All Secrets Great and Small

Delphine Demange[1] and David Sands[2]

[1] University of Rennes 1, France
[2] Chalmers University of Technology, Sweden

Abstract. Tools for analysing secure information flow are almost exclusively based on ideas going back to Denning's work from the 70's. This approach embodies an imperfect notion of security which turns a blind eye to information flows which are encoded in the termination behaviour of a program. In exchange for this weakness many more programs are deemed "secure", using conditions which are easy to check. Previously it was thought that such leaks are limited to at most one bit per run. Recent work by Askarov et al (ESORICS'08) offers some bad news and some good news: the bad news is that for programs which perform output, the amount of information leaked by a Denning style analysis is not bounded; the good news is that if secrets are chosen to be sufficiently large and sufficiently random then they cannot be effectively leaked at all. The problem addressed in this paper is that secrets cannot always be made sufficiently large or sufficiently random. Contrast, for example, an encryption key with an "hasHIV"-field of a patient record. In recognition of this we develop a notion of *secret-sensitive noninterference* in which "small" secrets are handled more carefully than "big" ones. We illustrate the idea with a type system which combines a liberal Denning-style analysis with a more restrictive system according to the nature of the secrets at hand.

1 Introduction

Most tools for analysing information flow in programs such as Jif [MZZ+08] and FlowCaml [Sim03] build upon ideas going back to Denning's work from the 70's [DD77]. These systems enforce an imperfect notion of information flow which has become known as *termination-insensitive noninterference* (TINI). Under this version of noninterference, information leaks are permitted if they are transmitted purely by the program's termination behaviour (i.e., whether it terminates or not). This imperfection is the price to pay for having a security condition which is relatively liberal (e.g. allowing while-loops whose termination may depend on the value of a secret) and easy to check.

How bad is termination-insensitive noninterference? Previously there have been informal arguments that termination-insensitive noninterference leaks at most one bit: either a program terminates or it does not, so at most one bit of information can be encoded in the termination state. However, recent work by Askarov *et al* [AHSS08] shows that for programs which perform output, an arbitrary amount of information can be leaked. The following program outputs an ascending sequence of natural numbers on a public channel until the secret has been output, at which point it goes into a silent loop:

G. Castagna (Ed.): ESOP 2009, LNCS 5502, pp. 207–221, 2009.
© Springer-Verlag Berlin Heidelberg 2009

```
for i = 0 to maxNat (
    output i on public_channel
    if (i = secret) then (while true do skip)
)
```

At the very least we can say that at each output step, the observer is able to narrow down the possible values of the secret. This program (in suitable variants) is accepted as secure by state-of-the-art information flow analysis tools such as Jif [MZZ+08], FlowCaml [Sim03], and the SPARK Examiner [BB03, CH04].

Askarov *et al* formalise the notion of termination-insensitive noninterference and show that although termination-insensitive noninterference can leak an arbitrary amount of information, it cannot do so any more efficiently than the above example. The revised intuition for programs performing public output is that the number of possible "termination states" that can be used to encode information is of the order of the number of public outputs performed by the program – since the program can diverge after 0 outputs, after 1 output, after 2 outputs, etc. This means that to leak n bits of information the program needs to perform 2^n outputs.

For Denning-style analyses this means that if secrets are sufficiently large and sufficiently random then programs are *computationally secure* in the sense that the probability of the attacker guessing the secret after observing a polynomial number of outputs (again, in the size of the secret) gives only a negligible advantage over guessing the secret without running the program.

What does this mean for information flow analysis in practice? Whereas previously the imperfections of a Denning-style analysis were viewed as a reasonable tradeoff between ease of analysis versus degree of security, we believe that in the light of [AHSS08] we need a different perspective. The leak caused by termination-insensitivity is only acceptable for sufficiently large and random secrets. But secrets, in general, are not always parametric: one cannot always freely choose to make a secret larger and more random. For example, an application cannot decide that a credit card CCV number should be made larger. An encryption key, on the other hand, might be something that the application can control, and decide to scale up.

In this paper we consider the information flow problem in an arbitrary multi-level security lattice. We present a way (Section 2) of refining each security level in an information-flow lattice into two levels: *big secrets*, that are sufficiently large and randomized to abide some leakage, and *small secrets*, for which even slow leakage is unacceptable. Then, we define a two-level noninterference (Section 3), following Askarov et al's recent work, which combines the demands of termination-insensitive noninterference (for big secrets) with the stricter requirements of termination-sensitive noninterference (for small secrets). A type system is provided (Section 4) that ensures this notion of noninterference. Additional novelties of the system are a somewhat more liberal treatment of small secrets than found in previous termination-sensitive type systems. Section 5 describes a strengthening of the definition of security to eliminate leakage correlations between big and small secrets.

2 A Refined Multilevel Lattice

In [AHSS08] a definition of termination-insensitive noninterference (**TINI**) was introduced which is suitable for programs with outputs, assuming only two security levels *low* and *high*. They proved that, even if programs verifying this condition can leak more than a bit of information, the attacker cannot reliably (i.e in a single run) learn a secret in polynomial time in the size of the secret. They also proved that, for programs satisfying TINI, if secrets are uniformly distributed, then a particular observation of a computation represents only a negligible hint for the attacker (Theorem 3).

The basic idea in this work is to refine the notion of *high* into two points *bhigh* and *shigh*. These will correspond to "big" secrets and "small" secrets respectively. We will define a notion of secret-sensitive noninterference which allows a low user to learn a little about big secrets, and nothing at all about small secrets (relative to the notion of observation that we model).

How are big and small secrets related? A key point here is that data labelled *bhigh* will depend only on *bhigh* or *low* data sources, whereas data labelled *shigh* might also depend on *shigh* data sources. Thus the label *bhigh* does not mean that the data *is* a large secret – it just means that it does not depend on (contain any information about) a small secret. We can then see that the resulting refined security lattice is as given in Figure 1.

Fig. 1. The refined 2-point lattice

Now we generalise this refinement to the case of an arbitrary multi-level lattice of information levels [Den76]. Denning's lattice model of information considers an arbitrary complete lattice $\langle \mathcal{L}, \sqsubseteq_{\mathcal{L}}, \sqcup_{\mathcal{L}}, \sqcap_{\mathcal{L}}, \bot_{\mathcal{L}} \rangle$ where $\mathcal{L}$ is the set of security *clearance levels* (henceforth just *levels*, ranged over by i, j), and $\sqsubseteq_{\mathcal{L}}$ is the ordering relation which determines when one level is higher than another. The idea is that a principal with a clearance level i is permitted to see data which is classified at level i or below according to the partial ordering. Information from any levels may be combined, in which case the classification for the resulting data is given deterministically by the least-upper-bound operation $\sqcup_{\mathcal{L}}$.

To refine this general case we note that we must split each level $i \in \mathcal{L}$, with the exception of the bottom level $\bot_{\mathcal{L}}$ (which can always be thought of as public data) into two points, corresponding to the big secrets (labelled b) and the small (labelled s). Thus any non-bottom element i will be refined to (i, b) and (i, s). To define the appropriate order between lattice elements we first note that $(i, b) \sqsubseteq (i, s)$ – with the same motivation as given for the refined two-point lattice. Similarly, when comparing secrets of the same kind we have $(i, a) \sqsubseteq (j, a)$ only when $i \sqsubseteq_{\mathcal{L}} j$.

What about the relationship between two points (i, b) and (j, s) – when can information flow between these points? The idea is that information at level b is potentially leaked via a covert channel, so that it may be leaked to *any* level. Because of this we can only permit flow from (i, b) to (j, s), and then only when $i \sqsubseteq_{\mathcal{L}} j$. If we permitted a small secret (i, s) to flow to any (j, b) for $(j \neq i)$ then we would be able to launder small secrets by first allowing them to flow to a big secret and then leaking via the covert channel from there. In summary, we define the refinement of a given security lattice:

Definition 1. *Let $\mathcal{S}$ denote the 2-point lattice formed from b and s under the ordering $b \sqsubseteq s$. We define the* refinement *of a security lattice $\mathcal{L}$ as the* partial product *of $\mathcal{L}$ and $\mathcal{S}$, which is the standard product lattice $\mathcal{L} \times \mathcal{S}$, quotiented by the equivalence $(\bot_{\mathcal{L}}, b) \equiv (\bot_{\mathcal{L}}, s)$ – and this bottom element will be simply denoted by $\bot_{\mathcal{L}}$.*

Example. Consider the example where $\mathcal{L} = \{secret, financial, medical, public\}$ is the set of the four security levels a program has to deal with, ordered according to the Hasse diagram in Figure 2. Motivating a refinement of the lattice, there could be medical data that is encrypted – or simply very large (e.g. high resolution image data) that could be safely allowed to leak slowly, and other medical data that are to be handled with more care, such as an "hasHIV" boolean flag in a patient record. The partial product of lattices $\mathcal{L}$ and $\mathcal{S}$ is presented in Figure 3.

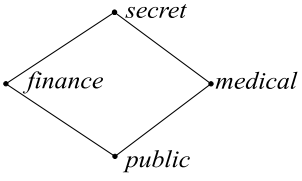

Fig. 2. Example $\mathcal{L}$

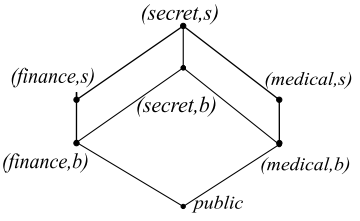

Fig. 3. The refinement of $\mathcal{L}$

3 Secret-Sensitive Noninterference

In this section we define the security goal for programs computing over data labelled with a refined lattice. This variant of the notion of noninterference, *secret-sensitive noninterference*, combines the demands of termination-insensitive noninterference for b-data, and the stronger termination-sensitive noninterference for s-data. Further, we develop a bisimulation-style characterisation of *secret-sensitive noninterference* which provides a convenient proof method.

Operational Semantics. We keep our presentation language independent, but we assume some basic structure for an operational semantics. We will consider simple imperative computation modelled by a standard small-step operational semantics defined over configurations of the form $\langle M, C\rangle$ where M is a memory (store) – a finite mapping from variables to values – and C (C', D etc.) is a command. Each variable x is assumed to have a fixed policy denoted $\Gamma(x)$, which we take to be a member of the refinement of some lattice $\mathcal{L}$.

We assume an operational semantics consisting of deterministic labelled transitions between configurations, where a label u is either (i) an observable output $i(v)$, meaning that a value v is output on a channel observable at level $i \in \mathcal{L}$ or above, or (ii) a silent action labelled τ. We write e.g. $\langle M, C\rangle \stackrel{i(v)}{\rightarrow} \langle M', C'\rangle$.

On top of the basic labelled transitions we define a family of transition systems labelled by a particular level:

Definition 2 (i-observable transitions). *We can define the transition relations $\xrightarrow{u}_i$, $i \in \mathcal{L}$ as:*

$$\frac{\langle M, C\rangle \xrightarrow{j(v)} \langle M', C'\rangle \qquad j \sqsubseteq_{\mathcal{L}} i}{\langle M, C\rangle \xrightarrow{v}_i \langle M', C'\rangle}$$

$$\frac{\langle M, C\rangle \xrightarrow{u} \langle M', C'\rangle \qquad u = \tau \text{ or } u = j(n) \text{ where } j \not\sqsubseteq_{\mathcal{L}} i}{\langle M, C\rangle \xrightarrow{\tau}_i \langle M', C'\rangle}$$

Thus the i-observable transitions are obtained from the raw transitions by filtering out (replacing by τ) all output actions that are not visible at level i. Note that the non-τ transitions are just the value which is observed and not the channel on which it is observed.

Now we define the "big step" transitions $\langle M, C\rangle \overset{u}{\Rightarrow}_i \langle M', C'\rangle$ as follows

$$\langle M, C\rangle \overset{\tau}{\Rightarrow}_i \langle M', C'\rangle \triangleq \langle M, C\rangle \xrightarrow{\tau}_i{}^{*} \langle M', C'\rangle$$
$$\langle M, C\rangle \overset{v}{\Rightarrow}_i \langle M', C'\rangle \triangleq \langle M, C\rangle \xrightarrow{\tau}_i{}^{*} \xrightarrow{v}_i \langle M', C'\rangle$$

We also define the multi-step observations $\langle M, C\rangle \overset{\vec{v}}{\Rightarrow}_i \langle M', C'\rangle$ with $\vec{v} = v_1 v_2 \cdots v_n$ as follows:

$$\langle M, C\rangle \overset{v_1}{\Rightarrow}_i \langle M_1, C_1\rangle \overset{v_2}{\Rightarrow}_i \langle M_2, C_2\rangle \overset{v_3}{\Rightarrow}_i \cdots \overset{v_{n-1}}{\Rightarrow}_i \langle M_{n-1}, C_{n-1}\rangle \overset{v_n}{\Rightarrow}_i \langle M', C'\rangle$$

for some sequence of intermediate configurations $\langle M_i, C_i\rangle$. We define the multi-step reduction for the empty vector to be synonymous with $\overset{\tau}{\Rightarrow}_i$.

Attacker's knowledge. Our presentation follows the style of Askarov *et al* [AHSS08] closely. The definition of noninterference developed here builds on the concept of *attacker knowledge* which is what an attacker (an observer of a given clearance level i) can deduce about the initial values of variables based on a particular observation of a program run.

The attacker i knows the initial low part of the memory. The low part of the memory from the perspective of a given level i is all variables with policy (i, s) or lower - and observes some output trace $\vec{v}$ that is not necessarily maximal, knows the program and is able to make perfect deductions about the semantics of the program. For a memory M we let M^i denote the low part of the memory from the perspective of an observer at level i, i.e. the part of the memory that he can see.

Definition 3 (Observations). *Given a program C and a low memory M^i, the i-observations is the set of all possible sequences of observable outputs that could arise from a run of C with a memory compatible with M^i. It is defined:*

$$Obs_i(C, M^i) = \{\vec{v} | \langle N, C\rangle \overset{\vec{v}}{\Rightarrow}_i \langle N', C'\rangle, \quad N^i = M^i\}$$

Definition 4 (Attacker's knowledge). *Given a program C, an initial choice M^i of the low part of the memory (for level i) and a trace of i-observable outputs $\vec{v}$, the attacker's knowledge gained from this observation is the set of all possible memories that could have lead to this observation.*

$$k_i(C, M^i, \vec{v}) = \{N | \langle N, C\rangle \overset{\vec{v}}{\Rightarrow}_i \langle N', C'\rangle, \quad N^i = M^i\}$$

Note that increase in knowledge corresponds to a decrease in the size of the knowledge set. Knowledge increases with outputs: the more outputs the attacker observes, the more precise is his knowledge [AS07]:

$$\forall C, M^i, \vec{v}, v. \quad k_i(C, M^i, \vec{v}v) \subseteq k_i(C, M^i, \vec{v})$$

In order to distinguish between what is learnt about the "big" secrets (variables at levels (i, b)) from what is learnt about the "small secrets" (variables at levels (i, s)) we define the projections of knowledge sets to the b- and s-parts.

Definition 5 (b- and s-restricted memories). *Given a memory M, and a security size $a \in \mathcal{S}$, we define $M|^i_a$ to be the restriction of M to those variables x such that $\Gamma(x) = (j, a)$, $j \not\sqsubseteq i$ – i.e. the "a-secrets" from i's perspective. We extend the definition pointwise to sets of memories.*

Definition 6 (b- and s-restricted knowledge). *Given a program C, a security size $a \in \mathcal{S}$ and an initial choice M^i of the low part of the memory and a trace of outputs $\vec{v}$, the a-restricted knowledge of the attacker i, written $k^a_i(C, M^i, \vec{v})$ is defined $(k_i(C, M^i, \vec{v}))|^i_a$.*

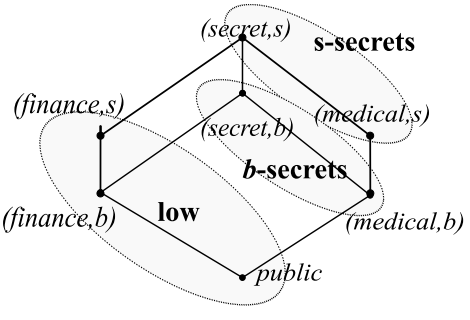

Fig. 4. The *finance*-perspective on the example refined lattice

Informally, the restricted knowledge $k^a_i(C, M^i, \vec{v})$ is i's knowledge about the a-secrets (from i's perspective) after having observed $\vec{v}$ from initial memory M^i.

The idea of "i's secrets" can be illustrated using the lattice presented in Figure 3. For example, the projection $M|^{finance}_s$ restricts M to just those variables with classifications $(medical, s)$ or $(secret, s)$. The *finance*-perspective on the lattice is illustrated in Figure 4, where the b-secrets and s-secrets are marked. The low part of the lattice, from the finance perspective, is also marked.

The s-restricted knowledge for an attacker at level *finance* is thus the knowledge that can be deduced about the s-secret part of the memory.

Noninterference. Several kinds of noninterference can be defined from the notion of knowledge. Here we adapt the definition of termination-(in)sensitive noninterference that was proposed in [AHSS08] and then propose a definition of a two-levelled noninterference.

Definition 7 (Termination-Sensitive Noninterference (TSNI)). *A program C satisfies TSNI if for all i, whenever $\vec{v}v \in Obs_i(C, M^i)$ then*

$$k_i(C, M^i, \vec{v}) = k_i(C, M^i, \vec{v}v).$$

TSNI means that at each step of output, nothing new about the high memory is learnt by the attacker.

Definition 8 (Termination-Insensitive Noninterference (TINI)). *A program C satisfies TINI if for all i, whenever $\vec{v}v \in Obs_i(C, M^i)$ then*

$$k_i(C, M^i, \vec{v}v) = \bigcup_{v'} k_i(C, M^i, \vec{v}v').$$

TINI allows leakage at each low output step, but only through the fact that there is *some* output step. The knowledge leaked by one output is the same as for any other.

In order to deal with our two different kinds of secret (b and s), the idea is here to combine both TSNI and TINI: although we only accept TSNI for s-data which must be handled with more care, we allow TINI for b-data, that abide some leakage since they are randomized and large enough.

Definition 9 (Secret-Sensitive Noninterference (SSNI)). *A program C satisfies SSNI if for all i, whenever $\vec{v}v \in Obs_i(C, M^i)$ then the following two properties hold:*

$$k_i^s(C, M^i, \vec{v}v) = k_i^s(C, M^i, \vec{v}) \qquad \textit{(s-TSNI)}$$
$$k_i^b(C, M^i, \vec{v}v) = \textstyle\bigcup_{v'} k_i^b(C, M^i, \vec{v}v') \qquad \textit{(b-TINI)}$$

3.1 Characterising SSNI

The knowledge based definitions are (in our opinion) lucid because they give a clear attacker perspective on the problem. However, for reasoning about secret-sensitive noninterference we find it convenient to work with a more conventional characterisation in terms of bisimulation relations. Here we develop this alternative characterisation, which we will employ in Section 4 in order to prove that the type system there guarantees secret-sensitive noninterference.

The basic idea is to establish the two components of SSNI via two forms of bisimulation relations between configurations.

Definition 10 (Termination-sensitive i-bisimulation (i-TSB)). *A symmetric relation $\mathcal{R}$ on configurations is a termination-sensitive i-bisimulation, if $\langle M, C\rangle \mathcal{R} \langle N, D\rangle$ implies:*

(i) $M^i = N^i$ *and* $M|_b^i = N|_b^i$, *and*
(ii) whenever $\langle M, C\rangle \xrightarrow{u}_i \langle M', C'\rangle$ *then* $\langle N, D\rangle \overset{u}{\Rightarrow}_i \langle N', D'\rangle$ *with* $\langle M', C'\rangle \mathcal{R} \langle N', D'\rangle$.

Two configurations are said to be i-TSB equivalent (denoted by $\cong_i$) if there exists a i-TSB relating them.

Here, the termination-sensitivity comes from the ability to produce the next output together with the symmetry of the relation.

Definition 11 (Termination-insensitive i-bisimulation (i-TIB)). *We say that a configuration $\langle M, C\rangle$ diverges for i, written $\langle M, C\rangle \Uparrow_i$, if $\langle M, C\rangle$ cannot perform any i-observable output transition $\xrightarrow{v}_i$.*

A symmetric relation $\mathcal{R}$ on configurations is defined to be a termination-insensitive i-bisimulation *if whenever $\langle M, C\rangle \mathcal{R} \langle N, D\rangle$ we have*

(i) $M^i = N^i$ *and*
(ii) if $\langle M, C\rangle \xrightarrow{u}_i \langle M', C'\rangle$ *then either* $\langle N, D\rangle \stackrel{u}{\Rightarrow}_i \langle N', D'\rangle$ *with* $\langle M', C'\rangle \mathcal{R} \langle N', D'\rangle$, *or* $\langle N, D\rangle \Uparrow_i$.

Two configurations are said to be i-TIB equivalent (denoted by $\simeq_i$) if there exists a i-TIB relating them.

Note that the notion of "divergence" used here is purely from the perspective of a remote observer who sees only the outputs on channels. We could make this more conventional if we made program termination an observable event for all levels. We have chosen not to do so, but the technical development in this paper does not depend in a crucial way on this fact.

Before we show how these relations are sufficient to characterise SSNI, we need the following lemmas about i-TSB and i-TIB.

Lemma 1
If $\langle M, C\rangle \cong_i \langle N, D\rangle$ and $\langle M, C\rangle \stackrel{\vec{v}}{\Rightarrow}_i \langle M', C'\rangle$ then $\langle N, D\rangle \stackrel{\vec{v}}{\Rightarrow}_i \langle N', D'\rangle$ with $\langle M', C'\rangle \cong_i \langle N', D'\rangle$.

Lemma 2
If $\langle M, C\rangle \simeq_i \langle N, D\rangle$ and $\langle M, C\rangle \stackrel{\vec{v}}{\Rightarrow}_i \langle M', C'\rangle$ then $\langle N, D\rangle \stackrel{\vec{v'}}{\Rightarrow}_i \langle N', D'\rangle$ for some $\vec{v'}$ such that either $\vec{v} = \vec{v'}$ and $\langle M', C'\rangle \simeq_i \langle N', D'\rangle$, or $\vec{v'}$ is a prefix of $\vec{v}$ and $\langle N', D'\rangle \Uparrow_i$.

Proof. (Lemmas 1 and 2) By induction on the number of outputs (length of $\vec{v}$), and in the base case by induction on the length of the raw transition sequence. □

Proposition 1
Suppose that for all levels i and all memories M and N such that $M^i = N^i$ and $M|_b^i = N|_b^i$ we have $\langle M, C\rangle \cong_i \langle N, C\rangle$. Then for all i, whenever $\vec{v}v \in Obs_i(C, M^i)$ then $k_i^s(C, M^i, \vec{v}v) = k_i^s(C, M^i, \vec{v})$.

Proof. See technical report [DS09]. □

A similar proposition can be stated about termination-insensitive noninterference concerning $bhigh$ data.

Proposition 2
Suppose that for all levels i and all M and N, such that $M^i = N^i$ we have that $\langle M, C\rangle \simeq_i \langle N, C\rangle$. Then $\vec{v}v \in Obs_i(C, M_i)$ implies $k_i^b(C, M^i, \vec{v}v) = \bigcup_{v'} k_i^b(C, M^i, \vec{v}v')$.

Proof. See technical report [DS09]. □

Clearly, then, putting the propositions together we get a proof technique for SSNI:

Corollary 1. *C satisfies $SSNI$ if, for all levels i and all M and N, we have*

- *$M^i = N^i$ implies $\langle M, C\rangle \simeq_i \langle N, C\rangle$, and*
- *$M^i = N^i$ and $M|_b^i = N|_b^i$ implies $\langle M, C\rangle \cong_i \langle N, C\rangle$.*

3.2 Computational Security

Definition 9 clearly enforces termination-sensitive noninterference for s-data. Regarding b-data, we can provide the computational security guarantees of [AHSS08] to show that b-secrets, if chosen uniformly, cannot be leaked in polynomial time in their size. To argue this we can first reclassify all secrets as b-data (or equivalently assume that there are no s-secrets). Then we are back in the standard security lattice, and we simply need to generalise the results of [AHSS08] from a two-point lattice to an arbitrary one. This is, as usual, unproblematic since from the perspective of each individual level i there are only two levels of interest: the levels which can be seen (i.e. the levels less than or equal to i) and those which cannot. The main result is that if b-data is randomly chosen, then an observer at level i learns a negligible amount of information (as a function of the size of the b-data) about the data which i cannot see. We will not further develop the details of this argument in the present article. The differences from the development in [AHSS08] would be minor.

4 Secret-Sensitive Noninterference by Typing

In this section, we describe a type system that enforces noninterference Definition 9: well-typed programs are secret-sensitive noninterfering. We study a classical deterministic while programming language defined with expressions and commands.

$$
\begin{aligned}
e &::= n \mid x \mid e \; op \; e \\
c &::= \mathbf{skip} \mid x := \; e \mid c \,;\; c \mid \mathbf{if}\; e \;\mathbf{then}\; c \;\mathbf{else}\; c \mid \\
&\qquad \mathbf{while}\; e \;\mathbf{do}\; c \mid \mathbf{for}\; e \;\mathbf{do}\; c \mid \mathbf{output}_i(e)
\end{aligned}
$$

Here n stands for any integer constant, x for any variable and op for any of the classical binary arithmetical operators. Booleans are represented by integers the classical way (0 is $false$, and everything else is $true$). We also assume that there are no exceptions raised: all binary operators are totally defined.

Note that the language provides two types of loops: **for** loops are always terminating, that is the guard expression is evaluated just once, leading to a constant that is decreased each time the end of the loop body is reached, and **while** loops are potentially non terminating. The distinction will be used in the type system to good effect.

The language includes the $\mathbf{output}_i$ primitive method that writes the value of its argument to a channel with level i. The operational semantics is standard and is given in the technical report [DS09].

4.1 Type System

This type system is based on the combination of a standard Denning-style analysis (in type system form [VSI96]) for enforcing the termination-insensitive security for b-secrets, and a more restrictive type system for handling the s-secrets. One such termination-sensitive type system is that described in [VS97], but that system is extremely restrictive: loops are only allowed if the guard does not refer to anything except data at the lowest lattice level, and if there is a branch on secret data at any level then no loops are allowed inside the branches. Instead we adapt an idea common to the type systems from [BC01] and [Smi01] for the termination-sensitive part. The idea is here to allow high while loops (i.e. loops with high guards or arbitrary while loops occurring in a high context) so long as no assignment or output to levels below the loop guards follows them.

The form of the typing judgements follows the style of [BC01] in that it handles indirect information flows by recording the write effect of a command (the lowest level to which it writes data). This gives the same power as Denning's popular approach which uses a "program counter" level.

Consider both lattices $\mathcal{L}$ and $\mathcal{S}$, and let $\mathcal{P}$ be their partial product as previously defined. A type is either an expression type denoted $e : \tau$, or a command type written $(\tau, \sigma, \delta)cmd$, where both τ and σ are in $\mathcal{P}$, the set of security levels, and δ, the *termination flag* is a member of the set $\{\downarrow, \uparrow\}$, where we order the elements $\downarrow \leq \uparrow$.

Type judgments are of the form

$$\Gamma \vdash C : (\tau, \sigma, \delta)cmd$$

where Γ is the typing environment i.e. a mapping from variables to variable types. In the following, Γ is kept implicit. The syntactic meaning of such a judgment is that

- τ is a lower bound on the security levels of variables that are assigned to in C.
- σ is the least upper bound on the levels of (for,if,while) guards occurring in C.
- δ is $\downarrow$ if C contains no while loops, and is $\uparrow$ otherwise.

The semantic implication of these typings is that

- τ is a lower bound on the the *write effect* of the command – i.e., the command only modifies variables of level τ or above, and
- σ is the *termination effect*: observing that C produces some output (i.e. "terminates") give us knowledge about data at level at most σ.
- δ is a *termination flag*: if $\delta = \downarrow$ then the command always terminates.

With these intended meanings of τ, σ and δ, there is a natural partial order on types which is contravariant in its first component and covariant in its second and third:

$$(\tau, \sigma, \delta)cmd \leq (\tau', \sigma', \delta')cmd \text{ if } \tau' \sqsubseteq_{\mathcal{P}} \tau \text{ and } \sigma \sqsubseteq_{\mathcal{P}} \sigma' \text{ and } \delta \leq \delta'$$

This relation is not used in the type system, but is used in the statement of e.g. the subject reduction property below.

For elements of $\mathcal{P}$ (the first two components of a command type in particular) we define the first and second projections in the obvious way: $fst(i, a) = i$ and $fst(\bot_{\mathcal{P}}) = \bot_{\mathcal{L}}$; $snd(i, a) = a$ and $snd(\bot_{\mathcal{P}}) = \bot_{\mathcal{S}} = b$.

Rules of the security type system are displayed in Figure 5, where we drop the subscript for the relation $\sqsubseteq_{\mathcal{P}}$.

Explicit flows are handled with rules for expressions, rules T-ASSIG, and T-OUT, while implicit flows are treated in T-IF, T-WHILE and T-FOR which demand that their body is at least as high as their guard level.

Most of the action takes place in the sequential composition rules. The interesting case is T-SEQ2 where the termination effect σ_1 of C_1 is an s-secret, and C_1 is indeed potentially nonterminating. This means that we cannot allow arbitrary assignments in C_2 since these might leak information about the s-secrets which affected the termination of C_1. Thus the write effect of C_2 is constrained so that it does not write below σ_1, the termination effect of C_1. For rule T-SEQ1 we are more liberal, since either the guards do not depend on s-secrets, or C_1 is always terminating.

The same reasoning is applied to while and for loops – their execution may be a sequential composition of the body of the loop and the loop itself.

$$\frac{}{\vdash n : \tau}\text{T-CONST} \qquad \frac{\Gamma(x) = \tau\ var}{\vdash x : \tau}\text{T-VAREXP}$$

$$\frac{\vdash e : \tau' \qquad \tau' \sqsubseteq \tau}{\vdash e : \tau}\text{T-SUBEXP} \qquad \frac{\vdash e_1 : \tau \qquad \vdash e_2 : \tau}{\vdash e_1\ op\ e_2 : \tau}\text{T-BINOP}$$

$$\frac{}{\vdash \textbf{skip} : (\top_{\mathcal{P}}, \bot_{\mathcal{P}}, \downarrow)cmd}\text{T-SKIP} \qquad \frac{\vdash e : \tau \qquad \Gamma(x) = \tau\ var}{\vdash x := e : (\tau, \bot_{\mathcal{P}}, \downarrow)cmd}\text{T-ASSIG}$$

$$\frac{\vdash e : \tau \qquad fst(\tau) \sqsubseteq_{\mathcal{L}} i}{\vdash \textbf{output}_i(e) : ((i, s), \bot_{\mathcal{P}}, \downarrow)cmd}\text{T-OUT}$$

$$\frac{\vdash C_i : (\tau_i, \sigma_i, \delta_i)cmd \qquad snd(\sigma_1) = b \quad \text{or} \quad \delta_1 = \downarrow}{\vdash C_1; C_2 : (\tau_1 \sqcap \tau_2, \sigma_1 \sqcup \sigma_2, \delta_1 \sqcup \delta_2)cmd}\text{T-SEQ1}$$

$$\frac{\vdash C_i : (\tau_i, \sigma_i, \delta_i)cmd \qquad \sigma_1 \sqsubseteq \tau_2 \qquad snd(\sigma_1) = s \qquad \delta_1 = \uparrow}{\vdash C_1; C_2 : (\tau_1 \sqcap \tau_2, \sigma_1 \sqcup \sigma_2, \uparrow)cmd}\text{T-SEQ2}$$

$$\frac{\vdash e : \theta \qquad \vdash C_i : (\tau_i, \sigma_i, \delta_i)cmd \qquad \theta \sqsubseteq \tau_i}{\vdash \textbf{if}\ e\ \textbf{then}\ C_1\ \textbf{else}\ C_2 : (\tau_1 \sqcap \tau_2, \sigma_1 \sqcup \sigma_2 \sqcup \theta, \delta_1 \sqcup \delta_2)cmd}\text{T-IF}$$

$$\frac{\vdash e : \theta \qquad \vdash C : (\tau, \sigma, \delta)cmd \qquad \theta \sqsubseteq \tau \qquad snd(\sigma) = s \Rightarrow \sigma \sqsubseteq \tau}{\vdash \textbf{while}\ e\ \textbf{do}\ C : (\tau, \sigma \sqcup \theta, \uparrow)cmd}\text{T-WHILE}$$

$$\frac{\vdash e : \theta \qquad \vdash C : (\tau, \sigma, \delta)cmd \qquad \theta \sqsubseteq \tau \qquad snd(\sigma) = s \wedge \delta = \uparrow \Rightarrow \sigma \sqsubseteq \tau}{\vdash \textbf{for}\ e\ \textbf{do}\ C : (\tau, \sigma, \delta)cmd}\text{T-FOR}$$

Fig. 5. The security type system

4.2 Type Soundness

In this section we prove some results about well typed programs with regard to the type system in Figure 5. The main proposition establishes that the type system indeed enforces the secret-sensitive noninterference property we defined in Section 3.

Proofs of the following results are only sketched here. A full version of the proofs can be found in the technical report corresponding to the present paper [DS09].

The first property is the standard notion of *subject reduction* which guarantees that execution preserves types.

Theorem 1 (Subject reduction). *If* $\vdash C : (\tau, \sigma, \delta)cmd$ *and* $\langle M, C\rangle \xrightarrow{u} \langle M', C'\rangle$, *then* $\vdash C' : (\tau', \sigma', \delta')cmd$ *with* $(\tau', \sigma', \delta')cmd \leq (\tau, \sigma, \delta)cmd$.

Proof. We proceed by induction on the typing derivation, and then by case analysis on the last rule of the operational semantics. □

We need some preliminary lemmas in order to prove the SSNI enforcement. The following lemmas (using the terminology from [VSI96]) confirm that the informal definitions we gave about both components of a command type in Section 4.1 are enforced by the type system.

Lemma 3 (Simple security). *If* $\vdash e : \tau$ *then every variable occurring in* e *has type* $\tau'\ var$ *where* $\tau' \sqsubseteq \tau$.

Lemma 4 (Confinement). *If* $\vdash C : (\tau, \sigma, \delta)cmd$, *then every variable assigned to in program* C *has type* $\theta\ var$ *with* $\tau \sqsubseteq \theta$.

Lemma 5 (Guard safety). *If* $\vdash C : (\tau, \sigma, \delta)cmd$, *then every while loop or conditional guard in program* C *has type* $\theta\ var$ *with* $\theta \sqsubseteq \sigma$.

Lemma 6 (Termination). *If* $\vdash C : (\tau, \sigma, \downarrow)cmd$, *then* C *terminates on all memories.*

These four lemmas can be easily proved by induction on the typing derivation.

In the formal development that follows for simplicity's sake we only treat the case of the three point lattice in Figure 1. The following results can be extended to the general case: for a given clearance level i in $\mathcal{L}$, as was depicted in the example of *finance*'s perspective in Figure 4, the refinement of $\mathcal{L}$ can be rethought of as a three point lattice - *low* level, *bhigh* and *shigh* secrets.

Proposition 3 (Noninterference of well typed commands)
If a command C *is typable, i.e.,* $\vdash C : (\tau, \sigma, \delta)cmd$, *then* C *satisfies SSNI.*

Proof. (Sketch; see technical report [DS09] for details) We use the proof technique provided by Corollary 1. In the construction of the specific bisimulations we adapt the proof from [BC01]. The first step is to show that $\vdash C : (\tau, \sigma, \delta)cmd$ implies $\langle C, M\rangle \cong_i \langle C, N\rangle$ for all levels i, to have the s-TSNI property of Definition 9. The interesting case is $i = low$ since $i = high$ is vacuous (memories and commands are in this case equal).

A command C is said to be *shigh* or *bhigh* if there exists τ and σ such that $\vdash C : (\tau, \sigma, \delta)cmd$ with respectively $\tau = shigh$ or $bhigh \sqsubseteq \tau$. We show that $\vdash C : (\tau, \sigma, \delta)cmd$ implies $\langle C, M\rangle \cong_l \langle C, N\rangle$ for all M and N that are equal on their low and bhigh parts. To do this we define a relation $\mathcal{R}_1 : \langle M, C\rangle \mathcal{R}_1 \langle N, D\rangle$ iff C and D are typable, $M^l = N^l$ and $M|_b^l = N|_b^l$, and one of the following four conditions holds:

(i) C and D are *shigh*; (ii) $C = D$
(iii) $C = C_1; C_2$, $D = D_1; C_2$ with $\langle M, C_1\rangle \mathcal{R}_1 \langle N, D_1\rangle$ and C_2 is *shigh*
(iv) C is *shigh*, $D = D_1; D_2$ with $\langle M, \mathbf{skip}\rangle \mathcal{R}_1 \langle N, D_1\rangle$ and D_2 is *shigh*

We then show that $\mathcal{R}_1$ is a l-TSB by induction on the definition of $\mathcal{R}_1$, and conclude using Proposition 1. By Clause (ii) and Proposition 1, we have that in well typed programs, there is no flow from $shigh$ data to $bhigh$ and low data.

The next step is to prove that the type system ensures TINI concerning the $bhigh$ data. We proceed in a similar way, providing a l-TIB $\mathcal{R}_2$ over configurations. The relation $\mathcal{R}_2$ is defined: $\langle M, C\rangle \mathcal{R}_2 \langle N, D\rangle$ iff C and D are typable, $M^l = N^l$, and one of the following holds:

(i) C and D are *bhigh* (ii) $C = D$
(iii) $\langle M, C\rangle \mathcal{R'}_2 \langle N, D\rangle$, where the relation $\mathcal{R'}_2$ is defined inductively as:

$$\frac{C, D \textit{ bhigh}}{\langle M, C; C'\rangle \ \mathcal{R'}_2 \ \langle N, D; C'\rangle}\text{R1} \qquad \frac{\langle M, C\rangle \ \mathcal{R'}_2 \ \langle N, D\rangle}{\langle N, C; C'\rangle \ \mathcal{R'}_2 \ \langle N, D; C'\rangle}\text{R2}$$

By Clause (ii) and Proposition 2, we then have the TINI property of well typed programs concerning their $bhigh$ data: there is no flow from $bhigh$ data to low data except via the termination channel. □

5 Correlation Leaks

In this section we mention a weakness in the definition of secret-sensitive noninterference which allows the attacker to observe *correlations* between big and small secrets. We show how the definition can be strengthened to remove such correlations, and conjecture that the type-system guarantees correlation-freedom without need for modification.

Suppose that b is *bhigh* and s is *shigh* (in the lattice in Figure 1). Somewhat surprisingly the program $output_{low}(b == s)$ is secret-sensitive noninterfering (note though that it is not typeable). This is because the low observer can say nothing about the value of e.g. s in isolation. The problem is that although the observer cannot deduce anything about the individual kinds of secret, he can deduce information about their *correlation* (in this example whether they are equal or not).

To eliminate the possibility of learning something about the correlation of big and small secrets we need to demand that the knowledge learnt about big and small secrets together is the same as for the combined knowledge learnt about them independently. To express this precisely we need some additional notation.

In the definitions of secret-sensitive noninterference we have dealt with knowledge as sets of projections of memories. We say that a memory M is *full* if $dom(M)$ is the set of all variables. In order to easily compare and combine knowledge sets we need to work with full memories. Define $M^\star$ to be the set of full memories obtainable by completing M:

$$M^\star = \{N \mid N|_{dom(M)} = M, N \text{ is full}\}.$$

Now lift $\cdot^\star$ to sets of memories K in the natural way by defining

$$K^\star = \bigcup_{M \in K} M^\star$$

Definition 12 (Correlation Freedom). *A program C is* Correlation Free *if for all $\vec{v} \in Obs_i(C, M^i)$, we have $k_i^{bs}(C, M^i, \vec{v})^\star = k_i^b(C, M^i, \vec{v})^\star \cap k_i^s(C, M^i, \vec{v})^\star$, where $k_i^{bs}(C, M^i, \vec{v}) = \{M|^i \mid M \in k_i(C, M^i, \vec{v})\}$ and $M|^i$ is the complement of M^i – i.e., the projection of M to the variables* not *visible at level i.*

In the case that C is secret-sensitive noninterfering we can show that this condition is equivalent to $k_i^{bs}(C, M^i, \vec{v})^\star = k_i^b(C, M^i, \vec{v})^\star$, which says that nothing more is learnt about the big and small secrets together than can be deduced from the big secrets alone.

Conjecture 1. Well-typed programs are correlation free.

We leave the proof of this conjecture to further work; the intuition here is that any "correlation information" will always be typed as s-level data, and hence cannot be leaked at all.

6 Conclusions

In this article we provided a way to refine an arbitrary complex security lattice in order to distinguish two levels of secret, the big secrets b and the small ones s. Big secrets can be handled more liberally on the grounds that they can be made sufficiently large and random for slow leakage to be tolerable. We introduced an accompanying notion of secret-sensitive noninterference which combines the relative merits of termination-sensitive and termination-insensitive noninterference. We illustrated the use of the definition in the soundness argument for a simple type system for verifying secret-sensitive noninterference.

Related Work. As mentioned previously, the starting point of this work is [AHSS08]. Our interpretation of the results there is that we need to treat different kinds of secrets in different ways, and to our knowledge this paper is the first to do so in a noninterference setting. It is, however, relatively common to give a special treatment to cryptographic keys as compared to other kinds of secret – e.g. [AHS06] – but usually the goal here is to deal with integrity (a key cannot be modified using a low value) or freshness (a key cannot be used more than once).

Our type system is essentially a fusion of a type-based version of Denning's system [VSI96], and a stricter system based on [BC01]. The latter system is stricter than a Denning-style analysis for quite a different purpose: to deal with multi-threaded programs. Our system, in a sequential setting, improves on [BC01] by additionally tracking whether a program is terminating.

Further Work. A natural and interesting next step would be to combine such a type system with cryptographic primitives (e.g. [Vol00][LV05][AHS06]). The notion of "big" and "small" secrets have a natural interpretation in the cryptographic setting, since "big" secrets correspond to e.g. cryptographic keys. In such a setting it might also be important to handle "size integrity", so that one could know that a variable is not only independent of small secrets, but that it *is* a big secret.

Acknowledgements. Thanks to Andrei Sabelfeld for pointing out the correlation problem discussed in Section 5, and to Niklas Broberg, David Pichardie, Thomas Jensen and the anonymous referees for very helpful comments on an earlier draft. This work was partly supported by grants from the Swedish funding agencies SSF, Vinnova (The Swedish Governmental Agency for Innovation Systems), VR, and by the European IST-2005-015905 MOBIUS project.

References

[AHS06] Askarov, A., Hedin, D., Sabelfeld, A.: Cryptographically-masked flows. In: Yi, K. (ed.) SAS 2006. LNCS, vol. 4134, pp. 353–369. Springer, Heidelberg (2006)

[AHSS08] Askarov, A., Hunt, S., Sabelfeld, A., Sands, D.: Termination-insensitive noninterference leaks more than just a bit. In: Jajodia, S., Lopez, J. (eds.) ESORICS 2008. LNCS, vol. 5283. Springer, Heidelberg (2008)

[AS07] Askarov, A., Sabelfeld, A.: Gradual release: Unifying declassification, encryption and key release policies. In: Proc. IEEE Symp. on Security and Privacy, pp. 207–221 (May 2007)

[BB03] Barnes, J., Barnes, J.G.: High Integrity Software: The SPARK Approach to Safety and Security. Addison-Wesley Longman Publishing Co., Inc., Amsterdam (2003)

[BC01] Boudol, G., Castellani, I.: Noninterference for concurrent programs. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 382–395. Springer, Heidelberg (2001)

[CH04] Chapman, R., Hilton, A.: Enforcing security and safety models with an information flow analysis tool. ACM SIGAda Ada Letters 24(4), 39–46 (2004)

[DD77] Denning, D.E., Denning, P.J.: Certification of programs for secure information flow. CACM 20(7), 504–513 (1977)

[Den76] Denning, D.E.: A lattice model of secure information flow. Comm. of the ACM 19(5), 236–243 (1976)

[DS09] Demange, D., Sands, D.: All secrets great and small. Technical report, Chalmers University of Technology, Sweden, Extended Version (2009)

[LV05] Laud, P., Vene, V.: A type system for computationally secure information flow. In: Liśkiewicz, M., Reischuk, R. (eds.) FCT 2005. LNCS, vol. 3623, pp. 365–377. Springer, Heidelberg (2005)

[MZZ+08] Myers, A.C., Zheng, L., Zdancewic, S., Chong, S., Nystrom, N.: Jif: Java information flow. Software release (July 2001-2008), `http://www.cs.cornell.edu/jif`

[Sim03] Simonet, V.: The Flow Caml system. Software release (July 2003), `http://cristal.inria.fr/~simonet/soft/flowcaml/`

[Smi01] Smith, G.: A new type system for secure information flow. In: Proc. IEEE Computer Security Foundations Workshop, pp. 115–125 (June 2001)

[Vol00] Volpano, D.: Secure introduction of one-way functions. In: CSFW 2000: Proceedings of the 13th IEEE workshop on Computer Security Foundations, p. 246. IEEE Computer Society, Washington (2000)

[VS97] Volpano, D., Smith, G.: Eliminating covert flows with minimum typings. In: Proc. IEEE Computer Security Foundations Workshop, pp. 156–168 (June 1997)

[VSI96] Volpano, D., Smith, G., Irvine, C.: A sound type system for secure flow analysis. J. Computer Security 4(3), 167–187 (1996)