. The difference between two observables was then measured using a supremum *norm*, and defined as the maximum over the absolute value of the difference between each constraint in the two observable. Thus, two agents A and B were *probabilistic identity confined* when the difference between their observables is zero in any context, i.e., for all agent S, and scheduling probabilities p and $q = 1 - p$:

$$\mathcal{O}(p : A \| q : S) = \mathcal{O}(p : B \| q : S)$$

This definition of *identity confinement* was based on the equivalence of the agents' behaviour. The authors then gave the identity definition of confinement on some similarity notion. The definition of *approximate confinement* was based on the idea of measuring how much the behaviour of two agents differed in a certain context. The intuition behind this was to look at how much difference there was between two agents' behaviours rather than qualitatively asserting whether they were identical or not. Two agents A and B were *approximate identity confined* (ε-confined) when

this difference was not less than zero: $\varepsilon \geq 0$. The authors also pointed out that the approximate and probabilistic confinement agreed with each other if both agents were ε-confined with $\varepsilon(p) = 0$ for all scheduling probability p.

The authors then developed a collecting semantics to calculate an exact ε to measure the confinement of a set of agents. They defined an additional agent as a *spy* which could interact with a system a finite number of times, determine the underlying probability distributions of the confined processes, and then determine how possible it is to distinguish the confined processes correctly via hypothesis testing. This *spy* thus provided enough information about a set of agents to compute their ε-confinement.

Finally, they defined an estimate of this probability by introducing an abstract semantics to calculate an approximation of the ε (bounds of the $m's$) in a more abstract way. The basic idea was to compute the range of possible observable and compare them to get an approximation of the ε, and then formulate the abstract semantics to get the bounds of the attackers.

However, the approach of approximate non-interference presented in this paper is based on the specific probabilistic declarative language PCCP. It seems difficult to automate in other frameworks.

3.6.3 Lowe's Information Flow Measurement in CSP

Gavin Lowe [37,38] measured information flow in CSP by counting refusals. He devised a formal definition of information quantity transmitted from a high level user to a low level user in a computing system. The definition was based on the number of different behaviours of High that can be distinguished from Low's point of view. According to Lowe, if there are n such distinguishable behaviours, then $\log_2 n$ bits of information were leaked to Low. Like other quantitative definitions, Lowe's definition was based on Shannon's information theory. However, unlike other models, this definition was sensitive to non-determinism.

Lowe chose discrete-time CSP as the modeling language for the consideration of the treatment of *non-determinism* and *information flow rate*. A *refusal trace* was introduced to represent the result of an experiment in which a sequence of refusals and performed events were recorded. Hence the *discrete-time refusal traces model* of CSP was presented to denote a process P by its refusal traces, written as $\mathcal{R}[\![P]\!]$. The details about the syntax, semantics, and the refusal traces of this discrete-time CSP were given in [55].

Lowe presented several example processes which had information leakage, and he quantified the information flow in each case. High and Low built the environment of the process. The models of High and Low were made so that they could observe the high and low level events respectively. Low could observe high level events in one time unit and then performed one of his own events. In this way, Low could infer whether or not the high level event is performed by observing the result of his own event. These examples gave some intuition of information flow quantity. In order to formalise it, Lowe defined information leakage in which High and Low interacted with the system to cause information to pass from High to Low. First, Lowe considered the way in which Low interacted with the systems. The Low user performed a *test*, *i.e.* the strategy of interacting with the system by repeatedly

offering sets of events to the system, and then observed the *results* with sequences of acceptances and refusals via events which represented the successful completion of a test. The *results* were considered as a set for the consideration of the possibility of non-determinism. Second, he considered a High user's communication strategy in which High can only block events of Low by preempting them with one of his own events rather than preventing any Low events. Lowe also defined the condition for processes to capture this requirement when defining strategies for High. Lowe then introduced the definition of information flow quantity as follows. Consider process P, suppose High strategies were given by $\mathcal{Q}$, Low's possible views of the system were represented by: $\{(P \overset{\|}{_\Sigma} \mathcal{Q}(k) \setminus H | k \in dom\ \mathcal{Q}\}$, and a particular test T was defined to distinguish all of these processes. The results were then obtained by the testing process T when interacting with these processes. Lowe used $ok(P, \mathcal{Q}, T)$ to describe a situation in which the strategies of High and Low were compatible, *i.e.* the result obtained by Low was exactly the value sent by High given strategies $\mathcal{Q}(k)$, T, and some refinement of P. Then the information flow quantity of the system P was defined as the maximum number of such cases satisfied $ok(P, \mathcal{Q}, T)$. However, it seems not easy to automate the procedure for computation of information quantity in a real system.

Specifically, Lowe gave a set of results about his model and definitions. First, he showed us that the information flow quantity would be 2 if Low user could tell the difference between the systems caused by two high level processes. Second, he pointed out that a process P had information flow quantity of 1 whenever High cannot change the way the process appears to Low given any refinements, *i.e.* no information flow in such case.

Lowe also defined the amount of information flow within some finite time period in the timed model, and hence introduced the *rate* of information flow. The *rate* of information flow was defined as:

$$r(P) \overset{\text{def}}{=} \lim_{t \to \infty} \frac{\log_2(IFQ_t(P))}{t}$$

where $IFQ_t(P)$ was the information flow quantity of P during the time t, *i.e.* the number of behaviours of H that L can distinguish in t time units. The rate of information flow was measured by the amount of information in bits within per time unit in this definition. However it seems difficult to distinguish the time unit for Low in some situations.

3.6.4 Approximation on Similarity of CCS

Aldini and Di Pierro [3] introduced a method of quantifying information flow on a probabilistic process system, The method is based on a process similarity relation with regard to an approximation of weak bisimulation of CCS.

The probabilistic model adopted in this paper was considered as allowing reactive behaviours. In particular, processes can react to the environment with a probabilistic choice on a set of inputs. The probabilistic framework was based on a probabilistic calculus [10] with non-determinism [1] which derived from a simple non-deterministic process algebra. Specifically, in order to focus on the continuous interactive behaviours, the visible actions were considered having two basic types: input and output. Unobservable actions were used to perform internal computa-

tions. Furthermore, the algebraic operators were extended to incorporate probabilities which built up a distribution due to that operator. The probabilities here can be viewed as a parameter of a distribution which guides the action of choices. An input action was treated as a reactive action and guarded by the environment. Once an action was triggered by the environment, the system started to move and reacted to the chosen action, a particular reactive action of that type was executed on the basis of a probability distribution. On the other hand, the output action was considered as a *generative* action in the model. The system automatically decided which output action should be produced to the environment and how to behave after such an event. This was also on the basis of a probability distribution. The operational semantics of the probabilistic process calculus was described as a transition graph represented by linear operators. The distance between processes was defined by *operator norms* which described the maximum difference of normalised vectors.

The measurement of information leakage due to the system discussed above was defined by calculating the maximum difference between the probability distribution based transitions observed by low level users for the cases of interacting and non-interacting with the high user. The quantity of such maximum difference was based on the concept of *behavioural distance* between process by approximating the probabilistic process equivalence introduced in [26,25]. This work had been applied to provide a quantitative security analysis of probabilistic protocols in [2].

3.6.5 Distance Measures in Reactive Processes

Backes [5] addressed the concern of quantifying information flow within the environment with reactive settings and allowing interactive behaviours. The author introduced a definition for measuring the quantity of information flow within interactive settings by measuring the distance between different behaviours of high user from low user's views.

The model considered in this paper was an asynchronous probabilistic execution model based on reactive simulatability [54,6]. In order to take the computational aspects required for cryptography into account, the model incorporated distributed scheduling which can be used to provide universal composability properties. The programs or protocols were executed by a set of interacting machines which started to move and produced relevant reactions if given certain inputs. Each *step* of the execution of the machine was defined as a tuple with the name of the active machine, its input, output, and old and new local state. A sequence of steps builds up a *run*. In the following, the definition of *view* is introduced to capture the different behaviours of the specific machines. Intuitively, the *view* of a (set of) machine(s) is defined as a restricted run containing only the steps of this (these) machine(s).

The key idea of the information flow quantity is to study the different behaviours of a high confidential user that result in different *views* for the low confidential user. The maximum distance between these different views provides a sound measure of the information flow quantity in the worst-case. The paper also showed such definition and structure satisfied the non-interference requirement. This meets our intuition: there is no information leaked to outside if there is no difference between high user's behaviours from low user's point of view. However, the paper

did not suggest a suitable metric for the distance between user views or probability distributions due to any specific attack models.

3.6.6 Boreale's Quantitative Models

Boreale [9] studied the quantitative models of information leakage in process calculus. He presented two quantitative models of information leakage in the process calculus: one for measuring absolute leakage and another for measuring the rate at which information was leaked. The *absolute leakage* measured in bits, presented the leakage of zero precisely when it satisfied secrecy. The *rate of leakage*, measured in bits per action, presented the maximum information extracted by repeated experiments coincided with the absolute leakage of the process.

The model for absolute leakage assumed that the attacker knew the process code $P(x)$ and controlled P's execution, up to the point that he can run at no cost as many copies of $P(x)$ as desired, corresponding to different instances of X. The definition of absolute leakage was based on conditional mutual information. The secure information was defined as a random variable, say X. The uncertainty of X was measured by Shannon entropy $\mathcal{H}(X)$, the conditional entropy $\mathcal{H}(X|Y)$ measures the uncertainty about X given that Y is known, expressed in bits. The process P, depending on both X and Y, induced a random variable $Z = P(X, Y)$ which took the observable output values. Then the conditional entropy $\mathcal{H}(X|Y, Z)$ quantified the uncertainty on X left after observing both Y and Z. Hence the absolute leakage was quantified by the difference $\mathcal{L}(P; X|Y) \stackrel{\text{def}}{=} \mathcal{I}(X; Z|Y) = \mathcal{H}(X|Y) - \mathcal{H}(X|Y, Z)$.

By adapting the testing equivalence framework and introducing a notion of *cost*, Boreale presented the model for the rate of leakage. Assume that the attacker can only conduct upon P repeated experiments $E1, E2, ...$ each returning a binary answer, success or failure. The model defined the rate at which P leaked information in terms of the maximum number of bits of information per visible action conveyed by an experiment on P. Hence the maximum number of bits per visible action conveyed by P was the rate at which it leaked information. The PIN-checking example gave a symbolic scenario, the attacker's action upon repeated experiments, each yielding a binary answer - success or failure. The interactive actions between the attacker and the program caused the information leaked. The amount of information leaked per interactive visible action plots the speed of information leakage during the whole process. The experiment E here was viewed as a closed process without using recursive definitions, it may succeed or not when composed in parallel with P. The notion of rate involved a ratio between the quantity of information and the *cost* of E. The *cost* of each single experiment E was defined as

$$|E| = \Sigma|s| : s \textit{ is successful for } E$$

The rate of P relative to X was then defined as

$$\mathcal{R}(P; X) = sup\frac{\mathcal{H}(E^*)}{|E|} = sup\frac{\mathcal{H}(p_s)}{|s|}$$

In each experiment E, each trace leading to success should be thought as a trial from the attacker. Each trial required a distinct execution of E, so the overall cost was actually the sum of the cost of individual trials. This was partly motivated by the interpretation of non-determinism in processes. For example, the cost of a

PIN-Checking process $\Sigma_{i\in 1...10} try(i)$ was ten, because it assumed that observing all possible outcomes of P-E took ten distinct executions of E, each of which cost one.

The relationship between the above two models showed that absolute leakage L coincides with the maximum information that can be extracted by repeated experiments, and this costs at least L/R, where R was the rate of P. It establishes that $\mathcal{H}(Z) = max_E I(X; E^*)$ is the maximal information that can be extracted by repeated binary experiments, and provided a lower bound on the cost necessary to extract this information in terms of the rate of P, thus providing a justification for the name *rate*. Due to the problem of iteration on processes, this paper defined sequential composition, and gave upper bounds for the rate of compound processes in terms of the component expressions. Boreale showed that under certain conditions, iteration preserved rate and thus gave another proof for the definition of rate. A weakness of the ratio formulation was that it was difficult to apply to recursive processes.

3.6.7 Jensen-shannon divergence as a measure of information flow in reactive processes

Mu [51,47] presented an approach for measuring information flow in reactive process descriptions with input, output, and probabilistic non-deterministic behaviours. The basic concept in this work was that the quantity of information flow was calculated by looking at the different behaviours of a high user from a low user's observations. A *metric space* was built upon the process trees, and the information flow was measured via metrics.

First, The reactive system was modelled by using *Probabilistic Labelled Transition System (PLTS)* as a basis. The probabilistic model was considered to be reactive in the sense that the system can react to the environment if fed with a parameterised high input action equipped with a probability distribution. The *observation* was defined to record the history traces of behaviours from the view of low users in a way of distributions. By executing a set of low level input-output actions triggered by the high input action, the system produced a set of probabilistic observation trees to the outside as a reaction.

Next, a probabilistic input-output security process algebra was presented, which was compatible with the semantics given by a PLTS. A set of traces can be extracted form the process tree built by the semantics structure. The observation was defined as the sum of the low projection on such traces. Under repeated observation on traces, the information on the projection of the high inputs from the traces can be deduced. A equivalence relation called probabilistic low bisimulation was also introduced. Such a coarser equivalence was considered more satisfactory as it will only distinguish processes that can be distinguished by external low level traces.

In order to capture the information flow from multiple parameterised high input actions during the execution, the author then introduced a transformation on the process tree obtained by the semantics. The transformation idea was based on the definition of interaction units (steps). Each interaction unit was started by a parameterised high input action $?H_i$, and stopped when another parameterised high input action $?H_{i+1}$ or a state of termination $\perp$ occurred. During each interaction unit, low users are communicating with the system via low level input-output

actions (high level output and internal actions are invisible to the outside). The transformation was built to capture the different behaviours of the process given a parameterised high input in each interaction unit. Intuitively, interaction units on the original process tree T were transformed in a recursive way:

$$T_0 = \bot,\ T_1 = ?H_0 \prec T_0,\ T_2 = ?H_1 \prec T_1,\ \ldots,\ T_{n+1} = ?H_{n+1} \prec T_n$$

where T_i denoted the transformation tree up to i^{th} interaction step, and $?H_i$ denoted the parameterised high input action triggered the i^{th} interaction step. The observation $\mathcal{O}_{i+1}$ on the transformation tree T_{i+1} due to interaction unit starty by $?H_i$ was the sum of the low projection of all possible traces on the subtrees T^j_{i+1}. For $1 \leq j \leq m$, T^j_{i+1} denoted the subtree started by an atomic action $?h_j$ in $?H_i$ with weight w_j: $?H_i = \{?h_j \rightarrow w_j | 1 \leq j \leq m\}$, m was the size of $?H_i$.

The author then introduced a method to provide a quantitative definition of information flow for reactive processes using metric spaces on the process domain. A *metric space* was built upon the transformation tree, and the information flow was measured via metrics. The metric chosen here was the square root of Jensen-Shannon divergence and corresponds to the framework of information theory. We therefore can compute the metrics on the transformation trees due to interaction units and merge them together as an leakage upper bound.

One weakness of this method is that the observer is very strong. The observer is able to observe all the possible actions of the system. Another weakness of this approach is that the leakage is a function of the semantics, in general, it is not executable due to the tractability problem.

3.7 Quantitative Information Flow on Side Channel Attacks

Köpf and Basin [34] developed a quantitative model for assessing a system's vulnerability to adaptive side-channel attacks. The model was devised to address the problem of determining the information flow quantity released to an attacker on side-channel attacks. The model considered attacking situations in which the attacker can make physical observations about a cryptographic function's execution (or a program's) but without the ability to access the secure inputs of the program. The attack can collect information and deduce possible values of the secret inputs via a sequence of query-response steps. The basic technique of the model was based on attack strategies which kept a log on the representations of the adaptive decisions made by the attacker during the attacks. A tree was built to formalise the attacker's decision with respect to his possible observations. In this tree, each attack step was represented by a node: the parent node represented the attacker's decision, and the children nodes formed a partition of its parent node. In this way, the decision of the attacker was refined gradually. The tree therefore described the history of the attacker's expected uncertainty about the secret after performing a set of side-channel attack steps following a given strategy. Intuitively, the uncertainty of the secret was reduced as the partition is refined. Such attack strategies were combined with entropy based measures. Information-theoretic entropy was then applied to measure the remaining uncertainty. The measures considered included Shannon's entropy [59], Massey's guessing entropy [40], and Pliam's marginal guesswork [56]. By evaluating attack strategies using such entropy measures, we obtained bounds

on the secret information that an attacker can achieve via a number of attack steps. The algorithms for computing the leakage bounds for a concrete system relied on an enumeration of the entire input space, so the weakness of this method was that it was also hard to apply to large systems.

3.8 Automatic analysis for quantitative information flow of programs

3.8.1 Measuring Maximum Flow via Dynamic Analysis

McCamant and Ernst [41] investigated techniques for quantifying information flow revealed by complex programs by building flow graphs and considering the weight of the maximum flow over it. It is known that precision, soundness, and feasibility are all important requirements of a good quantitative analysis too. However, most previous work in this area was focused on very small flows. It is useful to devise practical measurement techniques to be applied to real programing languages and to operate on large software without consuming too much time and resources. The paper proposed to develop a new effective method to measure the information leakage by observing execution of large software via *dynamic* analysis.

The basic steps of the algorithm applied in this work was sketched as follows. The first step is to build a flow graph which represents the possible secure flows through a program execution. The graph is devised to construct possible channels for information flow during the execution by using a dynamic shadow value analysis similar to a tainting analysis. Edges of the graph denoted the values incorporating with capacities showing the number of bits of the value held. Nodes represented basic operations on these values. The second step of the method was to combine such graphs from multiple executions of the same program into one single graph. The graphs were combined by merging all of the edges that correspond to a particular static program location and adding their capacities into it. The third step was to compute the maximum flow. The key idea here was to measure the information flow as a kind of network flow capacity. To compute a maximum flow, each edge was labelled with a bound on the amount of information it can convey. These bounds were obtained by counting the bits in each data value which might contain secret information by simultaneously performing a dynamic bit-width analysis. The analysis was actually implemented as dynamic tainting but at the level of bits. The amount of secret information leakage through a value was thus bounded by the number of its bits that were marked as secret.

This paper presented a different approach for determining information flow quantity about a program's confidential inputs can be deduced by its public outputs, without applying information theory and static analysis. The method can be used to provide an automatic *dynamic* quantitative analysis but it can not compute upper bounds on the information that a program can release as a static analysis can achieve.

3.8.2 Equivalence relations and QIF

Backes, Köpf, and Rybalchenko [7] presented an automatic method for information flow analysis that discovered what information was leaked and how much. The leaked information was considered in the form of an equivalence relation and was

represented as a logical assertion over program variables. The measurement procedure computed the number of discovered equivalence classes and their sizes.

First of all, programs were modelled as a transition system: (S, T, I, F), where S was a set of program states, T was a finite set of transitions, I was a set of initial states, F was a set of final states, and $I, F \subseteq S$. Specifically, the initial and final states are partitioned into low and high due to their security level: $I = I_{h_i} \times I_{l_i}$ and $F = F_{h_o} \times F_{l_o}$. The environment considered the situations in which an observer can see the program, the low components of the initial and final states, but cannot see the high components of the initial and final states.

Second, an equivalence relation $\mathcal{R}$ over I_{h_i}: $\mathcal{R} \subseteq I_{h_i} \times I_{h_i}$ was introduced in order to characterise the information leaked an an observer. Let $=_{h_i}$ denote the identity relation, *i.e.* the observer knew the total information on I_{h_i} under such relation. Let All_{h_i} denote the largest equivalence relation, *i.e.* the observer knew nothing about I_{h_i} under such relation. An equivalence relation $\mathcal{R}$, such that $=_{h_i} \leq \mathcal{R} \leq All_{h_i}$ expressed a partial knowledge on the high components in the initial states by observing the program's outputs.

Suppose the attacker can choose the low component of the initial states and start an experiment: $E \subseteq I_{l_o}$. Given a program and a set of experiments E, if there was a pair of computations induced by two sequences of program transitions π and η that start with $\mathcal{R}$-equivalent high components and equal low components in E, and lead to final states with different low components, then there was an information leak with respect to an equivalence relation $\mathcal{R}$ according to the authors. The notation $Confine_P(\mathcal{R}, E)$ was then introduced to confine the pair $\mathcal{R}$ and E such that the relation $\mathcal{R}$ provided a leakage upper bound when P was run on the experiments E. Intuitively, the largest equivalence relation $\mathcal{R}$ with $Confine_P(\mathcal{R}, E)$ was the most precise characterisation of the leaked information, denoted by $\approx_E = \bigcup\{\mathcal{R} | Confine_P(\mathcal{R}, E)\}$.

In the next step, the authors applied information theory to measure the quantity of equivalence relations $\mathcal{R}$ with $Confine_P(\mathcal{R}, E)$:

- count the size of each of the equivalence classes
- compute the attacker's initial uncertainty about the high component in the initial state: $\mathcal{H}(X)$, where X denotes the random variable of the high component in the initial state, and $\mathcal{H}$ denotes entropy as standard
- compute the final uncertainty after observing the output of the program: $\mathcal{H}(X|Y_{\approx})$, where $Y_{\approx}$ denotes the random variable of the equivalence relations $\mathcal{R}$ with $Confine_P(\mathcal{R}, E)$
- compute the leakage as the initial uncertainty minus the final uncertainty about high input

This paper proposed a method to automatically discover what information was leaked and compute the quantity of it by defining the equivalence classes on high input and computing the information entropy on the equivalence relations. The weakness of this method is that the observer is also very strong: he knows everything about the program and therefore the corresponding transition system.

3.8.3 Automate information flow analysis via abstract probabilistic semantics
Mu and Clark [49,52,50,48] developped an automatic analysis for measuring information flow within software systems. The basic scenario considered here was a program that receives high inputs and produces low observable behaviours. They consider ed that the high inputs satisfied a publicly-known probability distribution. By observing the behaviours of public outputs and the text of the program, an outsider can deduce some information about the high inputs. They then quantified the amount of information in high inputs leaked to public observers. The overall approach was static analysis: the analyser measured the information flow over all executions of a program given probability distribution based inputs. To allow automatic analysis, they applied Kozen's probabilistic semantics [35,36] as a distribution transformer and computed the exact leakage over it in the framework of information theory. To address the time complexity problem, they next applied Monniaux's [46] framework of abstract interpretation on Kozen's probabilistic semantics as a basis to introduce an approximation of the concrete leakage analysis.

In [49,52], they first presented an automatic analysis for measuring information flow within programs in a simple imperative while language. They quantified exact leakage in terms of information theory and incorporated this computation into Kozen's probabilistic semantics for programs [35,36]. One of Kozen's semantics for a probabilistic `while`-language is given in terms of interpreting programs as partial measurable functions on a measurable space. Their analysis provided information flow measurement for simple, imperative programs given secure inputs under any probability distribution. Specifically, while-loops were handled by applying the *entropy of generalised distributions* and the *entropy of partitions* in order to provide an analysis with respect to the time of observation for observers who can observe the iterations of the loops. They showed that this method calculated the same quantity as Malacaria's method [39], thereby providing correctness for his method relative to Kozen's semantics but in distinction to his approach, there was *no need for any initial (human) analysis* of loop behaviour and it was completely *automatic*. This approach worked well for small programs with small state spaces. However, as the scale increases the calculation suffered from a form of state space explosion and the time complexity growed.

To address the tractability problem, in [50,48], the authors suggested a method to scale up the programs and state spaces that can be handled albeit at the cost of replacing an exact result with an upper bound. To do this, they introduced abstraction on the state space via interval-based partitions, adapting an abstract interpretation framework introduced by Monniaux [46]. The user can define the partitions and the coarser the partitions, the coarser the resulting upper bound. In this work, they defined the abstract interpretation, showed its soundness, and proved that the result of an abstract computation was always an upper bound on the true leakage.

3.8.4 Automated Quantitative Analysis on an Algebraic Structure
Heusser and Malacaria [32] suggested a method to automatically calculate the information leakage of an algebraic structure described programs. The basic idea was that they defined the automatic interpretation of programs in the lattice of infor-

mation and then applied the related information theoretical computations on the lattice. The observations on a program were interpreted as equivalence relations on states and therefore as random variables in the lattice of information.

The method was based on two reachability analyses obtained by two algorithms. The first analysis produced a set of inputs S_{input} which lead to different outputs by executing the program by using SAT-based fixed point computation [12]. They developped a tool, AQuA (Automated Quantitative Analysis), which can be used to calculate the partition of input states $\pi(P)$ given a C program P without user interaction. Given S_{input}, in the next step, the second analysis counted the set of all inputs which led to the same output, *i.e.* the size of the equivalence classes using model counting. These two analyses were used to discover the partition of the input space according to the outputs of a program. Once the partition was calculated, we can apply a measure (*e.g.* Shannon entropy) to compute the difference between the uncertainty about the secret before and after observing the output of the program (*e.g* the conditional mutual information between the program and the secret given the low input).

Furthermore, as a case study, They quantified the information leaked by a database query. They modelled database queries as programs, and applied their analyser to calculate the partition of states and quantify the leakage of the data base queries.

3.8.5 Statistical Measurement of Information Leakage

Chatzikokolakis, Chothia and Guha [11] developed a tool to automatically calculate the information leakage from trial runs of a system in a statistical way.

The basic situation considered in this paper was as usual: secret user input data to the information -theoretic channel, and publicly observable actions output to the attacker. Mutual information was applied to measure the amount of the information that was sent across the channel, *i.e.* the information leakage about the secret inputs. Capacity was defined as the worst case leakage.

To measure the information leakage, the following issues were addressed in the paper:

- Find a probability transition matrix that reflects the system under test.
- Produce the conditional probabilities of the outputs given the inputs.
- Calculate capacity from this estimated matrix.

First, we need to define the inputs and outputs from samples. Then we can run trials of the system for each of the inputs and record the observable outcomes. These observations were used to construct an estimated matrix. Therefore, by running the system under test, an estimated probability transition matrix was built up. In the next step, the Blahut-Arimoto algorithm [4,8] was applied to this matrix to estimate the capacity and information leakage of the system. We can calculate the capacity to find the input distribution that maximise the information leakage and therefore calculate the worst-case scenario.

Furthermore, they calculated bound on the error of the estimate in order to measure how close the estimate of capcity is to the real value. They suggested two method to find such a bound. First, they estimated the error in each of the

matrix entries and then calculated the maximum effect on the final result by all of these errors. Another method was that we calculated the distribution of the result values in terms of the value to estimate. Given the input distribution and sample matrix to find the outputs, we computed the mean and variance of sampled mutual information to provide the confidence bound on the true capacity in terms of the estimated value. The calculation of the variance of the estimate value was used to judge whether the systems were too complex to successfully analyse or not.

4 Bridging Language-based and Process Calculi Security

We have presented a brief review of key papers in both language-based and process calculus-based quantified information flow analysis in the field of computer security. Language-based information security deals with preventing secret data from being leaked through the execution of program. Process calculus-based information security concerns how to prevent secret events from being revealed through the execution of communicating processes. In common with each other, non-interference is considered as the baseline security policy for both language-based and process calculi-based information security. Focardi, Rossi, and Sabelfeld [29] showed there is a link between a language-based specification of security and process-algebraic framework for security properties. They developed a formal connection between the language-based and process calculus security definition of information security. Specifically, via the following steps, a connection between a timing-sensitive properties for a simple imperative language and persistent security properties for a CCS-like process calculus was built:

- Define a simple imperative programming language and its semantics, and build it into the standard labelled transition system semantics of process calculi in which the labelled transitions are with regard to the observable actions of reading and writing memory.
- Define a CCS-like calculus and its operational semantics.
- Define a security calculus using bisimulation. Based on the semantics of the imperative language, the low level observations in the imperative language is formalised in terms of the bisimulation relation.
- Define a translation from an imperative programming language to a process algebra (calculus). The translation is based on a correspondence between *computation steps* in the imperative language and the *actions* of processes. Such a relation allows us to identify what the program security property corresponds to in the process-calculus world, which turns out to be the well understood property of persistent bisimulation based on non-deducibility on composition.
- Prove some properties of translation: the two security conditions match with each other via the translation.

In conclusion, the formal connection established above showed that a timing-sensitive security definition corresponds to persistent bisimulation-based non-deducibility on composition (P-BNDC), and also identified that process calculus-

based definition corresponds to compositional timing-sensitive language-based security. By considering a combination of programming language-based definitions and process-algebraic ones, it is possible to specify the security issues of computing systems in a more flexible way.

5 Conclusions and future works

Quantitative information flow has recently become an active research topic. There has been significant activity around the question of how to measure the amount of information flow caused by interference between variables by using information theory. This report presents a detailed survey in this area from 1980's to date. It is clear that this problem is far from being solved. Although quantitative analysis is becoming popular, the numerous information-theoretic models seems difficult to manipulate and reason about in static analysis. It is still a long-term research topic in the security community.

The main contribution of this work is a survey of most of the existing relevant works related to quantitative information flow for security, providing with an overall view on what has been done in the field and which are the available metrics that can help them in making decisions in the further development. This work will also help researchers to get a more comprehensive view of the direction that work in information flow measurement is taking.

By analyzing the related work and the execution mechanism of QIF, we think there are several meaningful research directions on measurement of information flow for security.

- A both automatic and precise leakage analysis tool for programming language is required. To improve the quality of analysis, time at some suitable level of abstraction is needed to account for. To automate the analysis, probability distribution sensitive semantics and the relative transformation functions need to be developed. Furthermore, various forms of partial probabilistic bisimulations on behaviours may be needed to be defined.
- QIF models with regard to different security policies and attack models need to be developed, e.g., timing, probabilistic systems, termination behaviours, etc. The classic confidentiality property *non-interference* is a guarantee that no secure information can be obtained by observing a program's public outputs, however it is rarely satisfied by real programs. How to distinguish acceptable flows from unacceptable flows need to be concerned in information flow security.
- Most information flow measurement techniques so far only applied to simple program languages and examples. There is still a long way to go before being actually used (e.g., due to the need for formal semantics for full-blown languages, and, due to poor runtime performance of the leakage analyzer implementation). It would be good to drive the flow measurement techniques to apply to real programming languages and operate on large programs and without using too much time and computing resources.
- We also need to address the concern of how to build interfaces to other system components to bear on the problems faced by existing software systems.

Such research would be beneficent to the analysis of software applications in security related domains such as the military, banks etc.

References

[1] A. Aldini, M. Bravetti, and R. Gorrieri. A process-algebraic approach for the analysis of probabilistic noninterference. *Journal of Computer Security*, 12(2):191–245, 2004.

[2] A. Aldini and A. Di Pierro. On quantitative analysis of probabilistic protocols, 2004.

[3] A. Aldini and A. Di Pierro. A quantitative approach to noninterference for probabilistic systems, 2004.

[4] S. Arimoto. An algorithm for computing the capacity of arbitrary memoryless channels. *IEEE Trans. on Inform. Theory*, IT-18(1):14–20, 1972.

[5] M. Backes. Quantifying probabilistic information flow in computational reactive systems. In *Proceedings of 10th European Symposium on Research in Computer Security (ESORICS)*, volume 3679 of *Lecture Notes in Computer Science*, pages 336–354. Springer, September 2005.

[6] M. Backes and B. Pfitzmann. Intransitive non-interference for cryptographic purposes. In *Proceedings of 24th IEEE Symposium on Security and Privacy*, pages 140–152, May 2003.

[7] Michael Backes, Boris Köpf, and Andrey Rybalchenko. Automatic discovery and quantification of information leaks. In *Proc. 30th IEEE Symposium on Security and Privacy (S& P '09)*, 2009.

[8] R. E. Blahhut. Computation of channel capacity and rate distortion functions. *IEEE Trans. on Inform. Theory*, IT-18(4):460–473, 1972.

[9] M. Boreale. Quantifying information leakage in process calculi. In *ICALP (2)*, pages 119–131, 2006.

[10] M. Bravetti and A. Aldini. Discrete time generative-reactive probabilistic processes with different advancing speeds. *Theoretical Computer Science*, 290:355–406, 2003.

[11] K. Chatzikokolakis, T. Chothia, and A. Guha. Statistical measurement of information leakage. In *TACAS*, pages 390–404, 2010.

[12] P. Chauhan, E. M. Clarke, and D. Kroening. Using sat based image computation for reachability analysis. Technical Report CMU-CS-03-151, School of Computer Science, Carnegie Mellon University, July 2003.

[13] H. Chen and P. Malacaria. Quantitative analysis of leakage for multi-threaded programs. In *PLAS'07: Proceedings of the 2007 workshop on Programming languages and analysis for security*, pages 31–40, New York, NY, USA, 2007. ACM.

[14] D. Clark, S. Hunt, and P. Malacaria. Quantitative analysis of the leakage of confidential data. In *Electronic Notes in Theoretical Computer Science*, volume 59, Elsevier, 2002.

[15] D. Clark, S. Hunt, and P. Malacaria. Quantified interference for a while language. In *Electronic Notes in Theoretical Computer Science 112*, pages 149–166, Elsevier, 2005.

[16] D. Clark, S. Hunt, and P. Malacaria. Quantitative information flow, relations and polymorphic types. *J. Log. and Comput.*, 15(2):181–199, 2005.

[17] D. Clark, S. Hunt, and P. Malacaria. A static analysis for quantifying information flow in a simple imperative language. *Journal of Computer Security*, 15:321–371, 2007.

[18] David Clark and Sebastian Hunt. Non-interference for deterministic interactive programs. In *Proc. 5th International Workshop on Formal Aspects in Security and Trust (FAST2008)*, Lecture Notes in Computer Science, Malaga, Spain, October 2008. Springer-Verlag.

[19] M. R. Clarkson, A. C. Myers, and F. B. Schneider. Belief in information flow. In *18th IEEE Computer Security Foundations Workshop*, pages 31–45, Aix-en-Provence, France, June 2005.

[20] M. R. Clarkson, A. C. Myers, and F. B. Schneider. Quantifying information flow with beliefs. *Journal of Computer Security*, 2007.

[21] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley Interscience, 1991.

[22] D. E. R. Denning. A lattice model of secure informatin flow. *Communications of the ACM*, 19(5):236–243, May 1976.

[23] D. E. R. Denning. *Cryptography and Data Security*. Addison-Wesley, 1982.

[24] A. Di Pierro, C. Hankin, and H. Wiklicky. Approximate non-interference. In *CSFW*, pages 3–17, 2002.

[25] A. Di Pierro, C. Hankin, and H. Wiklicky. Measuring the confinement of probabilistic systems abstract. In *Proceedings of WITS'03 - 2003 IFIP WG 1.7, ACM SIGPLAN and GI FoMSESS Workshop on Issues in the Theory of Security*, 2003.

[26] A. Di Pierro, C. Hankin, and H. Wiklicky. Quantitative relations and approximate process equivalences. In *CONCUR*, pages 498–512, 2003.

[27] A. Di Pierro, C. Hankin, and H. Wiklicky. Approximate non-interference. *Journal of Computer Security*, 12(1):37–82, 2004.

[28] A. Di Pierro, C. Hankin, and H. Wiklicky. Quantitative static analysis of distributed systems. *J. Funct. Program.*, 15(5):703–749, 2005.

[29] R. Focardi, S. Rossi, and A. Sabelfeld. Bridging language-based and process calculi security. In *FoSSaCS*, pages 299–315, 2005.

[30] J. Goguen and J. Meseguer. Security policies and security models. In *IEEE Symposium on Security and privacy*, pages 11–20. IEEE Computer Society Press, 1982.

[31] J. W. III Gray. Toward a mathematical foundation for informatin flow security. In *IEEE Security and Privacy*, pages 21–35, Oakland, California, 1991.

[32] Jonathan Heusser and Pasquale Malacaria. Applied quantitative information flow and statistical databases. In *Formal Aspects in Security and Trust*, pages 96–110, 2009.

[33] J. W. Gray III and P. F. Syverson. A logical approach to multilevel security of probabilistic systems. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 164–176, 1992.

[34] B. Köpf and D. Basin. An information-theoretic model for adaptive side-channel attacks. In *CCS '07: Proceedings of the 14th ACM Conference on Computer and Communications Security*, pages 286–296, New York, NY, USA, November 2007. ACM SIGSAC, ACM Press.

[35] D. Kozen. Semantics of probabilistic programs. In *20th Annual Symposium on Foundations of Computer Science*, pages 101–114, Long Beach, California, USA, October 1979.

[36] D. Kozen. Semantics of probabilistic programs. *Journal of Computer and System Sciences*, 22(3):328–350, 1981.

[37] G. Lowe. Quantifying information flow. In *Proceedings IEEE Computer Security Foundations Workshop*, pages 18–31, June 2002.

[38] G. Lowe. Defining information flow quantity. *Journal of Computer Security*, 12(3-4):619–653, 2004.

[39] P. Malacaria. Assessing security threats of looping constructs. In *Proceedings of the 34th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 225–235, Nice, France, 2007. ACM Press.

[40] James L. Massey. Guessing and entropy. In *In Proceedings of the 1994 IEEE International Symposium on Information Theory*, page 204, 1994.

[41] S. McCamant and M. D. Ernst. Quantitative information flow as network flow capacity. In *Proceedings of the ACM SIGPLAN 2008 Conference on Programming Language Design and Implementation*, Tucson, AZ, USA, June 2008.

[42] A. McIver and C. Morgan. A probabilistic approach to information hiding. In *Programming methodology*, pages 441–460, New York, NY, USA, 2003. Springer-Verlag New York.

[43] A. McIver and C. Morgan. *Abstraction, Refinement And Proof For Probabilistic Systems (Monographs in Computer Science)*. SpringerVerlag, 2004.

[44] J. McLean. Security models and information flow. In *Proceeding of the 1990 IEEE Symposium on Security and Privacy*, Oakland, California, May 1990.

[45] J. Millen. Covert channel capacity. In *Proceeding 1987 IEEE Symposium on Resarch in Security and Privacy*. IEEE Computer Society Press, 1987.

[46] D. Monniaux. Abstract interpretation of probabilistic semantics. In *SAS'00: Proceedings of the 7th International Symposium on Static Analysis*, pages 322–339, London, UK, 2000. Springer-Verlag.

[47] C. Mu. Jensen-shannon divergence as a measure of information flow in reactive processes. Technical Report TR-09-07, Department of Computer Science, King's College London, November 2009.

[48] C. Mu. An interval-based abstraction on quantifying information flow over probabilistic semantics. Technical Report TR-10-03, Department of Computer Science, King's College London, March 2010.

[49] C. Mu and D. Clark. Quantitative analysis of secure information flow via probabilistic semantics. Technical Report TR-08-08, Department of Computer Science, King's College London, November 2008.

[50] C. Mu and D. Clark. An interval-based abstraction on quantifying information flow over probabilistic semantics. In *Electronic Notes in Theoretical Computer Science*, volume 59, pages 119–141, Elsevier, 2009.

[51] C. Mu and D. Clark. Measuring information flow in reactive processes. In *ICICS*, pages 211–225, 2009.

[52] C. Mu and D. Clark. Quantitative analysis of secure information flow via probabilistic semantics. In *ARES*, pages 49–57, 2009.

[53] Kevin R. O'Neill, Michael R. Clarkson, and Stephen Chong. Information-flow security for interactive programs. In *Proceedings of the 19th IEEE Computer Security Foundations Workshop*, pages 190–201, July 2006.

[54] B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proceedings 22nd IEEE Symposium on Security and Privacy*, pages 184–200. IEEE Computer Society Press, 2001.

[55] I. Phillips. *Discrete analysis of continuous behaviour in real-time concurrent systems*. PhD thesis, Oxford university, 2000.

[56] J. O. Pliam. On the incomparability of entropy and marginal guesswork in brute-force attacks. In *INDOCRYPT '00: Proceedings of the First International Conference on Progress in Cryptology*, pages 67–79, London, UK, 2000. Springer-Verlag.

[57] A. Renyi. On measures of entropy and information. In *Proceedings of the 4th Berkeley Symposium on Mathematics, Statistics and Probability*, pages 547–561, 1961.

[58] A. Renyi. *Probability theory*. North-Holland Publishing Company, Amsterdam, 1970.

[59] C. E. Shannon. A mathematical theory of communication. *SIGMOBILE Mob. Comput. Commun. Rev.*, 5(1):3–55, 1948.

[60] Geoffrey Smith. On the foundations of quantitative information flow. In *FOSSACS*, pages 288–302, 2009.

[61] D. Sutherland. A model of information. In *Proceedings of the 9th national Computer Security Conference*, Sep 1986.

[62] D. Volpano and G. Smith. Verifying secrets and relative secrecy. In *Proc. 27th ACM Symp. on Principles of Programming Languages (POPL*, pages 268–276. ACM Press, 2000.

[63] D. G. Weber. Quantitative hook-up security for covert channel analysis. In *CSFW 1988*, pages 58–71, Franconia, NH, 1988. IEEE.

[64] J. Todd Wittbold and D. M. Johnson. Information flow in nondeterministic systems. In *IEEE Symposium on Security and Privacy*, pages 144–161, 1990.