UNLIMITED

BR115416 ②

Report No. 90020

Report No. 90020

ROYAL SIGNALS AND RADAR ESTABLISHMENT, MALVERN

AD-A229 043

UNIFYING INFORMATION FLOW POLICIES

Author: Simon N Foley*

*Cranfield Information Technology Institute

PROCUREMENT EXECUTIVE, MINISTRY OF DEFENCE
RSRE
Malvern, Worcestershire.

90 11 19 160

October 1990

UNLIMITED

ROYAL SIGNALS AND RADAR ESTABLISHMENT

REPORT 90020

Title: Unifying Information Flow Policies

Author: Simon N. Foley[†]

Date: October 1990

Abstract

This report proposes a structure for describing information flow policies that can express transitive, aggregation and separation (of duty) exceptions. Operators for comparing, composing and abstracting flow policies are described. These allow complex policies to be built from simpler policies. Many existing confidentiality (and by using a dual model, integrity) policies can be captured in this framework. A high water mark model is developed that can enforce these information flow policies. This model provides the basis for a taxonomy of existing high water mark mechanisms.

[†] Cranfield Information Technology Institute
Pitfield, Kiln Farm,
Milton Keynes, MK11 3LG

THIS PAGE IS LEFT BLANK INTENTIONALLY

Contents

1 Introduction

Confidentiality security is concerned with restricting the disclosure of information in systems. One way of achieving this is to use an information flow policy which defines the different classes of information (for example, classified, secret, etc.) that can exist in the system and a flow relation which describes how information may flow between these classes. System entities (users, processes, files, etc.) are considered to be the sources and sinks of information, and each is bound to a security class from the flow policy. This binding is interpreted as: if entity A is bound to class a then A may source information of class a or higher and may sink information of class a or lower. A system is considered multilevel secure if all flows between entities maintain the flow policy. Note that there are special cases where certain entities, such as trusted subjects, are allowed violate this requirement in a controlled manner.

Traditionally information flow policies have formed lattices[8], with good reason as it is easy to build state based enforcement mechanisms. However, it may be desirable to enforce non-transitive information flow policies. That is, a flow policy where information is allowed flow from class a to class b, and from b to class c, but not from a to c. An example of such a policy is reclassification: there are three classes of information, secret, classified, and downgrade; classified is allowed flow to secret; secret is allowed flow to downgrade, and downgrade is allowed flow to classified, but secret is not allowed flow (directly) to classified. A (trusted) security officer, cleared to downgrade, may read secrets and reclassify as downgraded information, which may flow to classified. Further examples of reflexive flow policies will be given in in this report.

Another desirable generalisation for flow policies concerns the consistency of the join operator (the lowest upper bound). Traditionally, if information may flow from a to c and from b to c, then the join of information of class a and b may flow to c. If this assumption is removed, we have a means of describing aggregation flow exceptions: a user of class c is allowed to sink information of class a or class b, but not their aggregate. That real aggregation policies exist, is demonstrated by Brewer and Nash in their popular Chinese Wall paper[6], and supported with further examples by Meadows[20]. Many models for aggregation policies have been proposed[6,16,19,12]. We seek a unified definition of an information flow policy that includes aggregation properties.

Separation of Duty rules are normally associated with integrity policies and models[7,15,21]. For example, a cheque must be proposed by a manager, and cleared by an accountant, or vice-versa, before it can be written. However, separation rules can also apply to information flow (confidentiality) policies. For example, a user of a dial-up database may only view database information if it is accompanied by charges information. This type of separation flow policy can be thought of as dual of an aggregation policy: only the aggregate may flow, not the individual classes.

This report proposes a single structure that can be used to describe information flow policies that may have transitivity, aggregation and separation exceptions; a high water mark mechanism is developed for enforcing these policies.

Section 3 examines the traditional notion of an information flow policy. Section 4 proposes a structure, a *conglomerate relation*, that can be used to describe our general information flow policies. Relations and operators over flow policies are developed that allow flow policies to be compared, composed and abstracted. We use the Z notation[23] as a convenient syntax for flow policies.

Section 5 introduces how conglomerate relations can be used to capture aggregation and separation exceptions in systems and in information flow policies, and defines, the abstract requirements for a secure system.

Section 6 shows how an arbitrary flow policy with no separation (of duty) exceptions can be enforced using the high water mark mechanism proposed in [11]. With a high water mark mechanism, each system entity is bound to a class that represents the join of all classes of information it has sunk to date. Thus as a system progresses, high water marks rise. Traditional high water marks rise to a single limit[26], in our model there is a set of possible limits. The policies that can be enforced by this model include quasi-ordered policies, and reflexive policies. We show how the mechanisms simplify for these cases to: the traditional lattice flow model[8] with static binding for quasi ordered policies; the interval confinement model[10] for reflexive flow policies. The interval confinement model can be viewed as a generalisation of models that associate an interval of classes with system entities, such as partially trusted subjects[3,22], and high water marks with limits[26]. This leads us to the conclusion that we can classify the type of confidentiality policy enforced by these systems as reflexive. Section 7 gives examples of reflexive and aggregation flow policies.

Section 8 develops an alternative high water mark model that can enforce any separation and/or aggregation information flow policy. As before, every entity has a high water mark that may rise according to the information it sinks. However, to implement separation, water marks may not rise linearly—certain class combinations may be invalid, representing separation exceptions. Section 9 gives examples of separation flow policies.

Section 10 considers the relationship between separation and aggregation policies, and shows how complementing one can give the other.

The structure used to describe an information flow policy can also be used to describe an integrity policy. Section 11 investigates the possibility of using conglomerate relations to describe integrity policies.

Section 12 contains a summary and discussion of the work described in this report. The appendix contains all the necessary proofs of properties proposed. The Z notation[23] is used throughout this report.

2 Lattices

Lattice structures will be used in this paper to describe information flow policies. This section gives a Z formulation of their structure.

The set of all partially ordered sets is

$$\begin{aligned} posets[X] == \{P : X \leftrightarrow X \mid & \operatorname{id} P \subseteq P \wedge \\ & P \mathbin{;} P^{-1} = \operatorname{id} P \wedge \\ & P \mathbin{;} P = P\} \end{aligned}$$

A partially ordered set is reflexive, antisymmetric and transitive. Given poset P, then $a \mapsto b \in P$ means that a is less than or equal to b in P, (a is dominated by b in P).

Given a poset P, then an element a of this set will have a set of upper bounds (everything that dominates it in P) given by *u-bounds* P a, and a set of lower bounds given by *l-bounds* P a.

$$\begin{array}{|l} [X] \\ \text{u-bound\_}, \\ \text{l-bound\_} : posets[X] \rightarrow (X \rightarrow \mathbb{P} X) \\ \hline \forall P : posets[X] \bullet \\ \qquad \text{u-bound} P = \{a : X \mid a \in \operatorname{dom} P \bullet a \mapsto \operatorname{ran}(\{a\} \lhd P)\} \\ \qquad \text{l-bound} P = \{a : X \mid a \in \operatorname{dom} P \bullet a \mapsto \operatorname{dom}(P \rhd \{a\})\} \end{array}$$

Functions *u-bound* and *l-bound* can be generalised to give the upper and lower bounds that are common to a set of elements of a poset:

$$\begin{array}{|l} [X] \\ \text{U-bound\_}, \\ \text{L-bound\_} : posets[X] \rightarrow (\mathbb{P} X \rightarrow \mathbb{P} X) \\ \hline \forall P : posets[X] \bullet \\ \qquad \text{U-bound} P = \{A : \mathbb{P}_1 X \mid A \subseteq \operatorname{dom} P \bullet A \mapsto \bigcap \text{u-bound} P (\!| A |\!)\} \\ \qquad \text{L-bound} P = \{A : \mathbb{P}_1 X \mid A \subseteq \operatorname{dom} P \bullet A \mapsto \bigcap \text{l-bound} P (\!| A |\!)\} \end{array}$$

We can now define what a lattice is. A lattice is a partially ordered set such that every set of elements have a unique upper bound that forms a lower bound on all other upper bounds on the group of elements; and a unique lower bound that forms an upper bound on all other lower bounds on the group of elements.

$$
\begin{array}{l}
\mathit{lattice}[X] == \\
\quad \{P : \mathit{posets}[X] \mid \\
\quad \forall A : \mathbb{P}X \bullet A \subseteq \mathrm{dom}\, P \Rightarrow \\
\qquad \#(\text{U-bound}P\ A) \cap (\text{L-bound}P(\text{U-bound}P\ A)) = 1 \land \\
\qquad \#(\text{L-bound}P\ A) \cap (\text{U-bound}P(\text{L-bound}P\ A)) = 1\}
\end{array}
$$

A lowest upper bound operator (lub) can be defined on a lattice such that it returns the lowest upper bound of a non-empty set of elements.

$$
\begin{array}{|l}
[X] \\
\hline
\bigoplus\_ : \mathit{lattice}[X] \rightarrow \mathbb{P}X \nrightarrow X \\
\hline
\forall L : \mathit{lattice}[X] \bullet \\
\quad \bigoplus L = \{A : \mathbb{P}_1 X;\ a : X \mid \\
\qquad (\text{U-bound}P\ A) \cap (\text{L-bound}P\ (\text{U-bound}P\ A)) = \{a\} \\
\qquad \bullet A \mapsto a\} \\
\hline
\end{array}
$$

A binary version of this operator can also be defined where $a \dotplus b = \bigoplus\{a, b\}$.

Given some set of elements from a lattice, the *maximums* on this set, is the set of components from this set such that there is no other component in the set that dominates these maximums. Note that if the lattice is total (i.e., every element either dominates or is dominated by, every other element) then there is just one maximum. There is a similar definition for minimums.

$$
\begin{array}{|l}
[X] \\
\hline
\mathrm{maxs}\_ : \mathit{lattice}[X] \rightarrow \mathbb{P}X \rightarrow \mathbb{P}X \\
\hline
\forall L : \mathit{lattice}[X] \bullet \\
\quad \mathrm{maxs}\ L = \{A, B : \mathbb{P}X \mid B \subseteq A \land A \subseteq \mathrm{dom}(L \rhd B) \land \\
\qquad\qquad \mathrm{ran}(B \lhd L) \cap A = B \bullet A \mapsto B\} \\
\hline
\end{array}
$$

A powerset lattice of a set of elements is a lattice with components drawn from the powerset of the elements; a partial ordering defined by subset, and bound operators defined by union and intersection.

$$
\begin{array}{|l}
[X] \\
\hline
\mathbb{P}_{\subseteq}\_ : \mathbb{P}X \rightarrow \mathit{lattice}[\mathbb{P}X] \\
\hline
\forall A : \mathbb{P}X \bullet \\
\quad \mathbb{P}_{\subseteq} A = \{B, C : \mathbb{P}X \mid C \in \mathbb{P}A \land B \subseteq C \bullet B \mapsto C\} \\
\hline
\end{array}
$$

A powerset lattice is distributive, i.e., its bound operators distribute over one another, and complementable, i.e., every component has a complement.

3 Secure Information Flow

An information flow policy can be thought of as defining the different classes or kinds of information that can exist in a system and how they may propagate. An example of this is the military flow policy which defines a set of information flow classes that represent the sensitivity of information in the system, and an ordering relation that describes relative sensitivity. Let us introduce the set of *information classes* as the basic type

$[classes]$

Rather than thinking of information classes as just security classes (as in the traditional sense), they can be thought of as a way of associating a simple representation of meaning with information in the system. We shall see later how the information itself is represented in the system. For the moment, we impose the traditional restrictions[8] on a policy, in that it should form a lattice. For a given flow policy, $P \in lattice[classes]$, if $a \mapsto b$ is a maplet of P, then it means that information of class a is permitted to flow to class b.

To apply a flow policy to a system we must, in some way, relate the information in the system to the security classes of the policy. We chose to do this by viewing the system in terms of system entities and the information flows between these entities. The entities of a system are the sources and sinks of information of interest in the system. They might correspond to subjects and objects such as those in [2], or the more abstract system interfaces of [13]. Introduce the set of all such entities as the basic type

$[ents]$

The functionality of a system will be abstracted to just flows between these entities. For the moment we will not concern ourselves with semantics for information flow, except that it is a relation between entities. Define the set of all possible abstracted systems to be

$$\begin{array}{|l}
systems : \mathcal{P}ents \leftrightarrow ents \\
\hline
systems = \{S : ents \leftrightarrow ents \mid \mathrm{id}\, S \subseteq S\}
\end{array}$$

If $S \in systems$ is a particular abstract system, then $e \mapsto f \in S$ means that information *can* flow from entity e to entity f over the system S. The only requirement we will place is that information flow should be at least reflexive: information can always flow from an entity to itself. Note that our characterisation of information flow is over the lifetime of the system; we will show later how a flow semantics might be given in terms of states and state transitions.

The relationship between a system and a flow policy can be established by binding each entity to an information flow class from the policy. The class of each entity represents the class of information that the entity is allowed to sink and source. The system is regarded as secure so long as the multilevel security requirement is maintained, i.e.,

> if information flows from class a to class b in a system implies, that $a \mapsto b$ must hold in the policy.

Thus we define a secure system by

$$
\begin{array}{|l}
\hline
\textit{Secure-System} \\
S : \textit{systems} \\
P : \textit{lattice}[\textit{classes}] \\
\beta\_ : \textit{ents} \nrightarrow \textit{classes} \\
\hline
\operatorname{dom} \beta\_ = \bigcup \operatorname{dom} S \\
\operatorname{ran} \beta\_ \subseteq \bigcup \operatorname{dom} P \\
\forall e, f : \textit{ents} \bullet \\
\quad e \mapsto f \in S \Rightarrow \beta e \mapsto \beta f \in P \\
\hline
\end{array}
$$

The schema states that a system S is secure by flow policy P, given the entity binding function β, if every flow between entities in the system is allowed by the flow policy. The other restrictions ensure that every entity is assigned an information class from the policy. Entity confinement is partial as it only maps entities of the system to information classes.

It is attractive to use lattice based information flow policies, as they lead to simple flow control mechanisms[8] that are inductively defined over secure states and secure transitions. An example of this is the star property and ss-condition[2] (under strong tranquillity); the tranquil Bell LaPadula model can be thought of as an 'unwinding' of our abstract notion of a secure system.

In [9] Denning shows how an arbitrary reflexive and transitive (quasi ordered) ordered set can be transformed into a lattice. This allows us to weaken our requirement that a flow policy should form a lattice, to one requiring that it form a quoset. Further weakenings of these requirements can be made. However as we shall see throughout this report, dynamic binding mechanisms are needed to enforce these more general policies. In [10], a method for enforcing non-transitive flow policies is proposed. This approach also results in simple inductive requirements for secure states and secure transitions which result in an implementation mechanism that is not unlike a high water mark mechanism[25]. Furthermore, non-transitive information flow policies are useful, and examples are given in [10,12].

Flow policies, whether transitive[8] or not[10], are assumed to have a consistent class combination operator $\oplus$ (giving a lowest upper bound on lattice classes), where if information of class a [illegible] flow to class c and information of class b can flow to c, then their aggregate $a \oplus b$ should also be allowed flow to c. This initially seems reasonable—a person who is allowed read access to one file or another should also be allowed read both—and it contributes to the simple flow control mechanisms. However, in [6,16,19,12], confidentiality policies are described, whose essence is that class combination does not form an upper bound operation; a consultant is allowed read information about bank A or bank B, but for conflict of interest reasons, not both[6]. Therefore, we propose to drop the requirement that an information flow policy should have consistent class combination operator (i.e., lowest upper bound operator). However to do this, it is no longer sufficient to represent an information flow policy as allowable flows between single classes. We need a new structure for a flow policy in which we enumerate not only flows between classes, but also flows between groups of classes.

4 A Theory of Conglomerates

In this section we will describe a structure that will be used to describe information flow policies. The structure, a *conglomerate relation*, will be defined generically as it will also be used to describe information flows in systems.

When we use a simple relation to describe a flow policy there is an implicit assumption that if a may flow to c and b may flow to c then the join of information of classes a and b may also flow to class c. To remove this assumption, we will need to enumerate whether information formed from the combination of a and b can flow to c. Thus the information flow relation will be a relation of type $(\mathcal{P}\mathit{classes}) \leftrightarrow \mathit{classes}$. This relation describes whether a *conglomerate* of information classes (i.e., information drawn from a set of classes), may flow to some class.

Define the set of valid conglomerate relations to be

$$\begin{array}{|l}
[X] \\
\hline
\mathcal{R} : \mathcal{P}(\mathcal{P}X \leftrightarrow X) \\
\hline
\mathcal{R} = \{P : \mathcal{P}X \leftrightarrow X \mid \\
\qquad\qquad \forall a : X \bullet a \in \bigcup \operatorname{dom} P \cup \operatorname{ran} P \Rightarrow \\
\qquad\qquad\qquad \bigcap \operatorname{dom} P \rhd \{a\} = \{a\}\} \\
\hline
\end{array}$$

For some conglomerate relation $P : \mathcal{R}[X]$, then $A \mapsto a \in P$ means that the conglomerate of elements A is dominated by a in P. This is akin to saying that the 'join' of elements of A is dominated by a. We can use this structure to capture relations that do not have consistent bounds. For example, we may have maplets $\{a, c\} \mapsto c$ and $\{b, c\} \mapsto c$, but not have $\{a, b, c\} \mapsto c$. The constraint on the set $\mathcal{R}[X]$ is such that an element is always dominated by itself, regardless of what conglomerate may be involved (a form of reflexivity). We shall see later when we consider the set $\mathcal{R}[\mathit{classes}]$ that these are reasonable restrictions to make for flow policies.

A conglomerate relation from $\mathcal{R}[X]$ is defined over a subset of its base type X. Define the *alphabet* of conglomerate relation to be this set. Since we have for any $P \in \mathcal{R}[X]$

$$\operatorname{ran} P = \bigcup \operatorname{dom} P$$

the alphabet of P is simply $\operatorname{ran} P$, and thus,

$$\begin{array}{|l}
[X] \\
\hline
\alpha\_ : \mathcal{R}[X] \rightarrow \mathcal{P}X \\
\hline
\alpha\_ = \mathrm{ran}\,\_ \\
\hline
\end{array}$$

Some useful definitions on the set of conglomerate relations are

$$\begin{array}{|l}
[X] \\
\hline
\top\_, \\
\bot\_ : \mathcal{P}X \rightarrow \mathcal{R}[X] \\
\hline
\forall\, C : \mathcal{P}X \bullet \\
\qquad \top C = \{a : X \mid a \in C \bullet \{a\} \mapsto a\} \\
\qquad \bot C = \{A : \mathcal{P}X;\ a : X \mid a \in A \wedge A \cup \subsetneq C \bullet A \mapsto a\} \\
\hline
\end{array}$$

Relation $\top A$ gives the smallest possible conglomerate relation for an alphabet A, and $\bot A$ the largest. Enumerating all possible relations can be rather tedious, so the operator $\rightsquigarrow$ should be used when possible

$$\begin{array}{|l}
[X] \\
\hline
\_ \rightsquigarrow \_, \\
\_ \hookrightarrow \_ : \mathcal{P}X \times X \rightarrow \mathcal{R}[X] \\
\hline
\forall\, A : \mathcal{P}X;\ a : X \bullet \\
\qquad A \rightsquigarrow a = \top A \cup \{A' : \mathcal{P}X \mid A' \subseteq A \bullet A' \cup \{a\} \mapsto a\} \\
\qquad A \hookrightarrow a = (\top A \cup \{a\}) \cup \{A \cup \{a\} \mapsto a\} \\
\hline
\end{array}$$

Example 1 As already seen, we can use conglomerate relations to describe relations that do not posses consistent upper bound operations. Consider a conglomerate relation R with alphabet $\{a, b, c\}$ and maplets:

$$\begin{array}{lll}
\{a\} \mapsto a & \{b\} \mapsto b & \{c\} \mapsto c \\
\{a, c\} \mapsto c & \{b, c\} \mapsto c &
\end{array}$$

or defined in shorthand as,

$$R = (\{a\} \rightsquigarrow c) \cup (\{b\} \rightsquigarrow c)$$

This relation does not have a consistent upper bound (join) on elements a and b: element a is dominated by c, element b is dominated by c, but the conglomerate $\{a, b, c\}$ (which represents their join) is not dominated by c. $\triangle$

A conglomerate relation $P : \mathcal{R}[X]$ contains an *aggregation exception* if there are conglomerates $A, B : \mathbb{P}X$, and $a : X$, such that

$$A \mapsto a \in P \wedge B \mapsto a \in P \wedge A \cup B \mapsto a \notin P$$

In the last example, relation R contained an aggregation exception. A dual to an aggregation exception is a *separation exception*, where there are conglomerates $A, B : \mathbb{P}X$, and $a : X$ such that

$$A \cup B \mapsto a \in P \wedge (A \mapsto a \notin P \vee B \mapsto a \notin P)$$

4.1 A Framework of Conglomerate Relations

In this section we will show that the set of conglomerate relations $\mathcal{R}[X]$ forms a lattice. Appendix A contains all the appropriate proofs.

The abstraction operator allows us to hide certain elements of a conglomerate relation. If P is a conglomerate relation and C some set of elements, then $P@C$ gives P abstracted to elements of C. This can be thought of as viewing P though the 'eyes' of the window C.

$$\begin{array}{|l}
[X] \\
\hline
\_@\_ : \mathcal{R}[X] \times \mathbb{P}X \rightarrow \mathcal{R}[X] \\
\hline
\forall P : \mathcal{R}[X];\ C : \mathbb{P}X \bullet \\
\qquad P@C = \{A : \mathbb{P}X \bullet A \cap C \mapsto A\} ⨾ (P \rhd C) \\
\hline
\end{array}$$

If $C \mapsto a$ appears in relation P, then $C \cap \alpha P \mapsto a$ appears in $P@C$ iff $a \in C$. The abstraction operator always returns a valid conglomerate policy, Some laws for abstraction are: given $P, Q : \mathcal{R}[X];\ B, C : \mathbb{P}X$, then

$$\begin{array}{rcl}
\alpha(P@C) & = & \alpha P \cap C \\
P@\{\} & = & \{\} \\
P@\alpha P & = & P \\
(P@B)@C & = & P@(B \cap C) \\
P \subseteq Q \wedge B \subseteq C & \Rightarrow & P@B \subseteq Q@C
\end{array}$$

Example 2 A conglomerate relation has alphabet $\{a, b, mid\}$ and is defined as

$$R = \{a\} \rightsquigarrow mid \cup \{a, mid\} \rightsquigarrow b$$

(a total ordering with $a < mid < b$). If element mid is hidden we get,

$$R@\{a,b\} = \{a\} \rightsquigarrow b$$

(a simple ordering $a < b$). If $R = \{a\} \rightsquigarrow mid \cup \{mid\} \rightsquigarrow b$ (i.e., non-transitive), then $R@\{a,b\} = \top\{a,b\}$. $\triangle$

Example 3 A conglomerate policy with alphabet $\{a,b,c\}$ is defined as

$$\begin{aligned} R &= \{a,b\} \hookrightarrow c \\ &= \top\{a,b,c\} \cup \{\{a,b,c\} \mapsto c\} \end{aligned}$$

Note how *only* the conglomerate of a and b can be dominated by c. If R is abstracted to $\{a,c\}$ we get

$$R@\{a,c\} = \{a\} \rightsquigarrow c$$

reflecting that when R is viewed though a window $\{a,c\}$, the viewer cannot see how b can affect the relation. and thus does not know when maplet $\{a,c\} \mapsto c$ is observed that it was a's join with b that made it possible. $\triangle$

There is a *partial ordering relation* $\sqsubseteq$ defined between conglomerate relations. If $P \sqsubseteq Q$ then we say that Q is more restrictive than P, in that any maplet that is not allowed by P will also not be allowed by Q. Note however, that Q may introduce additional elements.

$$\begin{array}{|l} [X] \\ \hline \_ \sqsubseteq \_ : \mathcal{R}[X] \leftrightarrow \mathcal{R}[X] \\ \hline \forall P, Q : \mathcal{R}[X] \bullet \\ \qquad P \sqsubseteq Q \Leftrightarrow \\ \qquad\qquad \alpha P \subseteq \alpha Q \wedge Q@\alpha P \subseteq P \\ \hline \end{array}$$

Conglomerate restrictiveness and abstraction are monotonic with respect to each other: given $P, Q : \mathcal{R}[X]$; $B, C : \mathbb{P}X$, then

$$B \subseteq C \wedge P \sqsubseteq Q \Rightarrow P@B \sqsubseteq Q@C$$

Restrictiveness is rather like a combination of refinement and concealment[14]. If $P \sqsubseteq Q$, we use abstraction (concealment) to hide any new elements introduced by Q, and then check that the result is a subset (refinement) of P.

The set of conglomerate relations $\mathcal{R}[X]$ forms a lattice with lowest upper and greatest lower bound operators defined by $\sqcup$ and $\sqcap$ respectively.

$$
\begin{array}{|l}
\hline
[X] \\
\_ \sqcup \_, \\
\_ \sqcap \_ : \mathcal{R}[X] \times \mathcal{R}[X] \rightarrow \mathcal{R}[X] \\
\hline
\forall P, Q : \mathcal{R}[X] \bullet \\
\quad P \sqcup Q = \{A : \mathcal{P}X;\ a : X | A \cup \{a\} \subseteq \alpha P \cup \alpha Q \wedge \\
\qquad\qquad (a \in \alpha Q \Rightarrow A \cap \alpha Q \mapsto a \in Q) \wedge \\
\qquad\qquad (a \in \alpha P \Rightarrow A \cap \alpha P \mapsto a \in P)\} \\
\quad P \sqcap Q = P@(\alpha P \cap \alpha Q) \cup Q@(\alpha P \cap \alpha Q) \\
\hline
\end{array}
$$

Note that for $P, Q : \mathcal{R}[X]$ we have,

$$
\begin{aligned}
\alpha(P \sqcup Q) &= \alpha P \cup \alpha Q \\
\alpha(P \sqcap Q) &= \alpha P \cap \alpha Q
\end{aligned}
$$

Example 4 Define the relations

$$
\begin{array}{llll}
P & = \{a\} \rightsquigarrow b & Q & = \{b\} \rightsquigarrow a \\
R & = \{b, c\} \rightsquigarrow a & S & = \{b\} \rightsquigarrow c
\end{array}
$$

The join of P and Q must preserve the restrictions of both policies, and thus $P \sqcup Q = \{a, b\}$, i.e., no flows are possible except reflexivity. In the join,

$$P \sqcup R = \top\{a, b, c\} \cup \{c\} \rightsquigarrow a$$

the maplet $\{a, b, c\} \mapsto a$ is not included, since it introduces a conflict with $\{a, b\} \mapsto b \in P$, when viewed through the alphabet of P. If one policy introduces new classes, then *any* flow involving those classes are valid, as long as they do not violate the original policy. Thus

$$P \sqcup S = \{a\} \rightsquigarrow b \cup \{a, b\} \rightsquigarrow c \cup \{c\} \rightsquigarrow a$$

$\triangle$

The set of all conglomerate relations with the same alphabet A forms an algebra that is a sublattice of $\mathcal{R}[X]$ with restrictiveness defined by superset; $\sqcup$ defined by intersection; $\sqcap$ defined by union, and universal upper and lower bounds defined by $\top A$ and $\bot A$, respectively. Since it forms an algebra, each conglomerate relation P has a complement defined by $\overline{P}$.

$$\begin{array}{|l}
\multicolumn{1}{l}{[X]} \\
\overline{\_} : \mathcal{R}[X] \to \mathcal{R}[X] \\
\hline
\forall P : \mathcal{R}[X] \bullet \\
\quad \overline{P} = \top \alpha P \cup (\bot \alpha P - P) \\
\hline
\end{array}$$

Since $\overline{P}$ is complement operator, we have

$$\begin{aligned} P \sqcup \overline{P} &= \top \alpha P \\ P \sqcap \overline{P} &= \bot \alpha P \end{aligned}$$

4.2 Classes of Conglomerate Relations

The set of conglomerate relations that do not possess any separation exceptions is defined as

$$\begin{aligned} \mathcal{R}_{-S}[X] == \{R : \mathcal{R}[X] | \forall A, A' : \mathcal{P}X;\ a : X \bullet \\ A \cup A' \mapsto a \in R \Rightarrow \\ \{A \mapsto a, A' \mapsto a\} \subseteq R\} \end{aligned}$$

We call $\mathcal{R}_{-S}[X]$ the set of *aggregation relations*. Conglomerate join is closed over aggregation relations.

The set of conglomerate relation that do not posses any aggregation exceptions is

$$\begin{aligned} \mathcal{R}_{-A}[X] == \{R : \mathcal{R}[X] | \forall A, A' : \mathcal{P}X;\ a : X \bullet \\ \{A \mapsto a, A' \mapsto a\} \subseteq R \Rightarrow \\ A \cup A' \mapsto a \in R\} \end{aligned}$$

We call this set the set of separation relations. Conglomerate join is closed over separation relations.

The set of conglomerate relations that posses neither aggregation nor separation exceptions is

$$\mathcal{R}_{\diamond}[X] == \mathcal{R}_{-A}[X] \cap \mathcal{R}_{-S}[X]$$

We call this set the set of reflexive relations. It is easy to show that any relation $R \in \mathcal{R}_{\diamond}[X]$ can be represented by a simple reflexive relation $X \leftrightarrow X$. Conglomerate join is closed over reflexive conglomerate relations, and is equivalent to the definition of join for reflexive relations defined in [10]

Finally, the set of quasi ordered conglomerate relations (reflexive, transitive, and no bound exceptions) can be defined as

$$\begin{aligned} \mathit{quoset}[X] == \{R : \mathcal{R}_{\diamond} | \forall\, a, b, c : X \bullet \\ \{a\} \rightsquigarrow b \cup \{b\} \rightsquigarrow c \subseteq R \Rightarrow \\ \{a\} \rightsquigarrow c \subseteq R\} \end{aligned}$$

4.3 Inconsistent Lower Bounds

The definition of a conglomerate relation cannot capture relations that have inconsistent lower bounds: if a is dominated by b and a is dominated by c, then it implicitly implies that a is dominated by b and c. If we wish to capture relations that have inconsistent lower bounds it is necessary to enumerate lower bounds as conglomerates, not single components. Thus a conglomerate relation would be of the form $\mathcal{P}X \leftrightarrow \mathcal{P}X$. By modelling inconsistent lower bounds, we can capture flow relations of the sort: information is allowed flow from a to b, or from a to c but not both.

In this report we develop a high water mark mechanisms that can enforce flow policies described as conglomerate relations. The mechanisms we propose are currently suitable only for enforcing conglomerate relations with consistent lower bounds, i.e., policies from $\mathcal{R}[\mathit{classes}]$. Thus, in this report we will not consider policies that may have inconsistent lower bounds.

5 Separation and Aggregation as Flow Properties

In this section we will show how a conglomerate relation might be used to represent the flows and flow policy for a system. We will illustrate how conglomerate relations can capture aggregation and separation (of duty) properties. Section 5.3 will define what is meant by secure information flow.

5.1 System Abstraction

A conglomerate of entities is used to represent a set of entities that source some information, in concert. Define the set of all possible abstract systems to be

$$systems == \mathcal{R}[ents]$$

Given some system $S : systems$, then for $E : \mathcal{P}ents$ and $f : ents$, $E \mapsto f \in S$ means that information can flow from conglomerate of entities E to entity f over the lifetime of the system S. This *can flow* relation is an abstraction of the functionality of the system and can capture more properties about the information flow than the simple entity to entity relation described in section 3.

How does the constraints defined on conglomerate relations reflect reasonable requirements for information flow in systems? The 'reflexivity' requirement ensures that information can always flow from an entity to itself.

Example 5 A simple system has two entities A and B, which correspond to a file and a user. This system will permit information to flow only from the file A to user B, and not in the other direction. The flows in the system can be captured by

$$Simple = \{A\} \leadsto B$$

This system contains no aggregation or separation exceptions. $\triangle$

Example 6 A system maintains coordinate information in files *Long* and *Lat*, and has a single user, *SysOp*. The system enforces an aggregation policy such that the system operator may only view one of the files. The information flows in this system can be captured by schema *Coord-System*.

Coord-System

$S : systems$

$S = \{Long\} \rightsquigarrow SysOp \cup$
$\quad \{Lat\} \rightsquigarrow SysOp$

We will use the schema notation as a convienent way of packaging up an abstract system.

In this system, there is no flow from conglomerate $\{Long,Lat\}$ to $SysOp$, implying that, while the individual flows can occur, a flow from the longitude and latitude file (an aggregate) is not possible.

This policy is not a particularly practical example of an aggregation policy, but gets across the idea of aggregation in a simple manner. Lunt [17], argues that by appropriate normalisation such a policy should not be expressed in terms of aggregation. However, Meadows[20] shows that there are useful policies that are best expressed as aggregation policies. We will examine these policies in later sections. $\triangle$

In addition to representing aggregation scenarios (as in the last example), conglomerate relations can also capture separation of duty: a flow $E \cup E' \mapsto f$ is allowed, but $E \mapsto f$ is not. Separation of duty policies are normally viewed as control or integrity policies, and examples of these will be given in section 11. The next example example looks at separation and information flow.

Example 7 A university examinations system maintains two files about students: *Health*, which gives (some) details on student health, and *Exams*, which gives their marks from recent exams. The Head of department scans though both files and determines the final mark for each student. College policy dictates that the health file must be consulted, so that a 'borderline' student with health problems may be considered favourably. Thus, the system must implement a form of separation policy, in that the head may look at both files in concert, but not individually. However, whether the head acts appropriately on the health information cannot be controlled—the policy only ensures that (some) health information was consulted. The flows of such a system might be described by

Results-DB

$S : systems$

$S = \{Health,Exams\} \hookrightarrow Head$

The presence of the flow $\{Exams, Health\}$ to *Head*, implies that conglomerate information from exams and health can flow to the head, however individually exams and health cannot flow to the head. Separation (flow) policies will be considered in section 8. △

5.2 Information Flow Policies

An information flow policy defines the flows that are allowed between the classes of information in the system. The structure of the flow policies discussed in section 3 imply the existence of consistent upper and lower bound operators. This implies that if information is allowed flow from a to c and from b to c, then an aggregate or join of information of class a and b is allowed flow to c. However, as has already been pointed out, we may wish to design systems where this does not hold, i.e., a may flow to c, b may flow to c, but a and b may not flow to c. Thus we will describe an information flow policy as a conglomerate relation, which will allow us to enumerate how collections of information may flow.

A conglomerate of classes is used to represent information that has originated from the classes in the conglomerate, in concert. Define the set of information flow policies to be

$$policies == \mathcal{R}[classes]$$

Given a policy $P : policies$, $A \mapsto b \in P$ means that information is allowed flow from class conglomerate A to class a.

The constraints on conglomerate relations also apply to policies: it is implicitly assumed that information may always flow to itself, regardless of the conglomerate.

Example 8 The traditional military ordering can be described as

```
Military
  P : policies
  ─────────────
  P = {classified} ⇝ secret ∪
      {classified,secret} ⇝ top-secret
```

Note how every permitted conglomerate flow must be enumerated. △

Example 9 Continuing with the coordinates example, a flow policy can be defined to describe the appropriate flow restrictions. Let classes `lat` and long represent latitude and longitude information, and let op represent the class of information that an operator is allowed handle. The flow policy can be captured as

Coords-Pol

$P : \mathit{policies}$

$P = \{\texttt{long}\} \leadsto \mathsf{op} \cup \{\texttt{lat}\} \leadsto \mathsf{op}$

In this policy latitude or longitude information is allowed flow to class operator, however, an *aggregate* of longitude and latitude (information sourced from class lat and long in concert) is not allowed flow to the operator. △

Example 10 A private hospital information system processes information of class **recs** (medical history); **treat** (treatments given to patients); **acc** (patient accounts); **dir** (shareholder information); and **mgmt** (information handled by management). How information may flow between these different classes is described by the reflexive relation in figure 1. Note how **treat**

$$\begin{tikzcd}
\texttt{recs} & & \texttt{dir} \\
& \texttt{mgmt} \arrow[ul] \arrow[ur] & \\
\texttt{treat} \arrow[uu] \arrow[ur] & & \texttt{acc} \arrow[uu] \arrow[ul]
\end{tikzcd}$$

Figure 1: Flow policy HOSPITAL

information is allowed flow to **recs** or **mgmt**, but fo. confidentiality reasons, canrot flow to class **dir**. Similarly, **acc** information is not allowed flow to **recs** (for profitability reasons). Management is allowed coordinate all this information given these constraints.

This policy (taken from [11]) can be described as the conglomerate relation

Hospital

$P : policies$

$$\begin{aligned} P = {} & \{\mathtt{treat,acc}\} \leadsto \mathtt{mgmt} \cup \\ & \{\mathtt{treat,mgmt}\} \leadsto \mathtt{rec} \cup \\ & \{\mathtt{acc,mgmt}\} \leadsto \mathtt{dir} \end{aligned}$$

$\triangle$

Example 11 Users of a dial-up stock market database must be aware of charges when they access stock information. This can be captured by the separation policy

Stock-Pol

$P : policies$

$$\begin{aligned} P = {} & \{\mathtt{charges}\} \leadsto \mathtt{user} \cup \\ & \{\mathtt{charges,stock}\} \hookrightarrow \mathtt{user} \end{aligned}$$

A user is always allowed sink charges information, but may only sink stock information as a conglomerate with charges information. $\triangle$

When describing a separation flow policy, one must realise that the separation is based on *flows*, not control. Thus we call them separation *flow* policies. In the example above the user is allowed sink information based on both stock and charges information. This user can in turn forward this aggregate information to any other user. Separation policies are considered further in sections 8 and 11.

5.3 Secure Information Flow

Recall the multilevel security requirement for information flow policies as,

> if information can flow from class a to class b in a system implies that $a \mapsto b$ must hold in the policy.

This can be restated for conglomerate flow polices as,

> if information can flow from class conglomerate A to class b in a system, implies that $A \mapsto b$ must hold in the policy.

The relationship between a system and a flow policy is established by binding each entity to a information class from the policy. The class of each entity represents the class of information that the entity is allowed sink and source. Thus the confidentiality policy can be captured by

FMC-Conf-Pol

$P : policies$

$\beta\_ : ents \nrightarrow classes$

$\operatorname{ran} \beta\_ \subseteq \alpha P$

Function $\beta\_$ gives the information class from the flow policy P for each entity of the system.

A system is secure so long as the multilevel security requirement is maintained. Thus a secure system can be defined by

FMC-Secure-System

FMC-Conf-Pol

$S : systems$

$\operatorname{dom} \beta\_ = \alpha S$

$\forall E : \mathcal{P} ents;\ f : ents \bullet$

$\quad E \mapsto f \in S \Rightarrow$

$\qquad \beta(\!| E |\!) \mapsto \beta \epsilon \in P$

This schema states that a system S is secure by a given confinement policy if for every flow from entity conglomerate E to entity f then the (conglomerate) class of the information sourced by E may flow to the class of f. The restriction $\operatorname{dom} \beta\_ = \alpha S$ ensures that every entity of the system S has an information class.

Example 12 Consider the coordinates examples (6 and 9). The system *Coord-DB* is secure by policy `Coord-Pol`, since every possible flow of the system is allowed by the policy. Suppose the system contained a trojan horse that generated flow from $\{Long, Lat\}$ to Op, then the system would no longer be secure since

$$\begin{aligned} \bigcup \beta(\!|\{Long, Lat, Op\}|\!) &= \{\texttt{long,lat,op}\} \\ \beta Op &= \texttt{op} \end{aligned}$$

and the flow $\{\texttt{long,lat,op}\} \mapsto \texttt{op}$ is not allowed in the policy `Coord-Pol`. $\triangle$

Note that our definition of a secure system is based on ensuring that flows between individual entities are secure, as opposed to ensuring flows between all entities of one class to all entities of another class are secure. In the coordinate example, we do not consider the possibility of two operators colluding: one operator reads longitude information, and another reads latitude information. This can be avoided by using entities to represent persistent knowledge environments[19]. If it was felt that operators in the same building would collude, then we could represent the building's network as an entity of class op. As soon as somebody on the network accesses longitude information, the network's class changes, and nobody in that building can subsequently access latitude information.

5.4 Choosing a Flow Semantics

In the following sections we will look at how we can construct general mechanisms for enforcing arbitrary conglomerate flow policies. We shall see in section 8 that it is not practical to implement state mechanisms that can enforce *any* conglomerate policy, and therefore we seek to collect a selection of mechanisms that can enforce interesting classes of policy. Section 6 looks at a general mechanism for enforcing aggregation policies, and section 8, separation policies.

One of the simplest classes of flow policies are conglomerate relations that form quasi ordered sets. We know that such policies are easily enforced using traditional state based flow models. The following holds for systems that enforce quoset policies,

$$\begin{array}{l}
\forall \mathit{FMC\text{-}Secure\text{-}System} \bullet \\
\quad P \in \mathit{quoset}[\mathit{classes}] \Rightarrow \\
\quad\quad \exists\, \mathit{Secure\text{-}System}' \bullet \\
\quad\quad P = P' \wedge \beta_{-} = \beta'_{-} \wedge \\
\quad\quad S \in \mathit{quoset}[\mathit{ents}] \wedge S' \sqsubseteq S
\end{array}$$

This theorem tells us that given any secure system that enforces a quoset flow policy, then it is always possible to add extra (secure) flows to the system so that the system also forms a quoset. Therefore if we are interested in only enforcing quoset policies, we need not look for an information flow semantics that can capture transitivity, aggregation or separation, exceptions. Similar conclusions can be made about reflexive policies (the semantics need not express aggregation or separation properties), and aggregation and separation policies.

6 Implementing Aggregation Policies

In [11,12] a model was proposed that could be used to enforce aggregation policies. We will now demonstrate how this model, the group confinement model, can be used to enforce any conglomerate policy that does not posses separation properties (i.e., policies from $\mathcal{R}_{-S}[classes]$). The group confinement model is implemented using a generalisation of a high water mark model.

6.1 FMG: Group Confinement Model

In the group confinement model the confidentiality policy is described by a lattice flow policy and entities are bound to classes from this lattice as described below.

$$
\begin{array}{|l}
\hline
\textit{FMG-conf-pol}[C] \\
L : \textit{lattice}[C] \\
\beta_{\bot}\_ : \textit{ents} \rightarrow\!\!\!\!\!\!+\; C \\
\beta_{\top_s}\_ : \textit{ents} \rightarrow\!\!\!\!\!\!+\; \mathcal{P}_1 C \\
\hline
\textit{dom}\, \beta_{\bot} = \textit{dom}\, \beta_{\top_s} \\
\forall\, e : \mathrm{dom}\, \beta_{\bot} \bullet \\
\qquad \beta_{\bot} e \in \mathrm{dom}(L \rhd \beta_{\top_s} e) \\
\hline
\end{array}
$$

Lattice L gives the flow policy. Its components can be instantiated as classes or if desired as sets of classes if a powerset lattice policy is used. Every entity is bound to a set of intervals from L (all with the same lower bound $\beta_{\bot}$), and the upper bounds are given by the set $\beta_{\top_s}$. This entity binding is interpreted as:

- entity e may source information at class $\beta_{\bot}$ or higher, and
- may sink information at any class $a \in \beta_{\top_s}$ or lower.

This binding is subject to the multilevel security requirement. The group confinement model can be defined in abstract terms as:

FMG-Secure-System$[C]$

$$\begin{array}{|l}
\textit{FMG-conf-pol} \\
S : \textit{systems} \\
\hline
\operatorname{dom} \beta_\perp = \alpha S \\
\forall E : \mathcal{P}\textit{ents};\ f : \textit{ents} \bullet \\
\quad E \mapsto f \in S \Rightarrow \\
\qquad \bigoplus \beta_\perp (\!| E |\!) \in \operatorname{dom}(L \rhd \beta_{\top_s} f) \\
\hline
\end{array}$$

A system is considered secure if for every conglomerate flow $E \mapsto f$ in the system, then the class of information generated by E (i.e., the lowest upper bound of the class of information each $e \in E$ can source) must be permitted to flow to entity f (i.e., there must be some class in $\beta_{\top_s} f$ that dominates the class of the source information).

Note how this characterisation of a secure system can enforce aggregation exceptions: an entity may be allowed sink information of class a or class b but not both if their join $(a \oplus b)$ is not dominated by some component of $\beta_{\top_s}$. We shall see later how the shape of the bindings of entities determine the class of policy enforced.

We now propose a function that transforms an arbitrary aggregation policy into a lattice, plus sets of intervals for each class (of the form defined in *FMG-conf-pol*). Given some aggregation policy $P : \mathcal{R}_{-S}[\textit{classes}]$ and $a, b \in \alpha P$, then b forms a bound on a, if it dominates a in the policy, and every thing that conglomerates involving b can flow to, then a can also flow to, i.e., if $a \leq b$ where

$$\begin{array}{lll}
(P)a \leq b & \Leftrightarrow & \{a, b\} \mapsto b \in P \wedge \\
& & \forall B : \mathcal{P}\textit{classes};\ c : \textit{classes} \bullet \\
& & \quad B \cup \{b\} \mapsto c \in P \Rightarrow B \cup \{a, b\} \mapsto c \in P
\end{array}$$

Define the set of all bounds of class a to be

$$\text{bounds } P\ a = \{b : \textit{classes} | (P) b \leq a\}$$

We now define a mapping from aggregation policy P to intervals of a powerset lattice $\mathcal{P}_{\mathcal{L}}\alpha P$ by

$$\begin{array}{|l}
\Phi_{\perp}\_ : \mathit{policies} \nrightarrow (\mathit{classes} \nrightarrow \mathbb{P}\mathit{classes}) \\
\Phi_{\top_s}\_ : \mathit{policies} \nrightarrow (\mathit{classes} \nrightarrow \mathbb{P}\mathbb{P}\mathit{classes}) \\
\hline
\forall P : \mathit{policies} \bullet \\
\quad P \in \mathcal{R}_{-S}[\mathit{classes}] \Rightarrow \\
\quad\quad \Phi_{\perp} P = \{a : \mathit{classes} | a \in \alpha P \bullet a \mapsto \text{bounds } P\ a\} \wedge \\
\quad\quad \Phi_{\top_s} P = \{a : \mathit{classes} | a \in \alpha P \bullet a \mapsto \text{maxs}(\text{dom } P \rhd \{a\})\}
\end{array}$$

The mapping is flow preserving, in the sense that for any aggregation policy $P : \mathcal{R}[\mathit{classes}]$, we have

$$\begin{array}{l}
\forall A : \mathbb{P}\mathit{classes};\ b : \mathit{classes} \bullet \\
\quad A \cup \{b\} \subseteq \alpha P \Rightarrow \\
\quad\quad (A \mapsto b \in P) \Leftrightarrow \exists B \in \Phi_{\top_s} P\ b \bullet \bigcup \Phi_{\perp}(\!| A |\!) \subseteq B
\end{array}$$

Note the similarity between the relation

$$\bigcup \Phi_{\perp}(\!| A |\!) \subseteq B$$

and the test for security in the *FMG-Secure-System* schema. We can enforce any aggregation policy using the group confinement model and the transformation above were: an entity e bound to class a in the aggregation policy $P : \mathcal{R}_{-S}[\mathit{classes}]$ will have

$$\begin{array}{rcl}
\beta_{\perp} e & = & \Phi_{\perp} P\ a \\
\beta_{\top_s} e & = & \Phi_{\top_s} P\ a
\end{array}$$

with a powerset lattice policy $\mathcal{P}_{\mathcal{L}} \alpha P$. Thus systems enforcing aggregation policies can be characterised by

$$\begin{array}{|l}
\textit{Aggregation-Secure-System} \\
\hline
\textit{FMG-Secure-System}[\mathbb{P}\mathit{classes}] \\
\textit{FMC-conf-pol} \\
\hline
P \in \mathcal{R}_{-S}[\mathit{classes}] \\
L = \mathcal{P}_{\mathcal{L}} \alpha P \\
\beta_{\perp}\_ = \beta\_ \mathbin{;} (\Phi_{\perp} P) \\
\beta_{\top_s}\_ = \beta\_ \mathbin{;} (\Phi_{\top_s} P) \\
\hline
\end{array}$$

Since Φ is flow preserving, it follows that any such (secure) group confinement system is also a secure conglomerate system, i.e.,

$$\begin{array}{l}
\forall \textit{Aggregation-Secure-System} \bullet \\
\quad \textit{FMC-Secure-System}
\end{array}$$

The flow preserving mapping maps an aggregate policy P to a powerset lattice $\mathcal{P}_{\mathcal{L}}\alpha P$. If P has a large number of components, then $\mathcal{P}_{\mathcal{L}}\alpha P$ will be even larger. We can improve on the mapping using the optimal flow preserving mapping proposed by Denning[9] as follows

1. Given a aggregate policy P, create a partially ordered set Q with components

   $$\Phi_{\perp}\ P(\!|\alpha P|\!) \cup \bigcup \Phi_{\top_s}\ P(\!|\alpha P|\!)$$

   and an ordering relation defined by subset.

2. Denning gives a terminating algorithm that takes an arbitrary poset (a subset of a powerset lattice), and adds additional components so that each element has a unique lowest upper and greatest lower bounds. Use this algorithm on the poset Q, to give the lattice to be enforced by the high water mark model.

Example 13 Consider the coordinates example 9. Table 1 gives the flow preserving mappings for classes op, lat, and long. The lattice to be used by

class	$\Phi_{\perp}$Coords	$\Phi_{\top_s}$Coords
op	{op}	{{op,long}.{op,lat}}
long	{long}	{{long}}
lat	{lat}	{{lat}}

Table 1: Bindings for Coordinate Flow Policy

FMG can be either the powerset lattice $\mathcal{P}_{\mathcal{L}}${op,lat,long}, or the lattice in figure 2, generated using Denning's transformation on the poset generated by Φ. △

6.2 iFMG: A State based Refinement of FMG

We will chose a system model that is an abstraction of the traditional mandatory access control models. Since we are concerned with just the mechanisms for enforcing the confidentiality policy, the model is considerably stripped down: we have a fixed set of entities throughout the the life of the system, and only flows due to accesses are modelled, not actual accesses.

$$\begin{tikzcd}
 & & \{\mathtt{op,long,lat}\} & & \\
 & \{\mathtt{op,long}\} \arrow[ur] & & \{\mathtt{op,lat}\} \arrow[ul] & \\
\{\mathtt{long}\} \arrow[ur] & & \{\mathtt{op}\} \arrow[ul] \arrow[ur] & & \{\mathtt{lat}\} \arrow[ul] \\
 & & \{\} \arrow[ull] \arrow[u] \arrow[urr] & &
\end{tikzcd}$$

Figure 2: Transformed Coordinate Lattice

Firstly, we identify two kinds of system entities: memorable ($memble$) and memoryless ($memless$)

$$\begin{array}{|l}
memless, \\
memble : \mathcal{P}ents \\
\hline
memble \cap memless = \{\}
\end{array}$$

A memorable entity is an entity that can 'remember' information sunk to it, and is liable to forward that information at any later state. An example of a memorable entity is a file: information written to a file can be retrieved later by performing a read. A memoryless entity is an entity, which having sunk information will, in essence, forget the information and is relied on not to source that information at a later state. An example of a memoryless entity is a trusted subject: it is trusted not to source (inappropriately) any information sunk. In the original definition[11] of the group confinement model only memorable entities were considered.

A state of a system captures the flows that could occur due to the accesses during that state. These 'access flows' are drawn from the set:

$$
\begin{array}{|l}
\textit{acc-flows} : \mathcal{P}(\textit{ents} \leftrightarrow \textit{ents}) \\
\hline
\textit{acc-flows} = \{A : \textit{ents} \leftrightarrow \textit{ents} \mid \\
\qquad\qquad \operatorname{dom} A \subseteq \textit{memble} \cup \textit{memless} \land \\
\qquad\qquad \operatorname{id} A \subseteq A \land \\
\qquad\qquad A \rhd \textit{memble} \mathbin{;} A \subseteq A\}
\end{array}
$$

Information flow is assumed to be reflexive. Information flow through memorable entities is assumed to be transitive: if entity e writes to memorable entity f and entity g can read entity f, then there is also a flow from e to g. A memoryless entity does not necessarily 'propagate' information from its sources to its sinks. Since we are modelling only flows due to accesses, we assume that there are no aggregation properties on memorable entities: i.e, if information can flow from entity e to memorable entity f and from entity g to entity f, then there there is also an implicit flow from conglomerate $\{e, g\}$ to f. If f is memoryless, then we assume that it forgets one while viewing the other. Thus we choose to represent the flows as a simple relation between single entities. Section 8 will consider a more elaborate scheme for modelling flows at a state.

A secure state in the IFMG model is defined as

$$
\begin{array}{|l}
\hline
\textit{Secure-State}[C] \\
\textit{FMG-conf-pol}[\delta_{hwm}/\beta_{\bot};\ \delta_{lims}/\beta_{\top_s}] \\
A : \textit{acc-flows} \\
\hline
\operatorname{dom} A = \operatorname{dom} \delta_{hwm} \\
\forall\, e, f : \textit{ents} \bullet \\
\qquad e \mapsto f \in A \rhd \textit{memble} \Rightarrow \\
\qquad\qquad \delta_{hwm} e \mapsto \delta_{hwm} f \in L \\
\qquad e \mapsto f \in A \rhd \textit{memless} \Rightarrow \\
\qquad\qquad \delta_{hwm} e \in \operatorname{dom}(L \rhd \delta_{lims} f) \\
\hline
\end{array}
$$

Each entity e has a current high water mark given by $\delta_{hwm} e$[1]. This can be thought of as representing the class of information that the entity has sunk to date. The set δ_{lims} gives the limits to which the high water mark may rise. The flows for the current state are given by A. A state is secure if, for every flow $e \mapsto f \in A$, (memorable f) then the class of information held by e must be dominated by the class of f. If f is memoryless, then any

[1]We use δ for any variable that can change as a system progresses

information sunk is immeaditally 'forgotten', and thus its high water mark need not reflect the information it has sunk. Therefore the flow is secure so long as f may sink the information.

The initial state does not permit any flows except reflexivity

$$
\begin{array}{|l}
\textit{Initial-State}[C] \\
\textit{Secure-State} \\
\hline
A = \mathrm{id}\, A \\
\hline
\end{array}
$$

When a system makes a transition from one state to another (as a result of some access requests), the lattice policy and system entities must remain fixed. However, the high water marks are allowed to float upwards to reflect the information they may hold. As the high water marks rise, certain limits may have to be removed so that the bindings are still valid (i.e., so that the δ_{lims} dominate the high water mark). Note that the bindings of memoryless entities remain static as they 'forget' between states. In [11] we show how to calculate the new high water marks (for memorable entities) in a precise manner. This is reflected in the definition of a secure transition:

$$
\begin{array}{|l}
\textit{Secure-Trans}[C] \\
\Delta \textit{Secure-State}[C] \\
\hline
L = L' \\
\mathrm{dom}\, A = \mathrm{dom}\, A' \\
\forall \epsilon : \mathit{ents} \bullet \\
\quad \epsilon \in (\mathrm{dom}\, \delta_{hwm}) \cap \mathit{memble} \Rightarrow \\
\quad\quad \delta'_{hwm}\epsilon = \bigoplus \delta_{hwm} (\!|\, \mathrm{dom}\, A' \rhd \{\epsilon\} \,|\!) \land \\
\quad\quad \delta'_{lims}\epsilon \subseteq \delta_{lims}\epsilon \cap \mathrm{dom}(\{\delta'_{hwm}\epsilon\} \lhd L) \\
\quad \epsilon \in (\mathrm{dom}\, \delta_{hwm}) \cap \mathit{memless} \Rightarrow \\
\quad\quad \delta'_{hwm}\epsilon = \delta_{hwm}\epsilon \land \\
\quad\quad \delta'_{lims}\epsilon = \delta_{lims}\epsilon \\
\hline
\end{array}
$$

This mechanism provides us with the ability to enforce aggregation policies: a high water mark may drift towards any limit, but as it does so, certain other limits are removed, and the entity may no longer access information at those classes.

A system is characterised by an initial state; a state transition function and a set of reachable states. The system is secure if the basic security theorem holds:

Theorem 1 A system is secure if it starts from a valid initial state, and if every state transition achievable by the transition function is a secure *transition. Such a secure system can be characterised by*

```
┌─ iFMG-Secure-System[C] ──────────────────────
│ Initial-State
│ Σ : ℙSecure-Trans
└──────────────────────────────────────────────
```

□

Example 14 Consider the coordinates example 13. Table 1 gave the flow preserving mappings (page 30). If the system operator is memorable then, from the initial state he may enter a state with access to longitude information, or a state with access to latitude information. He may not enter a state with access to both. If he enters a state with access to longitude information, his high water mark needs to rise to match the class of information he has sunk, i.e., class {`op,long`}, and this results in the removal of {`op,lat`} from his δ_{lims} ensuring that latitude information cannot be accessed.

Note that in this simple example, it is the class of the user (the system operator) *that changes, and thus there is no oppertunity for a covert channel* due to denial of access. However, suppose the system operator owned a file (memorable) that had class `op`. This file, if shared, can be used to create a covert channel: the system operator could first read some longitude information; a trojan horse will then access latitude information and if the coordinate is above the equator, it writes it to the operator's file, if it is below the equator it does nothing. By testing for denial of access on the file, the operator can determine what hemisphere the coordinate is. This problem is addressed further in [11]. △

6.3 Formal Basis for iFMG

To justify that an iFMG system is secure, in the sense of the abstract model, it is necessary to prove that *iFMG-Secure-System* is a refinement of *FMG-Secure-System* (and thus a suitable refinement of a secure conglomerate system enforcing aggregation policies). Thus we need to construct an abstraction mapping between the abstract FMG and the state based iFMG model, and prove that a system secure by the state based model will also be secure by the abstract model. This has already been done in [11], and it is not pursued further.

6.4 A Framework of High Water Mark Models

When enforcing an aggregation policy using the group confinement high water mark model, the binding of an entity at any state can be pictured as in figure 3. The entity is allowed sink any information that appears under δ_{lims}, and source information at or above δ_{hwm}. The area between δ_{hwm} and

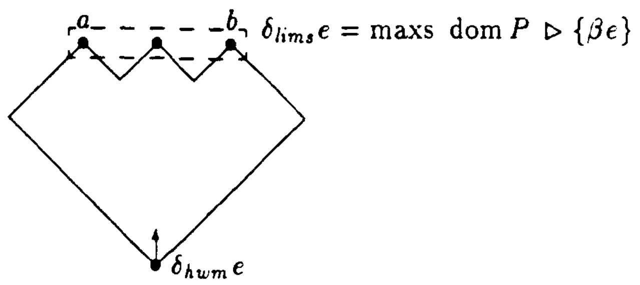

Figure 3: High Water Mark for Aggregation Policies

δ_{lims} gives the allowable future high water marks for the entity. As the high water mark rises, the set of possible future high water marks diminish. If δ_{hwm} heads towards the limit a, then the limit at b will eventually disappear, so that the entity no longer has a potential to read information at this class (b). This forms the basis for the implementation of aggregation policies.

Policy join ($\sqcup$) is closed over aggregation policies, implying that we can build a policy to be enforced by the group confinement model, based on a collection of smaller aggregation policies. An example of this is the Chinese wall policy in example 17.

6.4.1 Reflexive Policy Model

If a flow policy P is reflexive (i.e., a member of $\mathcal{R}_{\diamond}[classes]$) then each class will have only a single maximum in $\Phi_{\top_s} P$, since we know that for $A, B : \mathcal{P}classes$; $c : classes$ then

$$(A \mapsto c \in P) \wedge (B \mapsto c \in P) \Rightarrow A \cup B \mapsto c \in P$$

and we have

$$\Phi_{\top_s}\ P\ a = \{\bigcup \text{dom}\, P \rhd \{a\}\}$$

Given this, the conditions on states and state transitions simplify considerably: each entity is bound to a high water mark and a single static limit. The resulting state machine model is the interval confinement model described in [10].

Policy join ($\sqcup$) is closed over reflexive policies, implying that we can build a policy to be enforced by the interval model, based on a collection of smaller reflexive policies.

The binding of each entity can be pictured as in figure 4. An entity can

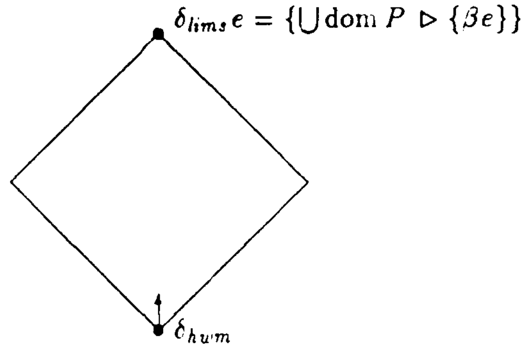
$\delta_{lims} e = \{\bigcup \text{dom}\, P \rhd \{\beta e\}\}$
δ_{hwm}

Figure 4: High Water Marks for Reflexive Policies

sink anything below δ_{lims}, and can source anything above δ_{hwm}. The high water mark may rise to any class in the area bounded by this interval. As the high water mark rises, there are (lower) classes that the entity can no longer source to. However, the entity can always sink everything under the single δ_{lims} (i.e., there are no aggregation exceptions).

This class of high water mark mechanism is similar to the high water mark mechanism adopted by the Compartmented Mode Workstation[26], with all entities memorable. Thus, we conclude that the policies enforced by the CMW can be classed as reflexive. The high water marks of memoryless entities are static, and they can be compared to partially trusted subjects[3].

6.4.2 Quasi Ordered Policy Model

If the policy to be enforced form a quoset, then the flow preserving mapping simplifies even further to, for P : quoset[$classes$]; a : $classes$,

$$\begin{aligned} \Phi_{\perp}\ P\ a &= \bigcup \text{dom}\, P \rhd \{a\} \\ \Phi_{\top_s}\ P\ a &= \{\bigcup \text{dom}\, P \rhd \{a\}\} \end{aligned}$$

which corresponds to the Birkoff mapping[5], that maps a quasi ordered set to a powerset lattice. In this case the high water mark model corresponds to the traditional lattice model, with each entity bound to a static class drawn from a lattice. Binding is static since δ_{hwm} is already at the limit, and thus cannot rise.

7 Examples

7.1 Reflexive Policies

Example 15 Consider the simple military policy in example 8. Suppose we wish to extend this by introducing two new classes **admiral** and **general**. Class **admiral** has no restrictions on the information classes it may sink and/or source. The only restriction on class **general** is that it cannot sink top-secret information. This policy can be defined as

Mil-2

$P : policies$

$P = \texttt{Military} \sqcup \bot\{\texttt{admiral}\} \sqcup \{\texttt{general}\} \rightsquigarrow \texttt{top-secret}$

Observe the use of conglomerate join: any flow is permitted so long as it does not violate the three operand policies **Military**, $\bot\{$**admiral**$\}$ (no restrictions on an admiral), and $\{$**general**$\} \rightsquigarrow$ **top-secret** (general is dominated by top-secret).

This policy can be transformed to a lattice policy by the mapping Φ, calculated in table 2. *The application of Denning's transformation* on these

class a	$\Phi_\bot$ Mil-2 a	$\Phi_\top$ Mil-2 a
classified	{c,g,a}	{c,g,a}
secret	{c,s,g,a}	{c,s,g,a}
top-secret	{c,s,t,g,a}	{c,s,t,g,a}
general	{c,g,a}	{c,s,g,a}
admiral	{c,g,a}	{c,s,t,g,a}

Table 2: Bindings for policy Mil-2

classes give a simple total ordering (lattice),

$$\{c,g,a\} \subset \{c,s,g,a\} \subset \{c,s,t,g,a\}$$

which is simply a re-labeling of the original military lattice. If an admiral A is bound to class **admiral**, then in the model he gets bound to the interval pair ({c,g,a}, {c,s,t,g,a}). This entity A is memoryless—he is trusted to handle **admiral** information appropriately, and therefore may simultaneously sink and source any class. If the general creates a (memorable) file, it would initially be assigned class **admiral**, and if secret information were sunk to that file, its high water mark needs to rise to reflect the file's contents.

Thus any memoryless entity bound to class **admiral** is like a trusted subject bound to the interval classified, top-secret; a memorable entity bound to class **admiral** is like an object with an initial high water mark classified, that may rise to top-secret. A memoryless entity bound to class **general** is like a partially trusted subject who is trusted to handle classified and secret information appropriately. A memorable entity bound to **general** is like an entity with an initial high water mark of **classified**, that is allowed rise to the limit **secret**. △

In the last example the new policy `Mil-2` can be thought of as a refinement of `Military`, i.e., we have

$$\texttt{Military} \sqsubseteq \texttt{Mil-2}$$

It is possible when developing a policy to end up with an unintentionally restrictive result. For example, if a policy was defined as

$$\texttt{Mil-3} = \texttt{Mil-2} \sqcup \{\texttt{top-secret}\} \leadsto \texttt{secret}$$

we have `Mil-2` $\sqsubseteq$ `Mil-3`, but this new policy no longer permits information flow from secret to top-secret. Thus, when building complex policies it is worth investigating how much of the original policies are preserved. For example 15, we have

$$\texttt{military} = \texttt{Mil-2}@_{\alpha}\texttt{Military}$$

which gives reassurance that the new policy preserves the original intentions of policy `Military`.

Example ~~16~~ The hospital policy (example 10) is transformed by the flow preserving mapping given in table 5 to the lattice with components

$$\{\{\},\{\texttt{t}\},\{\texttt{a}\},\{\texttt{t,m}\},\{\texttt{t,m,r}\},$$
$$\{\texttt{t,m,a}\},\{\texttt{d,m,a}\},\{\texttt{t,m,a,d,r}\}\}$$

where each class is denoted by its first letter. In this example, a hospital administrator would be bound to class **mgmt**. For the policy to work effectively, the administrator should be modelled as memoryless: he is trusted to handle treatment and accounts information appropriately, i.e., not forward treatments onto shareholders. Any files he may create should be memorable so that their high water marks reflect the information class they contain. △

class	$\Phi_{\bot}$Hospital	$\Phi_{\top_s}$Hospital
treat	{t}	{t}
acc	{a}	{a}
mgmt	{m}	{t,m,a}
recs	{t,m,r}	{t,m,r}
dir	{d,m,a}	{d,m,a}

Figure 5: Flow Preserving Mapping for Hospital-Pol

7.2 Aggregation Policies

Example 17 A stockmarket database holds confidential information on different organisations. A unique information class ('dataset') is used to denote the information of each company. Consultants are allowed access the information held in the database. We have,

$$\begin{array}{|l} \mathit{Orgs} : \mathbb{P}\,\mathit{classes} \\ \mathsf{cons} : \mathit{classes} \end{array}$$

where *Orgs*, is the set of all organisation information classes, and **cons** denotes the class of information that may be handled by consultants.

Organisations can be organised in terms of conflict (of interest) sets. Two companies occur in the same conflict set if they have conflicting interests. A conflict set is of type

$$\mathit{conflict\text{-}set} == \mathbb{P}_1\,\mathit{Orgs}$$

A number of (possibly intersecting) conflict sets may defined over a set of organisations. For example, a bank might also have an insurance business, and thus may appear in both the banks and insurance conflict of interest sets.

The information in a conflict class must be kept disjoint: one bank is not allowed find out anything about another bank. A consultant is allowed access to the information on any *one* bank, but no more. Thus if a consultant is modelled as memorable, the conflict policy for a conflict set C can be defined by *conflict-pol* C, where

$$\begin{array}{|l} \mathit{conflict\text{-}pol}\_ : \mathit{conflict\text{-}set} \rightarrow \mathit{policies} \\ \hline \forall C : \mathit{conflict\text{-}set} \bullet \\ \quad \mathit{conflict\text{-}pol}\ C = \top(C \cup \{\mathsf{cons}\}) \cup \\ \qquad\qquad \bigcup\{c : C \bullet \{c\} \rightsquigarrow \mathsf{cons}\} \end{array}$$

A consultant is allowed consult for any number of organisations so long he preserves the conflict policy for every conflict of interest class, i.e., he can consult for any group of organisations so long as they have no conflict of interest.

Suppose there are three conflict of interest sets for banks, insurance companies and oil companies. The flow policy in this case can be captured by the Chinese wall

Chinese-Wall

$P : \mathit{policies}$

$\mathtt{Banks}, \mathtt{Oil}, \mathtt{Insurance} : \mathit{conflict\text{-}sets}$

$P = \mathit{conflict\text{-}pol}\ \mathtt{Banks} \sqcup$
$\quad \mathit{conflict\text{-}pol}\ \mathtt{Oil} \sqcup$
$\quad \mathit{conflict\text{-}pol}\ \mathtt{Insurance}$

$\triangle$

Example 18 The chinese wall policy in the last example was only concerned with constructing a wall around the consultant, and did not consider how organisations that do not have conflicting interests should relate to each another. While conflicting classes are disjoint in Chinese-Wall, information from classes that do not conflict can flow freely between one another. For example, if $a, b \in \mathtt{Banks}$, and $c \in \mathtt{Oil}$ such that a, b are not conflicting with c, then $\{a, b\} \leadsto c$ is a valid flow. Therefore, we propose a modified chinese wall where *all* organisations are disjoint, as

China-2

$\mathtt{Chinese\text{-}Wall}[\mathit{old\text{-}P}/P]$
$P : \mathit{policies}$

$P = \mathit{old\text{-}P} \sqcup \top(\mathtt{Banks} \cup \mathtt{Oil} \cup \mathtt{Insurance})$

$\triangle$

The conglomerate join operator is useful for joining policies with different alphabets together. However, before using, the policy specifier should be clear on its semantics. In example 17, *conflict-Pol* only considers the flow restrictions between a conflict set and a consultant. When two conflict policies are joined *all* flows are allowed that do not violate the original two policies. Thus, since a single conflict policy does not express any restrictions between classes of different conflict sets, the resulting chinese wall reflects this. Thus a policy specifier should always remember that the flow restrictions described in a flow policy extend only to the alphabet of the policy.

Example 19 In any realistic implementation of a chinese wall, there will exist sanatized information to which the conflict policy need not apply. Define the class of such information as

| **sanatized** : *classes*

Sanatized information has no flow restrictions on it, and thus a new Chinese wall can be defined as

China-3
$\texttt{Chinese-wall}[Old\text{-}P/P]$
$P : policies$
$P = old\text{-}P \sqcup \bot\{\texttt{sanatized}\}$

In this policy information from all banks are allowed flow to class **sanitized**, and class **sanitized** is allowed flow to consultant, but transitivity does not follow. Therefore this policy can only be successfully implemented by reclassification: A memoryless entity of class **sanitize** (typically a trusted person) reads all conflict set information, adds sufficient noise etc., and reclassifies it as sanitized, so that it can be sunk (as sanitized) by any consultant. Note that 'reclassification' refers, not to an actual class binding being changed, but a (trusted) entity sinking information and sourcing it at a lower class. A memorable entity of class **sanitize** is also allowed sink all conflict set information, but in doing so its high water mark rises and it becomes inaccessible to the consultants. $\triangle$

Example 20 A telephone directory holds the names and numbers of individuals from a number of different departments. The information about each department (telephone numbers) is assigned a unique class. A quantity aggregation policy can be specified which states that no user is allowed access to the numbers of more than *limit* departments.

We can define a general quantity aggregation policy as

$$\begin{array}{|l}
\textit{qty-agg}(\_,\_,\_) : (\mathcal{P}\textit{classes} \times \aleph \times \textit{classes}) \rightarrowtail\!\!\!\!\rightarrow \mathcal{R}_{-S}[\textit{classes}] \\
\hline
\forall A : \mathcal{P}\textit{classes};\ \textit{lim} : \aleph;\ c : \textit{classes} \bullet \textit{lim} > 1 \Rightarrow \\
\qquad \textit{qty-agg}(A, \textit{lim}, c) = \{A' : \mathcal{P}\textit{classes} | \\
\qquad\qquad A' \subseteq A \wedge \#A \leq \textit{lim} \bullet A \cup \{c\} \mapsto c\}
\end{array}$$

Where, give a set of classes A, natural number N and a class c, then *qty-agg*(A, N, c), specifies that conglomerates drawn from A, not larger than N may flow to class c. Thus, if *Depts* give the set of departments and **user** the class for a user, then policy *qty-agg*(*Depts*, 10, **user**) ensures that a user may not discover the telephone numbers of more than ten departments.

The conflict policy in a chinese wall, is a quantity aggregation policy where a consultant may only read one (1) dataset.

A high water mark implementation of this policy will keep track of telephone numbers propagated between users (using the system). If one user accesses accounting and personell numbers, then this is reflected in his high water mark. When this user forwards information to another user, these numbers will be propagated and the latter user's high water mark will reflect this.

A variation on this policy might allocate limits based on the clearance (classified, secret, etc) of the user. This is left as an exercise for the reader. △

Example 21 The granularity of aggregation in a conglomerate policy extends to to information classes, not entities. Thus if we wished to generalise the last example to the traditional telephone-book example: no more that *limit* telephone numbers may be released to an individual; we need a separate class for each number:

$$|\ \textit{Numbers} : \mathcal{P}\textit{classes}$$

The policy is simply *qty-agg*(*Numbers*, *limit*, **user**). Due to the potential size of *Numbers*, it would not be realistic to directly implement such a policy using our high water mark model. But the high water mark model is just one possible refinement of the abstract conglomerate model. We could pick a restricted class of policies $\mathcal{R}_{\aleph}[\textit{classes}]$ that expressed only quantity aggregation policies and then build a new state model where entities' high water marks are numbers which reflect the number of telephone numbers sunk so far. This model would not be as precise as the group confinement model, since when one user forwards information to another, the destination's high water mark must be updated to the *sum* of the source high water marks. Precision is lost because they may share some telephone numbers. △

Example 22 A large corporation is divided into a number of divisions, and divisions into departments.

$$\mid \; \textit{Depts}, \textit{Divs} : \mathcal{P}\textit{classes}$$

It is possible construct a conglomerate policy that described all the allowable flows between individual departments and divisions (and their conglomerates). An alternative approach is to use military style classifications to associate degrees of trustworthiness with employees and how much information they are allowed access.

A classified employee is only allowed access to information about at most three (3) departments, and one (1) division:

$$\begin{aligned} \textit{Class-Emp} &= \textit{qty-agg}(\textit{Depts}, 3, \texttt{classified}) \sqcup \\ &\quad \textit{qty-agg}(\textit{Divs}, 1, \texttt{classified}) \end{aligned}$$

Note that there is no constraint on *what* departments and/or division is accessed.

A secret employee is permitted access to at most six (6) departments and two (2) divisions

$$\textit{Sec-Emp} = \textit{qty-agg}(\textit{Depts}, 6, \texttt{secret}) \sqcup \textit{qty-agg}(\textit{Divs}, 2, \texttt{secret})$$

A top-secret employee is permitted access to all departments and divisions

$$\textit{Top-Emp} = \textit{Divs} \cup \textit{Depts} \rightsquigarrow \texttt{top-secret}$$

The overall corporate security policy is thus

```
Corporate-Policy
  P : policies
  ----------------------------------------
  P = Class-Emp ⊔ Sec-Emp ⊔ Top-Emp⊔
      Military
```

Note that the military policy is still enforced. However, suppose the classified user has accessed three departments, and the secret user has accessed six other, different, departments, then the classified user may no longer forward information to the secret user. △

In all of these examples we concerned ourselves with what information a user may sink. We did not consider what information users may source. For example, it is desirable that classified users from example 22 should be allowed source departmental information. Thus we could redefine

$$\begin{aligned} \mathit{Class\text{-}Emp} = {} & (\mathit{qty\text{-}agg}(\mathit{Depts}, 3, \mathtt{classified}) \sqcup \\ & \mathit{qty\text{-}agg}(\mathit{Divs}, 1, \mathtt{classified})) \cup \\ & \bigcup \{ d : \mathit{Depts} \bullet \mathit{Depts} \cup \mathtt{classified} \leadsto d \} \end{aligned}$$

which allows a classified user source information to any department. Note that we cannot express any aggregation exceptions on how many different departments classified may source to: our framework does not cater for inconsistent lower bounds.

8 Implementing Separation Policies

In section 6 we described a general high water mark model that could enforce any aggregation policy from $\mathcal{R}_{-S}[\mathit{classes}]$. In this section we will describe an alternative high water mark model that can enforce *any* conglomerate policy. It is not practical to implement this model in its entirety, and therefore section 8.4 considers useful classes of separation (and aggregation) policies that have reasonable enforcement mechanisms.

8.1 Separation Flow Policies Revisited

A separation exception in a flow policy extends to all information at the classes involved. The granularity of separation of duty is at the class level, and not necessarily at the entity level. For example, a policy might state that information may flow from class `lo` to class `hi`, and from class `hi` to class `lo` only in the presence of a security officer `so`. This policy might be captured as:

$$\begin{aligned} P = &\{\texttt{lo},\texttt{so}\} \rightsquigarrow \texttt{hi} \\ &\{\texttt{hi},\texttt{so}\} \hookrightarrow \texttt{lo} \end{aligned}$$

The conglomerate of information $\{\texttt{hi},\texttt{so}\}$, may flow to class `lo`. Suppose there are entities H, L, and SO, representing a high user, a low user and a security officer. The high user can send `hi` information to the security officer, who in turn forwards it to the low user. This is a valid flow since the information that the low user sinks is a conglomerate (join) of `hi` and `lo` information. Information cannot flow directly from the high user to the low user. However, the low user can now be thought of as holding conglomerate information of class $\{\texttt{lo},\texttt{hi},\texttt{so}\}$, which can be combined with *any* additional item of `hi` information, and the result may still flow to `lo`. Thus the separation exception occurs at class granularity, not entity granularity.

If we reexamine the requirements of this downgrading policy, we see that the desired policy is not a separation (flow) policy, but a reclassification policy. A security officer is allowed read high information and reclassify it as low. This can be captured by the reflexive relation

$$\texttt{Reclassify} = \{\texttt{lo},\texttt{sso}\} \rightsquigarrow \texttt{hi} \cup \{\texttt{hi}\} \rightsquigarrow \texttt{sso} \cup \{\texttt{sso}\} \rightsquigarrow \texttt{lo}$$

Information is allowed flow from high to sso, and from sso to low, but not from high to low. Downgrading (high to low) can occur only through a memoryless (partially trusted) security officer.

Thus, when describing a separation flow policy, one must remember that the separation is based on *flows*, not control. In the example above, class so represents information generated by the security officer, not control, and as information it (as part of a conglomerate) could propagate through (memorable) entities. We will consider separation as control, when we use conglomerate relations to express integrity policies in section 11. The state based security mechanisms will be built with this in mind.

8.2 FMU: A Universal High Water Model

The confinement bindings in the FMU model are identical to the bindings in the group confinement model, except that instead of giving a set of limits ($\beta_{\top_s}$) on the classes of information an entity may sink, we give a set of sinks: the information classes an entity may sink.

FMU-conf-pol$[C]$

$L : lattice[C]$

$\beta_{\perp} : ents \rightarrow C$

$\xi_{sink} : ents \rightarrow \mathcal{P}C$

$\mathrm{dom}\,\beta_{\perp} = \mathrm{dom}\,\xi_{sink}$

$\mathrm{ran}\,\xi_{sink} \subseteq \mathrm{dom}\,L$

$\beta_{\perp}- \in \xi_{sink}-$

Lattice L gives the flow policy. Every entity is bound to a component of this lattice by $\beta_{\perp}$, which represents the class of information the entity may source. Every entity is also bound to a set ξ_{sink} which gives the set of classes of information that the entity may sink. The only restriction we place on these bindings is that every entity may sink the class of information it sources. The difference between this binding and group confinement binding is illustrated in figure 6: the holes represent areas where separation exceptions occur: an entity may not sink any class within the separation area, however as soon as additional classes are joined, they rise out of the separation area and the flow is valid. Aggregation exceptions occur on an entity whenever the aggregate (join) of two classes that are valid sinks of the entity leave the area defined by ξ_{sink} The area between $\perp$ (the lower bound of L) and $\beta_{\top_s}$ in the FMG binding represents the set of classes that the entity may sink. Note that there are no holes in this area, and thus no separation exceptions can be expressed. Aggregation exceptions may only occur at the extremities of the binding.

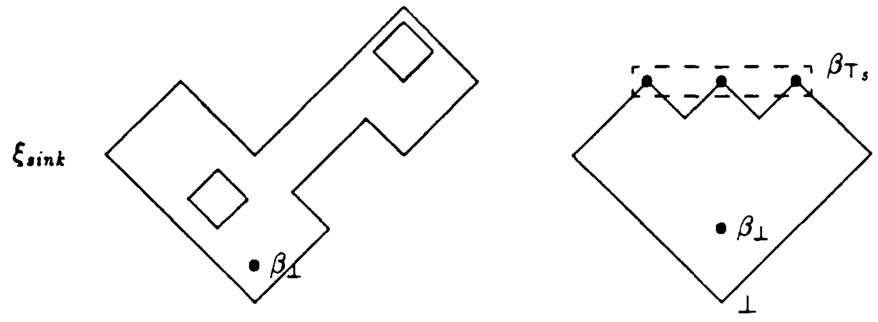

Figure 6: Typical FMU and FMG bindings

In a system, flows are constrained by the multilevel security requirement. Thus a secure system is defined as

```
── FMU-Secure-System[C] ──────────────
│ FMU-conf-pol[C]
│ S : systems
├──────────
│ dom β⊥ = αS
│ ∀ E : ℙents; f : ents •
│     E ↦ f ∈ S ⇒
│         ⊕ β⊥(|E|) ∈ ξ_sink f
└──────────────────────────────────────
```

A system is considered secure if for every conglomerate flow $E \mapsto f$ in the system, then the class of information generated by conglomerate E (i.e., the lowest upper bound of the class of information each $e \in E$ may source), is allowed to be sunk by f.

We can map any conglomerate policy P to a powerset lattice $\mathcal{P}_{\subseteq}\alpha P$ by

```
│ Φ⊥ : policies → (classes ⇸ ℙclasses)
│ Φ_sink : policies → (classes ⇸ ℙℙclasses)
├──────────
│ ∀ P : policies •
│     Φ⊥P = {a : αP • a ↦ {a}}
│     Φ_sink P = {a : αP • a ↦ dom P ▷ {a}}
```

This mapping is flow preserving in the sense that for P : *policies*

$$\forall A \mathcal{P} classes;\ b : classes \bullet$$
$$A \mapsto b \in P \Leftrightarrow \bigcup \Phi_\perp P (\!| A |\!) \in \Phi_{sink} P\ b$$

Combining this with the FMU model gives a model for enforcing conglomerate policies as

```
─ iFMC-Secure-System ──────────────────────
│ FMC-conf-pol
│ FMU-Secure-System[𝒫classes]
├──────────────
│ $L = \mathcal{P}_{C} P$
│ $\beta_{\perp -} = \Phi_{\perp} P \fatsemi \beta_{-}$
│ $\xi_{sink-} = \Phi_{sink} P \fatsemi \beta_{-}$
└──────────────────────────────────────────
```

Since Φ is flow preserving, it follows that any such secure iFMC system will be a secure conglomerate system, i.e.,

$$\forall \, \textit{iFMC-Secure-System} \bullet$$
$$\quad \textit{FMC-Secure-System}$$

8.3 iFMU: A State Based Refinement of FMU

In this section we will describe a high water mark model that can enforce arbitrary conglomerate flow policies. Section 8.3.1 views a system as a collection of entities that send messages to each other. We give an interpretation of information flow for such a system, and relate it to the abstract notion of information flow used in *FMU-Secure-System*. Given this view of a system, section 8.3.2 proposes a high water mark mechanism for enforcing conglomerate policies. Section 8.3.3 proves that any system that is secure by the high water mark mechanism will be secure by the abstract model FMU. Section 8.4 looks at restrictions on the class of conglomerate flow policies that result in simple tests and operations for the high water mark mechanisms.

8.3.1 System Abstraction

We represent a system as a collection of entities with an ability to send messages to one another. Entities represent the sources and sinks of information in this system for example, users, files, processes, windows, etc. The state of a system represents the current set of channels between entities, along which messages are being sent. We will assume that the existence of a channel represents an intention to use it to send and/or receive messages. The set of possible message flows between entities at any state are described by a conglomerate relation from the set

$$\left|\ \mathit{msg\text{-}flows} : \mathcal{PR}[\mathit{ents}]\right.$$

We will describe the constraints on this set shortly. Given any M : *acc-flows*, then $E \mapsto f \in M$ is interpreted as meaning that there is a commitment for information flow from conglomerate E to entity f. Thus a state M : *acc-flows* cannot express any aggregation exceptions, since every flow is a commitment and thus there is no opportunity for a choice between one flow or another.

There are two types of entities, memoryless and memorable

$$\begin{array}{|l}
\mathit{memble},\\
\mathit{memless} : \mathcal{P}\mathit{ents}\\
\hline
\mathit{memble} \cap \mathit{memless} = \{\}
\end{array}$$

A memorable entity is any entity that is liable to source any information it has previously sunk. Examples are files, variables, untrusted programs. An untrusted editor is memorable since it may contain a trojan horse that can 'remember' any secrets it edited. A memoryless entity will not inappropriately forward information it sinks. Trusted routines may be considered memoryless: a trusted windowing system that can correctly handle multi-level information will not steal a secret on one window and it reappear on a classified window. An appropriately classified user is prevented from cutting out a secret from one window and pasting it to a classified window: in this case there is an attempted flow from a secret window (entity) through a memorable paste buffer (entity), to a classified window (entity). Of course, the user could read the contents of the top-secret window and retype it into the classified window. However, there are easier ways of leaking a secret, for example using a telephone.

When a message is sent from a memorable entity e to entity f then in addition to the flow from e to f, there is a potential for transitive flows from everything that sinks to e, to f. Thus, if M : *msg-flows* represents the flows at the current state and given a memorable e, suppose $\{e, f\} \mapsto f \in M$ and $\{g, e\} \mapsto e \in M$; then $\{e, g, f\} \mapsto f \in M$ should also hold; however $\{g, f\} \mapsto f$ need not hold, since there there may not be an opportunity for a *direct flow* from g to f—it always goes through entity e. Given a memorable entity e, and current message flows M : *msg-flows*, the set of possible conglomerates it can forward are $P \rhd \{e\}$. Note that the forwarding of a conglomerate E always gets performed in concert with e itself. If E is a set of memorable entities, then the set of conglomerates it could forward is the set all-fwd(M, E), where

$$
\begin{array}{|l}
\text{all-fwd}(\_,\_) : \mathcal{R}[\mathit{ents}] \times \mathcal{P}\mathit{memble} \rightarrow \mathcal{P}\mathcal{P}\mathit{ents} \\
\hline
\forall M : \mathcal{R}[\mathit{ents}];\ E : \mathcal{P}\mathit{memble};\ e : \mathit{memble} \bullet \\
\quad \text{all-fwd}(M, \{\}) = \{\{\}\} \\
\quad \text{all-fwd}(M, E \cup \{e\}) = \\
\qquad \{F, G : \mathcal{P}\mathit{ents} | F \mapsto e \in M \wedge \\
\qquad\quad G \in \text{all-fwd}(M, E - \{e\}) \bullet \\
\qquad\quad F \cup G\}
\end{array}
$$

Memoryless entities are assumed not to propagate information in this way. Thus we can define the set of valid flow relations due to message communication at a state as

$$
\begin{array}{|l}
\mathit{msg\text{-}flows} : \mathcal{P}\mathcal{R}[\mathit{ents}] \\
\hline
\mathit{msg\text{-}flows} = \\
\quad \{M : \mathcal{R}[\mathit{ents}] | \\
\qquad M \in \mathcal{R}_{-A}[\mathit{ents}] \wedge \\
\qquad \forall E : \mathcal{P}\mathit{ents};\ f : \mathit{ents} \bullet \\
\qquad\quad E \mapsto f \in M \Rightarrow \\
\qquad\qquad (E \cap \mathit{memless}) \cup \text{all-fwd}(M, E \cap \mathit{memble}) \mapsto f \in M\}
\end{array}
$$

A variation of the all-fwd function gives the set of all entities that can be forwarded, in some way, by a conglomerate relation as

$$
\begin{array}{|l}
\text{all-fwd*}(\_,\_) : \mathcal{R}[\mathit{ents}] \times \mathcal{P}\mathit{ents} \rightarrow \mathcal{P}\mathit{ents} \\
\hline
\forall M : \mathcal{R}[\mathit{ents}];\ E : \mathcal{P}\mathit{ents} \bullet \\
\quad \text{all-fwd*}(M, E) = \bigcup \operatorname{dom}(M \rhd E)
\end{array}
$$

Note that $\text{all-fwd*}(M, E) = \bigcup \text{all-fwd}(M, E)$.

If our system makes a transition from a state with flows M to a state with flows M', we must agree with what the overall flows should be. Clearly, the flows of M will remain in effect as potential flows of the system. However, at state M', memorable entities will propagate any information they received during M to their sinks in M'. Suppose there was a flow $\{e,f\} \mapsto f \in M$, and a flow $\{f,g\} \mapsto g \in M'$, then overall there is also a flow from $\{e,f,g\}$ to g. If there are just flows $\{e,g\} \mapsto g$ and $\{f,g\} \mapsto g$ at state M and an only flow of $\{g,h\} \mapsto h$ at state M', then since the flows represent a flow *commitment*, then we can can say that there is a flow from $\{e,f,g,h\}$ to h overall, but not necessarily a flow from $\{e,g\}$ to h At state M', we know that entity g holds information from both e and f, and it is this conglomerate that is propagated, not the individual e and f.

Thus, if a system has made a series of transitions that result in overall flows $S : \mathcal{R}[ents]$, and enters a new state with flows M, then each memorable entity e at state M will propagate the conglomerate all-fwd*$(S, \{e\})$ to all its sinks in M. Therefore, if a system has gone through a sequence of states $t : \mathrm{seq}_1\ \textit{msg-flows}$, the overall flows can be defined as flows t, where

$$
\begin{array}{|l}
\text{flows}\_ : \mathrm{seq}_1\ \textit{msg-flows} \to \mathcal{R}[ents] \\
\hline
\forall t : \mathrm{seq}_1\ \textit{msg-flows};\ M : \textit{msg-flows} \bullet \\
\quad \text{flows}\ \langle M \rangle = M \land \\
\quad \text{flows}\ t ^\frown \langle M \rangle = \\
\qquad \text{flows}\ t \cup \\
\qquad \{E : \mathcal{P} ents;\ f : ents \mid E \mapsto f \in M \bullet \\
\qquad\qquad (E \cap memless) \cup \text{all-fwd*}(\textit{flows}\ t, E \cap memble) \mapsto f\}
\end{array}
$$

Note that we have assumed that every state along t has the same alphabet.

If a system is described by a prefix-closed set of state sequences T, then the overall flows are calculated as $\bigcup \text{flows}(\!| T |\!)$.

8.3.2 A High Water Mark Model

We are now in a position to describe a high water model that enforces flow policies from the FMU model for systems whose flows can be modelled as above (section 8.3.1).

A system state describes the current message flows and gives the policy binding according to *FMU-conf-pol*.

$$
\begin{array}{|l}
\textit{Secure-State}[C] \\
\hline
\textit{FMU-conf-pol}[C][\delta_{hwm}/\beta_\perp] \\
M : \textit{msg-flows} \\
\hline
\alpha M = \mathrm{dom}\,\delta_{hwm} \\
\forall E : \mathcal{P} ents;\ f : ents \bullet \\
\quad E \mapsto f \in M \Rightarrow \\
\qquad \bigoplus \delta_{hwm}(\!| E |\!) \in \xi_{sink} f \\
\hline
\end{array}
$$

Each entity e has a current high water mark given by δ_{hwm}. This represents the class of information that the entity currently holds. It differs slightly from the high water mark in the group confinement model in that it represents the classes of information the entity has sunk up to, but *not* including the current state. Conglomerate relation M represents the flows due to the message commitments at the current state. If there is a flow $E \mapsto f \in M$ then the class of information generated by the entities in E must be sinkable by f, i.e. in $\xi_{sink} f$. Note that unlike the group confinement model, we cannot check that the class of conglomerate E is dominated by some limit of $\xi_{sink} f$, since $\xi_{sink} f$ may contain holes (recall figure 6).

The initial state permits any flows so long as it is secure.

$$\begin{array}{|l}
\textit{Initial-State}[C] \\
\hline
\textit{Secure-State}[C] \\
\hline
\end{array}$$

When a system makes a transition from one state to another (as a result of some communication requests), only high water marks of memorable entities may change. The high water marks may rise upwards to reflect any new information a memorable entity has sunk. A secure transition is defined as

$$\begin{array}{|l}
\textit{Secure-Trans}[C] \\
\hline
\Delta \textit{Secure-State} \\
\hline
L' = L \\
\xi'_{sink} - = \xi_{sink} - \\
\forall e : ents \bullet \\
\quad e \in memorable \cap \alpha M \Rightarrow \\
\qquad \delta'_{hwm} e = \bigoplus \delta_{hwm} (\!| \text{all-fwd}^*(M, \{e\}) |\!) \\
\quad e \in memless \cap \alpha M \Rightarrow \\
\qquad \delta'_{hwm} e = \delta_{hwm} e \wedge \\
\hline
\end{array}$$

The new class for $\delta'_{hwm} e$ is a member of $\xi_{sink} e$ since: given that M does not contain any aggregation exceptions, then we have that

$$\text{all-fwd}^*(M, \{e\}) \mapsto e \in M$$

and since M is secure it follows that

$$\bigoplus \delta_{hwm} (\!| \text{all-fwd}^*(M, \{e\}) |\!) \in \xi_{sink} e$$

which implies, by its definition, $\delta'_{hwm} e \in \xi_{sink}$. Note that, unlike most high water mark schemes, a high water mark may not drift upwards arbitrarily, it rises only to include the classes of information it has sunk. As before, a memoryless entity does not change its high water mark.

A system is characterised by a single initial state; a state transition function, and a set of reachable states. The system is secure iff the basic security theorem holds.

Theorem 2 A system is secure if it starts from a valid initial state, and if every state transition achievable by the transition function is a secure transition. Such a system can be characterised by

```
── iFMU-Secure-System[C] ──────────────────
│ Initial-State[C]
│ Σ : ℙSecure-Trans[C]
└───────────────────────────────────────────
```

□

Example 23 Consider the stock market database example 11. Table 3 gives the flow preserving mapping Φ for this policy, where the first letter of each class is used to denote the class. If entities U, C, and S represent a user,

class	$\Phi_\perp$Stock-Pol	Φ_{sink}Stock-pol
user	{u}	{{u},{u,c},{u,c,s}}
charge	{c}	{{c}}
stock	{s}	{{s}}

Table 3: Flow Preserving Mapping for Stock-Pol

a charges file, and a stocks database, then $\Phi_\perp$ gives the initial high water mark for each entity. If the user attempts to directly access the database S, then there is a flow $\{S, U\} \mapsto U$, which is invalid, since

$$\begin{aligned}\delta_{hwm} S \cup \delta_{hwm} U &= \{\mathtt{u,s}\} \\ &\notin \xi_{sink} U\end{aligned}$$

However, if the user first reads the charges file (state s_1), he may then read the stock information (state s_2). This secure history is outlined in table 4. Once the user has read stock information his high water mark rises to {u,c}, and now the flow $\{S, U\} \mapsto U \in s_2.M$, is secure. Note that the user

state s_i	$s_i.M$	$s_i.\delta_{hwm}U$
s_1	$\{C\} \rightsquigarrow U$	{u}
s_2	$\{S\} \rightsquigarrow U$	{u,c}

Table 4: A Secure history for Stock Access

could read both charges and stock simultaneously at state s_1. This would be modelled as

$$s_1.M = \{C, S\} \hookrightarrow U$$

However, we would require some assurance that the program that allows the user access the database does not provide flow $\{S\} \rightsquigarrow U$. Compare this state with a state with flows $M = \{C, S\} \rightsquigarrow U$, which is not secure since $\{S, U\} \mapsto U \in M$. $\triangle$

8.3.3 Formal Basis for iFMU

To justify that an iFMU system is secure, we have to prove that it is a refinement of *FMU-Secure-System*. Thus we need to construct an abstraction mapping between the abstract *FMU* and the state based *iFMU*, and prove that a system secure by *iFMU* will also be secure by *FMU*.

The *FMU* and *iFMU* share the same set of entities. In the *iFMU* the initial high water marks of entities correspond to the bindings assigned in *FMU*. The fact that high water marks change as the system progresses is a artifact of the implementation. If T gives the set of all possible secure histories of a system, then set of flows over the entire system is the union of the flows over individual secure histories, i.e., $\bigcup \text{flows}(\!| T |\!)$. We can prove that for any secure history t of the system, then

$$\begin{array}{l} \forall E : \mathcal{P}ents;\ f : ents \bullet \\ \quad E \mapsto f \in \text{flows}\ t \Rightarrow \bigoplus \delta_{hwm}(\!| E |\!) \in \xi_{sink} f \end{array}$$

which implies that any system secure by *iFMU* will also be secure by *FMU*.

8.4 A Framework of High Water Mark Models

Recall figure 6, which illustrated the shape of ξ_{sink} for each entity. Not only does ξ_{sink} define the allowable classes of information that an entity may sink, it also defines the possible classes the high water mark may rise to. Since ξ_{sink} does not change along a secure history, when enforcing a conglomerate policy P : *policies*, it is sufficient to store one copy of $\Phi_{sink}\ P\ a$ for each class a, and have $\xi_{sink} a$ cross reference this. While this does save considerably on space (each entity has a current high watermark, and a class from αP indexing into $\Phi_{sink}\ P\ a$), the structure $\Phi_{sink}\ P\ a$, has a potential to be quite large. Inspection of the conditions for secure states and transitions reveal that it would be impractical to build a completely general security mechanism to enforce an arbitrary conglomerate policy. We therefore seek classes of useful conglomerate policies such that the shape of Φ_{sink} result in simple conditions on states and state transitions. We can also save on the size of the generated lattice policy, if desired, by using Denning's transformation.

Sections 8.4.1 to 8.4.3 outline how effective security mechanisms could be arrived at for reflexive, aggregation and separation policies.

8.4.1 Reflexive Flow Policies

If a flow policy P is reflexive (i.e., a member of $\mathcal{R}_{\diamond}[classes]$), then any class $a \in \alpha P$ will have a single maximum in $\Phi_{sink}\ P\ a$ defined as $\Phi_{\top} P\ a$, where

$$\Phi_{\top} P\ a = \bigcup \operatorname{dom} P \rhd \{a\}$$

and since P has no separation exceptions, the area in Φ_{sink} has no holes, and thus ξ_{sink} can be represented by its extremities. If βe gives the class of entity e under policy P, then its initial binding is calculated as

$$\begin{aligned} \delta_{hwm} e &= \Phi_{\perp} P\ (\beta\ e) \\ \xi_{\top} e &= \Phi_{\top} P\ (\beta\ e) \end{aligned}$$

and will have the typical shape illustrated in figure 7. Given this, the conditions on secure states and secure transitions simplify considerably. The condition on a secure state that

$$\bigoplus \delta_{hwm} (\!| E |\!) \in \xi_{sink} f$$

must hold if $E \mapsto f \in M$, can be simplified to

$$\bigcup \delta_{hwm} (\!| E |\!) \subseteq \xi_{\top} f$$

Since the policy does not express any aggregation or separation exceptions, this can be simplified further, so that state flows are viewed as a simple

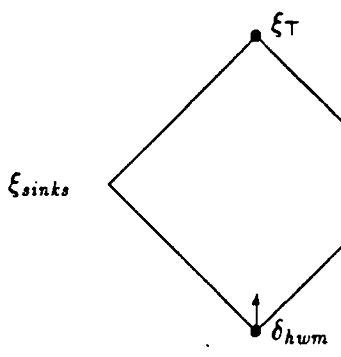

Figure 7: ξ_{sink} for Reflexive Policies

relation between single entities. Thus we result with variation of the interval confinement model[10].

Note again, the difference between the notion of a high water mark in the FMU, and a high water mark in the group confinement model: in FMG, δ_{hwm} gives the (join of the) class of all information that has been sunk by a memorable entity upto, and including, the current state; in FMU, δ_{hwm} gives the (join of the) class of all information that has been sunk by a memorable entity upto, but excluding, the current state. Both approaches are equally valid, however we found that it was easier to use the latter interpretation of a high water mark when developing the model for enforcing separation policies.

8.4.2 Aggregation Flow Policies

If the flow policy does not contain any separation exceptions, i.e., $P \in \mathcal{R}_{-A}[classes]$, then the typical shape of ξ_{sink} is as pictured in figure 8. Since there are no holes, then ξ_{sink} can be represented by its extremities. Each entity has a current high water mark δ_{hwm} and a set of limits on what it can sink, $\xi_{\top_s}$ (which can be calculated as the set of maximums on $\Phi_{sink}\ P$). As with group confinement, the high water mark will drift towards one limit from $\xi_{\top_{sink}}$, as it sinks information.

8.4.3 Continuous Separation Policies

While policies from $\mathcal{R}_{-S}[classes]$ do not contain any aggregation exceptions, the shape of the mapping Φ_{sink} can have an number of holes, which may

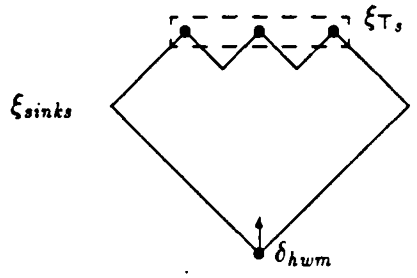

Figure 8: ξ_{sink} for Aggregation Policies

make implementation mechanisms impractical. We will therefore propose a restricted class of separation policies that do not contain any holes, so that ξ_{sinks} can be described by its extremities.

Define the set of *continuous* separation policies to be $\mathcal{R}_{-A*}[classes]$, where

$$\begin{aligned}
\mathcal{R}_{-A*}[X] == \{R : \mathcal{R}_{-A*}[X] \mid \forall A, B : \mathcal{P}X;\ c : X \bullet \\
\{A \mapsto c, B \mapsto c\} \subseteq R \land \\
\{c\} \subset A \land A \subseteq B \Rightarrow \\
\forall C : \mathcal{P}X \bullet A \subseteq C \land C \subseteq B \Rightarrow \\
C \mapsto c \in R\}
\end{aligned}$$

Given any $P : \mathcal{R}_{-A*}[classes]$, and $a : classes$, then the extremities of the area $(\Phi_{sink}\ P\ a) - \{\{a\}\}$ can be given by the mapping

$$\begin{array}{|l}
\Phi_{\bot_s} : \mathcal{R}_{-A*}[classes] \rightarrow classes \nrightarrow \mathcal{P}classes \\
\Phi_{\top} : \mathcal{R}_{-A*}[classes] \rightarrow classes \nrightarrow classes \\
\hline
\forall P : \mathcal{R}_{-A*}[classes] \bullet \\
\quad \Phi_{\bot_s} P = \{a : \alpha P \bullet a \mapsto \text{mins}\ (\text{dom}(P \rhd \{a\}) - \{\{a\}\})\} \\
\quad \Phi_{\top}\ P = \{a : \alpha P \bullet a \mapsto \bigcup(\text{dom}(P \rhd \{a\}))\}
\end{array}$$

Given this, we can prove that for any $P : \mathcal{R}_{-A*}[classes]$, then

$$\begin{aligned}
&\forall A : \mathcal{P}classes;\ b : classes \bullet A \neq \{b\} \Rightarrow \\
&\quad A \mapsto b \in P \Leftrightarrow \\
&\quad \exists B : \mathcal{P}classes \bullet B \in \Phi_{\bot_s} \land \\
&\qquad B \subseteq A \land A \subseteq \Phi_{\top}
\end{aligned}$$

Each entity will have a current high water mark δ_{hwm}, and ξ_{sinks} will be represented by $\xi_{\perp_s}$ and $\xi_{\top}$, giving the lower extremes and an upper extreme (there are no aggregation exceptions) of ξ_{sink}, respectively. This binding is pictured in figure 9. We know that a flow $E \mapsto f$ during some state is secure

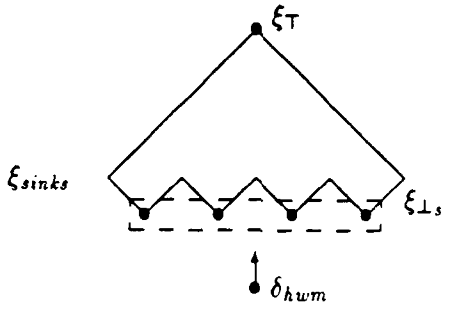

Figure 9: ξ_{sink} for Continuous Separation Policies

only if

$$\bigoplus \delta_{hwm}(\!| E |\!) \in \xi_{sinks} f$$

which can now be rewritten as

$$\begin{array}{l}(\bigoplus \delta_{hwm}(\!| E |\!) = \delta_{hwm} f) \vee \\ (\bigoplus \delta_{hwm}(\!| E |\!) \subseteq \xi_{\top} f \wedge \\ \exists B : \mathcal{P} classes \bullet B \in \xi_{\perp_s} f \wedge B \subseteq \bigoplus \delta_{hwm}(\!| E |\!))\end{array}$$

Notice how the high water mark must make an initial 'leap' from its initial value to the area bounded by $\xi_{\perp_s}$ and $\xi_{\top}$. This is how a separation exception gets enforced: only when sufficient classes are present can can information flow to the entity.

A similar mapping can be developed for continuous separation policies that allow aggregation exceptions (i.e., policies from $\mathcal{R}_{-A_*}[classes] \cup \mathcal{R}_{-S}[classes]$. Instead of a single top on Φ_{sink}, there are a number of tops. The bindings for such a mapping are illustrated in figure 10

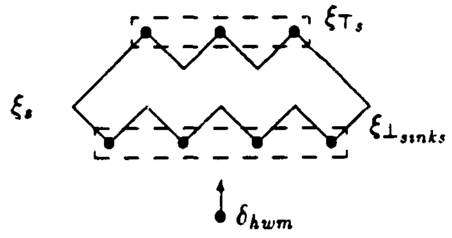

Figure 10: ξ_{sink} for Continuous Separation-Aggregation Policies

9 Examples

Example 24 The stocks database example 11, will have a flow preserving mapping described by table 5. and a lattice policy $\mathcal{P}_{\mathcal{L}}$`{u,c,s}` (or, Denning's

class	$\Phi_\perp$	$\Phi_{\perp_s}$	$\Phi_\top$
`user`	`{u}`	`{{u,c}}`	`{u,c,s}`
`charges`	`{c}`	`{c}`	`{c}`
`stock`	`{s}`	`{s}`	`{s}`

Table 5: Flow Preserving Mapping for `Stock-Pol`

transformation can be applied to the components given in table 5). △

Example 25 Separation exceptions can be used to prevent classes of information being made available until such time as it is appropriate. For example the policy

$$R = \{rel\} \rightsquigarrow b \cup \{\{rel, a\} \mapsto b\}$$

does not allow information flow from class a to class b unless is is accompanied by 'release' information of class rel. Suppose entities R, A and B are bound to rel, a and b, respectively. Initially, information may not flow from A to B. However, if there is a flow from R to A, and A is memorable, then the high water mark of A rises so that it is now allowed flow to B.

This can be used in the chinese wall example, whereby before a company 'goes public', its dataset is not accessible to consultants. By combining the dataset with release information, the company becomes public.

A conflict policy will give entities access only to classes that have been released:

$$\begin{array}{|l}
\textit{C-Rel-Pol\_} : \textit{conflict-set} \rightarrow \textit{policies} \\
\hline
\forall\, C : \textit{conflict-pol} \bullet \\
\quad \textit{C-Rel-Pol}\ C = \bigcup\{c : C \bullet \{c, \textit{rel}_c\} \hookrightarrow \texttt{cons} \\
\qquad\qquad\qquad\qquad \{\textit{rel}_c\} \rightsquigarrow \texttt{c}\}
\end{array}$$

If a company c has not been released, its high water mark is $\{c\}$ and a flow to the consultant is not possible. A company can be released by writing some release information to the entity holding the company information. This will cause the high watermark of the company entity to rise, so that it forms a conglomerate $\{c, \textit{rel}_c\}$, which may flow to consultant.

Note that to achieve the necessary granularity for this policy, it is necessary to have a different release class for each organisation class. If we had used a single release class for all organisations, then it could propagate from one dataset to a consultant, giving the consultant potential access to all other datasets. $\triangle$

Example 26 Consider a downgrading policy for the Military policy from example 15. We could use a reclassification scheme as described in section8.1. However, under this scheme the security officer is not accountable for what he reads and downgrades. A more satisfactory scheme would be to use a separation of duty rule which required the admiral to release information to the security officer for downgrading. The security officer is not permitted arbitrary access to secret and top-secret information. This policy can be specified as

$$\begin{array}{|l}
\texttt{Mil-Downgrade} \\
\hline
P : \textit{policies} \\
\hline
P = \texttt{Mil-2} \cup \\
\quad \{\texttt{admiral}\} \rightsquigarrow \texttt{so} \cup \{\texttt{so}\} \rightsquigarrow \texttt{classified} \\
\hline
\end{array}$$

The admiral can read top-secret information, reclassify it as **admiral** and forward it to the security officer. The security officer can take this information and reclassify it as classified. In both cases some trusted (memoryless) entity is necessary so that an admiral can sink a secret and source it as **admiral** (without its secret component); and a routine to allow the security officer sink admiral information and source it as **so**.

Now we have separation of duty: secrets cannot be downgraded without first being proposed by an admiral, and then cleared by a security officer. Note how the separation of duty is achieved by the use of memoryless entities, and a reflexive policy: no separation exceptions are in effect. This is an example of static separation of duty, where the separation occurs in a predefined sequence. This approach is similar to the technique in [15] for separation of duty for integrity policies, which uses partially trusted subjects (memoryless entities) bound to pairs of classes from a lattice integrity policy (implementing a reflexive integrity policy).

In section 11 we will give an example of a dynamic separation of duty integrity policy. $\triangle$

10 Relating Aggregation and Separation

Separation and aggregation exceptions can, in a sense, be thought of as duals of one another. An aggregation exception might state that A may flow to c and B may flow to c, but together they may not flow to c; a separation exception would state the opposite: together they may flow to c, but individually they may not. This section shows how this relationship can be formally established.

We can prove that, given any aggregate flow policy then its complement forms a separation policy, i.e.,

$$\forall P : \mathcal{R}_{-S}[classes] \bullet \overline{P} \in \mathcal{R}_{-A}[classes]$$

For example an aggregation policy that includes flows $\{a\} \rightsquigarrow c, \{b\} \rightsquigarrow c$, but not the flow $\{a, b\} \rightsquigarrow c$, when complemented will allow the flow $\{a, b\} \hookrightarrow c$. but not the flows $\{a\} \rightsquigarrow c$ or $\{b\} \rightsquigarrow c$.

The complement of a separation policy however, does not form an aggregation policy. For example a policy defined by

$$P = \top\{a, b, c\} \cup \{a\} \rightsquigarrow b$$

(b is disjoint) is a separation policy ($\mathcal{R}_{-A}[classes]$ includes all reflexive policies). However, $\overline{P}$ includes the flow $\{a, b, c\} \mapsto b$. but not the flow $\{a, b\} \mapsto b$, and thus $\overline{P}$ is not an aggregation policy. We can however, transform any continuous separation policy into a reflexive component and a separation component, and the complement of the separation component forms an aggregation policy.

A continuous separation policy $P : \mathcal{R}_{-A\bullet}[classes]$. can be made reflexive by filling in the 'hole' (area where separation occurs) to give a policy P_R defined as

$$P_R = \{A : \mathcal{P} classes;\ a : classes | A \subseteq \bigcup \operatorname{dom} P \rhd \{a\} \bullet A \cup \{a\} \mapsto a\}$$

This new policy is valid and reflexive ($P_R \in \mathcal{R}_{\diamond}[classes]$). The 'hole' that has been removed can be put into a new policy P_S that enforces only that hole (i.e., the separation exceptions), and has no other restrictions on flow. It is defined as

$$P_S = \bot \alpha P - (P_R - P)$$

and forms a continuous separation policy $P_S \in \mathcal{R}_{-A*}[classes]$. P_S can also be defined as

$$\begin{aligned} P_S &= (\bot\alpha P - P_R) \cup (\bot\alpha P \cap P) \\ &= P \cup (\bot\alpha P - P_R) \\ &= P \cup \top\alpha P \cup (\bot\alpha P - P_R) \\ &= P \sqcap \overline{P_R} \end{aligned}$$

and since a sublattice of policies with the same alphabet form an algebra, we can write

$$\begin{aligned} P_S \sqcup P_R &= (P \sqcap \overline{P_R}) \sqcup P_R \\ &= (P_R \sqcup P) \sqcap (P_R \sqcup \overline{P_R} \\ &= P \sqcup \top\alpha P \\ &= P \end{aligned}$$

Thus the policy P can be thought of as having a reflexive component P_R and a separation component P_S. We can prove that the complement of the separation component of P forms an aggregation policy, i.e.,

$$\forall\, P : \mathcal{R}_{-A*}[classes] \bullet \overline{P_S} \in \mathcal{R}_{-S}[classes]$$

Since $P = P_R \sqcup P_S$, we have

$$\begin{aligned} &\forall\, A : \mathcal{P}classes;\ b : classes \bullet \\ &\quad A \mapsto b \in P \Leftrightarrow \\ &\qquad (A \mapsto b \in P_R \wedge A \mapsto b \in P_S) \\ &\Leftrightarrow \\ &\forall\, A : \mathcal{P}classes;\ b : classes \bullet \\ &\quad A \mapsto b \in P \Leftrightarrow \\ &\qquad A = \{b\} \vee \\ &\qquad (A \mapsto b \in P_R \wedge \\ &\qquad\ A \mapsto b \notin \overline{P_S}) \end{aligned}$$

Thus P can be viewed as positively enforcing a reflexive component (P_R), and negatively enforcing an aggregation policy ($\overline{P_S}$). We can view this policy in terms of the binding pictures of section 8.4. Information may flow from class conglomerate A to class b iff $A = \{b\}$ or, A falls in $\Phi_{sink}\ P_R\ b$ (enforcing the reflexive part with the shape of figure 7), and if A does not fall in $\Phi_{sink}\ \overline{P_S}\ b$ (negatively enforcing the aggregation part, which has the shape in figure 8). Thus the shape of $\Phi_{sink}\ P\ b - \{\{b\}\}$ is simply the shape of $\Phi_{sink}\ P_R\ b$ minus the shape $\Phi_{sink}\ P_S\ b$, which turns out to be the original shape for a continuous policy described in figure 9.

11 Integrity Policies

A well known dual for the traditional lattice flow model is the Biba[4] integrity model. An integrity policy describes sets of clearances and a lattice relation between these clearances that defines their relative superiority. In this section we will illustrate how the conglomerate flow model can be used to enforce integrity policies described as conglomerate relations.

With an integrity model we are concerned with whether or not one entity may affect the integrity of another. We will use the same conglomerate structure to model a system, except that instead of information flow, we model integrity: if S : *systems*, then $E \mapsto f \in S$ means that the entities E can, as a conglomerate, affect in some way the integrity of entity f.

We can use the flow policy framework to describe integrity policies. Given integrity policy P : *policies*, then $A \mapsto b \in P$, means that, as a conglomerate A has higher clearance than b. Each entity in the system is assigned a single integrity clearance using the function β.

An integrity secure system is simply a system such that for every ability to change integrity $E \mapsto f \in S$, the integrity policy must be upheld, $\beta(\!|E|\!) \mapsto \beta f \in$ *policies*. Thus any system characterised by *FMC-Secure-System*, with the integrity interpretation, is a secure system.

Example 27 A cheque may only be written to by an accountant and a manager, but not by them individually. Thus a policy for writing cheques is the separation policy

Cheque-Pol

P : *policies*

$P = \top\{\texttt{acc},\texttt{mgr},\texttt{chk}\} \cup \{\texttt{acc},\texttt{mgr}\} \hookrightarrow \texttt{chk}$

△

We can use the conglomerate high water mark model to enforce integrity policies. However, when doing so, we must be careful about the propagation of integrity clearances i.e., integrity flow. If entity A can change the integrity of entity B, and B change the integrity of C, should the integrity clearance of A 'flow' to C? In the Biba model this is not an issue since the integrity policy is transitive, and thus it is implicit that A can affect C. However this question must be addressed if the integrity policy is non-transitive. We can model entities as memorable or memoryless, depending on whether or not they propagate integrity changes: affecting the integrity of a memorable entity has a potential to propagate those changes to everything the memorable entity can affect; affecting the integrity of a memoryless entity will have no propagating effect.

Example 28 The cheque writing policy can be implemented by the use of cheque transitions: the manager writes to a cheque transaction, and then forwards it to the accountant for clearance (or vice versa). Only when the transaction has been cleared can it be sent to the cheques file.

Define a set of unique labels to represent each cheque transaction

$CT : \mathcal{P} classes$

The cheque writing integrity policy can be specified as

Cheque-Trans

$P : policies$

$$P = \bigcup\{t : CT \bullet (\{\mathtt{mgr}\} \leadsto t) \cup (\{\mathtt{acc}\} \leadsto t)\} \cup \{\mathtt{mgr}, \mathtt{acc}, t\} \hookrightarrow \mathtt{chk}\}$$

To propose a cheque, a manager requests a transaction, writes to it and forwards that transaction to the accountant, who in turn clears it and send the transaction for printing. Note that an accountant is not permitted to change a transaction generated by the manager (and vice-versa). Compare this dynamic separation, achieved by aggregation exceptions, with the static separation in example 26. The cheque file may be updated only in the 'presence' of the manager, accountant and a transaction. We can prove that this policy is more restrictive that the original cheque policy, i.e.,

$\mathtt{Cheque} \sqsubseteq \mathtt{Cheque\text{-}Trans}$

Thus the original separation of duty rule for cheques is preserved in the more detailed policy.

To implement this policy, all entities must be memoryless except the cheque transactions, which should be memorable. Transactions are used to propagate 'clearances' to modify other entities. The cheques file must be memoryless, otherwise once the first transaction has been written, the clearance for writing subsequent transactions will be retained, regardless of whether they are cleared. A transaction, class t will have an initial high water mark $\{t\}$. It is not dominated by class **chk**. If a manager writes to a transaction, the security mechanisms will update the transaction water mark to {**mgr**, t}, and it still not dominated by **chk**. Now, the accountant may examine the cheque request given by t. The accountant is not however, allowed modify the transaction—an aggregation exception. If the cheque is valid, then the accountant will write it to the cheques file, (which is equivalent to a conglomerate write of accountant plus transaction (transaction plus manager) to cheques file).

Note that some assurance is required to ensure that transactions are used only once: we do not want the accountant printing multiple copies of the same cheque. △

In this last example, due to potential size of CT, it would not be practical to directly the policy as a powerset lattice. Like the telephone book example, some other mechanism should be derived which implements the high water mark and lattice more efficiently. For example, every cheque transaction class is identical in terms of what they can sink and source. Therefore, we could build a lattice with classes **mgr,acc,chk** and a distinguished class t which represents any transaction class. Whenever a t occurs in a water mark it is instantiated by an actual transaction identifier. Further research is required to develop refinements of the high water mark mechanisms to make use of similarities in *pools* of classes.

12 Discussion

This report proposes a structure that can be used to describe information flow policies that may have transitivity, aggregation and separation exceptions; a high water mark mechanism can be built for enforcing these policies.

In section 4 we examined how conglomerate relations can describe flow policies that may have an inconsistent join operator. Having inconsistent joins in a flow policy provides a way of describing aggregation and separation exceptions. In section 7 we capture many popular aggregation policies[6,19,20], using conglomerate relations. Research on separation of duty policies[7] seems to have concentrated primarily on separation rules for integrity. In this report we describe separation *flow* policies, and give some examples of how they might be used. Conglomerate flow policies are stateless specifications of security in a system. They describe the different classes of information that can exist in a system, and how they may propagate. The granularity of a flow policy extends only to the classes, and not the entities bound to those classes. This must be remembered when specifying flow policies, in particular, separation policies.

Section 11 examined how conglomerate relations could be used to describe integrity policies. A useful policy for cheque writing was described, which contained both aggregation (a cheque request may be written by either an accountant or a manager, not both), and separation (a cheque may written only by a conglomerate of accountant and manager).

Relations and operators for comparing, composing and abstracting conglomerate flow policies are described in section 4. These operators allow us to build a complex policy from simple components. Section 7 gave examples of how these relations and operators can be used. Section 10 showed how the policy complement operator relates separation and aggregation policies.

In this report we only considered flow policies that had inconsistent upper bounds (joins). A more general class of flow policies is one that includes policies that have inconsistent lower bounds. Such policies are of type $\mathcal{P}classes \leftrightarrow \mathcal{P}classes$, where a maplet $X \mapsto Y$ means that information may flow from class conglomerate X to class conglomerate Y. Such a structure could capture policies such as: mission-X may be learnt by agent A or agent B, but not both. Interesting integrity policies can be described: given classes (purchase order) and (cheque), a clerk may write one or the other but not both. In its current form, our high water mark model cannot enforce such policies. Further research into models for enforcing such policies would be worthwhile.

Sections 6 and 8 develop high water marks models that can enforce conglomerate flow policies.

The first model, based on the group confinement model[11,12], enforces aggregation policies (flow policies that do not express separation exceptions). This model is a generalisation of the traditional high water mark model. Each entity has a high water mark, and a set of limits to which it allowed rise. As the high water mark rises, some of the limits will 'dissappear'. This is how aggregation exceptions are enforced. This model enforces conglomerate policies by way of a flow preserving mapping, which takes an arbitrary aggregation policy, and maps it to a powerset lattice plus high water marks and limits, to be enforced by the high water mark model.

In the group confinement model, we identified two kinds of entities: memorable and memoryless. A memorable entity will propagate any information sunk to it. A memoryless entity does not propagate information. Thus, only the high water marks of memorable entities need to reflect the information they have sunk to date (it may be propagated). Each time a memoryless entity sources information, its class is the (static) class of the entity. Since the high water mark model is abstract, there are no restrictions on what entities should be. They represent all (interesting) sources and sinks of information. Examples are users, programs, windows, files, or even persistent knowledge environments[19]. If an entity is trusted to appropriately handle information at some class, then that entity can be considered memoryless. For example, an admiral (example 15) is trusted (by the very nature of his binding) to handle all admiral information appropriately; therefore he may be treated as memoryless—he is allowed read some secrets, 'forget' them, and then talk with a classified user. However, if he creates a file and puts a secret in it, the file is memorable and its high water mark must reflect the class of it contents. Trusted software can be thought of as memoryless, a certified windowing system should be allowed to simultaneously display secret and classified windows for an admiral.

Sometimes we must view a user as memorable. For example, in the chinese wall policy, consultants must be considered as memorable—they are liable to 'remember' information about conflicting organisations. If a user is trusted at some classes, but not others, it may be necessary to model the user as two entities, representing the memorable part and the memoryless part.

We discovered that the group confinement model generalises many existing flow policy models. Figure 11 summarises our observations. If a flow policy is quasi-ordered, then the flow preserving mapping from conglomer-

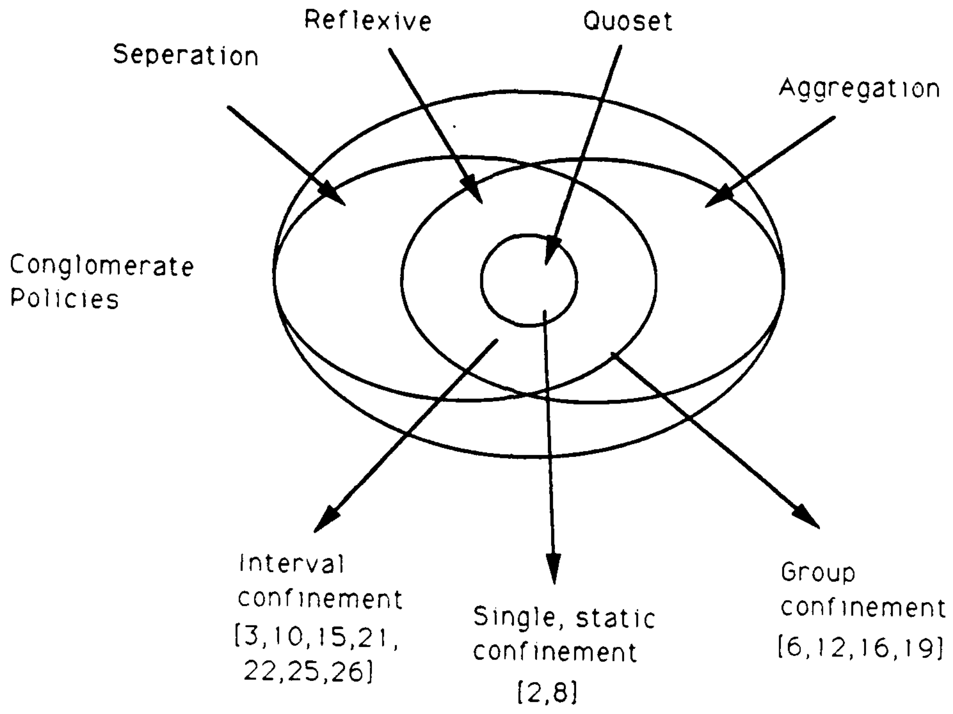
Reflexive
Quoset
Seperation
Aggregation
Conglomerate Policies
Interval confinement [3,10,15,21, 22,25,26]
Single, static confinement [2,8]
Group confinement [6,12,16,19]

Figure 11: Taxonomy of Policies and High Water Mark Models

ate policy to powerset lattice simplifies to the Birkoff mapping from quasi-ordered set to powerset lattice. In this situation, the conglomerate flow model corresponds to the traditional lattice flow model with static binding.

If the flow policy is reflexive, but has no aggregation exceptions, then each class from the flow policy maps to an interval pair from the lattice to be enforced by the high water mark model. Each entity is bound to an interval pair; the lower bound corresponds to the initial high water mark, and the upper bound gives its limit. This type of binding for memorable entities corresponds to the binding used in the Compartment Mode Workstation[26]. Thus we can think of the class of policy enforced by this system as reflexive. Memoryless entities bound to these intervals correspond to partially trusted subjects[3,22], and thus such systems can also be thought of as enforcing reflexive flow policies. Recall the admiral flow policy example (15).

The second high water mark model proposed in section 8 can enforce any conglomerate flow policy, separation and/or aggregation. The model is more general than the group confinement model, in the way it models the flows in a system. However, its notion of, and mechanism for maintaining, high water marks is virtually identical to the group confinement model. The only difference is that a high water mark of a memorable entity in the group confinement model represents the class of information that has been sunk to that entity upto and including the current state. In the universal high water mark model, it is the class of information sunk upto, but excluding the current state.

While the universal high water mark model can enforce any conglomerate policy, it would involve an unacceptable overhead to maintain the water marks. We therefore sought a smaller class of policies, continuous separation policies, that could be enforced efficiently. With these policies, a high water mark must make an initial 'leap' before it can rise as normal. This 'leap' corresponds to overcoming a separation exception.

In any system that uses high water marks, covert channels due to denial of access can arise[9]. We do not address this problem in our high water mark model; it is necessary to look outside the model to detect and/or avoid these flows. Any implementation oriented model for conglomerate policies should address this problem. A number of possible strategies are considered in [11].

We like to think of the two high water mark models as providing guidelines on how to enforce conglomerate policies: at present they need more refinement to be used in practice. It would be worthwhile to develop detailed systems oriented models, that enforce conglomerate policies. Only then can questions on denial of access, what constitutes an entity, and what are memorable and memoryless entities, be effectively dealt with.

Having the framework of conglomerate flow policies, and being able to consider various subsets of policies is appealing. It allows us to make comparisons between the policies enforced by existing models. For example, we discovered that systems with partially trusted subjects can be thought of as enforcing reflexive policies (section7). We saw how reflexive policies provide a form of static separation of duty (section 11). Are there more efficient mappings and mechanisms for restricted classes of policy? For example, quantity based aggregation policies? An interesting application of this ability would be to investigate if the reclassification policy described in [1] can be captured as a conglomerate policy.

13 Acknowledgement

I would like to thank all the members of the 'Thursday Security Club', who provided valuable discussions on this work. This research has been funded by the Royal Signals and Radar Establishment. I would like to express my gratitude to them, and in particular Simon Wiseman and Peter Bottomley.

THIS PAGE IS LEFT BLANK INTENTIONALLY

A A Framework of Conglomerate Relations

Lemma 1 Conglomerate abstraction is closed: given $P : \mathcal{R}[X]$ and $C : \mathcal{P}X$, then $P@A \in \mathcal{R}[X]$, i.e.,

$$\forall a : (\bigcup \mathrm{dom}\, P@C) \cup (\mathrm{ran}\, P@C) \bullet \bigcap \mathrm{dom}\, P \rhd \{a\} = \{a\}$$

PROOF Abstraction can be defined as

$$P@C = K_C \mathbin{⨾} P \rhd C$$

where $K_C = \{A : \mathcal{P}X \bullet A \cap C \cap \alpha P \mapsto A\}$.

Consider some $A : \mathcal{P}classes$; $a : classes$ such that $A \mapsto a \in P@C$. By the definiton of range restriction, we have $a \in \alpha P \cap C$. Furthermore, since the domain of K_C is drawn from subsets of $\alpha P \cap C$, then $A \subseteq \alpha P \cap C$ also. Given that $A \mapsto a \in P@C$, then there exists some $A' : \mathcal{P}classes$ such that $A \mapsto A' \in K_C$ and $A' \mapsto a \in P \rhd \{a\}$. But since P is a valid policy, we know that $a \in A'$, and therefore, since $a \in \alpha P \cap C$, then $a \in A$ also. Thus we have

$$\forall A : \mathcal{P}classes;\ a : classes \bullet A \mapsto a \Rightarrow a \in A \qquad (1)$$

For any $a \in \alpha P \cap C$, we have $\{a\} \mapsto a \in P$ and $\{a\} \mapsto \{a\} \in K_C$. This implies that $\{a\} \mapsto a \in P@C$. This with the fact that the range of $P@C$ is a subset of $\alpha P \cap C$, and the range is drawn from subsets of $\alpha P \cap C$, implies that

$$(\bigcup \mathrm{dom}\, P@C) \cup \mathrm{ran}\, P@C = \alpha P \cap C$$

which when combined with equation (1) proves the lemma.
COROLLARY The following laws follow

$$\begin{aligned} \alpha P@C &= \alpha P \cap C \\ P@\alpha P &= P \\ P@\{\} &= \{\} \end{aligned}$$

□

Lemma 2 Given conglomerate relation $P : \mathcal{R}[X]$, and $B, C : \mathcal{P}X$, then

$$(P@B)@C = P@(B \cap C)$$

PROOF Let

$$\begin{aligned} K_C &= \{A : \mathcal{P}X \bullet A \cap \alpha P \cap B \cap C \mapsto A\} \\ K_B &= \{A : \mathcal{P}X \bullet A \cap \alpha P \cap B \mapsto A\} \end{aligned}$$

We have by defintion,

$$(P@B)@C = K_C \mathbin{⨾} (K_B \mathbin{⨾} P \rhd B) \rhd C$$

Distributing,

$$(P@B)@C = K_C \mathbin{⨾} K_B \mathbin{⨾} (P \rhd B) \rhd C$$

But $P \rhd B \rhd C = P \rhd B \cap C$, and since $K_C \subseteq K_B$, then

$$\begin{aligned} (P@B)@C &= K_C \mathbin{⨾} P \mathbin{⨾} B \cap C \\ &= P@(B \cap C) \end{aligned}$$

□

Lemma 3 Conglomerate abstraction is monotonic with respect to subset, i.e., for $P, Q : \mathcal{R}[X]$, and $B : \mathcal{P}X$, then

$$P \subseteq Q \Rightarrow P@B \subseteq Q@B$$

PROOF The monotonicity of range restriction gives,

$$P \subseteq Q \Rightarrow P \rhd B \subseteq Q \rhd B$$

if $K_B = \{A : \mathcal{P}X \bullet A \cap B \mapsto A\}$, then

$$\begin{aligned} P \subseteq Q &\Rightarrow K_B \mathbin{⨾} P \rhd B \subseteq K_B \mathbin{⨾} Q \rhd B \\ &\Rightarrow P@B \subseteq Q@B \end{aligned}$$

COROLLARY Since domain restriction is monotonic in both arguments,

$$B \subseteq C \wedge P \subseteq Q \Rightarrow P \rhd B \subseteq Q \rhd C$$

But if $K_C = \{A : \mathcal{P}X \bullet A \cap C \mapsto A\}$, then $K_B \subseteq K_C$, which implies

$$\begin{aligned} P \subseteq Q \wedge B \subseteq C &\Rightarrow K_B \mathbin{⨾} P \rhd B \subseteq K_C \mathbin{⨾} Q \rhd C \\ &\Rightarrow P@B \subseteq Q@C \end{aligned}$$

□

Lemma 4 Conglomerate restrictiveness is a partial order.
PROOF Given some $P, Q : \mathcal{R}[X]$ then by defintion,

$$P \sqsubseteq Q \Leftrightarrow \alpha P \subseteq \alpha Q \wedge Q @ \alpha P \subseteq P$$

- Reflexivity follows since $Q @ \alpha Q = Q$.
- (Antisymmetry) The Antisymmetry of subset implies that

  $$P \sqsubseteq Q \wedge Q \sqsubseteq P \Rightarrow \alpha P = \alpha Q$$

  and it follows that

  $$P \sqsubseteq Q \wedge Q \sqsubseteq P \Rightarrow P \subseteq Q \wedge Q \subseteq P$$

  which implies $P = Q$.
- (Transitivity) Transitivity of subset gives

  $$P \sqsubseteq Q \wedge Q \sqsubseteq R \Rightarrow \alpha P \subseteq \alpha R$$

  We have, by the monotonicity of abstraction,

  $$\begin{aligned} Q \sqsubseteq R &\Rightarrow R @ \alpha Q \subseteq Q \\ &\Rightarrow (R @ \alpha Q) @ \alpha P \subseteq Q @ \alpha P \end{aligned}$$

  But $\alpha P \subseteq \alpha Q \Rightarrow R @ \alpha P \subseteq Q @ \alpha P$. Furthermore, $P \sqsubseteq Q \Rightarrow Q @ \alpha P \subseteq P$, thus we have

  $$\begin{aligned} P \sqsubseteq Q \wedge Q \sqsubseteq R &\Rightarrow R @ \alpha P \subseteq P \\ &\Rightarrow P \sqsubseteq R \end{aligned}$$

COROLLARY If the operands of $\sqsubseteq$ have the same alphabet then conglomerate restriction is equivalent to superset. □

Lemma 5 Conglomerate abstraction and restriction are monotonic with respect to each other, i.e., for $P, Q : \mathcal{R}[X]$ and $B, C : \mathcal{P}X$, then

$$B \subseteq C \wedge P \sqsubseteq Q \Rightarrow P @ B \subseteq Q @ C$$

PROOF We have by definition,

$$\begin{aligned} B \subseteq C \wedge \alpha P \subseteq \alpha Q &\Rightarrow \alpha P \cap B \subseteq \alpha Q \cap C \\ &\Rightarrow \alpha(P @ B) \subseteq \alpha(Q @ C) \end{aligned}$$

We have

$$\begin{aligned} P \sqsubseteq Q &\Rightarrow Q@\alpha P \subseteq P \\ &\Rightarrow (Q@\alpha P)@B \subseteq P@B \\ &\Rightarrow (Q@\alpha P)@(B \cap C) \subseteq P@B \\ &\Rightarrow (Q@C)@(B \cap \alpha P) \subseteq P@B \\ &\Rightarrow P@B \sqsubseteq Q@C \end{aligned}$$

□

Lemma 6 Conglomerate join is closed over $\mathcal{R}[X]$: given $P, Q : \mathcal{R}[X]$, then $P \sqcup Q \in \mathcal{R}[X]$, i.e.,

$$\forall a : (\bigcup \mathrm{dom}\, P \sqcup Q) \cup \mathrm{ran}\, P \sqcup Q \bullet \bigcap \mathrm{dom}(P \sqcup Q) \rhd \{a\} = \{a\}$$

PROOF Conglomerate join is defined as

$$\begin{aligned} P \sqcup Q = \{A : \mathcal{P}X, a : X | A \cup \{a\} \subseteq \alpha P \cup \alpha Q \,\wedge \\ (a \in \alpha P \Rightarrow A \cap \alpha P \mapsto a \in P) \wedge \\ (a \in \alpha Q \Rightarrow A \cap \alpha Q \mapsto a \in Q) \bullet \\ A \mapsto a\} \end{aligned}$$

First, it follows from the defintion of $\sqcup$ that

$$\bigcup \mathrm{dom}\, P \sqcup Q \cup \mathrm{ran}\, P \sqcup Q \subseteq \alpha P \cup \alpha Q \qquad (2)$$

Consider any $a \in \alpha P \cup \alpha Q$. If $a \in \alpha P$, then we know $\{a\} \mapsto a \in P$, and similarly if $a \in \alpha Q$, then $\{a\} \mapsto a \in \alpha Q$. Therfore we have

$$\forall a : \alpha P \cup \alpha Q \bullet \{a\} \mapsto a \in P \sqcup Q \qquad (3)$$

combining this with equation (2) gives

$$\alpha P \cup \alpha Q = \bigcup \mathrm{dom}(P \sqcup Q) \cup \mathrm{ran}(P \sqcup Q)$$

Now consider any $A \mapsto a \in P \sqcup Q$. If $a \in \alpha P$, we know that $\alpha P \cap A \mapsto a \in P$. But since P is valid, then $a \in A \cap \alpha P$, which implies that $a \in A$. Similarly, if $a \in \alpha Q$, then $\alpha Q \cap A \mapsto a \in Q$, and $a \in A$. Therefore we have

$$\forall A : \mathcal{P}classes;\ a : classes \bullet A \mapsto a \in P \sqcup Q \Rightarrow a \in A$$

combining this with equation (2) implies that $P \sqcup Q$ is also valid.
COROLLARY The alphabet of $P \sqcup Q$ is $\alpha P \cup \alpha Q$. □

Lemma 7 Conglomerate Join gives an upper bound, i.e., for $P, Q : \mathcal{R}[X]$, then

$$P \sqsubseteq P \sqcup Q \wedge Q \sqsubseteq P \sqcup Q$$

PROOF Since $\alpha P \subseteq \alpha(P \sqcup Q)$, then

$$\begin{aligned} P \sqcup Q \subseteq \{A : \mathcal{P}X, a : X | A \cup \{a\} \subseteq \alpha P \cup \alpha Q \wedge \\ a \in \alpha P \Rightarrow A \cap \alpha P \mapsto a \in P\} \end{aligned}$$

If $K_P = \{A : \mathcal{P}X \bullet A \cap \alpha P \mapsto A\}$, then monotonicity of abstraction gives

$$\begin{aligned} P \sqcup Q @ \alpha P \quad & \subseteq \quad K_P \,⨟\, \{A : \mathcal{P}X, a : X | A \cup \{a\} \subseteq \alpha P \cup \alpha Q \wedge \\ & \qquad (a \in \alpha P \Rightarrow A \cap \alpha P \mapsto a \in P) \bullet A \mapsto a\} \rhd \alpha P \\ & \subseteq \quad K_P \,⨟\, \{A : \mathcal{P}X, a : X | A \cup \{a\} \subseteq \alpha P \cup \alpha Q \wedge a \in \alpha P \wedge \\ & \qquad (a \in \alpha P \Rightarrow A \cap \alpha P \mapsto a \in P) \bullet A \mapsto a\} \\ & \subseteq \quad K_P \,⨟\, \{A : \mathcal{P}X, a : X | A \cup \{a\} \subseteq \alpha P \cup \alpha Q \wedge a \in \alpha P \wedge \\ & \qquad (A \cap \alpha P \mapsto a \in P) \bullet A \mapsto a\} \\ & \subseteq \quad \{A : \mathcal{P}X, a : X | A \cup \{a\} \subseteq \alpha P \wedge \\ & \qquad A \mapsto a \in P \bullet A \mapsto a\} \\ & \subseteq \quad P \end{aligned}$$

The symmetry of conglomerate join implies $Q \sqsubseteq P \sqcup Q$. □

Lemma 8 Conglomerate join gives the lowest upper bound of its arguments, i.e., for $P, Q : \mathcal{R}[X]$, then

$$\forall R : \mathcal{R}[X] \bullet P \sqsubseteq R \wedge Q \sqsubseteq R \Rightarrow P \sqcup Q \sqsubseteq R$$

PROOF We have $P \sqsubseteq R \wedge Q \sqsubseteq R \Rightarrow \alpha P \sqcup Q \sqsubseteq \alpha R$. Next,

$$\begin{aligned} P \sqsubseteq R \quad & \Rightarrow \quad P @ (\alpha P \cup \alpha Q) \sqsubseteq R @ (\alpha P \cup \alpha Q) \\ & \Rightarrow \quad (R @ (\alpha P \cup \alpha Q)) @ \alpha P \subseteq P \\ & \Rightarrow \quad \forall A : \mathcal{P}X;\ a : X \bullet \\ & \qquad A \mapsto a \in (R @ (\alpha P \cup \alpha Q)) @ \alpha P \Rightarrow A \mapsto a \in P \\ & \Rightarrow \quad \forall A : \mathcal{P}X;\ a : X \bullet \\ & \qquad A \mapsto a \in R @ (\alpha P \cup \alpha Q) \wedge a \in \alpha P \\ & \qquad\quad \Rightarrow A \cap \alpha P \mapsto a \in P \end{aligned}$$

And similarly for Q. Thus we get,

$$\begin{aligned}
P \sqsubseteq R \wedge Q \sqsubseteq R \ \Rightarrow\ & \forall A : \mathcal{P}X;\ a : X \bullet \\
& \quad (A \mapsto a \in R @ (\alpha P \cup \alpha Q) \wedge a \in \alpha P \\
& \quad\quad \Rightarrow A \cap \alpha P \mapsto a \in P) \wedge \\
& \quad (A \mapsto a \in R @ (\alpha P \cup \alpha Q) \wedge a \in \alpha Q \\
& \quad\quad \Rightarrow A \cap \alpha Q \mapsto a \in Q) \\
\Rightarrow\ & \forall A : \mathcal{P}X;\ a : X \bullet \\
& \quad A \mapsto a \in R @ (\alpha P \cup \alpha Q) \Rightarrow \\
& \quad\quad (a \in \alpha P \Rightarrow A \cap \alpha P \mapsto a \in P) \wedge \\
& \quad\quad (a \in \alpha Q \Rightarrow A \cap \alpha Q \mapsto a \in Q) \\
\Rightarrow\ & \forall A : \mathcal{P}X;\ a : X \bullet \\
& \quad A \mapsto a \in R @ (\alpha P \cup \alpha Q) \Rightarrow \\
& \quad\quad A \mapsto a \in P \sqcup Q \\
\Rightarrow\ & R @ (\alpha P \sqcup Q) \subseteq P \sqcup Q \\
\Rightarrow\ & P \sqcup Q \sqsubseteq R
\end{aligned}$$

COROLLARY If the operands of $\sqcup$ have the same alphabet then conglomerate join is equivalent to set intersection. □

Lemma 9 Conglomerate Intersection is closed: given $P, Q : \mathcal{R}[X]$, then $P \sqcap Q \in \mathcal{R}[X]$.
PROOF We have

$$P \sqcap Q = P @ (\alpha P \cap \alpha Q) \cup Q @ (\alpha P \cap \alpha Q)$$

and since both sides of the $\cup$ are valid conglomerate relations, and have the same alphabet, then $P \sqcap Q$ is also valid.
COROLLARY The alphabet of $P \sqcap Q$ is $\alpha P \cap \alpha Q$ □

Lemma 10 Conglomerate Intersection gives a lower bound, i.e., for $P, Q : \mathcal{R}[X]$

$$P \sqcap Q \sqsubseteq P \wedge P \sqcap Q \sqsubseteq Q$$

PROOF We have $\alpha P \sqcap Q \subseteq \alpha P$. The definition of $\sqcap$ implies that $P @ (\alpha P \cap \alpha Q \subseteq P \sqcap Q$, which implies $P \sqcap Q \sqsubseteq P$. Symmetry of definition implies $P \sqcap Q \sqsubseteq Q$. □

Lemma 11 Conglomerate Intersection gives the greatest lower bound of its arguments, i.e., for $P, Q : \mathcal{R}[X]$

$$\forall R : \mathcal{R}[X] \bullet R \sqsubseteq P \wedge R \sqsubseteq Q \Rightarrow R \sqsubseteq P \sqcap Q$$

PROOF We have $R \sqsubseteq P \wedge R \sqsubseteq Q \Rightarrow \alpha R \subseteq \alpha P \sqcap Q$, and

$$\begin{aligned} R \sqsubseteq P \wedge R \sqsubseteq Q &\Rightarrow P@\alpha R \cup Q@\alpha R \subseteq R \\ &\Rightarrow P@(\alpha P \cap \alpha Q)@\alpha R \cup Q@(\alpha P \cap \alpha Q)@\alpha R \subseteq R \end{aligned}$$

Distribuiting abstraction throught the arguments of $\cup$ gives

$$\begin{aligned} R \sqsubseteq P \wedge R \sqsubseteq Q &\Rightarrow (P@(\alpha P \cap \alpha Q) \cup Q@(\alpha capa Q))@\alpha R \subseteq R \\ &\Rightarrow P \sqcap Q@\alpha R \subseteq R \\ &\Rightarrow R \sqsubseteq P \sqcap Q \end{aligned}$$

COROLLARY If the operands of $\sqcap$ have the same alphabet then conglomerate intersection is equivalent to set union. □

Lemma 12 The conglomerate complement operator is closed over the set of conglomerate relations with the same alphabet: given $P : \mathcal{R}[X]$, then $\alpha P = \alpha \overline{P}$ and $\overline{P} \in \mathcal{R}[X]$.
PROOF Conglomerate complement is defined as

$$\overline{P} = \top \alpha P \cup (\bot \alpha P - P)$$

It follows directly from this that

$$\alpha P = \bigcup \operatorname{dom} \overline{P} \cup \operatorname{ran} \overline{P}$$

and since $\top \alpha P \subseteq \overline{P}$, then

$$\forall a : \alpha P \bullet \{a\} \mapsto a \in \overline{P} \tag{4}$$

If $A \mapsto a \in \overline{P}$, then we know that $A \mapsto a \in \bot \alpha P$, and since $\bot \alpha P$ is a valid policy, then $a \in A$. Combining this with (4) proves that $\overline{P}$ is a valid relation.
COROLLARY the alphabet of $\overline{P}$ is αP. □

Lemma 13 The conglomerate operator gives the complement of a relation in the subset of $\mathcal{R}[X]$ of relations with the same alphabet, i.e., for $P : \mathcal{R}[X]$

$$\begin{aligned} P \sqcup \overline{P} &= \top \alpha P \\ P \sqcap \overline{P} &= \bot \alpha P \end{aligned}$$

PROOF follows from the defintion of conglomerate complement. □

Theorem 3 The set of all conglomerate relations form a lattice with ordering relation $\sqsubseteq$; and lowest upper and greatest lower bound operators $\sqcup$ and $\sqcap$, respectively.

COROLLARY The set of conglomerate relations with the same alphabet forms a distributive and comlpementable sublattice of $\mathcal{R}[X]$. $\Box$

Lemma 14 Conglomerate Join is closed over the set of the set of conglomerate relations that do not express separation exceptions, i.e.,

$$\forall P, Q : \mathcal{R}_{-S}[X] \bullet P \sqcup Q \in \mathcal{R}_{-S}[X]$$

PROOF We have by definition for $P, Q : \mathcal{R}[X]$; $A, B : \mathcal{P}X$; $a : X$, that

$$\begin{aligned} A \cup B \mapsto a \in P \sqcup Q \quad \Leftrightarrow \quad & A \cup B \cup \{a\} \subseteq \alpha P \cup \alpha Q \wedge \\ & a \in \alpha P \Rightarrow (A \cup B) \cap \alpha P \mapsto c \in P \wedge \\ & a \in \alpha Q \Rightarrow (A \cup B) \cap \alpha Q \mapsto a \in Q \end{aligned}$$

But since $P \in \mathcal{R}_{-S}[X]$ then

$$\begin{aligned} & (A \cup B) \cap \alpha P \mapsto a \in P \\ \Rightarrow \quad & (A \cap \alpha P) \cup (B \cap \alpha P) \mapsto a \in P \\ \Rightarrow \quad & (A \cap \alpha P) \mapsto a \in P \wedge (B \cap \alpha P) \mapsto a \in P \end{aligned}$$

and similarly, since $Q \in \mathcal{R}_{-S}[X]$,

$$\begin{aligned} & (A \cup B) \cap \alpha Q \mapsto a \in Q \\ \Rightarrow \quad & (A \cap \alpha Q) \mapsto a \in P \wedge (B \cap \alpha Q) \mapsto a \in Q \end{aligned}$$

Thus if $A \cup B \mapsto a \in P \sqcup Q$ then,

$$\begin{aligned} & a \in \alpha P \Rightarrow (A \cap \alpha P) \mapsto a \in P \wedge \\ & a \in \alpha Q \Rightarrow (A \cap \alpha Q) \mapsto a \in Q \end{aligned}$$

which implies $A \mapsto a \in P \sqcup Q$, and similarly we can show that $B \mapsto a \in P \sqcup Q$. Therefore $P \sqcup Q \in \mathcal{R}_{-S}[X]$. $\Box$

Lemma 15 Conglomerate join is closed over the set of the set of conglomerate relations that do not express aggregation exceptions, i.e.,

$$\forall P, Q : \mathcal{R}_{-A}[X] \bullet P \sqcup Q \in \mathcal{R}_{-A}[X]$$

PROOF We have by definition for $P, Q : \mathcal{R}[X]$; $A, B : \mathcal{P}X$; $a : X$, that

$$\begin{aligned}
&A \mapsto a \wedge B \mapsto a \Rightarrow \\
&\qquad A \cup B \cup \{a\} \subseteq \alpha P \cup \alpha Q \wedge \\
&\qquad (a \in \alpha P \Rightarrow A \cap \alpha P \mapsto a \in P) \wedge \\
&\qquad (a \in \alpha P \Rightarrow B \cap \alpha P \mapsto a \in P) \wedge \\
&\qquad (a \in \alpha Q \Rightarrow A \cap \alpha P \mapsto a \in Q) \wedge \\
&\qquad (a \in \alpha Q \Rightarrow B \cap \alpha P \mapsto a \in Q)
\end{aligned}$$

and since $P \in \mathcal{R}_{-S}[X]$, this implies,

$$\begin{aligned}
& a \in \alpha P \Rightarrow \{(A \cap \alpha P) \mapsto a, (B \cap \alpha P) \mapsto a\} \subseteq P \\
\Rightarrow\ & a \in \alpha P \Rightarrow (A \cup B) \cap \alpha P \mapsto a \in P
\end{aligned}$$

Similarly, since $Q \in \mathcal{R}_{-A}[X]$, we get

$$a \in \alpha Q \Rightarrow (A \cup B) \cap \alpha Q \mapsto a \in Q$$

combining these results implies $A \cup B \mapsto a \in P \sqcup Q$, which implies $P \sqcup Q \in \mathcal{R}_{-A}[X]$. □

Lemma 16 Conglomerate Join is closed over the set of the set of conglomerate relations that do not express separation nor aggregation exceptions, i.e.,

$$\forall P, Q : \mathcal{R}_{\diamond}[X] \bullet P \sqcup Q \in \mathcal{R}_{\diamond}[X]$$

PROOF Follows from the results of the last two lemmas, since if $P, Q \in \mathcal{R}_{\diamond}[X]$, then P and Q are members of both $\mathcal{R}_{-S}[X]$ and $\mathcal{R}_{-A}[X]$, and so is their join. But we have that

$$\mathcal{R}_{\diamond}[X] = \mathcal{R}_{-A}[X] \cap \mathcal{R}_{-S}[X]$$

COROLLARY By inspection we see that our definition of join over policies from $\mathcal{R}_{\diamond}[classes]$ is identical to the (reflexive) policy join operator defined in [10]. □

THIS PAGE IS LEFT BLANK INTENTIONALLY

B Implementing Aggregation Policies

B.1 FMG: Group Confinement Model

Lemma 17 If P is a flow policy with no separation exceptions then the mapping Φ is flow preserving in the sense that given $P \in \mathcal{R}_{-S}[classes]$, then

$$\begin{array}{l} \forall A : \mathbb{P} classes;\ a : classes \bullet A \cup \{a\} \subseteq \alpha P \Rightarrow \\ \quad A \mapsto a \in P \Leftrightarrow \\ \qquad \exists B : \mathbb{P} classes \bullet B \in \Phi_{\top_s} P\ a \wedge \bigcup \Phi_\perp P\ (\!| A |\!) \subseteq B \end{array}$$

Recall that the definition of Φ gives: for policy P, and $a : classes$, then

$$\begin{array}{rcl} \Phi_\perp P\ a & = & \{b : classes | (P) b \leq a\} \\ \Phi_{\top_s} P\ a & = & \mathrm{maxs}(\mathrm{dom}\, P \rhd \{a\}) \end{array}$$

PROOF (if part) Suppose $B \mapsto c \in P$ holds. This can be written as $(B - \{b\}) \cup \{b\} \mapsto c \in P$ for any $b \in B$. By definition we have

$$\Phi_\perp P\ b = \{a : classes | (P) a \leq b\}$$

which implies, by definition of bound ($\leq$), that for any $a \in \Phi_\perp\ P\ b$, then

$$(B - \{b\}) \cup \{b\} \mapsto c \in P \Rightarrow (B - \{b\}) \cup \{a, b\} \mapsto c \in P$$

and we can progressively add components of $\Phi_\perp P\ b$ to give

$$(B - \{b\}) \cup \{b\} \mapsto c \in P \Rightarrow (B - \{b\}) \cup \{b\} \cup \Phi_\perp P\ b \mapsto c \in P$$

and we can repeat the process for every other conponent of $P - \{b\}$ to finally give

$$(B - \{b\}) \cup \{b\} \mapsto c \in P \Rightarrow B \cup \bigcup (\!| \Phi_\perp P\ B |\!) \mapsto c \in P$$

But since bound is reflexive, we get

$$\begin{array}{rcl} B - \{b\}) \cup \{b\} \mapsto c \in P & \Rightarrow & \bigcup \Phi_\perp P (\!| B |\!) \mapsto c \in P \\ & \Rightarrow & \bigcup \Phi_\perp P (\!| B |\!) \in \mathrm{dom}\, P \rhd \{c\} \\ & \Rightarrow & \exists C : \mathbb{P} classes \bullet \\ & & C \in \mathrm{maxs}\ \mathrm{dom}(P \rhd \{c\}) \wedge \bigcup \Phi_\perp P (\!| B |\!) \subseteq C \\ & \Rightarrow & \exists C : \mathbb{P} classes \bullet \\ & & C \in \Phi_{\top_s} \wedge \bigcup \Phi_\perp P (\!| B |\!) \subseteq C \end{array}$$

PROOF (only if) Suppose that

$$\exists\, C : \mathcal{P} classes \bullet \\ C \in \Phi_{\top_s} \wedge \bigcup \Phi_{\perp} P(\!| B |\!) \subseteq C$$

Then since the C for which the above holds is a member of $\Phi_{\top_s}$ we have,

$$\exists\, C : \mathcal{P} classes \bullet C \mapsto c \wedge \bigcup \Phi_{\perp}(\!| B |\!) \subseteq C$$

But we already know that $B \subseteq \Phi_{\perp} P(\!| B |\!)$ (reflexivity of bound), and by transitivity of subseteq gives $B \subseteq C$. Since P is an aggregation policy, then if $C \mapsto c \in P$ then all subsets of C may flow to c. Therfore $B \mapsto c \in P$, and the lemma is proven. □

C Implementing Separation Policies

C.1 FMU: A Universal High Water Mark Model

Lemma 18 The mapping Φ is flow preserving, in the sense that for any policy $P : policies$, then

$$\begin{array}{l} \forall A : \mathcal{P}classes;\ b : classes \bullet \\ \quad A \mapsto b \in P \Leftrightarrow \bigcup \Phi_{\perp} P (\!| A |\!) \in \Phi_{sink} P\ b \end{array}$$

PROOF Given $A : \mathcal{P}classes;\ b : classes$ the defintion of Φ gives

$$\begin{array}{rcl} \bigcup \Phi_{\perp} P (\!| A |\!) A & = & A \\ \Phi_{sink} P\ b & = & \operatorname{dom} P \rhd \{b\} \end{array}$$

and if $A \mapsto b \in P$, then $\bigcup \Phi_{\perp} P (\!| A |\!) \in \Phi_{sink} P\ b$ follows and vice-versa. □

C.2 Formal Basis for iFMU

Lemma 19 Given a history t of states from *msg-flows*, and a state M : *msg-flows*, then for every memorable entity $e \in \alpha M$

$$\begin{array}{l} \text{all-fwd}^*(\text{flows } t\,\widehat{}\,\langle M \rangle, \{e\}) = \\ \quad \textit{all-fwd}^*(M, \{e\}) \cap \textit{memless} \cup \\ \quad \textit{all-fwd}^*(\text{flows } t, \textit{all-fwd}^*(M, \{e\}) \cap \textit{memble}) \end{array}$$

PROOF We have by the defintion of flows t and all-fwd*, that

$$\begin{array}{ll} \text{all-fwd}^*(\text{flows } t\,\widehat{}\,\langle M \rangle, \{e\}) = & \\ \quad \bigcup \operatorname{dom}(\text{flows } t \rhd \{e\}) \cup & (5) \\ \quad \bigcup \{E : \mathcal{P}ents \mid E \mapsto e \in M \bullet & (6) \\ \qquad (E \cap \textit{memless}) \cup \bigcup(\operatorname{dom} \text{flows } t \rhd (E \cap \textit{memble}))\} & \end{array}$$

Since M does not posses any aggregation exceptions, equation (6) can be written as

$$\begin{array}{lll} & \bigcup \operatorname{dom}(M \rhd \{e\}) \cap \textit{memless} \cup & \\ & \bigcup \{E : \mathcal{P}ents \mid E \in \operatorname{dom} M \rhd \{e\} \bullet \bigcup(\operatorname{dom} \text{flows } t \rhd (E \cap \textit{memble}))\} & \\ = & \bigcup \operatorname{dom}(M \rhd \{e\}) \cap \textit{memless} \cup & \\ & \bigcup \operatorname{dom}(\text{flows } t \rhd (\bigcup \operatorname{dom} M \rhd \{e\}) \cap \textit{memble}) & (7) \end{array}$$

But e is memorable, and thus

$$e \in \bigcup \mathrm{dom}(M \rhd \{e\}) \cap \mathit{memble}$$

which implies that the result of equation (5) is a subset of equation (7), and therefore we have

$$\begin{aligned}&\bigcup \mathrm{dom}(\mathrm{flows}\ t^\frown\langle M\rangle \rhd \{e\}) =\\ &\quad \bigcup \mathrm{dom}(M \rhd \{e\}) \cap \mathit{memless} \cup\\ &\quad \bigcup \mathrm{dom}(\mathrm{flows}\ t \rhd (\bigcup \mathrm{dom}(M \rhd \{e\}) \cap \mathit{memble}))\end{aligned}$$

and the lemma follows. □

Lemma 20 For any secure history t (a sequence of secure states formed by a series of secure transitions), and a secure state s_n such that $t^\frown\langle s_n\rangle$ also forms a secure history, then we can determine the high water mark of each memorable entity at state s_n in terms of the inital high water marks of all the sources of ϵ, i.e.,

$$s_n.\delta_{hwm}\,e = \bigoplus \delta_{hwm}(\!|\text{all-fwd}^*(\text{flows } t, \{\epsilon\})|\!)$$

where $\delta_{hwm} = t\ 1.\delta_{hwm}$, the initial high water marks, and the defintion of flows is extended to sequences of secure states.

PROOF We will use induction over the length of t.

- (base case) If $\#t = 1$, then

  $$\text{all-fwd}^*(\mathit{flows}\ t, \{\epsilon\}) = \text{all-fwd}^*(t\ 1.M, \{\epsilon\})$$

  and since the transition from state $(t\ 1)$ to state s_2 is secure, we have by *Secure-Transition*,

  $$s_2.\delta_{hwm}\epsilon = \bigoplus \delta_{hwm}(\!|\text{all-fwd}^*(\text{flows } t, \{\epsilon\})|\!)$$

  and the lemma holds for the base case.

- (inductive hypothesis) Suppose that the lemma holds for histories of length n. Now consider a history $t^\frown\langle s_n, s_{n+1}\rangle$ of length $n+1$. By the definition of a secure transition from s_n to s_{n+1} we have for a memorable ϵ,

  $$\begin{aligned} s_{n+1}.\delta\epsilon &= \bigoplus s_n.\delta_{hwm}(\!|\text{all-fwd}^*(s_n.M, \{\epsilon\})|\!) && (8)\\ &= \bigoplus s_n.\delta_{hwm}(\!|\text{all-fwd}^*(s_n.M, \{\epsilon\}) \cap \mathit{memble}|\!) \dotplus && (9)\\ &\quad \bigoplus s_n.\delta_{hwm}(\!|\text{all-fwd}^*(s_n.M, \{\epsilon\}) \cap \mathit{memless}|\!) && (10)\end{aligned}$$

Since the high water marks of memoryless entities remain static, equation (9) can be written as

$$\bigoplus \delta_{hwm}(\!|\text{all-fwd}^*(s_n.M, \{e\}) \cap \mathit{memble}|\!) \tag{11}$$

However, the high water marks of memorable entities change, and applying the inductive hypothesis to equation (10) implies that for any memorable $f \in$ all-fwd*$(s_n.M, \{e\})$, then

$$s_n.\delta_{hwm}f = \bigoplus \delta_{hwm}(\!|\text{all-fwd}^*(\text{flows } t, \{f\})|\!)$$

and therefore, equation (10) can be written as

$$\begin{aligned} &\bigoplus\{f : \mathit{ents} | f \in \text{all-fwd}^*(s_n.M, \{e\}) \cap \mathit{memble} \bullet \\ &\qquad \bigoplus \delta_{hwm}(\!|\text{all-fwd}^*(\text{flows } t, \{f\})|\!)\} \\ = \; &\bigoplus \delta_{hwm}(\!|\text{all-fwd}^*(\text{flows } t, \text{all-fwd}^*(s_n.M, \{e\}) \cap \mathit{memble})|\!) \end{aligned}$$

and combining this equation with the memoryless case gives

$$\begin{aligned} &s_{n+1}.\delta_{hwm}e = \\ &\quad \bigoplus \delta_{hwm}(\!|\text{all-fwd}^*(s_n.M, \{e\}) \cap \mathit{memless} \cup \\ &\qquad\qquad \text{all-fwd}^*(\text{flows } t, \text{all-fwd}^*(s_n.M, \{e\}) \cap \mathit{memble})|\!) \end{aligned}$$

which, by replacing all-fwd* by it defintion and applying the result of lemma 19 gives,

$$s_{n+1}.\delta_{hwm}e = \bigoplus \delta_{hwm}(\!|\text{all-fwd}^*(\text{flows } t^\frown\langle s_n\rangle, \{e\})|\!)$$

i.e., the lemma holds for histories of length $n+1$.

Thus by induction the lemma is proven.
COROLLARY Given a memorable conglomerate $E \subseteq \mathit{memble}$, then

$$\bigoplus s_n.\delta_{hwm}(\!|E|\!) = \bigoplus \delta_{hwm}(\!|\text{all-fwd}^*(\text{flows } t, E)|\!)$$

□

Lemma 21 Given any non empty history t of secure states, arrived at via a series of secure transitions, then

$$\begin{aligned} &\forall E : \mathbb{P}\mathit{ents}; \; f\mathit{ents} \bullet \\ &\quad E \mapsto f \in \text{flows } t \Rightarrow \\ &\qquad \bigoplus \delta_{hwm}(\!|E|\!) \in \xi_{sink} f \end{aligned}$$

PROOF We will use induction on the length of the history t.

- (base case) If $\#t = 1$, then flows t = $(t\ 1).A$, and the lemma follows from the defintion of a secure state.
- (inductive hypothesis) Suppose the lemma holds for histories t of length n. Consider a secure history $t^\frown\langle s_{n+1}\rangle$ of length $n+1$. By the defintion of function flows_, we know that if $E \mapsto f \in$ flows $t^\frown\langle s_{n+1}\rangle$, then either

  $$E \mapsto f \in \text{flows } t$$

  in which case, by the inductive hypothesis the lemma holds; or if $E \mapsto f$ is not in flows t, then

  $$E \mapsto f \in \{G : \mathcal{P}ents;\ f : ents \mid G \mapsto f \in s_{n+1}.M \bullet \\ (G \cap memless) \cup \text{all-fwd*}(\text{flows } t, G \cap memble) \mapsto f\}$$

  Pick a $G : \mathcal{P}ents$ such that

  $$G \mapsto f \in s_{n+1}.M$$

  and G propagates E to f, i.e.,

  $$E = (G \cap memless) \cup \text{all-fwd*}(\text{flows } t, G \cap memble) \qquad (12)$$

  Since s_{n+1} is secure, we have

  $$\bigoplus s_{n+1}.\delta_{hum}(\!|G|\!) \in \xi_{sink} f \qquad (13)$$

  (since ξ_{sink} is static, we have $\xi_{sink} = s_{n+1}.\xi_{sink}$). But δ_{hum} for memorable entities in G can be calculated according to lemma 20, along with the fact that δ_{hum} is unchanged for memoryless entities in G. Thus we have,

  $$\bigoplus s_{n+1}.\delta_{hum}(\!|G|\!) = \bigoplus \delta_{hum}(\!|(G \cap memless) \cup \\ \text{all-fwd*}(\text{flows } t, G \cap memble)|\!)$$

  applying equation (12) to this result gives

  $$\bigoplus s_{n+1}.\delta_{hum}(\!|G|\!) = \bigoplus \delta_{hum}(\!|E|\!)$$

  But this result, along with equation (12) (state s_{n+1} is secure) implies that

  $$\bigoplus \delta_{hum}(\!|E|\!) \in \xi_{sink} f$$

  i.e., the lemma holds for any secure history of length $n+1$.

By induction the lemma holds for all secure histories. □

C.3 A Framework of High Water Mark Models

C.3.1 Continuous Separation Policies

Lemma 22 Given a continuous separation policy $P : \mathcal{R}_{-A\bullet}[classes]$, then

$$\begin{aligned}
&\forall A : \mathcal{P} classes;\ b : classes \bullet A \neq \{\} \wedge A \neq \{b\} \Rightarrow \\
&\quad A \mapsto b \in P \Leftrightarrow \\
&\quad \exists B : \mathcal{P} classes \bullet B \in \text{mins}(\text{dom}(P \rhd \{b\}) - \{\{\}, \{b\}\})) \wedge \\
&\qquad B \subseteq A \wedge \\
&\qquad A \subseteq \bigcup \text{dom}\, P \rhd \{b\}
\end{aligned}$$

PROOF Suppose that $A \mapsto b$ and $A \neq \{\}$ and $A \neq \{b\}$. It follows that

$$A \subseteq \bigcup \text{dom}(P \rhd \{b\}) \tag{14}$$

and since $A \mapsto b$, we also have

$$A \in (\text{dom}(P \rhd \{b\})) - \{\{\}, \{b\}\}$$

thus there must be a minimum less than A, i.e.,

$$\exists B : \mathcal{P} classes \bullet B \in \text{mins}(\text{dom}(P \rhd \{b\}) - \{\{\}, \{b\}\})$$

combining this with equation (14) gives the if part of the lemmma.

If we have

$$\begin{aligned}
&A \subseteq \bigcup \text{dom}\, P \rhd \{b\} \wedge \\
&\exists B : \mathcal{P} classes \bullet B \in \text{mins}(\text{dom}(P \rhd \{b\}) - \{\{\}, \{b\}\})
\end{aligned}$$

then since P is non-aggregating, we have for $C : \mathcal{P} classes$

$$\begin{aligned}
&C \in \text{dom}\, P \rhd \{b\} \in P \wedge C' \in \text{dom}\, P \rhd \{b\} \in P \Rightarrow \\
&\quad C \cup C \mapsto b \in P
\end{aligned}$$

Which implies that for the above,

$$\bigcup \text{dom}(P \rhd \{b\}) \mapsto b \in P \tag{15}$$

Also given our inital assumption, we have some minimum $B : \mathcal{P} classes$ such that $B \mapsto b \in P$. But $B \subseteq \bigcup \text{dom}(P \rhd \{b\})$, and P is continuous, which implies every conglomerate between B and the top may flow to b. But we have $B \subseteq A$ and $A \subseteq \bigcup \text{dom}(P \rhd \{b\})$ which implies that $A \mapsto b \in P$, and the lemma is proven. □

THIS PAGE IS LEFT BLANK INTENTIONALLY

D Relating Aggregation and Separation

Lemma 23 The complement of any policy not exhibiting separation properties does not exhibit aggregation policies, i.e.,

$$\forall P : \mathcal{R}_{-S}[classes] \bullet \overline{P} \in \mathcal{R}_{-A}$$

PROOF By definition we have

$$\overline{P} = \top\alpha P \cap (\bot\alpha P - P)$$

Thus, for $A, B : \mathbb{P} classes;\ a : classes$ then $\{A \mapsto a, B \mapsto a\} \subseteq P$ implies that:

- If $\{A \mapsto a, B \mapsto a\} \subseteq \top\alpha P$, then it follows that $A \cup B \mapsto a \in \top\alpha P$, and thus $A \cup B \mapsto a \in \overline{P}$.
- If $A \mapsto a \in (\bot\alpha P - P)$, then we have $A \mapsto a \notin P$. But policy $P \in \mathcal{R}_{-s}[classes]$, and thus does not exhibit any separation properties: if $A \mapsto a \notin P$, then $A \cup C \mapsto a \notin P$ must hold for any C—if a C existed such that $A \cup C \mapsto a \in P$ then this would violate the requirement

  $$A \cup C \mapsto a \in P \Rightarrow A \mapsto a \in P \wedge C \mapsto a \in P$$

  Therefore we must have $A \cup B \mapsto a \notin P$ and thus $A \cup B \mapsto a \in (\bot\alpha P - P)$, implying that $A \cup B \mapsto a \in \overline{P}$. By symmetry, if $B \mapsto a \in (\bot\alpha P - P)$, then $A \cup B \mapsto a \in \overline{P}$ also.

Thus we have for any $P \in \mathcal{R}_{-S}[classes]$, that

$$\begin{aligned}&\forall A, B : \mathbb{P} classes;\ a : classes \bullet\\ &\quad \{A \mapsto a, B \mapsto a\} \subseteq \overline{P} \Rightarrow\\ &\quad A \cup B \mapsto a \in \overline{P}\end{aligned}$$

i.e., $\overline{P} \in \mathcal{R}_{-A}[classes]$. □

Lemma 24 Given a continuous separation policy $P : \mathcal{R}_{-A^*}[classes]$, then P_R is a valid policy, where

$$P_R = \{A : \mathbb{P} classes;\ a : classes | A \subseteq \bigcup \mathrm{dom}\, P \rhd \{a\} \bullet A \cup \{a\} \mapsto a$$

PROOF We have

$$\alpha P = \bigcup \mathrm{dom}\, P_R \cup \mathrm{ran}\, P_R$$

and for any $a \in \alpha P$, then $\{a\} \mapsto a \in P$, which implies

$$\{a\} \mapsto a \in P_R \tag{16}$$

For any $A : \mathcal{P}classes$; $a : classes$, we have $A \mapsto a \in P_R$ implies, by the definition of P_R, that $a \in A$. Combining this with (16) implies that

$$\forall\, a : \alpha P \bullet \bigcap \mathrm{dom}\, P \rhd \{a\} = \{a\}$$

i.e., P_R forms a valid policy. Note also that $\alpha P = \alpha P_R$.
COROLLARY Since $P_S = P \sqcap \overline{P_R}$, and P_R is a valid policy, then P_S must also be a valid policy. □

Lemma 25 If $P \in \mathcal{R}_{-A\bullet}[classes]$, then $P_R \in \mathcal{R}_{\diamond}[classes]$, where P_R is defined as in the previous lemma.
PROOF The definiton of P_R states that for any c, then all subsets of $\bigcup \mathrm{dom}\, P \rhd \{c\}$ may flow to c. Thus if $A \cup B$ may flow to c, then so too may their subsets, i.e., $A \mapsto c \in P_R$ and $B \mapsto c \in P_R$.

Similarly, if A may flow to c and B may flow to c then both A and B are subsets of $\bigcup \mathrm{dom}\, P \rhd \{c\}$, and thus so too is $A \cup B$ a subset. Therefore $A \cup B \mapsto c$. □

Lemma 26 Given a continuous separation policy $P : \mathcal{R}_{-A\bullet}[classes]$, then the complement of its separation component $P_S = P \sqcap \overline{P_R}$, where P_R is defined as above, forms an aggregation policy, i.e., $\overline{P_S} \in \mathcal{R}_{-S}[classes]$
PROOF Since a sublattice of policies with the same alphabet forms a boolean algebra, and $\alpha P = \alpha P_R$, we have

$$\begin{aligned}
\overline{P_S} &= \overline{P \sqcap \overline{P_R}} \\
&= \overline{P} \sqcup P_R \\
&= (\top\alpha P \cup (\bot\alpha P - P)) \cap P_R \\
&= (P_R \cap \top\alpha P) \cup (P_R \cap (\bot\alpha P - P)) \\
&= \top\alpha P \cup (P_R \cap \bot\alpha P - P) \\
&= \top\alpha P \cup (P_R - P)
\end{aligned}$$

For $A, B : \mathcal{P}classes$; $a : classes$, if $A \cup B \mapsto a \in \overline{P_S}$, then

$$A \cup B \mapsto a \;\in\; \top\alpha P \;\vee \tag{17}$$
$$A \cup B \mapsto a \;\in\; (P_R - P) \tag{18}$$

If equation (17) holds, it follows that $\{A \mapsto a, B \mapsto a\} \subseteq \top\alpha P$, which implies

$$A \mapsto a \in \overline{P_S} \wedge B \mapsto a \in \overline{P_S} \tag{19}$$

For equation (18) we have

$$A \mapsto a \in P_R \wedge A \cup B \mapsto a \notin P \tag{20}$$

Since $P_R \in \mathcal{R}_\diamond[classes]$, then $A \cup B \mapsto a \in P_R$ implies that $A \mapsto a \in P_R$ and $B \mapsto a \in P_R$. Since P_R is simply policy P with its holes filled up, then if $A \cup B \mapsto a \in P_R$, then there must be some conglomerate C containing both A and B, with $C \mapsto a \in P$, i.e.,

$$\exists\, C : \mathcal{P} classes \bullet A \cup B \subseteq C \wedge C \mapsto a \in P \tag{21}$$

If $A \cup B \mapsto a \notin P$, then since P does not have aggregation properties, we have

$$A \mapsto a \notin P \vee B \mapsto a \notin P \tag{22}$$

Suppose that $A \mapsto a \in P$ holds. Then by equation (21), there exists some C such that $C \mapsto a$ and $A \cup B \subseteq C$. But P is a continuous separation policy which implies that all conglomerates between A and C (including $A \cup B$) may flow to a, i.e., $A \cup B \mapsto a$, which is a contradiction. Therefore, if $A \cup B \mapsto a \notin P$ holds, then

$$A \mapsto a \notin P \wedge B \mapsto a \notin P$$

combining this with equation (20) gives

$$A \mapsto a \in (P_R - P) \wedge B \mapsto a \in (P_R - P)$$

and thus,

$$A \mapsto a \in \overline{P_S} \wedge B \mapsto a \in \overline{P_S}$$

combining this with (19) gives the desired result. □

THIS PAGE IS LEFT BLANK INTENTIONALLY

References

[1] L. Badger. Providing a Flexible Security Override for Trusted Systems. In *Computer Security Foundations Workshop III*, IEEE Computer Society Press, Franconia, June 1990.

[2] D.E. Bell and L.J. LaPadula. *Secure Computer Systems: Mathematical Foundations and Model.* Technical Report M74-244, The MITRE Corporation, Bedford, MA., October 1974.

[3] D.E. Bell. Security Policy Modeling for the Next-Generation Packet Switch. In *1988 IEEE Symposium on Security and Privacy*, pages 212–216, IEEE Computer Society, Oakland, CA., April 1988.

[4] K.J. Biba. *Integrity Considerations for Secure Computer Systems.* Technical Report MTR-3153, The Mitre Corporation, Bedford, MA., April 1977.

[5] G. Birkhoff. *Lattice Theory.* Volume XXV of *American Mathematical Society Colloqium Publications*, American Mathematical Society, 3 edition, 1967.

[6] D.F. Brewer and M.J. Nash. The Chinese Wall Security Policy. *Proceedings 1989 Symposium on security and Privacy*, pages 206–258, IEEE Computer Society, Oakland, CA., May 1989.

[7] D.D. Clark and D.R. Wilson. A Comparison of Commercial and Military Computer Security Models. In *Proceedings 1987 Symposium on security and Privacy*, IEEE Computer Society, Oakland, CA., April 1987

[8] D.E. Denning. A Lattice Model of Secure Information Flow. *Communications of the ACM*, 19(5):236–243, May 1976.

[9] D.E. Denning. *On the Derivation of Lattice Structured Information Flow Policies.* Technical Report CSD TR180, Purdue University, 1976.

[10] S.N. Foley. A Model for Secure Information Flow In *1989 IEEE Symposium on Security and Privacy*, IEEE Computer Society, Oakland, May, 1989.

[11] S.N. Foley. *Lattices for Security Policies* Report 90005, Royal Signals and Radar Establishment, April 1990.

[12] S.N. Foley. Secure Information Flow using Security Groups In *Computer Security Foundations Workshop III*, IEEE Computer Society Press, Franconia, June, 1990.

[13] J.A. Goguen, J. Messeguer. Security Policies and Security Models In *1982 IEEE Symposium on Security and Privacy*, IEEE Computer Society, Oakland, April, 1982.

[14] C.A.R. Hoare. Communicating Sequential Processes, Prentice Hall.

[15] T.M.P. Lee. Using Mandatory Integrity to Enforce "Commercial Security", In *1988 IEEE Symposium on Security and Privacy*, IEEE Computer Society, Oakland, April, 1988.

[16] T.Y. Lin. Chinese Wall Security Policy - An Aggressive Model. In *Proceedings of the 5th Aerospace Symposium on Computer Security and Privacy*, 1989.

[17] T.F. Lunt. Aggregation and Inference: Facts and Fallacies, In *1988 IEEE Symposium on Security and Privacy*, IEEE Computer Society, Oakland, April, 1988.

[18] J. McLean. The Algebra of Security. In *1988 IEEE Symposium on Security and Privacy*, pages 2–7, IEEE Computer Society, Oakland, CA., April 1988.

[19] C. Meadows. Extending the Brewer-Nash Model to a Multi-level Context. In *1990 IEEE Symposium on Security and Privacy*, IEEE Computer Society, Oakland, CA., May 1990.

[20] C. Meadows. Aggregation Problems: A Position Paper. *RADC Workshop on Dynamic Security Policies*, June 1990.

[21] W. R. Shockley. Implementing Clark/Wilson Integrity Policy Using Current Technology. In *Proceedings 11th National Computer Security Conference*, October, 1988.

[22] M Branstad, et. al. Trusted Mach Design Issues. In *Proceedings of the Third Aerospace Computer Security Conference*, December 1987.

[23] J.M. Spivy. *The Z Notation*. Prentice Hall, 1989.

[24] P.F. Terry and S.R. Wisemman. A 'New' Security Policy Model. In *1989 IEEE Symposium on Security and Privacy*, Oakland CA., May 1989.

[25] C. Weissman. Security Controls in the ADEPT-50 Time Sharing System. In *Proceedings of the 1969 AFIPS Fall Joint Computer Conference*, pages 119–133, AFIPS Press, Arlington, Va, 1969.

[26] J.P.L. Woodward. Expoliting the Dual Nature of Sensitivity Labels. In *1987 IEEE Symposium on Security and Privacy*, Oakland CA., April 1987.

THIS PAGE IS LEFT BLANK INTENTIONALLY

REPORT DOCUMENTATION PAGE DRIC Reference Number (if known) ....

Overall security classification of sheet ....Unclassified...

(As far as possible this sheet should contain only unclassified information. If it is necessary to enter classified information, the field concerned must be marked to indicate the classification eg (R), (C) or (S).

Originators Reference/Report No. REPORT 90020	Month OCTOBER Ye⸗ 1990
Originators Name and Location RSRE, St Andrews Road Malvern, Worcs WR14 3PS	
Monitoring Agency Name and Location	
Title UNIFYING INFORMATION FLOW POLICIES	
Report Security Classification UNCLASSIFIED	Title Classification (U, R, C or S) U
Foreign Language Title (in the case of translations)	
Conference Details	
Agency Reference	Contract Number and Period
Project Number	Other References
Authors FOLEY, S N	Pagination and Ref 94

Abstract

This report proposes a structure for describing information flow policies that can express transitive, aggregation and separation (of duty) exceptions. Operators for comparing, composing and abstracting flow policies are described. These allow complex policies to be built from simpler policies. Many existing confidentiality (and by using a dual model, integrity) policies can be captured in this framework. A high water mark model is developed that can enforce these information flow policies. This model provides the basis for a taxonomy of existing high water mark mechanisms.

Abstract Classification (U,R,C or S)
U

Descriptors

Distribution Statement (Enter any limitations on the distribution of the document)

UNLIMITED

580/48

THIS PAGE IS LEFT BLANK INTENTIONALLY