# Assessing Security Threats of Looping Constructs

Pasquale Malacaria
Department of Computer Science, Queen Mary, University of London
pm@dcs.qmul.ac.uk

## Abstract

There is a clear intuitive connection between the notion of leakage of information in a program and concepts from information theory. This intuition has not been satisfactorily pinned down, until now. In particular, previous information-theoretic models of programs are imprecise, due to their overly conservative treatment of looping constructs. In this paper we provide the first precise information-theoretic semantics of looping constructs. Our semantics describes both the amount and rate of leakage; if either is small enough, then a program might be deemed "secure". Using the semantics we provide an investigation and classification of bounded and unbounded covert channels.

***Categories and Subject Descriptors*** D.4.6 [*Security and Protection*]: Information flow controls; H.1.1 [*Systems and Information Theory*]: Information theory

***General Terms*** Security,Theory, Languages

***Keywords*** Information Theory, Security, Language Semantics.

## 1. Introduction

There is a basic conceptual issue that lies at the heart of the foundations of security: The problem is that "secure" programs do leak small amounts of information. An example is a password checking program

$\texttt{if}\ (\texttt{l} == \texttt{h})\ \texttt{access else deny}$

where an attacker will gain some information by observing what the output is (by observing `deny` he will learn that his guess `l` was wrong). This makes non-interference[1] [11] based models of security [24, 1] problematic; they judge far too many programs to be "insecure". As elegantly put in [22]

> *In most non-interference models, a single bit of compromised information is flagged as a security violation, even if one bit is all that is lost. To be taken seriously, a non-interference violation should imply a more significant loss. Even ... where timings are not available, and a bit per millisecond is not distinguishable from a bit per fortnight ... a channel that compromises an unbounded amount of information is substantially different from one that cannot.*

Of course, using declassification it is still possible to use a non-interference model to limit, rather than eliminate, the areas in a program where information will be leaked. But, non-interference does not itself help us in deciding whether to declassify. Again, [22] raises the question: how we decide that a region is safe to declassify?

To illustrate, consider the following program containing a secure variable `h` and a public variable `l`:

```
l=20; while ( h < l) {l=l-1}
```

The program performs a bounded search for the value of the secret `h`. Is it safe to declassify that program? One could argue that the decision should depend on the size of the secret; the larger the secret the more declassifiable it becomes. How to give a precise meaning to this argument? Is the previous program secure if `h` is a 10-bit variable?
Is it secure if `h` is a 16-bit variable? And shouldn't the answer depend also on the attacker's knowledge of the distribution of inputs e.g. if she/he knew that 0 is a much more likely value for `h` than any other value?

The main objective of the present work is to develop a theory where this kind of questions can be mathematically addressed. To this aim we will develop an information theoretical semantics of looping commands. The semantics is quantitative: outcomes are real numbers measuring security properties of programs.

The appeal of Shannon's information theory [23] in this context is that it combines the *probability* of an event with the *damage* the happening of that event would cause. In this sense information theory provides a *risk assessment analysis* of language based security. Consider again the password checking program and suppose `l, h` are 2-bit variables and the distribution of values of `h` is uniform (all values are equally likely). We identify the damage associated to an event with the difference between the size of the search space for the secret before and after the event has happened. The more is revealed by an event-the larger the difference-the bigger the damage. The damage for the event `observe access` happening will be gaining information of the whole secret $2 = \texttt{log}(4)$ bits[2] while the damage for `observe deny` will be gaining information of one possibility being eliminated. Formally:

1. `observe access`:
   - probability $=\frac{1}{4}$,
   - damage $= \texttt{log}(4) - \texttt{log}(1) = \texttt{log}(\frac{4}{1}) = 2$
2. `observe deny`:
   - probability $=\frac{3}{4}$,
   - damage $= \texttt{log}(4) - \texttt{log}(3) = \texttt{log}(\frac{4}{3})$

*POPL'07* January 17–19, 2007, Nice, France.

[1] Intuitively interference from `x` to `y` means changes in `x` affect the state of `y`. Non-interference is the lack of intererence

[2] In the paper `log` stands for base 2 logarithm.

Combining damages with probabilities we get

$\frac{1}{4}\mathtt{log}(\frac{4}{1}) + \frac{3}{4}\mathtt{log}(\frac{4}{3})$

an instance of $\sum \mathtt{p_i log}(\frac{1}{\mathtt{p_i}})$, Shannon's entropy formula.

This paper introduces tools to compute the leakage in loops; first information theoretical formulas characterizing leakage are extracted by the denotational semantics of loops: these formulas are the basis for defining:

1. channel capacity: the maximum amount of leakage of a loop as a function of the attacker's knowledge of the input.
2. rate of leakage: the amount of information leaked as a function of the number of iterations of the loop.

These definitions are then used in a classification of loops. This is an attempt to answer questions like:

1. is the amount of leakage of the loop unbounded as a function of the size of the secret?
2. How does the rate change when the size of the secret changes?

Notice that in sequential programs there are no natural cases of unbounded covert channels unless loops are present; for this reason we claim that a major achievement of this work is the identification of and mathematical reasoning about unbounded covert channels [22]

> *Characterization of unbounded channels is suggested as the kind of goal that would advance the study of this subject, and some creative thought could no doubt suggest others.*

To motivate the relevance of this paper in the above contexts some case studies are presented. We hope that by seeing the definitions at work in these cases the reader will be satisfied that the semantics is:

1. natural: i.e. in most cases agrees with our intuition about what the leakage should be and when it doesn't it provides new insights.
2. helpful: i.e. it provides clear answers for situations where the intuition doesn't provide answers.
3. general: although some ingenuity is required case by case, the setting is not ad hoc.
4. innovative: it provides a fresh outlook on reasoning about covert channels in programs in terms of quantitative reasoning.

To complete the work we also address the following basic question: what is the meaning of information theoretical measures in the context of programming language interference? For example what does it mean that the above program "leaks 2.6 bits for a 10-bit variables under uniform distribution"? Based on recent work by Massey [17], Malone and Sullivan[16] it will be argued that this quantity is a lower bound on the attacker effort to guess the secret using a binary search or a dictionary attack.

### 1.1 Contribution and related work

Pioneering work by Denning [7, 8] shows the relevance of information theory to the analysis of flow of information in programs. She worked out semantics for assignments and conditionals, and gave persuasive arguments and examples. However, she did not show how to do a semantics of a full, Turing-complete programming language, with loops. As a consequence, some of the examples we consider involving unbounded channels are beyond the theory there.

Further seminal work relating information theory and non-interference in computational systems was done by Millen, McLean, Gray [18, 13, 19]; none of this work however concentrate on programming languages constructs.

In the context of programming languages the relations between information theory and non-interference [11, 20] relevant to the present work have been studied in a series of papers by Clark, Hunt, Malacaria [2, 3, 4], where the background for the present work is introduced: the main ingredients are an interpretation of programs and program variables in terms of random variables and a definition of leakage in terms of conditional mutual information.

Other quantitative approaches to non-intereference have also recently been studied; Lowe [15] defines channel capacity in the context of CSP. Di Pierro, Hankin, Wiklicky propose a probabilistic approach to approximate non-interference in a declarative setting[9] and more recently in distributed systems [10]. A probabilistic beliefs-based approach to non-interference has been suggested by Clarkson, Myers, Schneider [5].

Quantitative approaches to covert channel analysis in somewhat different contexts have been proposed by Gray and Syverson [12], Weber [25] and Wittbold [27].

To the best of our knowledge no work so far has provided a reasonable quantitative analysis of loops in imperative languages; the bounds in [2, 3, 4] are over pessimistic (if any leakage is possible in a loop, the loop leaks everything). Hence the analysis here presented is original, and because of the relationship between unbounded covert channels and loops this paper provides an original quantitative analysis for covert channels in the context of programming languages.

### 1.2 Structure of the work

The article is structured as follows:

- Section 2 reviews some basic definitions from information theory and presents an interpretation of program variables and commands in terms of random variables.
- Section 3 define an information theoretical formula for the leakage of the command `while e M`. From the leakage formula some definitions are derived, like rate of leakage, channel capacity, security, ratio of leakage.
- Based on these definitions section 3.3 classifies loops according to their leakage and rate of leakage.
- Section 4 provides case studies justifying the usefulness of these notions.
- Section 5 provides a justification of the information theoretical measures in this work. This justification is based on bounds on a dictionary attack scenario.

## 2. Preliminaries

### 2.1 Entropy, interaction, interference

We begin by reviewing some basic concepts of information theory relevant to this work; additional background is readily available both in textbooks [6] and on the web (e.g. the wikipedia entry for Entropy).

Given a space of events with probabilities $\mathtt{P} = (\mathtt{p_i})_{\mathtt{i}\in\mathtt{N}}$ ($\mathtt{N}$ a set of indices) the Shannon's entropy is defined as

$\mathtt{H(P)} = -\Sigma_{\mathtt{i}\in\mathtt{N}}\mathtt{p_i log(p_i)}$.

It is usually said that this number measure the average uncertainty of the set of events: if there is an event with probability 1 then the entropy will be 0 and if the distribution is uniform i.e. no event is more likely than any other the entropy is maximal, i.e. $\mathtt{log}(|N|)$. The entropy of a random variable is the entropy of its distribution.

An important property of entropy which we will use says that if we take a partition of the events in a probability space, the entropy of the space can be computed by summing the entropy of the partition with the weighted entropies of the partition sets. We call this the *partition property*; formally: given a distribution $\mu$ over a set $\mathtt{S} = \{\mathtt{s}_{1,1}, \ldots, \mathtt{s}_{n,m}\}$ and a partition of $\mathtt{S}$ in sets $(\mathtt{S}_i)_{1 \leq i \leq n}$, $\mathtt{S}_i = \{\mathtt{s}_{i,1}, \ldots, \mathtt{s}_{i,m}\}$:

$$\mathtt{H}(\mu(\mathtt{s}_{1,1}), \ldots, \mu(\mathtt{s}_{n,m})) = \mathtt{H}(\mu(\mathtt{S}_1), \ldots, \mu(\mathtt{S}_n)) + \sum_{i=1}^{n} \mu(\mathtt{S}_i)\mathtt{H}\left(\frac{\mu(\mathtt{s}_{i,1})}{\mu(\mathtt{S}_i)}, \ldots, \frac{\mu(\mathtt{s}_{i,m})}{\mu(\mathtt{S}_i)}\right)$$

where $\mu(\mathtt{S}_i) = \sum_{1 \leq j \leq m} \mu(\mathtt{s}_{i,j})$.

Given two random variables $\mathtt{X}, \mathtt{Y}$ the conditional entropy $\mathtt{H}(\mathtt{X}|\mathtt{Y})$ is the average of all entropies of $\mathtt{X}$ conditioned to a given value for $\mathtt{Y}$, $\mathtt{Y} = \mathtt{y}$, i.e.

$$\Sigma_{\mathtt{Y}=\mathtt{y}}\mu(\mathtt{Y} = \mathtt{y})\mathtt{H}(\mathtt{X}|\mathtt{Y} = \mathtt{y})$$

where $\mathtt{H}(\mathtt{X}|\mathtt{Y} = \mathtt{y}) = -\Sigma_{\mathtt{X}=\mathtt{x}}\mu(\mathtt{X} = \mathtt{x}|\mathtt{Y} = \mathtt{y})\mathtt{log}(\mu(\mathtt{X} = \mathtt{x}|\mathtt{Y} = \mathtt{y}))$.

The higher $\mathtt{H}(\mathtt{X}|\mathtt{Y})$ is the lower is the correlation between $\mathtt{X}$ and $\mathtt{Y}$. It is easy to see that if $\mathtt{X}$ is a function of $\mathtt{Y}$, $\mathtt{H}(\mathtt{X}|\mathtt{Y}) = 0$ and if $\mathtt{X}$ and $\mathtt{Y}$ are independent $\mathtt{H}(\mathtt{X}|\mathtt{Y}) = \mathtt{H}(\mathtt{X})$.

Mutual information is defined as

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

This quantity measures the correlation between $\mathtt{X}$ and $\mathtt{Y}$. This follows from what we saw about conditional entropy: if $\mathtt{X}$ is a function of $\mathtt{Y}$, $\mathtt{I}(\mathtt{X};\mathtt{Y}) = \mathtt{H}(\mathtt{X}) - \mathtt{H}(\mathtt{X}|\mathtt{Y}) = \mathtt{H}(\mathtt{X}) - 0$ and if $\mathtt{X}$ and $\mathtt{Y}$ are independent $\mathtt{I}(\mathtt{X};\mathtt{Y}) = \mathtt{H}(\mathtt{X}) - \mathtt{H}(\mathtt{X}) = 0$.

Mutual information is a measure of binary *interaction*. In fact so far we have only defined unary or *binary* concepts.

As we will see conditional mutual information, a form of ternary interaction will be used to quantify *interference*. Conditional mutual information measures the interference of a random variable on a binary interaction, i.e.

$$\mathtt{I}(\mathtt{X};\mathtt{Y}|\mathtt{Z}) = \mathtt{H}(\mathtt{X}|\mathtt{Z}) - \mathtt{H}(\mathtt{X}|\mathtt{Y},\mathtt{Z}) = \mathtt{H}(\mathtt{Y}|\mathtt{Z}) - \mathtt{H}(\mathtt{Y}|\mathtt{X},\mathtt{Z})$$

Conditional mutual information is always non negative; however it can affect interaction in a positive or negative way as these examples show:

$$\begin{array}{lcll}
\mathtt{I}(\mathtt{X};\mathtt{Y}|\mathtt{X}\oplus\mathtt{Y}) & = & \mathtt{H}(\mathtt{Y}|\mathtt{X}\oplus\mathtt{Y}) - \mathtt{H}(\mathtt{Y}|\mathtt{X}\oplus\mathtt{Y},\mathtt{X}) & = \\
1\ -\ 0 & > & 0 & = \\
\mathtt{I}(\mathtt{X};\mathtt{Y}) & & & \\
\mathtt{I}(\mathtt{X};\mathtt{X}\wedge\mathtt{Y}|\mathtt{X}) & = & \mathtt{H}(\mathtt{X}|\mathtt{X}) - \mathtt{H}(\mathtt{X}|\mathtt{X},\mathtt{X}\wedge\mathtt{Y}) & = \\
0\ -\ 0 & < & 0.32 & = \\
\mathtt{I}(\mathtt{X};\mathtt{X}\wedge\mathtt{Y}) & & &
\end{array}$$

where $\mathtt{X}, \mathtt{Y}$ are independent random variables taking boolean values and $\wedge, \oplus$ are boolean conjunction and exclusive or.

A positive interference $\mathtt{I}(\mathtt{X};\mathtt{Y}|\mathtt{Z})$ means $\mathtt{Z}$ increase the interaction between $\mathtt{X}$ and $\mathtt{Y}$ by contributing new relevant information, whereas negative interference means $\mathtt{Z}$ removes information which was present in the interaction.

In the previous example $\mathtt{X} \oplus \mathtt{Y}$ contributes to the interaction of two independent random variables $\mathtt{X}, \mathtt{Y}$ by bringing the information if they have the same value or not, whereas $\mathtt{X}$ doesn't bring any new information to the interaction between $\mathtt{X}$ and $\mathtt{X} \wedge \mathtt{Y}$; in fact knowledge of $\mathtt{X}$ is detrimental to the interaction between $\mathtt{X}$ and $\mathtt{X} \wedge \mathtt{Y}$ because that knowledge is removed from the interaction.

### 2.2 Random variables and programs

The language we are considering is a simple imperative language with assignments, sequencing, conditionals and loops. Further in the paper we will add to this language a probabilistic choice operator. Syntax and semantics for the language are standard and so we omit them. The expressions of the language are arithmetic expression, with constants $0, 1, \ldots$ and boolean expressions with constants $\mathtt{tt}, \mathtt{ff}$.

Following denotational semantics commands are state transformers, informally maps which change the values of variables in the memory and expressions are maps from the memory to values; we will denote by $\llbracket \mathtt{M} \rrbracket$ the standard denotational semantics of the program $\mathtt{M}$ [26]. We assume there are two input variables $\mathtt{H}, \mathtt{L}$, the high (confidential) and low (public) input, and we assume that inputs are equipped with a probability distribution, so we can consider them as random variables (the input is the joint random variable $(\mathtt{H}, \mathtt{L})$). A deterministic program $\mathtt{M}$ can hence be seen as a random variable itself, the output random variable where the probability on an output value of the program is the sum of probabilities of all inputs evaluating via $\mathtt{M}$ to that value $\mu(\mathtt{M} = \mathtt{o}) = \Sigma\{\mu(\mathtt{h}, \mathtt{l}) | \llbracket \mathtt{M} \rrbracket(\mathtt{h}, \mathtt{l}) = \mathtt{o}\}$.

More formally

1. Our probability space is $(\Omega, \mathtt{A}, \mu)$ where

   $$\Omega = \{\sigma | \sigma : \{\mathtt{H}, \mathtt{L}\} \to \mathtt{N}\}$$

   $\mathtt{A} = \mathcal{P}(\Omega)$ (the power set) and $\mu$ a probability distribution over $\Omega$.

   An element $\sigma \in \Omega$ is a memory state (environment), i.e. a map from names of variables to values.

   A state $\sigma$ is naturally extended to a map from arithmetic expressions to $\mathtt{N}$ by

   $$\sigma(\mathtt{e}(\mathtt{x}_1, \ldots, \mathtt{x}_n)) = \mathtt{e}(\sigma(\mathtt{x}_1), \ldots, \sigma(\mathtt{x}_n))$$

   i.e. the $\sigma$ evaluation of an expression is the value obtained by evaluating all variables in the expression according to $\sigma$.

2. A random variable $\mathtt{M}$ is a partition (an equivalence relation) over $\Omega$[3]. For a command $\mathtt{M}$ the equivalence relation would identify all $\sigma$ which have the same observable output state for the command; i.e. $\sigma \equiv_{\mathtt{M}} \tau$ iff $\mathtt{M}(\sigma) \upharpoonright_{\mathtt{Ob}} = \mathtt{M}(\tau) \upharpoonright_{\mathtt{Ob}}$. Here we will take as observable the output values of the variable $\mathtt{L}$, i.e. $\mathtt{Ob} = \mathtt{L}$; for example if $\mathtt{M}$ is the command $\mathtt{L} = \mathtt{H}$ then $\sigma \equiv_{\mathtt{M}} \tau$ iff $\sigma_{[\mathtt{L}=\llbracket \mathtt{H} \rrbracket]} \upharpoonright_{\mathtt{Ob}} = \tau_{[\mathtt{L}=\llbracket \mathtt{H} \rrbracket]} \upharpoonright_{\mathtt{Ob}}$ iff $\sigma_{[\mathtt{L}=\llbracket \mathtt{H} \rrbracket]} \upharpoonright_{\mathtt{L}} = \tau_{[\mathtt{L}=\llbracket \mathtt{H} \rrbracket]} \upharpoonright_{\mathtt{L}}$. The notation $\sigma_{\mathtt{x}=\llbracket \mathtt{e} \rrbracket}$ means $\sigma$ where the variable $\mathtt{x}$ is evaluated to $\llbracket \mathtt{e} \rrbracket$. Hence $\sigma_{[\mathtt{L}=\llbracket \mathtt{H} \rrbracket]} \upharpoonright_{\mathtt{L}} = \tau_{[\mathtt{L}=\llbracket \mathtt{H} \rrbracket]} \upharpoonright_{\mathtt{L}}$ holds for any $\sigma, \tau$ which agree (have the same value) on the variable $\mathtt{H}$.

   The probability distribution on a command random variable $\mathtt{M}$ is defined as

   $$\mu(\mathtt{M} = \tau') = \Sigma_{\tau \in \Omega}\{\mu(\tau) | \mathtt{M}(\tau) \upharpoonright_{\mathtt{Ob}} = \tau' \upharpoonright_{\mathtt{Ob}}\}$$

   If $\mathtt{M}$ is a non terminating program the definition of random variable as an equivalence relation still holds; now we will have an additional class which is all states which will be non terminating; For the probability distribution we extend the above definition with the clause:

   $$\mu(\mathtt{M} = \perp) = \Sigma_{\tau \in \Omega}\{\mu(\tau) | \mathtt{M}(\tau) = \perp\}$$

   Instantiating the above definition we get the following random variables associated to particular commands:

   - $\mathtt{M}$ is the command $\mathtt{x} = \mathtt{e}$: this is the equivalence relation $\sigma \equiv_{\mathtt{x}=\mathtt{e}} \tau$ iff $\sigma_{\mathtt{x}=\llbracket \mathtt{e} \rrbracket} \upharpoonright_{\mathtt{Ob}} = \tau_{\mathtt{x}=\llbracket \mathtt{e} \rrbracket} \upharpoonright_{\mathtt{Ob}}$.
   - $\mathtt{M}$ is $\mathtt{if\ e\ c\ else\ c'}$: then $\sigma \equiv_{\mathtt{if\ e\ c\ else\ c'}} \tau$ iff
     if $\sigma(\mathtt{e}) = \mathtt{tt} \neq \mathtt{ff} = \tau(\mathtt{e})$ then $\llbracket \mathtt{c} \rrbracket(\sigma) \upharpoonright_{\mathtt{Ob}} = \llbracket \mathtt{c}' \rrbracket(\tau) \upharpoonright_{\mathtt{Ob}}$
     and $\sigma(\mathtt{e}) = \tau(\mathtt{e})$ and $\tau(\mathtt{e}) = \mathtt{tt}$ implies $\sigma \equiv_{\mathtt{c}} \tau$ and
     $\tau(\mathtt{e}) = \mathtt{ff}$ implies $\sigma \equiv_{\mathtt{c}'} \tau$.
   - $\sigma \equiv_{\mathtt{c};\mathtt{c}'} \tau$ iff $\sigma \not\equiv_{\mathtt{c}} \tau$ implies $\llbracket \mathtt{c} \rrbracket(\sigma) \equiv_{\mathtt{c}'} \llbracket \mathtt{c} \rrbracket(\tau)$.

   Given a command $\mathtt{M}$ we will use the random variable

[3] The conventional mathematical definition of a random variable is of a map from a probability space to a measurable space. In those terms we are considering the kernel of such a map.

$\mathtt{M}^n \equiv \mathtt{M}; \dots ; \mathtt{M}$

for the n-th iteration of M. This is a generalization of the sequential composition. For example $\sigma \equiv_{(\mathtt{x}=\mathtt{x}+1)^5} \tau$ iff $\sigma \equiv_{\mathtt{x}=\mathtt{x}+5} \tau$ and

$$\mu((\mathtt{x} = \mathtt{x} + 1)^5 = \sigma) = \Sigma\{\mu(\tau) | (\mathtt{x} = \mathtt{x} + 5)(\tau) \restriction_{\mathtt{Ob}} = \sigma \restriction_{\mathtt{Ob}}\}$$

3. Similarly we will have random variables corresponding to boolean expressions (we take as boolean values the integers $0, 1$); again an equivalence class is the set of states evaluated to the same (boolean) value:

$$\sigma \equiv_{\mathtt{e}} \tau \Leftrightarrow \sigma(\mathtt{e}) = \tau(\mathtt{e})$$

$$\mu(\mathtt{e} = \mathtt{tt}) = \Sigma_{\tau \in \Omega}\{\mu(\tau) \mid \tau(\mathtt{e}) = \mathtt{tt}\}$$

for example for $\mathtt{e}_1 == \mathtt{e}_2$

$$\sigma \equiv_{\mathtt{e}_1 == \mathtt{e}_2} \tau \Leftrightarrow \sigma(\mathtt{e}_1) = \sigma(\mathtt{e}_2) = \tau(\mathtt{e}_1) = \tau(\mathtt{e}_2)$$

$$\mu((\mathtt{e}_1 == \mathtt{e}_2) = \mathtt{tt}) = \Sigma_{\tau \in \Omega}\{\mu(\tau) | \tau(\mathtt{e}_1) = \tau(\mathtt{e}_2)\}$$

Given an expression e guarding a command M we define the random variable $\mathtt{e}^n$ as e where the variables in e are evaluated following $\mathtt{n} - 1$ iterations of M. For example if e is $\mathtt{x} > 0$, M is $\mathtt{x} = \mathtt{x} + 1$ then $\mathtt{e}^3$ is $\mathtt{x} + 2 > 0$. e is hence an abbreviation for $\mathtt{e}^1$.

Following [2] and inspired by works by Dennings, McLean, Gray, Millen [7, 8, 18, 13, 19], *interference* (or leakage of confidential information) in a program M is defined as

$$\mathtt{I}(\mathtt{O}; \mathtt{H}|\mathtt{L})$$

i.e. the conditional mutual information between the output and the high input of the program given knowledge of the low input.

Notice:

1. O is just another name for the random variable corresponding to the program seen as a command, i.e. $\mathtt{O} = \mathtt{M}$.
2. This is a input-output model i.e. it doesn't model an attacker who could have knowledge of some intermediate state of the program. One implication of this model is that only *global* timing attacks can in principle be analyzed.
3. For deterministic programs we have

$$\begin{array}{lcl} \mathtt{I}(\mathtt{O};\mathtt{H}|\mathtt{L}) & = & \mathtt{H}(\mathtt{O}|\mathtt{L}) - \mathtt{H}(\mathtt{O}|\mathtt{H},\mathtt{L}) \\ & = & \mathtt{H}(\mathtt{O}|\mathtt{L}) - \mathtt{H}([\![\mathtt{M}]\!](\mathtt{H},\mathtt{L})|\mathtt{H},\mathtt{L}) \\ & = & \mathtt{H}(\mathtt{O}|\mathtt{L}) \end{array}$$

i.e. interference becomes the uncertainty in the output of M given knowledge of the low input.

A motivating result for this definition of leakage is that for deterministic programs $\mathtt{I}(\mathtt{O};\mathtt{H}|\mathtt{L}) = 0$ iff the program is non-interfering [4].

To see why $\mathtt{H}(\mathtt{O}|\mathtt{L})$ is not enough for measuring leakage in non-deterministic setting, consider the following simple program: $\mathtt{l} = \mathtt{random}(0, 1)$ i.e. the output is 0 or equally likely 1. Since the output is independent from the inputs $\mathtt{H}(\mathtt{O}|\mathtt{L}) = \mathtt{H}(\mathtt{O})$ and $\mathtt{H}(\mathtt{O}) = 1$. So we would conclude that there is 1 bit of leakage. This is clearly false as there is no secret information in the program. However
$\mathtt{I}(\mathtt{O};\mathtt{H}|\mathtt{L}) = \mathtt{H}(\mathtt{O}|\mathtt{L}) - \mathtt{H}(\mathtt{O}|\mathtt{H},\mathtt{L}) = \mathtt{H}(\mathtt{O}) - \mathtt{H}(\mathtt{O}) = 1 - 1 = 0$

Let's now investigate the quantity $\mathtt{H}(\mathtt{O}|\mathtt{L})$.

Consider for example the program

$\mathtt{M} \equiv \mathtt{l} = 3; \mathtt{if}\ (\mathtt{l} == 5)\ \mathtt{l} = \mathtt{h}\ \mathtt{else}\ \mathtt{l} = 0$

Here $\mathtt{H}(\mathtt{M}|\mathtt{l}) = \mathtt{H}(\mathtt{M})$ because l is initialized in the program, hence there is no dependency from low inputs outside the program. Also, because the above program is equivalent to $\mathtt{l} = 0$ there is no leakage of information, i.e. $\mathtt{H}(\mathtt{M}) = 0$.

Consider now the program where l is not initialized, i.e.

$\mathtt{M} \equiv \mathtt{if}\ (\mathtt{l} == 5)\ \mathtt{l} = \mathtt{h}\ \mathtt{else}\ \mathtt{l} = 0$

Then $\mathtt{H}(\mathtt{M}|\mathtt{l})$ will be the weighted sum of $\mathtt{H}(\mathtt{M}|\mathtt{l} = 5)$ and $\mathtt{H}(\mathtt{M}|\mathtt{l} \neq 5)$; formally

$$\begin{array}{lr} \mathtt{H}(\mathtt{M}|\mathtt{l}) & = \\ \mu(\mathtt{l} = 5)\mathtt{H}(\mathtt{if}\ (\mathtt{l} == 5)\ \mathtt{l} = \mathtt{h}\ \mathtt{else}\ \mathtt{l} = 0|\mathtt{l} = 5) & + \\ \mu(\mathtt{l} \neq 5)\mathtt{H}(\mathtt{if}\ (\mathtt{l} == 5)\ \mathtt{l} = \mathtt{h}\ \mathtt{else}\ \mathtt{l} = 0|\mathtt{l} \neq 5) & = \\ \mu(\mathtt{l} = 5)\mathtt{H}(\mathtt{h}) + 0 & \end{array}$$

However if the attacker were to choose the input $\mathtt{l} = 5$ then $\mathtt{M} \equiv \mathtt{l} = \mathtt{h}$ and so $\mathtt{H}(\mathtt{M}) = \mathtt{H}(\mathtt{h})$.

Hence by considering the non conditional entropy

$\mathtt{max}_{\mathtt{v} \in \mathbb{N}} \mathtt{H}(\mathtt{O}|\mathtt{L} = \mathtt{v})$

we will get an upper bound on $\mathtt{H}(\mathtt{O}|\mathtt{L})$. This will provide the leakage of the attack where the attacker can choose the inputs (to maximize his gain). The other extreme is

$\mathtt{min}_{\mathtt{v} \in \mathbb{N}} \mathtt{H}(\mathtt{O}|\mathtt{L} = \mathtt{v})$

The case of the least devastating attack.

Hence instead of computing $\mathtt{H}(\mathtt{O}|\mathtt{L})$ we will compute a non conditional entropy $\mathtt{H}(\mathtt{M}_{\mathtt{l}=\mathtt{v}})$ where v is a defined value for l. According to the cases such v will be calibrated to the power of the attacker. As no confusion arises we will drop the subscript and just write $\mathtt{H}(\mathtt{M})$.

Finally notice that we do not model an attacker able to choose high inputs, i.e. we are not modeling a *spy* trying to communicate with an external accomplice but an *intruder* (e.g. a trojan horse or a dictionary attack on passwords).

## 3. Analysis of loops

### 3.1 Loops as disjoint union of functions

#### 3.1.1 Entropy of disjoint union of functions

This subsection contains the technical backbone of the main definitions of the paper.

Consider a function $\mathtt{f} : \mathtt{X} \longrightarrow \mathtt{Y}$, which is the union of a family of functions $(\mathtt{f}_\mathtt{i})_{\mathtt{i} \in \mathtt{I}}$with disjoint domains $(\delta\mathtt{f}_\mathtt{i})_{\mathtt{i} \in \mathtt{I}}$, i.e. for each $\mathtt{i}, \delta\mathtt{f}_\mathtt{i} \subseteq \mathtt{X}$ is the domain of $\mathtt{f}_\mathtt{i}$ and $(\delta\mathtt{f}_\mathtt{i})_{\mathtt{i} \in \mathtt{I}}$ is a partition of X.

We will note f by $\Sigma_{\mathtt{i} \in \mathtt{I}}(\mathtt{f}_\mathtt{i}|\delta\mathtt{f}_\mathtt{i})$ when we want to stress that $\mathtt{f}(\mathtt{x}) = \mathtt{f}_\mathtt{i}(\mathtt{x})$ for the unique i such that $\mathtt{x} \in \delta\mathtt{f}_\mathtt{i}$.

Define $\{[\mathtt{y}] = \mathtt{f}^{-1}(\mathtt{y})|\mathtt{y} \in \mathtt{Y}\}$; clearly this is also a partition of X. Define the entropy of f as the entropy of its inverse images, i.e. $\mathtt{H}(\mu([\mathtt{y}_1]), \dots, \mu([\mathtt{y}_n]))$. The aim now is to characterize the entropy of f.

Assume that f is *collision free* i.e. the family $(\mathtt{f}_\mathtt{i})_{\mathtt{i} \in \mathtt{I}}$ has also disjoint codomains. In that case $(\delta\mathtt{f}_\mathtt{i})_{\mathtt{i} \in \mathtt{I}}$ can also be seen as a partition on the partition $[\mathtt{y}] = \mathtt{f}^{-1}(\mathtt{y})$: $\delta\mathtt{f}_\mathtt{i}$ is the set of all $[\mathtt{y}]$ for y in the codomain of $\mathtt{f}_\mathtt{i}$. Let's write $[\mathtt{y}_1]^\mathtt{j}, \dots, [\mathtt{y}_m]^\mathtt{j}$ for the classes in $\delta\mathtt{f}_\mathtt{j}$

From now on to ease the notation we will often use events instead of their probability when no confusion arise, for example in a computation $[\mathtt{y}]$ will stand for $\mu[\mathtt{y}]$ the probability of the event $[\mathtt{y}]$, i.e. $\sum\{\mu(\mathtt{x})|\mathtt{x} \in [\mathtt{y}]\}$. Similarly $\mathtt{H}([\mathtt{y}_1], \dots, [\mathtt{y}_n])$ will stand for $\mathtt{H}(\mu[\mathtt{y}_1], \dots, \mu[\mathtt{y}_n])$ etc.

By using the partition property from section 2.1 we have:

PROPOSITION 1. *For a collision free function* f*:*

$$\mathtt{H}([\mathtt{y}_1], \dots, [\mathtt{y}_n]) = \mathtt{H}(\delta\mathtt{f}_1, \dots, \delta\mathtt{f}_n) + \sum_{\mathtt{j} \in \mathtt{I}} \delta\mathtt{f}_\mathtt{j} \mathtt{H}(\frac{[\mathtt{y}_1]^\mathtt{j}}{\delta\mathtt{f}_\mathtt{j}}, \dots, \frac{[\mathtt{y}_m]^\mathtt{j}}{\delta\mathtt{f}_\mathtt{j}})$$

Let's now consider the case where f has collisions. Remember a collision is a $\mathtt{y} \in \mathtt{Y}$ in the image of two different functions, i.e. $[\mathtt{y}] \cap \delta\mathtt{f}_\mathtt{j} \neq \emptyset \neq [\mathtt{y}] \cap \delta\mathtt{f}_\mathtt{i}$ for $\mathtt{i} \neq \mathtt{j}$. In this case let's define $\mathtt{Y}'$ as Y extended with enough new elements to eliminate collisions and let $\mathtt{f}' : \mathtt{X} \longrightarrow \mathtt{Y}'$ be the derived function with no collisions, so $\mathtt{f}'$ is the union of the family of funtions $(\delta f'_i)_{i \in I}$ with disjoint domain and codomain. $f'_i$ is defined as

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

(So $(f_i(x), i)$ are the new elements added to $Y$)

Let's define $C_f(Y)$ as the set of collisions of $f$ in $Y$, and write $x^y_1, \ldots, x^y_m$ for the elements of $[y]$. By using again the partition property we have:

PROPOSITION 2.

$$H([y_1], \ldots, [y_n]) = H([y'_1], \ldots, [y'_{n'}]) - \sum_{y \in C_f(Y)} [y] H(\frac{x^y_1}{[y]}, \ldots, \frac{x^y_m}{[y]})$$

This means that the entropy of a function defined as a union of functions with disjoint domains is given by the entropy of the derived function with no collisions minus the weighted sum of the entropies of the collisions. To ease the notation we can rewrite Proposition 2 as.

$H(f) = H(f') - \sum H(C_f(Y))$

Let's call *disambiguation* of $f$ the function $f'$.

Notice that Proposition 2 implies that the the entropy of $f$ is a lower bound on the entropy of the disambiguation of $f$.

As an example let's consider the function $f = f_1 \oplus f_2 \oplus f_3$ defined by

$f_1(x_1) = y_1, f_1(x_2) = y_2 = f_2(x_3), f_2(x_4) = y_4,$
$f_3(x_5) = y_5 = f_3(x_6)$

and assume uniform distribution on the inputs. $f$ has one collision $y_2$ so to compute $H(f)$ we first extend the codomain with a new element $y'_2$ so to have $f'_1(x_2) = y_2, f'_2(x_3) = y'_2$

Computing $H(f)$ using proposition 2 gives:

$$\begin{aligned} H(f) &= H(f') - \sum H(C_f(Y)) \\ &= H(\tfrac{1}{3}, \tfrac{1}{3}, \tfrac{1}{3}) + 2\tfrac{1}{3}H(\tfrac{1}{2}, \tfrac{1}{2}) + \tfrac{1}{3}H(1, 0) - \tfrac{1}{3}H(\tfrac{1}{2}, \tfrac{1}{2}) \\ &= 1.585 + \tfrac{2}{3} + 0 - \tfrac{1}{3} \\ &= 1.918 \end{aligned}$$

### 3.1.2 Entropy of loops

Let `while e M` be a terminating loop. From a denotational point of view we can see it as a map

$$F = \Sigma_{1 \leq i \leq n} F_i$$

where $n$ is an upper bound on the number of iteration of the loop and all $F_i$ have disjoint domain: each $F_i$ is the map which iterates $M$ $i$ times under the condition that the guard has been true up to that moment and it will be false after the $i - th$ iteration of $M$. The domain of $F_i$ is hence given by all states $\sigma$ such that $F_j(\sigma)(e) = tt, 0 \leq j \leq i$ and $F_{i+1}(\sigma)(e) = ff$.

Formally

$$\text{while } e\ M = \Sigma_{0 \leq i \leq n}(M^i | e^{<i>})$$

where

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

and $M^0 = \text{skip}$. Notice

1. $e^{<i>}$ are events and not random variables
2. the assumption that $n$ is an upper bound on the number of iterations of the loop implies
$$\Sigma_{0 \leq i \leq n}\, \mu(e^{<i>}) = 1$$
3. the events $e^{<0>}, \ldots, e^{<n>}$ constitute a partition of the set of states: given any initial state $\sigma$ the loop will terminate in $< n$ iterations; exactly one of the $e^{<i>}$ must be true for $\sigma$ i.e. $\sigma \in e^{<i>}$, e.g. for $i > 1$
$\sigma(e) = tt \wedge \cdots \wedge M^i(\sigma)(e) = tt \wedge M^{i+1}(\sigma)(e) = ff$
To prove that this is a partition suppose it isn't, i.e.
$\sigma \in e^{<i>} \cap e^{<i+j>}$; then $M^{i+1}\sigma(e) = ff$ because of $e^{<i>}$ and $M^{i+1}\sigma(e) = tt$ because of $e^{<i+j>}$: a contradiction, hence the $e^{<i>}$ are disjoint sets, i.e. a partition.

By applying proposition 1,2 for a `while` we have:

PROPOSITION 3. *For a collision free loop* `while e M` *bounded by* $n$ *iterations*

$$\begin{aligned} H(\text{while } e\ M) \quad &= \quad H(\mu(e^{<0>}), \ldots, \mu(e^{<n>}))+ \\ &\quad\ \Sigma_{1 \leq i \leq n}\mu(e^{<i>})H(M^i | e^{<i>}) \end{aligned}$$

In the case of a loop with collisions, following proposition 2 equality is achieved as follows:

$$\begin{aligned} H(\text{while } e\ M) \quad &= \quad H(\mu(e'^{<0>}), \ldots, \mu(e'^{<n>}))+ \\ &\quad\ \Sigma_{1 \leq i \leq n}\mu(e'^{<i>})H(M'^i | e'^{<i>})- \\ &\quad\ \textstyle\sum_{\sigma \in C_{\text{while } e\ M}(\Omega)} [\sigma] H(\frac{\tau^\sigma_1}{[\sigma]}, \ldots, \frac{\tau^\sigma_m}{[\sigma]}) \end{aligned}$$

Notice that the disambiguation of a collisions free loops is the loop itself. This entails:

PROPOSITION 4. *For a command* `while e M` *bounded by* $n$ *iterations*

$$\begin{aligned} H(\text{while } e\ M) \quad &\leq \quad H(\mu(e'^{<0>}), \ldots, \mu(e'^{<n>}))+ \\ &\quad\ \Sigma_{1 \leq i \leq n}\mu(e'^{<i>})H(M'^i | e'^{<i>}) \end{aligned}$$

*with equality iff the loop is collision free.*

Collisions do not present a conceptual change in the framework but add some computational burden; also collisions are not very frequent in loops; for a collision in a loop to arise two different iteration of the loop should give the same values for all read and written low variables in the loop and the guard should be false on these values. For example all loops using a counter, a variable taking a different value at each iteration don't contain collisions.

For these reason from now on we will concentrate on collision free loops.

## 3.2 Basic definitions

Define

$$\begin{aligned} W(e, M)_n \quad &= \quad H(\mu(e^{<0>}), \ldots, \mu(e^{<n>}), 1 - \Sigma_{0 \leq i \leq n}\mu(e^{<i>}))+ \\ &\quad\ \textstyle\sum_{1 \leq i \leq n} \mu(e^{<i>})H(M^i | e^{<i>}) \end{aligned}$$

as the leakage of `while e M` up to n iterations.

PROPOSITION 5. $\forall n \geq 0,\ W(e, M)_n \leq W(e, M)_{n+1}$

*Proof*: we only need to prove

$H(\mu(e^{<0>}), \ldots, \mu(e^{<n>}), 1 - \Sigma_{0 \leq i \leq n}\mu(e^{<i>})) \leq$
$H(\mu(e^{<0>}), \ldots, \mu(e^{<n>}), \mu(e^{<n+1>}), 1 - \Sigma_{0 \leq i \leq n+1}\mu(e^{<i>}))$

which can be rewritten as

$$H(p_1, \ldots, p_n, q_{n+1} + p_{n+1}) \leq H(p_1, \ldots, p_n, p_{n+1}, q_{n+1})$$

the inequality then follows from

$H(p_1, \ldots, p_n, p_{n+1}, q_{n+1}) =$
$H(p_1, \ldots, p_n, p_{n+1} + q_{n+1})+$
$(p_{n+1} + q_{n+1})H(\frac{p_{n+1}}{p_{n+1}+q_{n+1}}, \frac{q_{n+1}}{p_{n+1}+q_{n+1}})$

■

The *leakage* of `while e M` is defined as

$$\lim_{n \to \infty} W(e, M)_n \qquad (1)$$

In the case of a loop with collisions the definition is modified in the obvious way:

$$\mathtt{lim}_{n\to\infty}\mathtt{W}'(\mathtt{e},\mathtt{M})_n - \sum \mathtt{H}(\mathtt{C}(\mathtt{W}'(\mathtt{e},\mathtt{M}))) \qquad (2)$$

i.e. we first compute the leakage in the disambiguation of the loop and then we subtract the weighted entropies of the collisions

The *rate of leakage* is

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

Hence in the case of terminating loops the rate will be the total leakage divided by the number of iterations. This can be considered a rough measure of rate: for example if the first iteration were to leak all secret and the following billion nothing the rate would still be one billionth of the secret size. However as in our model the attacker can only perform observations on the output and not on intermediate states of the program the chosen definition of rate will give indication of the timing behavior of the channel in that context.

A fundamental concept in information theory is *channel capacity*, i.e. the maximum amount of leakage over all possible input distributions, i.e.

$$\mathtt{max}_{\mu}\ \mathtt{lim}_{n\to\infty}\mathtt{W}(\mathtt{e},\mathtt{M})_n \qquad (3)$$

In our setting we will look for the distribution which will maximize leakage. Informally such a distribution will provide the setting for the most devastating attack: we will refer to this as the *channel distribution*.

Also we will use the term *channel rate* for the rate of leakage of the channel distribution. Again this should be thought of as the average maximal amount of leakage per iteration.

To define rate and channel capacity on the case of collisions the above definitions should be applied on the definition of leakage for loops with collisions.

### 3.2.1 Leakage vs security

Consider a simple assignment $\mathtt{l} = \mathtt{h}$ where the variables are $\mathtt{k}$-bit variables. We know that the assignment transfer all information from $\mathtt{h}$ to $\mathtt{l}$, so we would be tempted to say that the leakage is $\mathtt{k}$. That is not correct. Suppose $\mathtt{h}$ is a 3-bit variable (so possible values are $0\ldots 7$ ) and suppose the attacker knows $\mathtt{h}$ is even (so the possible values are $0,2,4,6$). The uncertainty on $\mathtt{h}$ before executing $\mathtt{l} = \mathtt{h}$ is hence $\mathtt{H}(\frac{1}{4},\frac{1}{4},\frac{1}{4},\frac{1}{4}) = 2$. The leakage is not 3 but

$$\mathtt{H}(\mathtt{l} = \mathtt{h}) = \mathtt{H}(\frac{1}{4},\frac{1}{4},\frac{1}{4},\frac{1}{4}) = 2$$

i.e. the information of $\mathtt{h}$. The *security* of the program is the difference between the uncertainty before execution and the leakage (the uncertainty after execution). Hence security of the previous example of $\mathtt{l} = \mathtt{h}$ is 2-2=0. Notice that when the program reveal everything this notion is invariant w.r.t. the chosen distribution, i.e. while the leakage of $\mathtt{l} = \mathtt{h}$ will depend on the distribution, its security will always be 0, all that can be revealed is revealed.

Formally security is defined [2] as

$$\mathtt{Sec}(\mathtt{O}) = \mathtt{H}(\mathtt{H}|\mathtt{L}) - \mathtt{H}(\mathtt{O}|\mathtt{L}) = \mathtt{H}(\mathtt{H}|\mathtt{L},\mathtt{O})$$

The last equality is proven as follows:

$$\begin{array}{lcll}\mathtt{H}(\mathtt{H}|\mathtt{L},\mathtt{O}) & = & \mathtt{H}(\mathtt{H},\mathtt{L},\mathtt{O}) - \mathtt{H}(\mathtt{L},\mathtt{O}) & = \\ \mathtt{H}(\mathtt{H},\mathtt{L}) - \mathtt{H}(\mathtt{L},\mathtt{O}) & = & \mathtt{H}(\mathtt{H},\mathtt{L}) - \mathtt{H}(\mathtt{L}) - \mathtt{H}(\mathtt{L},\mathtt{O}) + \mathtt{H}(\mathtt{L}) & = \\ \mathtt{H}(\mathtt{H}|\mathtt{L}) - \mathtt{H}(\mathtt{O}|\mathtt{L}) & & & \end{array}$$

Using arguments similar to the ones presented at the end of section 2.2 most of the times we will consider the simplified version where there are no dependencies on $\mathtt{L}$, i.e. $\mathtt{H}(\mathtt{H}) - \mathtt{H}(\mathtt{O})$. In fact $\mathtt{H}(\mathtt{H}|\mathtt{L})$ can be reduced to $\mathtt{H}(\mathtt{H})$ when (as it is normally the case) the secret is independent of the public input.

Another notion we will use is the *leakage ratio* i.e. $\frac{\mathtt{H}(\mathtt{O}|\mathtt{L})}{\mathtt{H}(\mathtt{H}|\mathtt{L})}$ the amount leaked divided by the maximum amount leakable. This is a number in the interval $[0,1]$ which measures the percentage of the secret leaked by the program, so the ratio has minimum 0 iff the leakage is 0 and maximum 1 iff all the secret is revealed by the program.

### 3.3 Classification of looping channels

The following classification combines the previous definitions with variations in the size of the secret. For example a bounded loop is one where even if we were able to increase arbitrarily the size of the secret we would not be able to increase arbitrarily the amount leaked.

For the purposes of this investigation loops are classified as:

**a** C-*bounded* if the leakage is upper bounded by a constant C.

**b** *Bounded* if the leakage is C-bounded independently of the size (i.e. number of bits) of the secret. It is unbounded otherwise.

**c** *Stationary* or *constant rate* if the rate is asymptotically constant in the size of the high input.

**d** *Increasing* (resp decreasing) if the rate is asymptotically increasing (resp decreasing) as a function of the size of the high input.

**e** *Mixed* if the rate is not stationary, decreasing or increasing.

Clearly all loops are C-bounded by the size of the secret and by the channel capacity; the interesting thing is to determine better bounds. For example if we are studying a loop where we know the input distribution has a specific property we may found better bounds than the size of the secret.

From a security analysis point of view the most interesting case is the one of unbounded covert channels, i.e. loops releasing all secret by indirect flows. Notice that a guard cannot leak more than 1 bit so the rate of a covert channel cannot exceed the number of guards in the command.

Notice also that the rate of leakage is loosely related to timing behaviour. In loops with decreasing rate if the size of the secret is doubled each iteration will (on average) reveal less information than each iteration with the original size. We will spell out the timing content of rates in some of the case studies.

## 4. Case studies

We will now use the previous definition. The aim is to show that the definitions make sense and the derived classification of channels helps in deciding when a loop is a threat to the security of a program and when is not.

The programs studied are simple examples of common loops: linear, bounded and bitwise search, parity check etc.

Most of the arguments will use a separation property of the definition of leakage: in fact Definition 1 neatly separates the information flows in the guard and body of a loop, so if there is no leakage in the body (e.g. no high variable appears in the body of the loop) (1) becomes

$$\mathtt{lim}_{n\to\infty}\{\mathtt{H}(\mu(e^{<0>}),\ldots,\mu(e^{<n>}),1-\Sigma_{0\leq i\leq n}\mu(e^{<i>}))\} \quad (4)$$

On the other side if there is no indirect flow from the guard (e.g. $\mathtt{e}$ doesn't contain any variable affected by high variables) then (1) becomes

$$\mathtt{lim}_{n\to\infty}\Sigma_{1\leq i\leq n}\mu(e^{<i>})\mathtt{H}(\mathtt{M}^i|e^{<i>}) \qquad (5)$$

Unless otherwise stated we are assuming uniform distribution for all input random variables (i.e. all input values are equally likely).

Also to simplify notations we will consider that a $\mathtt{k}$-bit variable assume values $0.\ldots,2^k-1$ (i.e. no negative numbers).

**Table 1.** Summary of analysis for loops; loop `i` is the loop presented in section 4.`i` of the paper

| | loop 1 | loop 2 | loop 3 | loop 4 | loop 4a | loop 5 | loop 6 | loop 7 |
|---|---|---|---|---|---|---|---|---|
| Bound | $\infty$ | 4.3219 | 1 | 16 | $\infty$ | 0 | $\mathtt{log(C)}$ | $\infty$ |
| Channel Rate | $\downarrow$ | = | = | = | = | = | $\downarrow$ | = |
| Capacity | $\mathtt{k}$ | 4.3219 | 1 | 16 | k | 0 | $\mathtt{log(C)}$ | $\frac{k}{2}$ |
| Channel leakage ratio | 1 | $\leq \frac{4.3219}{k}$ | $\leq \frac{1}{k}$ | $\leq \frac{16}{k}$ | 1 | 0 | $\leq \frac{\mathtt{log(C)}}{k}$ | $\leq \frac{1}{2}$ |

A summary of this section results is shown in table 1[4].

### 4.1 An unbounded covert channel with decreasing rate

Consider

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

Under uniform distribution $\max W(e, M)_n$ is achieved by

$$H(\mu(e^{<0>}), \dots, \mu(e^{<2^k-1>})) + \sum_{0 \leq i \leq 2^k-1} \mu(e^{<i>})H(M^i|e^{<i>})$$

Notice that no high variable appears in the body, so there is no leakage in the body, i.e

$$\sum_{0 \leq i \leq 2^k-1} \mu(e^{<i>})H(M^i|e^{<i>}) = 0$$

We hence only need to study

$$H(\mu(e^{<0>}), \dots, \mu(e^{<2^k-1>}))$$

notice now that

$$e^{<i>} = \begin{cases} 0 = h, \text{if } i = 0 \\ 0 \neq h, \dots, i \neq h \wedge i+1 = h, \text{ if } i > 0 \end{cases}$$

hence $\mu(e^{<i>}) = \frac{1}{2^k}$. This means

$$H(\mu(e^{<0>}), \dots, \mu(e^{<2^k-1>})) = H(\frac{1}{2^k}, \dots, \frac{1}{2^k}) = \log(2^k) = k$$

As expected all `k`-bit of a variable are leaked in this loop, for all possible `k`; however to reveal `k` bits $2^k$ iterations are required. We conclude that this is an unbounded covert channel with decreasing rate $\frac{k}{2^k}$. To attach a concrete timing meaning to this rate let $t_1, t_2$ be the time (in milliseconds) taken by the system to evaluate the expression $!(l = h)$ and to execute the command $l = l + 1$ respectively. Then the above program leaks $\frac{k}{2^k}$ bits per $t_1 + t_2$ milliseconds.

Notice that uniform distribution maximizes leakage, i.e. it achieves channel capacity.

Consider for example the following input distribution for a 3-bit variable:

$$\mu(0) = \frac{7}{8}, \mu(1) = \mu(2) \dots = \mu(7) = \frac{1}{56}$$

In this case the attacker knows, before the run of the program, that 0 is much more likely than any other number to be the secret, so the amount of information revealed by running the program is below 3 bits (below capacity). In fact we have

$$H(\frac{7}{8}, \frac{1}{56}, \dots, \frac{1}{56}) = 0.8944838$$

Notice however that whatever the distribution the security of this program is 0 and leakage ratio 1.

### 4.2 A bounded covert channel with constant rate

$$l = 20;\ \mathtt{while}(h < l)\{l = l - 1\}$$

After executing the program `l` will be 20 `if` $h \geq 20$ and will be `h` `if` $0 \leq h < 20$ i.e. `h` will be revealed if it is in the interval 0..19.

The random variables of interest are:

$$M^n \equiv l = 20 - n$$

The events associated to the guard are:

$$e^{<n>} = \begin{cases} h < 20 - n \wedge h \geq 20 - (n+1) \quad \equiv & \\ h = 20-(n+1), & n>0 \\ & \\ h \geq 20, & n=0 \end{cases}$$

and

$$\mu(e^{<n>}) = \begin{cases} \frac{2^k-20}{2^k} \text{ if } n = 0 \\ \\ \frac{1}{2^k} \text{ if } 0 < n \leq 20 \\ \\ 0 \text{ if } n > 20 \end{cases}$$

Again since the body of the loop doesn't contain any high variable

$$\Sigma_{1 \leq i \leq n} \mu(e^{<i>})H(M^i|e^{<i>}) = 0$$

The leakage is hence given by

$$\begin{aligned} & H(\mu(e^{<1>}), \dots, \mu(e^{<n>})) && = \\ & H(\tfrac{2^k-20}{2^k}, \tfrac{1}{2^k}, \dots, \tfrac{1}{2^k}, 0, \dots, 0) && = \\ & -\tfrac{2^k-20}{2^k}\log(\tfrac{2^k-20}{2^k}) - 20(\tfrac{1}{2^k}\log(\tfrac{1}{2^k})) && \end{aligned}$$

This function is plotted in figure 1 for $k = 6 \dots 16$. The interesting thing in the graph is how it shows that for `k` around 6 bits the program is unsafe (more than 2.2 bits of leakage) whereas for `k` from 14 upwards the program is safe (around 0 bits of leakage).

We conclude that this is a bounded covert channel with decreasing rate.

However uniform distribution is not the channel distribution. The capacity of this channel is 4.321928 and is achieved by the distribution where the only values with non zero probability for `h` are in the range $0 \dots 19$ and have uniform distribution[5].

Notice that the channel distribution ignores values of `h` higher than 20, so the channel rate is constant $\frac{4.321928}{20} = 0.2160$.

### 4.3 A 1-bounded channel with constant rate

Consider the following program

```
h=BigFile
i=0;
l=0;
while (i<N)
{
```

[4] In the Channel leakage ratio row in the table quantities greater than 1 should be ignored.

[5] We are ignoring the case where $k < 5$ where the capacity is less than 4.321928

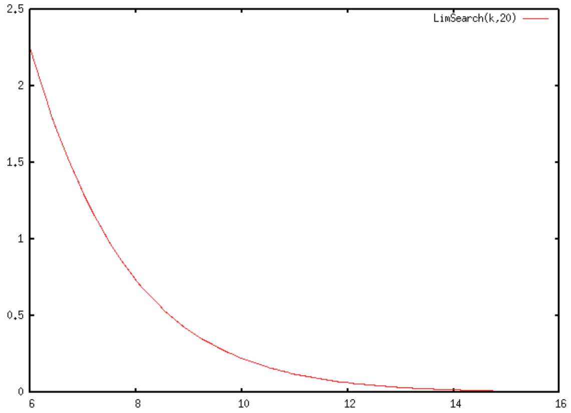

**Figure 1.** leakage in `l=20; while ( h < l) {l=l-1}`

```
l= Xor(h[i],l);
i=i+1;
}
```

This program take a large confidential file and performs a parity check, i.e. write in l the `Xor` of the first `N` bits of the file. The n-ary `Xor` function returns 1 if its argument has an odd number of 1s and 0 otherwise. This is a yes/no answer so its entropy has maximum 1 which is achieved by uniform distribution. Hence

$$H(M^n|e^{<n>}) = H(h[0] \oplus \dots \oplus h[n-1]) = 1$$

Notice that

$$e^{<n>} \equiv n < N \wedge n+1 \geq N$$

henceforth

$$\mu(e^{<i>}) = 0 \text{ if } i \neq N-1 \text{ and } \mu(e^{<N-1>}) = 1$$

We deduce the leakage is:
$H(\mu(e^{<0>}),\dots,\mu(e^{<n>})) + \Sigma_{1\leq i\leq n}\mu(e^{<i>})H(M^i|e^{<i>}) =$
$0 + \mu(e^{<N>})H(M^N|e^{<N>}) = 1$

This is a 1-bounded channel with constant rate and capacity 1. Notice however that if the number of iterations were a function of the secret size, for example by inserting in the second line of the program the assignment $N = size(h)$, (where $size(h) = k$ the size of the secret) then it becomes a 1 bounded channel with decreasing rate $\frac{1}{k}$ and capacity 1.

Again there are distributions which do not achieve channel capacity, for example one where values of `h` with odd number of bits equal to 1 are less likely than other values.

### 4.4 A 16-bounded stationary channel

Consider the program

```
int c = 16, low = 0;
while (c >= 0) {
    int m = (int)Math.pow(2,c);
    if (high >= m) {
        low = low + m;
        high = high - m;
    }
    c = c - 1;
}
System.out.println(low);
```

Here the guard of the loop doesn't contain variables affected by `high`, hence we only need to use formula 5 where `M` is

```
int m = (int)Math.pow(2,c);
          if (high >= m) {
              low = low + m;
              high = high - m;
          }
          c = c - 1;
```

To compute $H(M^n)$ notice that the $n$-th iteration of `M` test the $n$-th bit of `high`, i.e. `high` $>=$ `m` is true at the n-th iteration iff the n-th bit of `high` is 1(this is because $m = 2^{16-n}$) and copies that bit into `low`

The variables of interests are:

$$M^n \equiv low = nBits(high)$$

$$e^{<n>} = 16 - n \geq 0 \wedge 16 - (n+1) < 0$$

$$\mu(e^{<n>}) = \begin{cases} 1 \text{ if } n = 16 \\ 0 \text{ otherwise} \end{cases}$$

Because of this the leakage of the guard is 0 and for the total leakage we only need to compute $H(M^{16}|e^{<16>}) = 16$. This mean that the rate is 1.

This is hence an example of a 16-bounded stationary channel. However if we were to replace the first assignment `int c = 16` with `c = size(l)` i.e.

```
int c = size(l), low = 0;
while (c >= 0) {
    int m = (int)Math.pow(2,c);
    if (high >= m) {
        low = low + m;
        high = high - m;
    }
    c = c - 1;
}
System.out.println(low);
```

then we would have an unbounded stationary channel (assuming that `h,l` be of the same size) with constant channel rate 1.

Again channel capacity is achieved by uniform distribution. For example a distribution where we already know few bits of `high` will not achieve channel capacity,

### 4.5 A never terminating loop

```
while (0== 0)
        low =   high;
```

Here $\mu(e^{<i>}) = 0$ for all $i$, hence for all $n$ the formula

$$W(e, M)_n = H(\mu(e^{<0>}),\dots,\mu(e^{<n>}), 1 - \Sigma_{0\leq i\leq n}\mu(e^{<i>})) + \Sigma_{1\leq i\leq n}\mu(e^{<i>})H(M^i|e^{<i>})$$

becomes

$$H(0,\dots,0,1) + \Sigma_{1\leq i\leq n}\, 0\, H(M^i|e^{<i>}) = 0$$

from which we conclude that the leakage, rate and capacity are all 0.

The reason why the program is secure even if the whole secret is assigned to a low variable is that only observations on final states of the command are allowed (none in this case because of non

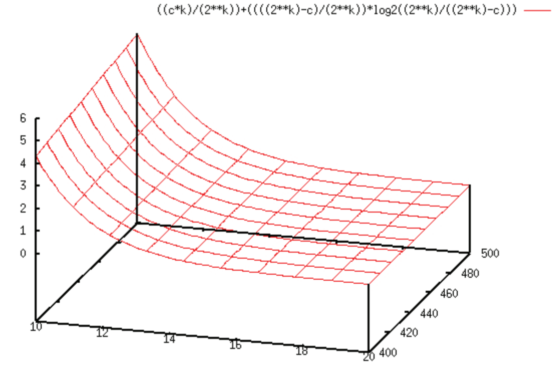

**Figure 2.** leakage for program in section 4.6

termination); again this is feature of our model where the observer cannot see intermediate values of the computation, in which case this program would leak everything.

### 4.6 A may terminating loop

```
l=0;
flag=tt;
       while (flag or l<h)
               {
               if (h<= C) flag=ff;
               l=l+1;
               }
```

This loop will terminate if $\mathtt{h} \leq \mathtt{C}$ and in that case $\mathtt{l} = \mathtt{h}$. The event $\mathtt{e}^{<\mathtt{i}>}$ corresponds to $\mathtt{i} = \mathtt{h} \wedge \mathtt{h} \leq \mathtt{C}$, hence

$$\mu(\mathtt{e}^{<\mathtt{i}>}) = \frac{1}{\mathtt{C}} \frac{\mathtt{C}}{2^{\mathtt{k}}} = \frac{1}{2^{\mathtt{k}}} \text{ if } \mathtt{i} \leq \mathtt{C}$$

Notice that as the information $\mathtt{h} \leq \mathtt{C}$ is known by knowing $\mathtt{e}^{<\mathtt{i}>}$ we conclude that for all $\mathtt{i}$, $\mathtt{H}(\mathtt{M}^{\mathtt{i}}|\mathtt{e}^{<\mathtt{i}>}) = 0$.

The leakage of this channel (under uniform distribution) is hence
$\mathtt{H}(\frac{1}{2^{\mathtt{k}}}, \ldots, \frac{1}{2^{\mathtt{k}}}, \frac{2^{\mathtt{k}}-\mathtt{C}}{2^{\mathtt{k}}}) = \frac{\mathtt{Ck}}{2^{\mathtt{k}}} - \frac{2^{\mathtt{k}}-\mathtt{C}}{2^{\mathtt{k}}}\mathtt{log}(\frac{2^{\mathtt{k}}-\mathtt{C}}{2^{\mathtt{k}}})$
This function is similar to the one from section 4.2. Again channel capacity is achieved not by the uniform distribution but from the one where the first $\mathtt{C}$ values have probability $\frac{1}{\mathtt{C}}$: in that case the program reveal all the secret.

Figure 2 shows the leakage for $\mathtt{k}$ between 10 and 20 and $\mathtt{C}$ between 400 and 500 under uniform distribution.

### 4.7 Probabilistic operators

When defining leakage in section 2.2 it was shown that the conditional entropy $\mathtt{H}(\mathtt{O}|\mathtt{L})$ would overestimate leakage for a program like

$\mathtt{l} = \mathtt{random}(0, 1)$

where $\mathtt{random}(0, 1)$ a probabilistic operator returning 0 with probability $\mathtt{p}$ and 1 with probability $1 - \mathtt{p}$.

However we could interpret $\mathtt{l} = \mathtt{random}(0, 1)$ as the program $\mathtt{l} = \mathtt{x}$ where $\mathtt{x}$ is an "unknown input" variable taking value 0 with probability $\mathtt{p}$ and 1 with probability $1 - \mathtt{p}$. Then computing $\mathtt{H}(\mathtt{O}|\mathtt{L}, \mathtt{X})$ gives $\mathtt{H}(\mathtt{O}|\mathtt{L}, \mathtt{X}) = \mathtt{H}(\mathtt{O}|\mathtt{X}) = 0$, all uncertainty in the output comes from "the random"$\mathtt{x}$ so it can be eliminated by conditioning on it.

This suggests that an analysis of probabilistic programs can be developed by introducing a new random variable to cater for the probabilistic operator; the leakage formula becomes $\mathtt{H}(\mathtt{O}|\mathtt{L}, \mathtt{X})$; the effect of this formula is to subtract from the uncertainty in the output the uncertainty coming from the low input and from the probabilistic operator; i.e. the uncertainty in $\mathtt{H}(\mathtt{O}|\mathtt{L}, \mathtt{X})$ comes from the secret. As usual we can simplify the formula to $\mathtt{H}(\mathtt{O}|\mathtt{X})$ by hardwiring the low inputs into the probability distribution as shown at the end of Section 2.2.

In the cases of loops using a probabilistic operator we take $\mathtt{X}$ as a stream of bits; the i-th bit in the stream is the i-th outcome of the operator.

We can compute the leakage of probabilistic programs by using the definition of conditional entropy

$$\mathtt{H}(\mathtt{O}|\mathtt{X}) = \sum \mu(\mathtt{X} = \mathtt{x}_{\mathtt{i}})\mathtt{H}(\mathtt{O}|\mathtt{X} = \mathtt{x}_{\mathtt{i}})$$

As an example consider the program $\mathtt{P}$

```
int i=0;  low = 0;
while (i< size(high)) {
    if (Coin[i]==0 )
        low[i] = high[i];
        i=i+1;
}
System.out.println(low);
```

where $\mathtt{Coin}$ is a stream of unknown bits such that $\mathtt{Coin}[\mathtt{i}] = 0$ with probability $\mathtt{p}_{\mathtt{i}}$. Then at the end of the program the $\mathtt{i} - \mathtt{th}$ bit of $\mathtt{high}$ will be copied in $\mathtt{low}$ with probability $\mathtt{p}_{\mathtt{i}}$.

To compute the leakage of the program , i.e. $\mathtt{H}(\mathtt{P}|\mathtt{Coin})$ we proceed as follow:

1. Compute, using formula 1, the entropies $\mathtt{H}(\mathtt{P}_{\mathtt{s}_1}), \ldots, \mathtt{H}(\mathtt{P}_{\mathtt{s}_{\mathtt{n}}})$ where $\mathtt{H}(\mathtt{P}_{\mathtt{s}_{\mathtt{i}}})$ is the above program where the vector $\mathtt{Coin}$ is instantiated to a specific sequence $\mathtt{s}_{\mathtt{i}}$.
2. Compute $\sum \mu(\mathtt{s}_{\mathtt{i}})\mathtt{H}(\mathtt{P}_{\mathtt{s}_{\mathtt{i}}}) = \mathtt{H}(\mathtt{P}|\mathtt{Coin})$.

Given a stream $\mathtt{s}_{\mathtt{i}}$ and $\mathtt{high}$ a $\mathtt{k}$-bit variable, the bits of $\mathtt{high}$ copied in $\mathtt{low}$ are those corresponding to the positions in $\mathtt{s}_1$ with value 0. For example if $\mathtt{high}$ is a 4-bit variable and $\mathtt{s}_{\mathtt{i}} = 1001\ldots$ then $\mathtt{low}$ will be the sequence $0\mathtt{h}[1]\mathtt{h}[2]0$. The leakage of $\mathtt{H}(\mathtt{P}_{\mathtt{s}_{\mathtt{i}}}) =$ number of 0s in $\mathtt{s}_{\mathtt{i}}$

For example if we assume $\mathtt{high}, \mathtt{Coin}$ are uniformly distributed, i.e. any bit in $\mathtt{high}, \mathtt{Coin}$ has $1/2$ chance of being 0 or 1 and $\mathtt{high}$ is a 4-bit variable then there will be 4 sequences with 1 zero, 6 with 2 zeros, 4 with 3 zeros and 1 with 4 zeros (the general formula is $\frac{\mathtt{k}!}{(\mathtt{k}-\mathtt{i})!\mathtt{i}!}$ where $\mathtt{i}$ is the number of zeros) . The leakage will hence be

$$\frac{4}{16} + \frac{6}{16}2 + \frac{4}{16}3 + \frac{1}{16}4 = \frac{1}{2} + \frac{3}{2} = 2$$

the general formula being

$$\sum_{1 \leq \mathtt{i} \leq \mathtt{k}} \frac{\mathtt{k}!}{(\mathtt{k}-\mathtt{i})!\mathtt{i}!}\mathtt{i}\frac{1}{2^{\mathtt{k}}} = \sum_{1 \leq \mathtt{i} \leq \mathtt{k}} \frac{\mathtt{k}!}{(\mathtt{k}-\mathtt{i})!\mathtt{i}!}\mathtt{i}\frac{1}{2^{\mathtt{i}}}\frac{1}{2^{\mathtt{k}-\mathtt{i}}} = \frac{\mathtt{k}}{2}$$

This is hence an unbounded channel leaking $\frac{\mathtt{k}}{2}$ bits with rate $\frac{1}{2}$.

Notice that in the presence of probabilistic operators all definitions introduced, leakage, rate, channel, leakage ratio have an additional parameter, i.e. the distribution on this unknown input. The leakage for the above program given $\mathtt{Coin}[\mathtt{i}] = 0$ with probability $\mathtt{p}$ is $\mathtt{pk}$. This is obtained by the expected value of the binomial distribution:

$$\sum_{1\leq i\leq k} \frac{k!}{(k-i)!i!} i\, p^{i}(1-p)^{k-i} = pk$$

For example by changing the distribution in $\mathtt{Coin}$ such that for all i, $\mathtt{Coin}[i] = 0$ with probability 1 the above program become the unbounded stationary channel studied in section 4.4 whereas if for all i, $\mathtt{Coin}[i] = 0$ with probability 0 the above program become secure.

The analysis of probabilistic programs should hence return a number if the probabilities of the probabilistic operator are known and a distribution when the probabilities are unknown.

## 5. Justifying entropy as a measure of leakage

We now address the questions: “how is leakage as defined in this work related to computer security?”

A basic result proved in [4] is that for a terminating deterministic program the leakage is 0 if and only if the program is non interfering. Similar results had been previously proved in different contexts by Millen[19] and Gray [13]. The idea is to see a non interfering program as a function $F(h, l)$ (its denotational semantics) which is constant on the $h$ component, i.e. for all $h \neq h'$, $F(h, l) = F(h', l)$. Let’s now consider $H(F|l)$: because $F$ is constant on the $h$ component there will be no uncertainty on $F$ if we know $l$, hence $H(F|l) = 0$; on the other side any denotation of a program which satisfy $H(F|l) = 0$ has to be constant on the $h$ component so has to denote a non interfering program.

Let’s now address the remaining part of this section’s question: if the leakage is $n > 0$ what does that mean?

The idea here is that $n$ is a lower bound on the effort of the attacker in guessing the secret given observations on the output of the program. In the following we will use work from Massey [17], Malone and Sullivan[16]. The following argument extends one from [2].

Suppose the attacker has available a distribution $p = (p_i)_{i\in I}$ for the secret. He can then mount a *dictionary attack* i.e. he will try to guess the secret starting from the most likely guess and so on. The expected number of guesses is then $G(p) = \sum_i i p_i$. In case of the uniform distribution $G(p) = \frac{n+1}{2}$. This inspires the information theoretic definition

$H_G(p) = \frac{2^{H(p)}+1}{2}$

In the setting of the present work, $p$ is the distribution after observing the program, and so $H(p)$ is the uncertainty of the secret after running the program, i.e. $\mathtt{Sec}(M)$, the security of the program as defined in section 3.2.1.

Massey has shown that $0.7H_G(p) \leq G(p)$ (his precise bound is $G(p)/H_G(p) \leq 2/e$). This supports the view that security provides a lower bound on the average effort required to guess the secret using a dictionary attack. Another possible yet less realistic scenario of attack is where the attacker may guess sets of values and been told if the secret is in that set. In this case the connection with entropy is even stronger as the average number of guesses becomes $H(p)$ (again this is $\mathtt{Sec}(M)$).

In the case of the dictionary attack, how good is the lower bound $0.7H_G(p) \leq G(p)$? In an experimental study [16] one million random distributions for a set between 2 and 20 values were generated. These experiments show that the following relation holds:

$0.7H_G(p) \leq G(p) \leq H_G(p)$

This suggest that in normal situations the bound is very tight.

Massey however has shown that there are distributions for which the inequality $G(p) \leq H_G(p)$ doesn’t hold; an example is the distribution $p_1 = 1 - b/n, p_2 = \cdots = p_n = b/(n^2 - n)$. For $n \to \infty$ we have $G(p)$ tends to $1 + b/2$ while $H(p)$ tends to 1. Because $b$ is arbitrary we conclude that $G(p)$ can be arbitrary larger than $H(p)$.

## 6. Conclusion and further work

This paper has given the first precise, information-theoretic account of the constructs in a Turing-complete programming language. The central point is our information theoretical semantics of leakage in loops. The theory consists of several notions: absolute leakage, rate of leakage, channel capacity, and leakage ratio. We have given a classification of loops with the aim to determine which loop presents a security threat, and then presented several case studies in an attempt to show that the definitions and classification are useful in individuating security threats and are natural.

We believe that the ideas in this paper could provide a springboard for further applications of information theory in security and programming languages. Some immediate directions for investigation are the following.

1. STATIC ANALYSIS. This work could pave the way for more powerful static analyses based on information theory. As the case studies show the analysis requires some ingenuity, for example to determine which events the $e^{<i>}$ represents. This reasoning usually involves the ability to detect interaction between several random variables. It may be possible that by combining techniques from theorem proving, model checking and quantitative static analysis like [4, 3] some reasonable static analysis may be built. The central point, though, is that with a precise semantics of loops in place, we have a reference semantics that potential abstract domains should over-approximate, in which case loops could be soundly analyzed via fixed-point iteration.
2. TIMING ATTACKS. As already noted, there is some information about timing in the notion of rate of leakage, rate being an indication of the average time needed to release some information; for example a low rate suggests little amount of secret is released in each iteration, a decreasing rate indicates that the channel take longer to transmit information as the size of the secret increases. However many timing attacks are not covered in our current model, for example those whose study requires intermediate states of execution to be *observable*; hence more work is required to address important issues in timing attacks.
3. CONCURRENCY, NON DETERMINISM. Integrating this work with a concurrency framework could open the way to the analysis of interesting protocols.
4. SEPARATION LOGIC. O’Hearn, Reynolds and Isthiaq[14, 21] have introduced a logic to reason about heaps based on some sort of non-interference between different parts of the code. Quantified interference may suggest a weaker separation logic which could be interesting to explore.

### 6.1 Acknowledgments

I’m very grateful to Fabrizio Smeraldi, Peter O’Hearn and Sebastian Hunt for very useful comments on this work. This research was supported by the EPSRC grant EP/C009967/1 Quantitative Information Flow.

## References

[1] D.Bell and L. LaPadula, “Secure computer systems: Unified exposition and Multics interpretation”, *Technical Report MTR-2997, MITRE Corp*, 1997.

[2] D. Clark and S. Hunt and P. Malacaria “Quantitative Analysis of the Leakage of Confidential Data” *Electronic Notes in Theoretical Computer Science* volume 59, issue 3, Elsevier, 2002

[3] D. Clark and S. Hunt and P. Malacaria, “Quantified Interference for a While Language”, Elsevier, *Electronic Notes in Theoretical Computer Science 112*, pages 149 – 166, 2005.

[4] D. Clark and S. Hunt and P. Malacaria, “Quantitative Information Flow, Relations and Polymorphic Types”, *Journal of Logic and Computation, Special Issue on Lambda-calculus, type theory and natural language,* 2005, volume 18, number 2, pages 181–199

[5] M. R. Clarkson and A. C. Myers and F. B. Schneider, “Belief in Information Flow”, *Proc. 18th IEEE Computer Security Foundations Workshop (CSFW 18),* IEEE Computer Society Press, 2005.

[6] T. M. Cover and J. A. Thomas, “Elements of Information Theory”, 1991, Wiley Interscience.

[7] D. E. R. Denning, “A Lattice Model of Secure Information Flow”, *Communications of the ACM,* volume 19, number 5, May 1976.

[8] D. E. R. Denning, “Cryptography and Data Security”, 1982, Addison-Wesley.

[9] A. Di Pierro and C. Hankin and H. Wiklicky, “Probabilistic confinement in a declarative framework”, *Electronic Notes in Theoretical Computer Science,* volume 48, Elsevier 2001.

[10] A. Di Pierro and C. Hankin and H. Wiklicky, “Quantitative static analysis of distributed systems”, *Journal of Functional Programming*,2005.

[11] J. Goguen and J. Meseguer, “Security Policies and Security Models”, *IEEE Symposium on Security and Privacy*, pages 11–20, IEEE Computer Society Press, 1982.

[12] J. W. Gray III and P. F. Syverson, “A Logical Approach to Multilevel Security of Probabilistic Systems”, *Distributed Computing*, volume 11, number 2, 1998, pages 73-90.

[13] W. Gray, III, James, “Toward a Mathematical Foundation for Information Flow Security”, *Proc. 1991 IEEE Symposium on Security and Privacy,* Oakland, CA, May1991, pages 21–34.

[14] S. Isthiaq and P.W. O’Hearn, “BI as an assertion language for mutable data structures”, pages = ”14–26”, *28th POPL* London 2001.

[15] G. Lowe, “Quantifying Information Flow”, *Proceedings of the Workshop on Automated Verification of Critical Systems*, 2001.

[16] D.Malone and W. Sullivan, “Guesswork and entropy”, *IEEE Transactions on Information Theory*, volume 50, number 3, March 2004.

[17] J. L. Massey, “‘Guessing and entropy”, *Proc. IEEE International Symposium on Information Theory*, 1994, Trondheim, Norway.

[18] J. McLean, “Security models and information flow”, *Proceedings of the 1990 IEEE Symposium on Security and Privacy*, 1990, Oakland, California.

[19] J. Millen, “Covert channel capacity”, *Proc. 1987 IEEE Symposium on Research in Security and Privacy*, IEEE Computer Society Press, 1987.

[20] J. C. Reynolds, “Syntactic control of interference”, *Conf. Record 5th ACM Symp. on Principles of Programming Languages* 1978.

[21] J. Reynolds, “Separation logic: a logic for shared mutable data structures”, Invited Paper, *LICS’02*, 2002.

[22] P. Y. A. Ryan and J. McLean and J. Millen and V. Gilgor, “Non-interference, who needs it?”, *Proceedings of the 14th IEEE Security Foundations Workshop*, Cape Breton, Nova Scotia, Canada, June 2001,

[23] C. Shannon, “A mathematical theory of communication”, *The Bell System Technical Journal*, volume 27, July and October, 1948, pages 379–423 and 623–656.

[24] D. Volpano and G. Smith, “A Type-Based Approach to Program Security”, *Proceedings of TAPSOFT ’97 (Colloquium on Formal Approaches in Software Engineering)*, April 1997, Lecture Notes in Computer Science, number 1214, pages 607–621.

[25] D. G. Weber, “Quantitative Hookup security for covert channel analysis”, *Proceedings of the 1988 Workshop on the Foundations of Computer Security*, 1988, Fanconia, New Hampshire, U.S.A..

[26] G. Winskel, “The formal semantics of programming languages: an introduction”, MIT Press 1993.

[27] T. Wittbold, “Network of Covert Channels”, *Proceedings of the 1990 Workshop on the Foundations of Computer Security*, 1990.