Electronic Notes in Theoretical Computer Science 59 No. 3 (2001)
URL: http://www.elsevier.nl/locate/entcs/volume59.html 14 pages

Quantitative Analysis of the Leakage of Confidential Data

David Clark [1]

Department of Computer Science
King's College London

Sebastian Hunt [2]

Department of Computing
City University, London

Pasquale Malacaria [3]

Department of Computer Science
Queen Mary, University of London

Abstract

Basic information theory is used to analyse the amount of confidential information which may be leaked by programs written in a very simple imperative language. In particular, a detailed analysis is given of the possible leakage due to equality tests and **if** statements. The analysis is presented as a set of syntax-directed inference rules and can readily be automated.

1 Introduction

We use basic information theory to analyse the amount of confidential information which may be leaked by programs written in a very simple imperative language (no iteration). We deal with the same issues that are the subject of [VS00] but use different methods.

Our work is motivated by the fact that it is quite common for programs to leak acceptably small amounts of information about sensitive data. The archetype is a program which performs a password check before allowing access to a sensitive resource. While the password check protects the resource, it also

[1] Email: david@dcs.kcl.ac.uk
[2] Email: seb@soi.city.ac.uk
[3] Email: pm@dcs.qmw.ac.uk

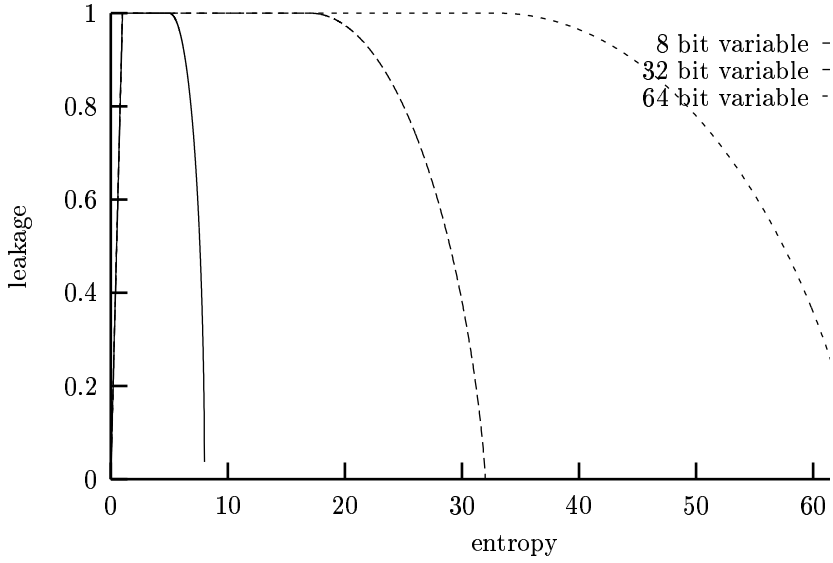

Fig. 1. leakage of a password check as a function of the password entropy

leaks a small amount of information about the password file, since a failed attempt (marginally) narrows the space which an attacker needs to explore. We begin by formalising what is meant by an 'amount of information', using very basic information theory. We go on to develop syntax-directed rules which allow bounds on the amount of information leaked by a program to be automatically determined.

To illustrate the key idea, consider a program which performs a single password check, where the password is stored in a k bit variable. There are 2^k possible values for the password. If passwords are randomly allocated, we can assume a uniform probability distribution for the variable. According to the basic information-theoretic measure of entropy (see sect. 2.1) the variable then 'contains' its maximum information content of k bits (but note that it is perfectly possible in general for a variable which occupies k bits in memory to contain fewer than k bits of information). It will be very difficult to discover the password in one check in this case, especially for large k. However, if the distribution of possible values is made less uniform, by making some values significantly more likely than others, it becomes easier to learn the password by a single test. The analysis presented below hinges on a precise account of how the effectiveness of such a test, measured as the number of bits leaked, varies with k and with the entropy of the variable. Figure 1 plots the maximum possible leakage of a simple equality test against entropy, for a few values of k. Note that as the entropy increases beyond a certain point $(1 + \frac{1}{2}\log(2^k - 1))$, the maximum possible leakage falls to a near zero minimum.

1.1 Related work

The work we describe in this paper is not the first attempt to apply information theory to the analysis of confidentiality properties. The earliest example

of which we are aware is in Denning's book [Den82] where she gives some examples of how information theory may be used to calculate the leakage of confidential data via some imperative language program constructs. However she does not develop a systematic, formal approach to the question as we do in this paper. Another early example is that of Jonathan Millen [Mil87] which points to the relevance of Shannon's use of finite state systems in the analysis of channel capacity. More recent is the work of James W. Gray [WG91], which develops a quite sophisticated operational model of computation and relates non-interference properties to information theoretic properties. However, neither of these deals with the analysis of programming language syntax, as we do here. By contrast, much more has been done with regard to syntax directed analysis of non-interference properties. See particularly the work of Sands and Sabelfeld [SS99,SS00].

2 Leakage

We suppose that the variables Var of a program are partitioned by two vectors:

$\vec{\mathtt{x}}_{\mathrm{H}}$ the *high* security variables

$\vec{\mathtt{x}}_{\mathrm{L}}$ the *low* security variables

We assume fixed some probability distribution p on Σ: this is the distribution on the *initial* values taken by σ, ie the inputs to the program being analysed. We note here that the analysis developed in sect. 3 does not require p to be fully specified.

Our concern in this paper is with the amount of information initially in $\vec{\mathtt{x}}_{\mathrm{H}}$ which an "attacker" can learn by observing a run of a program. The attacker has only low security privileges: it is unable to observe $\vec{\mathtt{x}}_{\mathrm{H}}$ directly but *is* able to observe the initial and final values of $\vec{\mathtt{x}}_{\mathrm{L}}$. For $\mathtt{x} \in \mathrm{Var}$, the *leakage into* $\mathtt{x}$ is intended to model the amount that can be learnt about the initial values of $\vec{\mathtt{x}}_{\mathrm{H}}$ from the final value of $\mathtt{x}$. Our intention is that if the leakage is zero the program satisfies a non-interference property.

2.1 The use of information theory

As mentioned above, the use of information theory in the analysis of security is not novel. Nonetheless, its relevance to the problem we address may not be immediately obvious. We attempt here to motivate its use by appeal to an intuitively reasonable operational model of a class of possible attackers.

We start by recalling Shannon's basic definition. [Sha48]. For our purposes, a random variable in a finite set V is a variable v taking values in V and equipped with a probability distribution. $P(\mathrm{v} = v)$ is the probability that v takes the value v. Shannon defines the following measure of the information

content of V, which he calls *entropy*:

$$H(\mathrm{V}) \stackrel{\text{def}}{=} \sum_v P(\mathrm{V}=v) \log \frac{1}{P(\mathrm{V}=v)} \tag{1}$$

Here and in rest of the paper, log is to the base 2. Suppose we determine (somehow) that, according to this measure, the information content of the confidential inputs to a program, minus the amount of information leaked, is n bits. What does this tell us about how hard it is to determine our secret? Of course, it depends on what the attacker is able to do. We suppose that the attacker is able to guess at the value of the secret V and has an oracle which can (cheaply) confirm or refute the guess. It is crucial here that the oracle can only answer questions of the form "is $\mathrm{V} = v$?" and not more general Yes/No questions. Supposing that the attacker knows the distribution for V, the best strategy is clearly to start by guessing the most likely v and proceed in decreasing order of likelihood until the secret is determined. The relevance of entropy is then given by the following result, due to James L. Massey [Mas94]: the average number of guesses required to guess the secret is at least $(1/4)2^n + 1$.

2.2 Random variables and program variables

Suppose $f : A \to B$. By $\mathrm{B} \stackrel{\text{def}}{=} f(\mathrm{A})$ we mean that B is the random variable in B such that

$$P(\mathrm{B}=b) = \sum_{a \in f^{-1}b} P(\mathrm{A}=a)$$

Suppose given random variable A in A, functions $f : A \to B$ and $g : A \to C$, and the associated random variables $\mathrm{B} \stackrel{\text{def}}{=} f(\mathrm{A})$ and $\mathrm{C} \stackrel{\text{def}}{=} g(\mathrm{A})$. We derive new random variables from B and C in two main ways:

pairing We write $<\mathrm{B}, \mathrm{C}>$ for the random variable $<f, g>(\mathrm{A})$. This is extended to vectors of higher arity in the obvious way. We usually write $P(\mathrm{B}_1 = b_1, \mathrm{B}_2 = b_2)$ instead of $P(<\mathrm{B}_1, \mathrm{B}_2> = <b_1, b_2>)$.

specialisation Given $c \in C$, we write $\mathrm{B}_{\mathrm{C}=c}$ for the random variable such that

$$P(\mathrm{B}_{\mathrm{C}=c} = b) = P(\mathrm{B}=b | \mathrm{C}=c) \stackrel{\text{def}}{=} \frac{P(\mathrm{B}=b, \mathrm{C}=c)}{P(\mathrm{C}=c)}$$

Let S_0 be the random variable in Σ such that $P(\mathrm{S}_0 = \sigma) = p(\sigma)$. All the random variables of interest in the remaining sections will be functions of S_0. For $\mathtt{x} \in \mathrm{Var}$, we let $\mathrm{X} \stackrel{\text{def}}{=} (\lambda\sigma.\sigma\mathtt{x})(\mathrm{S}_0)$. Hence $P(\mathrm{X}=n)$ is calculated as follows:

$$\begin{aligned} P(\mathrm{X}=n) &= P((\lambda\sigma.\sigma\mathtt{x})(\mathrm{S}_0) = n) \\ &= P(\mathrm{S}_0\mathtt{x} = n) \\ &= \sum_{\forall\sigma.\sigma\mathtt{x}=n} p(\sigma) \end{aligned}$$

In contexts where a state transformer $f : \Sigma \to \Sigma$ is understood, we also define the *output* random variable $\mathrm{X}' \stackrel{\text{def}}{=} (\lambda\sigma.f\sigma\mathtt{x})(\mathrm{S}_0)$. With the vectors $\vec{\mathtt{x}}_{\mathrm{H}}$ and $\vec{\mathtt{x}}_{\mathrm{L}}$ of

high and low security variables, we associate the random variables:

$$\mathrm{H} \stackrel{\text{def}}{=} (\lambda\sigma.\sigma^*(\vec{\mathtt{x}}_{\mathrm{H}}))(\mathrm{S}_0)$$
$$\mathrm{L} \stackrel{\text{def}}{=} (\lambda\sigma.\sigma^*(\vec{\mathtt{x}}_{\mathrm{L}}))(\mathrm{S}_0)$$

where σ^* is the element-wise extension of σ to a vector of variables. We let h and l range over the values taken by H and L, respectively.

2.3 Definition of leakage

Our definition of leakage can be seen as a specialisation of a proposal made by Gray [WG91] based on the notion of *mutual information* between two systems. Here we need the notion of *conditional entropy*. The conditional entropy of A given B is:

$$H(\mathrm{A}|\mathrm{B}) \stackrel{\text{def}}{=} \sum_b P(\mathrm{B}=b)H(\mathrm{A}_{\mathrm{B}=b}) \tag{2}$$

Re-stated in these terms and specialised to the current setting, Gray's proposal is to define the amount of information leaked into $\mathtt{x}$ as:

$$I(\mathrm{H}, \mathrm{X}' \mid \mathrm{L}) \stackrel{\text{def}}{=} H(\mathrm{X}' \mid \mathrm{L}) - H(\mathrm{X}' \mid \mathrm{H}, \mathrm{L}) \tag{3}$$

The intuition is that $I(\mathrm{H}, \mathrm{X}' \mid \mathrm{L})$ is the amount of information that H and X′ have in common given that L is known and it is defined to be $H(\mathrm{X}' \mid \mathrm{L})$, i.e. the uncertainty in the final value of $\mathtt{x}$ given full knowledge of the low inputs, less $H(\mathrm{X}' \mid \mathrm{H}, \mathrm{L})$, the 'intrinsic uncertainty' in the final value of X, that is, the uncertainty about the final value of $\mathtt{x}$ which would remain even given full knowledge of the initial values of all input variables. In the current setting, $H(\mathrm{X}' \mid \mathrm{H}, \mathrm{L})$ is always 0, because our programming language is deterministic (the state transformer f is a function). This is easy to check.

Let $f : \Sigma \to V$ and let $\mathrm{V} \stackrel{\text{def}}{=} f(\mathrm{S}_0)$. Define:

$$\mathcal{L}(\mathrm{V}) \stackrel{\text{def}}{=} H(\mathrm{V} \mid \mathrm{L}) = \sum_l P(\mathrm{L}=l)H(\mathrm{V}_{\mathrm{L}=l}) \tag{4}$$

Our use of V is meant to suggest a random variable that takes on the values of expressions as well as (like X′) those of variables. For a program with state transformer $f : \Sigma \to \Sigma$, we define the leakage into $\mathtt{x}$ to be $\mathcal{L}(\mathrm{X}')$.

2.4 Observations on the definition

Our choice of measure implies that the total amount of confidential information available to be leaked is given by:

$$H(\mathrm{H} \mid \mathrm{L}) = \sum_l P(\mathrm{L}=l)H(\mathrm{H}_{\mathrm{L}=l}) \tag{5}$$

and, indeed, this is an upper bound for $\mathcal{L}(\mathrm{L}')$. We also note that when H and L are independent: $H(\mathrm{H} \mid \mathrm{L}) = H(\mathrm{H})$.

$$c \in \text{Com} \qquad \mathtt{x} \in \text{Var} \qquad e \in \text{Exp} \qquad n \in \text{Num}$$

$$c ::= \textbf{skip} \;\mid\; \mathtt{x} := e \;\mid\; c_1 \,;\, c_2 \;\mid\; \textbf{if } e \textbf{ then } c_1 \textbf{ else } c_2$$

$$e ::= \mathtt{x} \;\mid\; \mathtt{n} \;\mid\; e_1 + e_2 \;\mid\; e_1 - e_2 \;\mid\; e_1 * e_2 \;\mid$$
$$\neg e \;\mid\; e_1 \wedge e_2 \;\mid\; e_1 < e_2 \;\mid\; e_1 \leq e_2 \;\mid\; e = \mathtt{n}$$

Table 1
the language

Rather than measuring leakage we could have chosen to measure 'secureness':

$$\mathcal{S}(\text{V}) \stackrel{\text{def}}{=} H(\text{H} \mid \text{L}, \text{V}) = \sum_{l,v} P(\text{L} = l, \text{V} = v) H(\text{H}_{<\text{L},\text{V}>=<l,v>})$$

Intuitively, this is the uncertainty in H given full knowledge of L and V: the more uncertain the attacker, the more secure the program. It is easy to show that $\mathcal{S}(\text{V})$ is just the part of $H(\text{H} \mid \text{L})$ which is not leaked:

$$\mathcal{S}(\text{V}) = H(\text{H} \mid \text{L}) - \mathcal{L}(\text{V})$$

3 The Analysis

The language we analyse is presented in table 1. We assume a standard state-transformer semantics of the following form:

$$V \stackrel{\text{def}}{=} \{0,1\}^k \text{ (vectors of } k \text{ bits)}$$
$$\sigma \in \Sigma \stackrel{\text{def}}{=} \text{Var} \to V$$
$$C[\![\cdot]\!] : \text{Com} \to \Sigma \to \Sigma$$
$$E[\![\cdot]\!] : \text{Exp} \to \Sigma \to V$$

To simplify the presentation we have opted for a single k-bit integer data type, treating 0 as false and 1 as true wherever boolean values are required. We assume a basic well-formedness check that guarantees expressions do not take values other than 0, 1 in boolean contexts. The + operator in the semantics should be understood as an operation on binary words, such as twos-complement addition (our analysis is too crude for the exact choice to matter). With the above provisos, the details of the semantics are routine and are omitted.

This is clearly a very limited language. The most serious limitation is that it provides no form of iteration. It is fairly straightforward to extend our analysis to cope with bounded iteration but unbounded iteration raises more fundamental issues. The point is discussed further in sect. 3.8. A less obvious limitation is that the language's equality test requires one parameter to be a constant. However, the analysis can be extended to cope with a general

equality test, and this point is also discussed in sect. 3.8.

3.1 How the analysis works

The ultimate aim of analysing a program is to place an upper bound on $\mathcal{L}(\mathrm{x}')$ for each low-security variable $\mathtt{x}$. We aim to analyse programs in a compositional and incremental fashion. We start with a partial specification of a probability distribution on the inputs (s_0 - see sect. 2.2). More precisely, we start by specifying, for each variable $\mathtt{x}$, an interval which includes $H(\mathrm{x}|\mathrm{l})$. To analyse programs directly for the values of $H(\mathrm{x}'|\mathrm{l})$ would be problematic because it would require knowledge of the distribution of input values for the low variables $\vec{\mathtt{x}}_{\mathrm{L}}$ and such knowledge might not be available. Worse still, the attacker may actually be *providing* the low inputs. The analysis we present avoids this problem by calculating bounds on the best and worst cases for leakage over the complete set of possible input values for the low variables. These bounds are defined as follows:

(6) $\mathcal{L}^-(\mathrm{v}) \stackrel{\text{def}}{=} \min_l H(\mathrm{v}_{\mathrm{L}=l})$

(7) $\mathcal{L}^+(\mathrm{v}) \stackrel{\text{def}}{=} \max_l H(\mathrm{v}_{\mathrm{L}=l})$

The minima and maxima here are taken over all l for which $H(\mathrm{v}_{\mathrm{L}=l})$ is defined (that is, all l such that $P(\mathrm{l} = l) > 0$). These quantities bound $\mathcal{L}(\mathrm{v})$:

Proposition 3.1

$$\mathcal{L}^-(\mathrm{v}) \leq \mathcal{L}(\mathrm{v}) \leq \mathcal{L}^+(\mathrm{v})$$

3.2 Inference rules

The analysis is presented as a collection of inference rules in tables 2 and 3. The rule NoAss for commands makes use of an auxiliary function $X_{:=}$ on commands: this just returns the set of all variables $\mathtt{x}$ which occur as the target of an assignment anywhere in the command; its definition is obvious for the simple language under consideration and is omitted.

We note that these rules can readily be presented in functional form and automated. The rules Eq(2) and Eq(3) require some simple numerical methods, discussed briefly in sect. 3.5.

In these rules, Γ is a partial function from Var to closed intervals $[a, b]$ in the range $0 \leq a \leq b \leq k$. We write $\mathtt{x} : [a, b]$ to mean the partial function sending $\mathtt{x}$ to $[a, b]$ and having domain $\{\mathtt{x}\}$. Γ_1, Γ_2 denotes the union of partial functions with disjoint domains.

Informally, the meaning of an entailment $\Gamma \vdash c \downarrow \mathtt{x} : [a, b]$ is that the program c modifies $\mathtt{x}$ in such a way that $\mathcal{L}^-(\mathrm{x}') \leq a$ and $b \leq \mathcal{L}^+(\mathrm{x}')$. To formalise this, given $f : \Sigma \to \Sigma$, for any $\mathtt{x} \in \mathrm{Var}$, let $\mathrm{x}^f \stackrel{\text{def}}{=} (\lambda\sigma. f\sigma\mathtt{x})(\mathrm{s}_0)$. Say that $f \models \mathtt{x} : [a, b]$ if $\mathcal{L}^-(\mathrm{x}^f) \leq a$ and $b \leq \mathcal{L}^+(\mathrm{x}^f)$. Say further, that $f \models \Gamma$ if $f \models \Gamma(\mathtt{x})$ for all $\mathtt{x} \in \mathrm{dom}(\Gamma)$. Then the meaning of $\Gamma \vdash c \downarrow \Gamma'$ is given by the following theorem.

$$\text{EConj}\frac{\Gamma \vdash e : [a_1, b_1] \qquad \Gamma \vdash e : [a_2, b_2]}{\Gamma \vdash e : [\max(a_1, a_2), \min(b_1, b_2)]}$$

$$\text{Const}\frac{}{\Gamma \vdash \mathtt{n} : [0, 0]} \qquad \text{Var}\frac{}{\Gamma, \mathtt{x} : [a, b] \vdash \mathtt{x} : [a, b]}$$

$$\text{And}\frac{\Gamma \vdash e_i : [\_, b_i]}{\Gamma \vdash (e_1 \wedge e2) : [0, b_1 +_1 b_2]} \qquad \text{Neg}\frac{\Gamma \vdash e : [a, b]}{\Gamma \vdash \neg e : [a, b]}$$

$$\text{Plus}\frac{\Gamma \vdash e_i : [\_, b_i]}{\Gamma \vdash (e_1 + e2) : [0, b_1 +_k b_2]} \qquad \text{Eq(1)}\frac{\Gamma \vdash e : [\_, b]}{\Gamma \vdash (e = \mathtt{n}) : [0, \min(1, b)]}$$

$$\text{Eq(2)}\frac{\Gamma \vdash e : [a, \_]}{\Gamma \vdash (e = \mathtt{n}) : [\mathcal{B}(q), 1]} \quad q \leq \tfrac{1}{2^k},\ \mathcal{U}_k(q) \leq a$$

$$\text{Eq(3)}\frac{\Gamma \vdash e : [a, \_]}{\Gamma \vdash (e = \mathtt{n}) : [0, \mathcal{B}(q)]} \quad \tfrac{1}{2^k} \leq q \leq \tfrac{1}{2},\ \mathcal{U}_k(q) \leq a$$

Table 2
leakage inference for expressions

$$\text{CConj}\frac{\Gamma \vdash c \downarrow x : [a_1, b_1] \qquad \Gamma \vdash c \downarrow x : [a_2, b_2]}{\Gamma \vdash c \downarrow x : [\max(a_1, a_2), \min(b_1, b_2)]}$$

$$\text{Join}\frac{\Gamma \vdash c \downarrow \Gamma_1 \qquad \Gamma \vdash c \downarrow \Gamma_2}{\Gamma \vdash c \downarrow \Gamma_1, \Gamma_2} \quad \mathrm{dom}(\Gamma_1) \cap \mathrm{dom}(\Gamma_2) = \emptyset$$

$$\text{Ass}\frac{\Gamma \vdash e : [a, b]}{\Gamma \vdash \mathtt{x} := e \downarrow \mathtt{x} : [a, b]} \qquad \text{Seq}\frac{\Gamma \vdash c_1 \downarrow \Gamma' \qquad \Gamma' \vdash c_2 \downarrow \Gamma''}{\Gamma \vdash c_1 \,;\, c_2 \downarrow \Gamma''}$$

$$\text{NoAss}\frac{}{\Gamma, \mathtt{x} : [a, b] \vdash c \downarrow \mathtt{x} : [a, b]} \quad \mathtt{x} \notin X_{:=}(c)$$

$$\text{If(1)}\frac{\Gamma \vdash e : [\_, b] \qquad \Gamma \vdash c_i \downarrow \mathtt{x} : [\_, b_i]}{\Gamma \vdash \textbf{if } e \textbf{ then } c_1 \textbf{ else } c_2 \downarrow \mathtt{x} : [0, b +_k b_1 +_k b_2]}$$

$$\text{If(2)}\frac{\Gamma \vdash e : [0, 0] \qquad \Gamma \vdash c_i \downarrow \mathtt{x} : [a_i, b_i]}{\Gamma \vdash \textbf{if } e \textbf{ then } c_1 \textbf{ else } c_2 \downarrow \mathtt{x} : [\min(a_1, a_2), \max(b_1, b_2)]}$$

Table 3
leakage inference for commands

Theorem 3.2 *For all commands c and for all $f : \Sigma \to \Sigma$, if $\Gamma \vdash c \downarrow \Gamma'$ then*

$$f \models \Gamma \Rightarrow f; C[\![c]\!] \models \Gamma' \tag{8}$$

Proof: by induction on the height of the derivation of $\Gamma \vdash c \downarrow \Gamma'$.

We omit the details of the proof for reasons of space.

Corollary 3.3 *The analysis gives correct results for any initial Γ such that $[\mathcal{L}^-(\mathrm{X}), \mathcal{L}^+(\mathrm{X})] \subseteq \Gamma(\mathtt{x})$ for all $\mathtt{x} \in \mathrm{dom}(\Gamma)$ (recall that X is the random variable describing the* input *distribution for $\mathtt{x}$).*

Note that $\mathcal{L}^-(\mathrm{X}) = \mathcal{L}^+(\mathrm{X}) = 0$ for low-security $\mathtt{x}$ and, when the high-security and low-security inputs are independent, $\mathcal{L}^-(\mathrm{Y}) = \mathcal{L}^+(\mathrm{Y}) = H(\mathrm{Y})$ for high-security $\mathtt{y}$.

The expression rules for $-, *, <, \leq$ are essentially the same as those for $+$ and $\wedge$ and are omitted. Many of the rules are crudely conservative, based simply on the observation that the entropy of a function of a vector of random variables cannot be less than 0 and cannot exceed the sum of the entropies of the component random variables. This applies to And, Plus and If(1). The less obvious of the remaining rules are discussed below.

3.3 Logical rules

EConj, CConj and Join are generic logical rules allowing the results of sub-analyses to be conjoined. Their correctness is more or less obvious.

3.4 Assignment

The rule Ass is immediate since assignment makes the random variable for the variable coincide with the random variable associated with the expression. The rule NoAss reflects the fact that the leakage into a variable is unchanged by any command which leaves that variable unaltered (though see sect. 3.8 on this point).

3.5 Analysis of equality tests

The analysis of a test of the form $e = \mathtt{n}$ is determined by the Eq rules.

Eq(1) is straightforward. It is justified by the simple observations that the total amount of information which can be leaked by a boolean expression cannot exceed 1 and, since the meaning of $e = \mathtt{n}$ is solely a function of the meaning of e, the amount leaked by $e = \mathtt{n}$ cannot exceed the amount leaked by e.

Eq(2) and Eq(3) are justified by the answer to the following question. Suppose that V is a k-bit random variable (ie, there are at most 2^k possible values which V can take). Now suppose that $P(\mathrm{V} = n) = q$, for some q. What is the maximum possible value for the entropy $H(\mathrm{V})$?

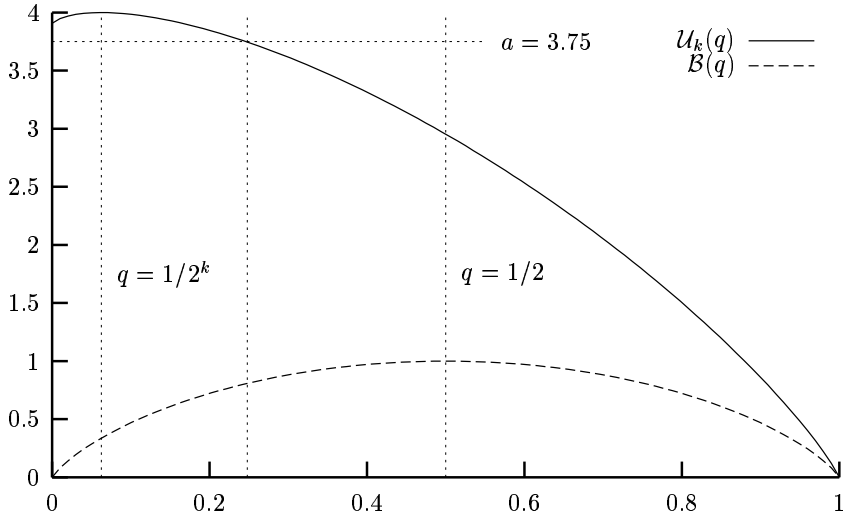

Fig. 2. the upper entropy for q in 4 bits

We call this maximum the *upper entropy for q in k bits*, denoted $\mathcal{U}_k(q)$. Since entropy is maximised by uniform distributions, the maximum value possible for $H(\mathrm{V})$ is obtained in the case that $P(\mathrm{V} = n')$ is uniformly distributed for all $n' \neq n$. There are $2^k - 1$ such n' and applying the definition of H (eqn. 1) immediately gives.

$$\mathcal{U}_k(q) \stackrel{\text{def}}{=} q \log \frac{1}{q} + (1-q) \log \frac{2^k - 1}{1-q} \tag{9}$$

The rules Eq(2) and Eq(3) are based on the observation that q is constrained to have a value such that $\mathcal{L}^-(\mathrm{V}) \leq \mathcal{U}_k(q)$. Once we have determined a range of possible values for q, we can compute corresponding bounds on the entropy for (the random variable determined by) $e = n$ using the formula for the entropy of a 2-element space where one element has probability q:

$$\mathcal{B}(q) \stackrel{\text{def}}{=} q \log \frac{1}{q} + (1-q) \log \frac{1}{1-q} \tag{10}$$

The idea is illustrated by the example shown in fig. 2. This plots $\mathcal{U}_k(q)$ and $\mathcal{B}(q)$ against q for $k = 4$ and shows that for a lower entropy bound of $a = 3.75$, q is bounded by $0 \leq q \leq 0.25$ (the precise upper bound is slightly lower than this).

It is easily seen that $\mathcal{B}(q)$ achieves its maximum value of 1 when $q = 1/2$ and $\mathcal{U}_k(q)$ achieves its maximum value of k when $q = 1/2^k$. This leaves two regions of interest: $q \leq 1/2^k$ and $1/2^k \leq q \leq 1/2$, corresponding to the rules Eq(2) and Eq(3), respectively. To find maximum and minimum q in these regions we need to solve equations of the form $\mathcal{U}_k(q) - a = 0$ and, for this, simple numerical techniques suffice [LF89]. (In fact, as k increases, $\mathcal{U}_k$ becomes close to linear and so a very simple geometrical approximation may be adequate.)

3.6 Analysis of if-then-else

As mentioned above, If(1) is very conservative. We note that the contribution of the entropy of the condition e corresponds to what Denning calls an *implicit* information flow.

If(2) applies in the case that the boolean condition has zero entropy. This means that the value of the condition is completely independent of the values of the high-security inputs. Thus, the choices of low-security inputs which minimise and maximise the overall leakage of the program will each select just one of the two branches and it is safe to take the smaller of the lower bounds and the larger of the upper bounds.

3.7 Example

Let c be the command **if** $\mathtt{y} = 0$ **then** $\mathtt{x} := 0$ **else** $\mathtt{x} := 1$ with $\mathtt{y}$ high-security and $\mathtt{x}$ low-security. Suppose that $k = 32$ and the input distribution makes $\mathtt{y}$ uniform over its 2^{32} possible values and independent of $\mathtt{x}$. Thus we can analyse c starting with $\Gamma_0 = \{\mathtt{x} : [0,0], \mathtt{y} : [32,32]\}$. The rules presented above are easily seen to derive:

$$\Gamma_0 \vdash \mathtt{y} = 0 : [\epsilon, \epsilon]$$

where $\epsilon = \mathcal{B}(1/2^{32}) \approx 7.8 \times 10^{-7}$. (The final rule is EConj, with premises given by Eq(2) and Eq(3).) Thus, using If(1), we derive:

$$\Gamma_0 \vdash c \downarrow \mathtt{x} : [0, \epsilon]$$

3.8 Improvements to the analysis

The first improvement that can readily be made is to record in Γ explicit bounds on the cardinality of the space over which the random variables are defined. Currently the rules implicitly assume the bound 2^k where k is the length of the primitive data type. Specifying a bound explicitly would allow us to deal well with input distributions which are uniform, or near uniform, but sparse (assigning probability 0 to many of the 2^k possible inputs). It would also be of help in generalising the Eq rules to arbitrary equality tests of the form $e_1 = e_2$. The issue here is that, without bounds on the cardinalities of the ranges of the e_i, we may naively make the worst case assumption that q could be uniformly distributed over the entire diagonal, of size 2^k. In fact we can do much better than this in many cases where the entropy of one of the expressions is very low, by calculating an upper bound on the size of the set which can carry a given q.

We can also do better for **if** statements. The rule If(2) is weak because it takes no account of the relative probabilities of either branch being chosen. Bounds on the probabilities will in many cases be available (provided, for example, by the analysis of equality tests). To illustrate the weakness of If(2), let c' be the command **if** $\mathtt{y} = 0$ **then** $\mathtt{x} := \mathtt{y}$ **else** $\mathtt{x} := 1$, This is semantically

equivalent to c in sect. 3.7 but the best we can derive for c' is the totally uninformative:

$$\Gamma_0 \vdash c' \downarrow \mathtt{x} : [0, 32]$$

The problem is caused by the statement $\mathtt{x} := \mathtt{y}$ which, in isolation, would leak all the information in $\mathtt{y}$ into $\mathtt{x}$. But, in the context of this **if** statement, it actually leaks *no* information. A careful analysis of conditionals gives the following result.

Proposition 3.4 *Let c be the command* **if** e **then** c_1 **else** c_2. *Let* V *be a random variable in* Σ, *let* $\mathrm{B} \stackrel{\text{def}}{=} [\![e]\!](\mathrm{V})$, $\mathrm{T} \stackrel{\text{def}}{=} (\lambda\sigma.[\![c_1]\!]\sigma\mathtt{x})(\mathrm{V})$, $\mathrm{F} \stackrel{\text{def}}{=} (\lambda\sigma.[\![c_2]\!]\sigma\mathtt{x})(\mathrm{V})$, $\mathrm{I} \stackrel{\text{def}}{=} (\lambda\sigma.[\![c]\!]\sigma\mathtt{x})(\mathrm{V})$. *Let* $q = P(\mathrm{B} = 1)$ *(hence* $1 - q = P(\mathrm{B} = 0)$*). Then*

$$0 \leq H(\mathrm{I}) - (qH(\mathrm{T}_{\mathrm{B}=1}) + (1-q)H(\mathrm{F}_{\mathrm{B}=0})) \leq H(\mathrm{B}) \tag{11}$$

An alternative (but more complicated) version of If(2) can be derived using this result, giving much better results for examples such as c'. This result can also be used to decompose a non-uniform input distribution into two more nearly uniform distributions on a pair of subsets which partition the domain, and then to combine the results of separate analyses for the partitions (here B would characterise the property which partitions the domain).

Finally, we briefly address the thorny issue of iteration. It would be possible, on the basis of the existing rules, to handle bounded forms of iteration, simply by treating them as nested **if** statements of finite depth. In many security-sensitive contexts, bounded iteration may be sufficient and so it would be worthwhile to pursue the details of such an extension. However, ultimately we wish to be able usefully to analyse quite general programs and so we need to deal with unbounded iteration. But, of course, this takes us outside the simple state-transformer semantics we assume here, due to the possibility of non-termination. This raises a fundamental theoretical problem since (at a minimum) it becomes necessary to define a measure of information which makes sense for the images of partial functions. We conjecture that the formal definition of entropy can be applied, *mutatis mutandis*, in the case when the 'distributions' may sum to less than 1, and that this would give a reasonable measure (intuitively, non-termination is not something which can be observed, rather it is the *absence* of an observation). Even if this is true, significant problems remain. Firstly, the possibility of non-termination would invalidate the NoAss rule: using high-security inputs to affect termination, a **while** loop could leak information even without making any variable assignments. Secondly, we believe that, in the presence of iteration (even bounded iteration) it becomes more appropriate to consider the *rate* at which information may be leaked, rather than the absolute amount leaked. This is the subject of ongoing research.

4 Conclusions and Future Work

We have presented preliminary results in an ongoing attempt to develop a general and practically useful analysis for confidentiality properties. Our results deal with programs which may leak some confidential information and our concern is to determine bounds on the *quantity* of information leaked. Like others before us, we have applied information theory to this problem. Where our work differs from previous work is in our emphasis on analysis which may be *automated* and which, therefore, must be defined directly in terms of the *syntax* of programs (though, of course, correctness is established with respect to the semantics of the language). Unlike Volpano and Smith in [VS00] we are concerned not with establishing an asymptotic limit but with quantifying the actual amount leaked. Our approach is able to give good results for programs which leak small amounts of information via equality tests. We are, as yet, unable to deal in a satisfactory way with unbounded iteration. Our model of what attackers can observe is too simple to address more subtle forms of leakage such as can result from timing properties.

Clearly, there is much to be done. As priorities, we wish to broaden the work described above to more realistic models of what can be observed by an attacker, and to move from the emphasis on absolute leakage to the analysis of rates of leakage as in [VS00]. This suggests a shift from a simple denotational semantic setting to an operational one (or a more sophisticated denotational one). In particular, it will be interesting to see if the operational setting of [SS00] can be adapted.

References

[Den82] D. E. R. Denning. *Cryptography and Data Security*. Addison-Wesley, 1982.

[LF89] Richard L.Burden and J. Douglas Faires. *Numerical Analysis*. PWS-KENT, 1989. ISBN 0-534-93219-3.

[Mas94] James L. Massey. Guessing and entropy. In *Proc. IEEE International Symposium on Information Theory*, Trondheim, Norway, 1994.

[Mil87] Jonathan Millen. Covert channel capacity. In *Proc. 1987 IEEE Symposium on Research in Security and Privacy*. IEEE Computer Society Press, 1987.

[Sha48] Claude Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423 and 623–656, July and October 1948. Available online at http://cm.bell-labs.com/cm/ms/what/shannonday/paper.html.

[SS99] Andrei Sabelfeld and David Sands. A per model of secure information flow in sequential programs. In *Proc. European Symposium on Programming*, Amsterdam, The Netherlands, March 1999. ACM Press.

[SS00] Andrei Sabelfeld and David Sands. Probabilistic noninterference for multi-threaded programs. In *Proc. 13th IEEE Computer Security Foundations Workshop*, Cambridge, England, July 2000. IEEE Computer Society Press.

[VS00] Dennis Volpano and Geoffrey Smith. Verifying secrets and relative secrecy. In *Proc. 27th ACM Symposium on Principles of Programming Languages*, pages 268–276, Boston MA, Jan 2000.

[WG91] 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.