Information theoretical analysis of bounded and unbounded channels in programming languages

Malacaria, Pasquale

ISSN 1470-5559

Information theoretical analysis of bounded and unbounded channels in programing languages

Pasquale Malacaria

Queen Mary
University of London
Department of Computer Science

RR-06-06 June 2006

Information theoretical analysis of bounded and unbounded channels in programing languages

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

ABSTRACT

This work proposes a quantitative study of the behaviour of looping constructs in programs, both in terms of absolute and rate of leakage, the main interest being the investigation and classification of bounded and unbounded covert channels. This analysis of looping programs is based on information theoretical formulas derived from the denotational semantics of the `while` statement.

1. INTRODUCTION

In recent years a considerable effort has been devoted to devise formalisms and techniques to deal with insecure programs in a safe way.

The problem is that "secure" programs do leak small amounts of information. An example is a password checking program
`if` $(\mathtt{l} == \mathtt{h})$ `access else deny`
here an attacker will always gain some information by observing what the output is (by observing `deny` he will learn that his guess `l` was wrong).

This makes non-iterference[1] [10] based models of security[24, 6] problematic, at least from a foundational point of view.

As elegantly put in [21] "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."

Even successful approaches like declassification[22] don't provide a satisfactory answer to the this problem. In declassification regions of a program using confidential data may be allowed to be observable by low level observers. The idea is that a declassified region would leak only a marginal amount of secure information and so its release to low level observers doesn't compromise the security of the system.

The issue raised in [21] is however:
"how we decide that a region is safe to declassify?"
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 bits variable?
Is it secure if `h` is a 16 bits variable?
And shouldn't the answer depend also on the attacker's knowledge of the distribution of inputs e.g. if he[2] 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 analysis of looping commands. The analysis 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 $\mathtt{l}, \mathtt{h}$ are 2 bits variables and the distribution of values of `h` is uniform (all values are equally likely). The damage for the event `observe access` happening will be gaining information of the whole secret $2 = \mathtt{log}(4)$ bits[3] while the damage for `observe deny` will be gaining information of one possibility being eliminated. Formally:

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

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

[2] When referring to an attacker he is an abbreviation for she/he

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

.

Combining damages with probabilities we get
$\frac{1}{4}\log(\frac{4}{1}) + \frac{3}{4}\log(\frac{4}{3})$
an instance of $\sum p_i \log(\frac{1}{p_i})$, 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 presents; for this reason we claim that a major achievement of this work is the identification of and mathematical reasoning about unbounded covert channels [21] "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 cases studies are presented. We hope that by seeing the definitions at work in these cases the reader will be satisfied that the analysis is:

1. natural: i.e. in most cases agrees with our intuition about what the leakage should be and when not 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 languages interference? For example what does it mean that the above program "leaks 2.6 bits for a 10 bits variables under uniform distribution"? Based on recent work by Massey [14], Malone and Sullivan[9] 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.

Further seminal works relating information theory and non-interference in computational systems were done by Millen, McLean, Gray [15, 25, 16]; none of this work however concentrate on programming languages constructs.

In the context of programming languages the relations between information theory and non-interference [10, 20] relevant to the present work have been studied in a series of papers by Clark, Hunt, Malacaria [2, 1, 3], 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 [13] defines channel capacity in the context of CSP. DiPierro, Hankin, Wiklicky propose a probabilistic approach to approximate non-interference in a declarative setting[17] and more recently in distributed systems [18]. A probabilistic beliefs-based approach to non-interference has been suggested by Clarkson, Myers, Schneider [4].

Quantitative approaches to covert channel analysis in somewhat different contexts have been proposed by Gray and Syverson [11], Weber [26] 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, 1, 3] 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, secureness, ratio of leakage.
- Based on these definitions section 3.3 classify loops according to their leakage and rate of leakage
- Section 4 provides case studies justifying the goodness 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 [5] and on the web (e.g. the wickipedia entry for Entropy).

Given a space of events with probabilities $\mathtt{P} = (\mathtt{p}_i)_{i\in\mathtt{N}}$ ($\mathtt{N}$ a set of indices) the Shannon's entropy is defined as
$\mathtt{H}(\mathtt{P}) = -\Sigma_{i\in\mathtt{N}}\mathtt{p}_i\mathtt{log}_2(\mathtt{p}_i)$.

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. $\mathtt{log}_2(|N|)$. 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 to the weighted entropies of the partition sets. We call this the *partition property*; formally: given a distribution μ over a set $\mathtt{S} = \{\mathtt{s}_{1,1}, \ldots, \mathtt{s}_{n,m}\}$ and a partition of $\mathtt{S}$ in sets $(\mathtt{S}_i)_{1\leq i\leq n}$, $\mathtt{S}_i = \{\mathtt{s}_{i,1}, \ldots, \mathtt{s}_{i,m}\}$:

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

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 $\mathtt{H}(\mathtt{X}|\mathtt{Y} = \mathtt{y}) = -\Sigma_{\mathtt{X}=\mathtt{x}}\mu(\mathtt{X} = \mathtt{x}|\mathtt{Y} = \mathtt{y})\mathtt{log}_2(\mu(\mathtt{X} = \mathtt{x}|\mathtt{Y} = \mathtt{y}))$

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:

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

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

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 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 $\mu(\mathtt{M} = \mathtt{o}) = \Sigma\{\mu(\mathtt{h},\mathtt{l})|[\![\mathtt{M}]\!](\mathtt{h},\mathtt{l}) = \mathtt{o})\}$.

More formally

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

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

   $\mathtt{A} = \mathcal{P}(\Omega)$ and μ a probability distribution over Ω.

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

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

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

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

2. A random variable $\mathtt{M}$ is a partition(an equivalence relation) over Ω[4]. For a command $\mathtt{M}$ the equivalence relation would identify all σ which have the same observable output state for the command; i.e. $\sigma \equiv_{\mathtt{M}} \tau$ iff $\mathtt{M}(\sigma)\restriction_{\mathtt{Ob}} = \mathtt{M}(\tau)\restriction_{\mathtt{Ob}}$. Here we will take as observable output states low output values, i.e. $\mathtt{Ob} = \mathtt{L}$; for example if $\mathtt{M}$ is the command $\mathtt{L} = \mathtt{H}$ that would be the equivalence relation $\sigma \equiv \tau$ iff $\sigma_{[\mathtt{L}=[\![\mathtt{H}]\!]]}\restriction_{\mathtt{Ob}} = \tau_{[\mathtt{L}=[\![\mathtt{H}]\!]]}\restriction_{\mathtt{Ob}}$ iff $\sigma_{[\mathtt{L}=[\![\mathtt{H}]\!]]}\restriction_{\mathtt{L}} = \tau_{[\mathtt{L}=[\![\mathtt{H}]\!]]}\restriction_{\mathtt{L}}$ i.e. any σ, τ.

   The probability distribution on a random variable is defined as

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

   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:

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

   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 $\sigma_{\mathtt{x}=[\![\mathtt{e}]\!]}\restriction_{\mathtt{Ob}} = \tau_{\mathtt{x}=[\![\mathtt{e}]\!]}\restriction_{\mathtt{Ob}}$[5]

[4] 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.

[5] That is σ where $\mathtt{x}$ is evaluated to $[\![\mathtt{e}]\!]$ is equal to τ where $\mathtt{x}$ is evaluated to $[\![\mathtt{e}]\!]$

- M is `if e c else c'`. Then $\sigma \equiv_{\texttt{if e c else c'}} \tau$ iff if $\sigma(\texttt{e}) = \texttt{tt} \neq \texttt{ff} = \tau(\texttt{e})$ then $[\![\texttt{c}]\!](\sigma) \restriction_{\texttt{Ob}} = [\![\texttt{c}']\!](\tau) \restriction_{\texttt{Ob}}$ and $\sigma(\texttt{e}) = \tau(\texttt{e})$ and $\tau(\texttt{e}) = \texttt{tt}$ implies $\sigma \equiv_{\texttt{c}} \tau$ and $\tau(\texttt{e}) = \texttt{ff}$ implies $\sigma \equiv_{\texttt{c}'} \tau$
- $\sigma \equiv_{\texttt{c;c}'} \tau$ iff $\sigma \not\equiv_{\texttt{c}} \tau$ implies $[\![\texttt{c}]\!](\sigma) \equiv_{\texttt{c}'} [\![\texttt{c}]\!](\tau)$

Given a command M we will use the random variable $\texttt{M}^{\texttt{n}} \equiv \texttt{M};\ldots;\texttt{M}$, the n-th iteration of M. This is a generalization of the sequential composition. For example $\sigma \equiv_{(\texttt{x=x+1})^5} \tau$ iff $\sigma \equiv_{\texttt{x=x+5}} \tau$ and

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

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:

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

$$\mu(\texttt{e} = \texttt{tt}) = \Sigma_{\tau \in \Sigma}\{\mu(\tau) \mid \tau(\texttt{e}) = \texttt{tt}\}$$

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

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

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

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

Following [2] and inspired by works by Dennings, McLean, Gray, Millen [7, 8, 15, 25, 16], *interference* (or leakage of confidential information) in a program M is defined as

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

i.e. the conditional mutual information between the output and the high input of the program given knowledge of the low input. Notice that O is just another name for the random variable corresponding to the program seen as a command, i.e. $\texttt{O} = \texttt{M}$.

Notice this is a input-output model i.e. it doesn't model an attacker who could have knowledge of intermediate state of the program. One implication of this model is that only *global* timing attacks can in principle be analyzed.

Notice that for deterministic programs we have

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

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 $\texttt{I}(\texttt{O}; \texttt{H} | \texttt{L}) = 0$ iff the program is non-interfering [3].

To see why $\texttt{H}(\texttt{O} | \texttt{L})$ is not enough for measuring leakage in non-deterministic setting, consider the following simple program: $\texttt{l} = \texttt{random}(0, 1)$ i.e. the output is 0 or equally likely 1. Since the output is independent from the inputs $\texttt{H}(\texttt{O} | \texttt{L}) = \texttt{H}(\texttt{O})$ and $\texttt{H}(\texttt{O}) = 1$. 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
$\texttt{I}(\texttt{O}; \texttt{H} | \texttt{L}) = \texttt{H}(\texttt{O} | \texttt{L}) - \texttt{H}(\texttt{O} | \texttt{H}, \texttt{L}) = \texttt{H}(\texttt{O}) - \texttt{H}(\texttt{O}) = 1 - 1 = 0$

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

Consider for example the program

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

Here $\texttt{H}(\texttt{M} | \texttt{l}) = \texttt{H}(\texttt{M})$ 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 $\texttt{l} = 0$ there is no leakage of information, i.e. $\texttt{H}(\texttt{M}) = 0$.

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

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

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

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

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

Hence by considering the non conditional entropy

$\texttt{max}_{\texttt{v} \in \omega}\texttt{H}(\texttt{O} | \texttt{L} = \texttt{v})$

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

$\texttt{min}_{\texttt{v} \in \omega}\texttt{H}(\texttt{O} | \texttt{L} = \texttt{v})$

The case of the least devastating attack.

Hence instead to compute $\texttt{H}(\texttt{O} | \texttt{L})$ we will compute a non conditional entropy $\texttt{H}(\texttt{M}_{\texttt{l}=\texttt{v}})$ 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 $\texttt{H}(\texttt{M})$.

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

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

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

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 $[\texttt{y}]$ will stand for $\mu[\texttt{y}]$ the probability of the event $[\texttt{y}]$, i.e. $\sum\{\mu(\texttt{x}) | \texttt{x} \in [\texttt{y}]\}$. Similarly $\texttt{H}([\texttt{y}_1], \ldots, [\texttt{y}_\texttt{n}])$ will stand for $\texttt{H}(\mu[\texttt{y}_1], \ldots, \mu[\texttt{y}_\texttt{n}])$ etc.

By using the partition property from section 2.1 we have:

PROPOSITION 1. *For a collision free function* $\mathtt{f}$*:*

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

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

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

(So $(\mathtt{f}_i(\mathtt{x}),\mathtt{i})$ are the new elements added to $\mathtt{Y}$)

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

PROPOSITION 2.

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

This means that the entropy of a function defined as an 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.

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

Let's call *disambiguation* of $\mathtt{f}$ the function $\mathtt{f}'$.

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

As an example let's consider the function $\mathtt{f}=\mathtt{f}_1\oplus\mathtt{f}_2\oplus\mathtt{f}_3$ defined by
$\mathtt{f}_1(\mathtt{x}_1)=\mathtt{y}_1,\mathtt{f}_1(\mathtt{x}_2)=\mathtt{y}_2=\mathtt{f}_2(\mathtt{x}_3),\mathtt{f}_2(\mathtt{x}_4)=\mathtt{y}_4$,
$\mathtt{f}_3(\mathtt{x}_5)=\mathtt{y}_5=\mathtt{f}_3(\mathtt{x}_6)$
and assume uniform distribution on the inputs. $\mathtt{f}$ has one collision $\mathtt{y}_2$ so to compute $\mathtt{H}(\mathtt{f})$ we first extend the codomain with a new element $\mathtt{y}'_2$ so to have $\mathtt{f}'_1(\mathtt{x}_2)=\mathtt{y}_2,\mathtt{f}'_2(\mathtt{x}_3)=\mathtt{y}'_2$

Computing $\mathtt{H}(\mathtt{f})$ using proposition 2 gives:

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

3.1.2 Entropy of loops

Let $\mathtt{while\ e\ M}$ be a terminating loop. From a denotational point of view we can see it as a map

$$\mathtt{F}=\Sigma_{1\le i\le n}\mathtt{F}_i$$

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

Formally

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

where

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

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

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

By applying proposition 1,2 for a $\mathtt{while}$ we have:

PROPOSITION 3. *For a collision free loop* $\mathtt{while\ e\ M}$ *bounded by* $\mathtt{n}$ *iterations*

$$\begin{array}{rcl}\mathtt{H}(\mathtt{while\ e\ M}) & = & \mathtt{H}(\mu(\mathtt{e}^{<0>}),\dots,\mu(\mathtt{e}^{<n>}))+ \\ & & \Sigma_{1\le i\le n}\mu(\mathtt{e}^{<i>})\mathtt{H}(\mathtt{M}^i|\mathtt{e}^{<i>})\end{array}$$

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

$$\begin{array}{rcl}\mathtt{H}(\mathtt{while\ e\ M}) & = & \mathtt{H}(\mu(\mathtt{e}'^{<0>}),\dots,\mu(\mathtt{e}'^{<n>}))+ \\ & & \Sigma_{1\le i\le n}\mu(\mathtt{e}'^{<i>})\mathtt{H}(\mathtt{M}'^i|\mathtt{e}'^{<i>})- \\ & & \sum_{\sigma\in\mathtt{C}_{\mathtt{while\ e\ M}}(\Sigma)}[\sigma]\mathtt{H}(\frac{\tau^\sigma_1}{[\sigma]},\dots,\frac{\tau^\sigma_m}{[\sigma]})\end{array}$$

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

PROPOSITION 4. *For a command* $\mathtt{while\ e\ M}$ *bounded by* $\mathtt{n}$ *iterations*

$$\begin{array}{rcl}\mathtt{H}(\mathtt{while\ e\ M}) & \le & \mathtt{H}(\mu(\mathtt{e}'^{<0>}),\dots,\mu(\mathtt{e}'^{<n>}))+ \\ & & \Sigma_{1\le i\le n}\mu(\mathtt{e}'^{<i>})\mathtt{H}(\mathtt{M}'^i|\mathtt{e}'^{<i>})\end{array}$$

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

$$\begin{array}{rcl}\mathtt{W}(\mathtt{e},\mathtt{M})_n & = & \mathtt{H}(\mu(\mathtt{e}^{<0>}),\dots,\mu(\mathtt{e}^{<n>}),1-\Sigma_{0\le i\le n}\mu(\mathtt{e}^{<i>}))+ \\ & & \sum_{1\le i\le n}\mu(\mathtt{e}^{<i>})\mathtt{H}(\mathtt{M}^i|\mathtt{e}^{<i>})\end{array}$$

as the leakage of $\mathtt{while\ e\ M}$ up to n iterations.

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

PROOF. we only need to prove
$\mathtt{H}(\mu(\mathtt{e}^{<0>}),\ldots,\mu(\mathtt{e}^{<n>}),1-\Sigma_{0\leq i\leq n}\mu(\mathtt{e}^{<i>}))\leq$
$\mathtt{H}(\mu(\mathtt{e}^{<0>}),\ldots,\mu(\mathtt{e}^{<n>}),\mu(\mathtt{e}^{<n+1>}),1-\Sigma_{0\leq i\leq n+1}\mu(\mathtt{e}^{<i>}))$
which can be rewritten as

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

the inequality then follows from
$\mathtt{H}(p_1,\ldots,p_n,p_{n+1},q_{n+1}) =$
$\mathtt{H}(p_1,\ldots,p_n,p_{n+1}+q_{n+1})+$
$(p_{n+1}+q_{n+1})\mathtt{H}(\frac{p_{n+1}}{p_{n+1}+q_{n+1}},\frac{q_{n+1}}{p_{n+1}+q_{n+1}})$

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

$$\mathtt{lim}_{n\to\infty}\mathtt{W}(\mathtt{e},\mathtt{M})_n \tag{1}$$

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

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

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

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

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 distribution, i.e.

$$\mathtt{max}_\mu\ \mathtt{max}\{\mathtt{W}(\mathtt{e},\mathtt{M})_n|n\in\omega\} \tag{3}$$

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 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 secureness

Consider a simple assignment $\mathtt{l}=\mathtt{h}$ where the variables are `k` bits variables. We know that the assignment transfer all information from `h` to `l`, so we would be tempted to say that the leakage is `k`. That is not correct. Suppose `h` is a 3 bits variable (so possible values are `0...7`) and suppose the attacker knows `h` is even (so the possible values are $0,2,4,6$). The uncertainty on `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 `h`. The *secureness* of the program is the difference between the uncertainty before execution and the leakage (the uncertainty after execution). Hence secureness of the previous example of $\mathtt{l}=\mathtt{h}$ is 2-2=0. Notice that when the program reveal everything this notion is invariant wrt the chosen distribution, i.e. while the leakage of $\mathtt{l}=\mathtt{h}$ will depend on the distribution, its secureness will always be 0, all that can be revealed is revealed.

Formally secureness is defined [2] as

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

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 `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 ha minimum 0 iff the leakage is 0 and maximum 1 iff all the secret is revealed by the program.

3.3 Looping channel classification

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.

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

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

d *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 program studied are simple examples of common loops: linear, bounded and bitwise search, parity check.

Most of the arguments will use a separation property of the definition of leakage: in fact Definition 2 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 appear in the body of the loop) (2) becomes

$$\mathtt{max}\{\mathtt{H}(\mu(\mathtt{e}^{<0>}),\ldots,\mu(\mathtt{e}^{<n>}),1-\Sigma_{0\leq i\leq n}\mu(\mathtt{e}^{<i>})|\mathtt{n}\in\omega\}) \tag{4}$$

On the other side if there is no indirect flow from the guard (e.g. e doesn't contain any variable affected by high variables) then (2) becomes

$$\Sigma_{1\leq i}\mu(\mathtt{e}^{<i>})\mathtt{H}(\mathtt{M}^{i}|\mathtt{e}^{<i>}) \tag{5}$$

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 k bits variable assume values $0.\ldots,2^{k}-1$ (i.e. no negative numbers).

A summary of this section results is shown in table 1

4.1 An unbounded covert channel with decreasing rate

Consider

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

Under uniform distribution $\mathtt{max}\ \mathtt{W}(\mathtt{e},\mathtt{M})_{n}$ is achieved by

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

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

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

We hence only need to study

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

notice now that

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

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

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

As expected all k bits 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 $\mathtt{t}_1,\mathtt{t}_2$ the time (in milliseconds) taken by the system to evaluate the expression $!(\mathtt{l}=\mathtt{h})$ and to execute the command $\mathtt{l}=\mathtt{l}+1$ respectively. Then the the above program leaks $\frac{k}{2^{k}}$ bits per $\mathtt{t}_1+\mathtt{t}_2$ milliseconds.

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

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

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

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). Indeed we have

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

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

4.2 A bounded covert channel with constant rate

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

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

The random variables of interest are:

$$\mathtt{M}^{n}\equiv\mathtt{l}=20-\mathtt{n}$$

The events associated to the guard are:

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

and

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

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

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

The leakage is hence given by

$$\begin{array}{ll}\mathtt{H}(\mu(\mathtt{e}^{<1>}),\ldots,\mu(\mathtt{e}^{<n>})) & =\\ \mathtt{H}(\frac{2^{k}-20}{2^{k}},\frac{20}{2^{k}},\ldots,\frac{20}{2^{k}},0,\ldots,0) & =\\ -\frac{2^{k}-20}{2^{k}}\mathtt{log}_2(\frac{2^{k}-20}{2^{k}})-20(\frac{20}{2^{k}}\mathtt{log}_2(\frac{20}{2^{k}})) & \end{array}$$

This function is plotted in figure 1 for $\mathtt{k}=10\ldots20$. The interesting thing in the graph is how it shows that for k around 10 bits the the program is unsafe (more than 2.6 bits of leakage) whereas for k from 15 upwards the program is safe (around 0 bits of leakage).

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
Bound	∞	4.3219	1	16	∞	0	$\mathtt{log(K)}$
Channel Rate	$\downarrow$	$=$	$=$	$=$	$=$	0	$\downarrow$
Capacity	$\mathtt{k}$	4.3219	1	16	k	0	$\mathtt{log(K)}$
Leakage ratio	1	$\leq \frac{\mathtt{H(h)}-4.3219}{\mathtt{H(h)}}$	$\leq \frac{\mathtt{H(h)}-1}{\mathtt{H(h)}}$	1	1	0	$\leq \frac{\mathtt{log(K)}}{2^{\mathtt{k}}}$

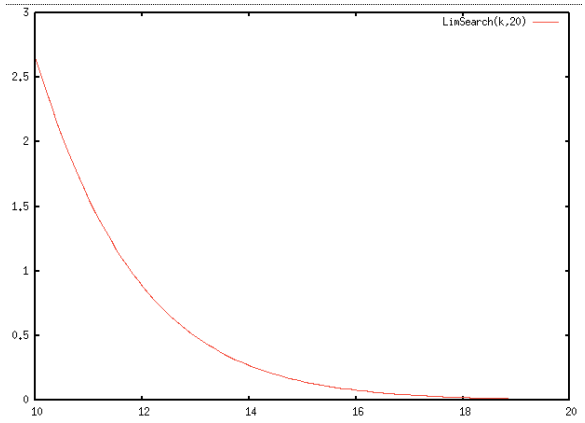

Figure 1: leakage in `l=20; while ( h < l) {l=l-1}`

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...19` and have uniform distribution[6].

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)
{
        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

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

[6]We are ignoring the case where $\mathtt{k} < 5$ where the capacity is less than 4.321928

Notice that

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

henceforth

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

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

This is a 1-bounded channel with constant rate and capacity 1. Notice however that if number of iterations were a function of the secret size, for example by inserting the assignment $\mathtt{N} = \mathtt{size(h)}$, (i.e. $2^{\mathtt{k}}$ the size of the secret) then it becomes a 1 bounded channel with decreasing rate $\frac{1}{2^{\mathtt{k}}}$ 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 $\mathtt{H(M^n)}$ 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[7] and copies that bit into `low`

The variables of interests are:

[7]This is because $\mathtt{m} = 2^{16-\mathtt{n}}$

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

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

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

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

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

becomes

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

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

The reason 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 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 and l<h)
                {
                if (h<= K) flag=ff;
                l=l+1;
                }
```

This loop will terminate if $h \leq K$ and in that case $l = h$.

The event $e^{<i>}$ corresponds to $l = h \wedge h \leq K$, hence $\mu(e^{<i>}) = \frac{1}{K}$. Notice that as the information $h \leq K$ is known by knowing $e^{<i>}$ we conclude that for all i, $H(M^i|e^{<i>}) = 0$.

The leakage of this channel (under uniform distribution) is hence

$H(\frac{1}{K},\ldots,\frac{1}{K}) = \log(K)$.

This is also the channel capacity as it is an upper bound for the entropy of any distribution over $\log(K)$ elements.

4.7 Probabilistic operators

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

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

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

However we could interpret $l = \mathtt{random}(0,1)$ as the program $l = x$ where x is an "unknown input" variable taking value 0 with probability p and 1 with probability $1-p$. Then computing $H(O|L,X)$ gives $H(O|L,X) = H(O|X) = 0$, all uncertainty in the output comes from "the random"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 $H(O|L,X)$; 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 $H(O|L,X)$ comes from the secret. As usual we can simplify the formula to $H(O|X)$ 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 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

$$H(O|X) = \sum \mu(X = x_i)H(O|X = x_i)$$

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

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

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

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

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 `high` is a 4 bits variable 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

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

From $\sum_{1 \leq \mathtt{i} \leq \mathtt{k}} \frac{\mathtt{k}!}{(\mathtt{k}-\mathtt{i})!\mathtt{i}!}\mathtt{i} \gg 2^{\mathtt{k}}$ we deduce that this is an unbounded channel.

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. For example by changing the distribution in `Coin` such that for all i, $\mathtt{Coin}[\mathtt{i}] = 0$ with probability 1 the above program become the unbounded stationary channel studied in section 4.4 whereas if for all i, $\mathtt{Coin}[\mathtt{i}] = 0$ 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 [3] 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[16] and Gray [25]. The idea is to see a non interfering program as a function $\mathtt{F}(\mathtt{h}, \mathtt{l})$ (its denotational semantics) which is constant on the $\mathtt{h}$ component, i.e. for all $\mathtt{h} \neq \mathtt{h}'$, $\mathtt{F}(\mathtt{h}, \mathtt{l}) = \mathtt{F}(\mathtt{h}', \mathtt{l})$. Let's now consider $\mathtt{H}(\mathtt{F}|\mathtt{l})$: because $\mathtt{F}$ is constant on the $\mathtt{h}$ component there will be no uncertainty on $\mathtt{F}$ if we know $\mathtt{l}$, hence $\mathtt{H}(\mathtt{F}|\mathtt{l}) = 0$; on the other side any denotation of a program which satisfy $\mathtt{H}(\mathtt{F}|\mathtt{l}) = 0$ has to be constant on the $\mathtt{h}$ 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 $\mathtt{n} > 0$ what does that mean?

The idea here is that $\mathtt{n}$ 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 [14], Malone and Sullivan[9]. A related argument can be found also in [2].

Suppose the attacker has available a distribution $\mathtt{p}$ 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 $\mathtt{G}(\mathtt{p}) = \sum_{\mathtt{i}} \mathtt{i}\mathtt{p}_{\mathtt{i}}$ where $(\mathtt{p}_i)_{i \in I}$. In case of the uniform distribution $\mathtt{G}(\mathtt{p}) = \frac{\mathtt{n}+1}{2}$. This inspires the information theoretic definition $\mathtt{H}_{\mathtt{G}}(\mathtt{p}) = \frac{2^{\mathtt{H}(\mathtt{p})}+1}{2}$.

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

Massey has shown that $0.7\mathtt{H}_{\mathtt{G}}(\mathtt{p}) \leq \mathtt{G}(\mathtt{p})$ (his precise bound is $\mathtt{G}(\mathtt{p})/\mathtt{H}_{\mathtt{G}}(\mathtt{p}) \leq 2/\mathtt{e}$). This supports the view that secureness 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 $\mathtt{H}(\mathtt{p})$ (again this is $\mathtt{Sec}(\mathtt{M})$).

In the case of the dictionary attack, how good is the lower bound $0.7\mathtt{H}_{\mathtt{G}}(\mathtt{p}) \leq \mathtt{G}(\mathtt{p})$? In an experimental study [9] one million random distributions for a set between 2 and 20 values was generated. These experiments show that the following relation holds:

$0.7\mathtt{H}_{\mathtt{G}}(\mathtt{p}) \leq \mathtt{G}(\mathtt{p}) \leq \mathtt{H}_{\mathtt{G}}(\mathtt{p})$

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

Massey however has shown that there are distributions for which the inequality $\mathtt{G}(\mathtt{p}) \leq \mathtt{H}_{\mathtt{G}}(\mathtt{p})$ doesn't hold; an example is the distribution $\mathtt{p}_1 = 1 - \mathtt{b}/\mathtt{n}, \mathtt{p}_2 = \cdots = \mathtt{p}_{\mathtt{n}} = \mathtt{b}/(\mathtt{n}^2 - \mathtt{n})$. For $\mathtt{n} \to \infty$ we have $\mathtt{G}(\mathtt{p})$ tends to $1 + \mathtt{b}/2$ while $\mathtt{H}(\mathtt{p})$ tends to 1. Because $\mathtt{b}$ is arbitrary we conclude that $\mathtt{G}(\mathtt{p})$ can be arbitrary larger than $\mathtt{H}(\mathtt{p})$.

6. CONCLUSION AND FURTHER WORKS

This work introduced an information theoretical analysis of leakage in looping constructs.

The analysis consists of several notions, like absolute leakage, rate of leakage, channel capacity, leakage ratio.

A classification of loops has been then defined with the aim is to determine which loop presents a security threat.

Several cases studies have been presented with the aim to show that the definitions and classification are useful in individuating security threats and are natural.

Some unanswered questions in this paper are:

1. Is there a good static analysis derivable from this work? As the case studies show the analysis requires some ingenuity, for example to determine which events the $\mathtt{e}^{<\mathtt{i}>}$ 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 [3, 1] some reasonable static analysis may be built.
2. What about 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 require as *observable* intermediate states of execution; 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[12, 19] 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.

7. REFERENCES

[1] 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.

[2] David Clark, Sebastian Hunt, and Pasquale Malacaria. Quantitative analysis of the leakage of confidential data. *Electronic Notes in Theoretical Computer Science*, 59, 2002.

[3] David Clark, Sebastian Hunt, and Pasquale Malacaria. Quantitative information flow, relations and polymorphic types. *Journal of Logic and Computation, Special Issue on Lambda-calculus, type theory and natural language*, 18(2):181–199, 2005.

[4] Michael R. Clarkson, Andrew C. Myers, and Fred B. Schneider. Belief in information flow. In *Proc. 18th IEEE Computer Security Foundations Workshop (CSFW 18)*. IEEE Computer Society Press, 2005.

[5] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley Interscience, 1991.

[6] D.Bell and L. LaPadula. Secure computer systems: Unified exposition and multics interpretation. Technical Report MTR-2997, MITRE Corp, 1997.

[7] D. E. R. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5), May 1976.

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

[9] D.Malone and W. Sullivan. Guesswork and entropy. *IEEE Transactions on Information Theory*, 50(3), March 2004.

[10] 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.

[11] James W. Gray III and Paul F. Syverson. A logical approach to multilevel security of probabilistic systems. *Distributed Computing*, 11(2):73–90, 1998.

[12] S. Isthiaq and P.W. O'Hearn. BI as an assertion language for mutable data structures. In *28th POPL*, pages 14–26, London, January 2001.

[13] Gavin Lowe. Quantifying information flow. In *Proceedings of the Workshop on Automated Verification of Critical Systems*, 2001.

[14] James L. Massey. Guessing and entropy. In *Proc. IEEE International Symposium on Information Theory*, Trondheim, Norway, 1994.

[15] John McLean. Security models and information flow. In *Proceedings of the 1990 IEEE Symposium on Security and Privacy*, Oakland, California, May 1990.

[16] Jonathan Millen. Covert channel capacity. In *Proc. 1987 IEEE Symposium on Research in Security and Privacy*. IEEE Computer Society Press, 1987.

[17] Alessandra Di Pierro, Chris Hankin, and Herbert Wiklicky. Probabilistic confinement in a declarative framework. In Agostino Dovier, Maria Chiara Meo, and Andrea Omicini, editors, *Electronic Notes in Theoretical Computer Science*, volume 48. Elsevier, 2001.

[18] Alessandra Di Pierro, Chris Hankin, and Herbert Wiklicky. Quantitative static analysis of distributed systems. *Journal of Functional Programming*, 2005.

[19] J. Reynolds. Separation logic: a logic for shared mutable data structures, 2002.

[20] J. C. Reynolds. Syntactic control of interference. In *Conf. Record 5th ACM Symp. on Principles of Programming Languages*, pages 39–46, Tucson, Arizona, 1978. ACM, New York.

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

[22] Andrei Sabelfeld and David Sands. Dimensions and principles of declassification. In *Proceedings of the 18th IEEE Computer Security Foundations Workshop*, pages 255–269, Cambridge, England, 2005. IEEE Computer Society Press.

[23] Claude Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423 and 623–656, July and October 1948. Available on-line at http://cm.bell-labs.com/cm/ms/what/shannonday/paper.html.

[24] Dennis Volpano and Geoffrey Smith. A type-based approach to program security. In *Proceedings of TAPSOFT '97 (Colloquium on Formal Approaches in Software Engineering)*, number 1214 in Lecture Notes in Computer Science, pages 607–621, Lille, France, 1997.

[25] James W. Gray, III. Toward a mathematical foundation for information flow security. In *Proc. 1991 IEEE Symposium on Security and Privacy*, pages 21–34, Oakland, CA, May 1991.

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

[27] T. Wittbold. Network of covert channels. In *Proceedings of the 1990 Workshop on the Foundations of Computer Security*, 1990.