Information Flow, Distributed Systems, and Refinement, by Example

Joshua D. Guttman

The MITRE Corporation and
Worcester Polytechnic Institute

1 Introduction

Non-interference is one of the foundational notions of security stretching back to Goguen and Meseguer [3]. Roughly, a set of activities C is non-interfering with a set D if any possible behavior at D is compatible with anything that could have occurred at C. One also speaks of "no information flow" from C to D in this case. Many hands further developed the idea and its variants (e.g. [15,12]), which also flourished within the process calculus context [6,13,1,2]. A. W. Roscoe contributed a characteristically distinctive idea to this discussion, in collaboration with J. Woodcock and L. Wulf. The idea was that a system is secure for flow from C to D when, after hiding behaviors at the source C, the destination D experiences the system as *deterministic* [8,11]. In the CSP tradition, a process is deterministic if, after engaging in a sequence t of events, it can refuse an event a, then it always refuses the event a after engaging in t [9].

One advantage of this approach via determinism is that it disposed of the so-called "refinement paradox" of non-interference (for which C. Morgan [7] cites J. Jacob [6], who does not use the term). Namely, a system might display non-interference, but refine to a system that caused impermissible information flows. Refinement does not preserve ignorance, in Morgan's words. However, if the system is already deterministic to the destination, no refinement can provide the destination with information about the behavior of the source.

Unfortunately, non-interference is too strong a property to be desirable except rarely. One rarely would design a system that has the activities C, D when C should not interfere with D in any way at all. One would instead like to design systems in which there are at least clear limitations on how that interference may occur. For instance, perhaps there is a responsible intermediary M such that C may influence M and M may then decide what information to make visible to the destination D. Thus, writing "may influence directly" as $\rightsquigarrow$, we have $C \rightsquigarrow M \rightsquigarrow D$, although $C \not\rightsquigarrow D$. In this case, the "may-influence" relation is not transitive. One may view this intransitive non-interference as a kind of declassification, one in which the permissible intermediaries are trusted to decide what information may reach the destination. From this point of view, it is a kind of "who" declassification, in which the policy identifies which domains M are permitted to choose what information to allow to pass from C to D [14].

A second advantage of Roscoe's determinism idea turned out to be its surprising and attractive applicability to intransitive non-interference, developed with M. Goldsmith [10]. Non-interference given an intransitive "may-influence" relation meant that, *hiding* the behavior of the sensitive source C, and *fixing* the behavior of the permissible intermediary M, the destination D again experiences the system as deterministic.

However, suppose we have explicit specifications of what we would like to permit D to learn about C? For instance, the buyer should be able to learn what the president had for breakfast, so as to replenish the larder, but not who she vetted for the court opening. This is called "what" declassification, since the content determines what D may learn and what not. The determinism point of view does not seem to provide an explanation of "what" declassification, which would be attractive.

We think it also attractive to recast the notions in a context that makes the graph structure of distributed systems explicit, and allows us to use the graph structure as a guide to information flow properties [4]. In this paper, we aim to explain, largely by example, three aspects of information flow in distributed systems that are governed by "what" declassification policies:

1. How to define policies bounding "what" declassification, i.e. upper bounds on information flow, and also functionality goals expressed as lower bounds on information flow;
2. How to represent distributed systems as directed graphs in which the nodes are processing elements and the arcs are message channels, in which these policies are meaningful;
3. How to ensure that these conclusions are preserved when a system is refined using a surprisingly simple but still useful principle.

Functionality goals as lower bounds on information flow are new in this paper, as is the simple refinement principle.

2 An Example System

We will consider a system EpiDB with very simple, but nevertheless useful, behavior. We do not focus on the realism of EpiDB, as we will use it simply to stimulate intuition for the information flow considerations at hand. EpiDB is suggested by a related unpublished demonstration system written by two colleagues.

2.1 The EpiDB idea

EpiDB serves as a database for epidemiological information. Imagine that health-care providers deposit two kinds of records into the system. First, we have a table of *disease records*, that say of a particular person that they had a particular disease during a period of time. Second, we have a table of *personal encounter records*, that say of an unordered pair of people that they had an encounter on a particular date, or that they encounter each other habitually, possibly because they belong to the same family or school class.

EpiDB will be used by public health analysts who seek to understand the propagation of diseases through this population. Thus, an analyst A asks a query about a person p_1, a disease d, and a time t_0. If that query is permitted from A, and p_1 had the disease d at a time t_1 near t_0, then the system will return a set of tuples (p_2, e_2, t_2) such that p_2 encountered p_1 at time e_2, and had disease d at time t_2, where e_2 and t_2 are near t_0. For simplicity, we will choose a parameter ϵ, and take "ϵ-near" to mean that $|t' - t| < \epsilon$. Thus, the query takes a sort of join on the two tables, containing the disease and personal encounter records, restricted to times near t_0.

If the query is permitted from A but p_1 did not have the disease d near t_0, it returns a distinctive value *unsick* denying the diagnosis. If the query is impermissible from A, it returns a distinctive value *imperm* denying permission. Perhaps some analysts are responsible only for certain diseases, and if they start querying for sexually transmitted diseases instead of influenza (e.g.), they are letting their curiosity get the better of them. Alternatively, some analysts may be authorized to ask about some patients but not others, or some time periods. In this example system, we will assume that permission is independent of the contents of the database, and does not change as it operates.

If the database's state consists of the tables of disease records with contents T^d and personal encounter records with contents T^e, then we will write $\mathsf{ans}(A, q, T^d, T^e)$ for the result when query $q = (p_1, d, t_0)$ is received on c, where:

$$\mathsf{ans}(c, q, T^d, T^e) = \begin{cases} \mathit{unsick} & \text{if not sick } \epsilon\text{-near } t_0 \\ \mathit{imperm} & \text{if not permitted} \\ \{(p_2, e_2, t_2) \colon \exists t_1 \,.\; & (p_1, d, t_1) \in T^d,\ (p_2, d, t_2) \in T^d, \\ & (\{p_1, p_2\}, e_2) \in T^e, \\ & \text{and } t_1, t_2, e_2 \text{ are } \epsilon\text{-near } t_0\} \end{cases}$$

To simplify the statement of information flow upper and lower bounds, we will assume one type of coordination between the analysts and the data provider. Namely, we will assume that the data provider remains up-to-date, while the analysts are not concerned with very recent events. Thus, we will assume that if an analyst ever makes a query q about a time t, and a provider ever deposits a record r concerning a related time $t' < t + 2\epsilon$, then in fact the system received r before q. As a consequence, no query ever has a result that would have been altered by records received subsequently. In particular, the analyst can never detect the order of arrival of records by a sequence of queries.

2.2 Simplest EpiDB system

Thus, the simplest version of our system EpiDB takes the form shown in Fig. 1, in which a provider PR delivers data into the database E itself, which can be queried by an analyst A_1. We assume that E starts empty, so that its contents at any time is just what PR has delivered over channel 3.

We regard the whole graph as the system, rather than simply the node E, partly because in subsequent steps there are additional nodes, but also because

$$\begin{tikzcd}
\text{Analyst} & \text{DB} & \text{Provider} \\
A_1 \arrow[r, bend left, "1"] & E \arrow[l, bend left, "2"] & PR \arrow[l, "3"']
\end{tikzcd}$$

Fig. 1. Schematic System EpiDB

the security and functionality goals of the system are about A_1 and PR. In particular, A_1 is authorized to learn certain aspects of the behavior of PR. A_1 can learn which records PR has submitted that are relevant to a permissible query. If PR submits records that are not related to any permissible query of A_1, then EpiDB is obliged to ensure that they can have no effect on what A_1 observes on channels $1, 2$.

We do not need to specify the behavior of A_1 and PR, since the goals should hold regardless of their actions. Thus, we regard them as always willing to send or receive any message on their outgoing or incoming channels.

By contrast, E has a specification. We can describe it as a state machine where the state includes two sets of tuples, representing the tables T^d, T^e. An additional state component records the not-yet-processed query (p_1, d, t_0) or else $\perp$ if every query has already received a response. A new record r_d, r_e may be deposited at any moment, even between receiving a query and answering it, so this state component remembers any as-yet unanswered query. We do, however, maintain the upper bound t of the times mentioned in all queries we have received; we refuse to receive a new record whose time does not exceed $t + 2\epsilon$. We record this maximum query time in the state component m, and we require that when a record r is received, its time is greater than $m + 2\epsilon$. We write $\mathsf{time}(q)$ or $\mathsf{time}(r)$ for the last component of q, r, which is its time component.

We give the labeled transition relation in Fig. 2. Notice that E does not accept a new query until it has answered the previous one, and restored $\perp$ to the first state component. Also, channel 2 carries a *set* of records (p_2, e_2, t_2), or else a symbol *unsick*, *imperm*.

2.3 The intended information flow

EpiDB is intended to limit information flow from the provider PR to the analyst A_1. In particular, the access control system is intended to limit flow to information for which A_1 is authorized. The remainder of the system is intended to maximize flow subject to authorization, and relevance to the queries A_1 asks.

For definiteness, we will assume that each analyst A has been assigned:

$\mathsf{persons}(A)$**:** A set of persons of interest for A;
$\mathsf{diseases}(A)$**:** A set of diseases A is authorized to consider;
$\mathsf{start}(A)$**:** an earliest time about which to query; and
$\mathsf{finish}(A)$**:** a most recent time about which to query.

Pre-State	Label (channel, message)	Post-State
$(\bot,\, T^d,\, T^e,\, m)$	$(1,\, q)$	$(q,\, T^d,\, T^e,\, m')$
$(q,\, T^d,\, T^e,\, m)$	$(2,\, a)$	$(\bot,\, T^d,\, T^e,\, m)$
$(x,\, T^d,\, T^e,\, m)$	$(3,\, r_d)$	$(x,\, T^d \cup \{r_d\},\, T^e,\, m)$
$(x,\, T^d,\, T^e,\, m)$	$(3,\, r_e)$	$(x,\, T^d,\, T^e \cup \{r_e\},\, m)$

where $a = \mathsf{ans}(A_1, q, T^d, T^e)$, $m' = \max(m, \mathsf{time}(q))$
q is a non-$\bot$ query, x is any value,
and r_d, r_e are respectively a disease record and an encounter record,
with $\mathsf{time}(r_d), \mathsf{time}(r_e) > m + 2\epsilon$

Fig. 2. Labeled transition relation for E

Since A will be able to learn about disease records within ϵ of the time t_0 in a query, we will write $\mathsf{Int}(A) = [\mathsf{start}(A) - \epsilon, \mathsf{finish}(A) + \epsilon]$ to define the the interval of disease records A is authorized to learn about.

The analyst A who queries p, d, t will learn whether p had disease d at time t' near t, as long as $p \in \mathsf{persons}(A)$, $d \in \mathsf{diseases}(A)$, and $t' \in \mathsf{Int}(A)$. Or more precisely, A learns whether PR has registered this fact in the relevant portion of its run. The set of permissible queries creates a region R_0 of the space of disease records that A can learn about directly.

The relevant portion of PR's run also contains a set of encounter records of the form $(\{p, p'\}, e)$, and these records create an *adjacency* relation between records $r_d \in R_0$ and other disease records involving p', d, and a nearby time t'. We will refer to the set of disease records adjacent to R_0 as R_1.

Essentially, the authorization mechanism entails that A should learn nothing about what disease records and encounter records PR has submitted, except as they help to determine R_0 and R_1. In particular, PR messages that provide encounter records not connected to R_0 should be invisible to A. Moreover, given a set of encounter records, PR messages that provide disease records not in R_0 or R_1 are also invisible.

In particular, A's observations as a consequence of a single query must remain unchanged, regardless of variation in PR's messages containing encounter records unconnected to R_0 and regardless of variation in disease records not in $R_0 \cup R_1$. A query imposes no ordering requirement on PR's messages.

By submitting a sequence of queries, A can learn conjunctions of the conclusions returned by the individual queries. But by the timing constraints, A cannot exclude any particular order in which the records may have arrived. In particular, a record can have been absent from an earlier response if it is found in a later response.

Thus, the purpose of the EpiDB system is essentially a *what*-declassification, where the regions R_0, R_1 for each permissible query q determine what aspects of the sensitive PR runs should be "declassified" and made available to the analyst A asking q.

$\mathcal{LO}, \mathcal{CH}, \mathcal{DA}, \mathcal{ST}, \mathcal{EV}$

$\mathsf{sndr} : \mathcal{CH} \to \mathcal{LO}$	$\mathsf{rcpt} : \mathcal{CH} \to \mathcal{LO}$
$\mathsf{chan} : \mathcal{EV} \to \mathcal{CH}$	$\mathsf{msg} : \mathcal{EV} \to \mathcal{DA}$
$\mathsf{lts} : \mathcal{LO} \to \textsc{lts}$	

Table 1. Signature of frames

Later (Section 5) we will refine the schematic version of EpiDB from Figs. 1–2 into a more complex system with separate components that guide an efficient and reliable implementation.

3 Information Flow in the Frame Model

In this section, we will summarize the key notions of [4]. Systems (or *frames*), represented as directed graphs, have *executions*; the local portions of an execution are called *local runs*; and an observer who sees one local run is trying to infer information at a source, by determining what local runs at that source are *compatible* with the observations. An information flow specification, which we call a *blur*, is a specific kind of closure condition on the set of compatible local runs at the information source.

3.1 Frames and Executions

We formalize systems such as EpiDB by structures we call *frames.* A frame $\mathcal{F}$ consists of a directed graph, the nodes (or locations) of which are processing elements each defined by a labeled transition system, and the arcs of which carry messages. An *execution* of a frame $\mathcal{F}$ is a partially ordered set of events, where each event e has a channel $\mathsf{chan}(e)$ and a message $\mathsf{msg}(e)$. The events associated with a single node n must be linearly ordered, and moreover must form a possible trace of $\mathsf{lts}(n)$. However, events on two channels that are not attached to a common node may be unordered, unless some causal sequence of events connects them. We will use the words "node" and "location" synonymously.

Definition 1. *Let $\mathcal{LO}, \mathcal{CH}, \mathcal{DA}, \mathcal{ST}, \mathcal{EV}$ be domains that we will call* locations*,* channels*,* data*,* states*, and* events*, resp.*

1. *A* labeled transition relation *is a ternary relation $\leadsto \subseteq \mathcal{ST} \times \mathcal{EV} \times \mathcal{ST}$. A* labeled transition system *is a pair $(\leadsto, s_0)$ of a labeled transition relation and an "initial state" $s_0 \in \mathcal{ST}$.* LTS *is the set of labeled transition systems.*
2. *When $\ell \in \mathcal{LO}$, we define* $\mathsf{chans}(\ell) = \{c \in \mathcal{CH} \colon \mathsf{sndr}(c) = \ell$ *or* $\mathsf{rcpt}(c) = \ell\}$.
3. *A* frame *is a structure $\mathcal{F}$ containing the domains and functions shown in Table 1 satisfying the following properties:*
   (a) For all $e_1, e_2 \in \mathcal{EV}$, if $\mathsf{chan}(e_1) = \mathsf{chan}(e_2)$ *and* $\mathsf{msg}(e_1) = \mathsf{msg}(e_2)$*, then for all $\ell \in \mathcal{LO}$ and $s, s' \in \mathcal{ST}$, $s \overset{e_1}{\leadsto}_\ell s'$ iff $s \overset{e_2}{\leadsto}_\ell s'$.*
   (b) For all $s, s' \in \mathcal{ST}$, $e \in \mathcal{EV}$, and $\ell \in \mathcal{LO}$, $s \overset{e}{\leadsto}_\ell s'$ implies $\mathsf{chan}(e) \in \mathsf{chans}(\ell)$.

   where we let $(\leadsto_\ell, \mathsf{initial}(\ell)) = \mathsf{lts}(\ell)$. ///

The *histories* of an LTS $(\rightsquigarrow, s_0)$ are all finite or infinite alternating sequences $h = \langle s_0, e_0, s_1, \ldots, s_i, e_i, s_{i+1}, \ldots\rangle$ starting with s_0, such that $(s_j, e_j, s_{j+1}) \in \rightsquigarrow$ whenever e_j is well defined. In particular, s_{j+1} is well defined whenever e_j is, so that h does not end with an event e_j. A *trace of* $(\rightsquigarrow, s_0)$ is a finite or infinite sequence of events $tr = \langle e_0, e_1 \ldots\rangle$ such that there is a history h where tr enumerates the events in h.

An execution is a partially ordered set of events that—when projected onto $\mathsf{chans}(\ell)$—always yields a trace for ℓ.

Definition 2. $\mathcal{A} = (E, \preceq)$ *is an* execution *for a frame* $\mathcal{F}$*, written* $\mathcal{A} \in \mathsf{Exc}(\mathcal{F})$*, iff* $E \subseteq \mathcal{EV}$ *and* $\preceq$ *is a well-founded partial ordering on* E*, and, for all* $\ell \in \mathcal{LO}$*, letting* $tr_{\mathcal{A}}(\ell)$ *be the set* $\{e \in E \colon \mathsf{chan}(e) \in \mathsf{chans}(\ell)\}$*,*

1. $tr_{\mathcal{A}}(\ell)$ *is linearly ordered by* $\preceq$*; and*
2. $tr_{\mathcal{A}}(\ell)$ *ordered by* $\preceq$ *is a trace of* $\mathsf{lts}(\ell)$*.* ///

When $\mathcal{LO}$ is finite, the "well-founded" condition is redundant. If $\mathcal{A} = (E, \preceq)$ is an execution, and $\preceq'$ is a partial order that is stronger than $\preceq$, i.e. $\preceq \subseteq \preceq'$, then $\mathcal{A}' = (E, \preceq')$ is also an execution. The weakest partial order is generated from the sequential traces of the individual locations, and extended to events at other locations when they share an event on some channel that connects them. However, any strengthening of this order determines another execution based on the same set E of events.

Our notion of execution ignores what states the locations ℓ reach after engaging in the events $tr_{\mathcal{A}}(\ell)$, and thus ignores the effects of nondeterminism. A similar theory can be developed including the resulting states, which would let us talk about refusals as well as traces, but we will postpone that opportunity for now.

We have here a synchronous notion of communication; a message m passes over channel c only if both endpoints can take a transition with label c, m. Thus, the sender learns that the recipient is willing to accept m over c now. Information flows over channels in both directions.

3.2 Local Runs and Compatibility

We can now define what an observer with access to a particular set of channels sees, or what a source of information does. We will assume that the observer or the source has access to a set of channels $C \subseteq \mathcal{CH}$. Often C is of the form $C = \mathsf{chans}(\ell)$ for some $\ell \in \mathcal{LO}$ or $C = \bigcup_{\ell \in L} \mathsf{chans}(\ell)$ for some $L \subseteq \mathcal{LO}$, but this is not always the case.

A local run at C is just the result of restricting the events in some execution to the channels C.

Definition 3. *Let* $\mathcal{B} = (E, R)$ *be a partially ordered set of events, and* $C \subseteq \mathcal{CH}$*.*

1. *The* restriction $\mathcal{B} \restriction C$ *is* (B_0, R_0)*, where*
   $B_0 = \{e \in E \colon \mathsf{chan}(e) \in C\}$*, and*

$R_0 = R \cap (B_0 \times B_0)$.

2. *$\mathcal{B}$ is a C-*run *of $\mathcal{F}$ iff for some $\mathcal{A} \in \mathsf{Exc}(\mathcal{F})$, $\mathcal{B} = \mathcal{A} \restriction C$.*
3. *C-$\mathsf{runs}(\mathcal{F}) = \{\mathcal{B} \colon \mathcal{B}$ is a C-run of $\mathcal{F}\}$.*

We write C-runs when $\mathcal{F}$ is understood, and, when C is understood, we speak of local runs*.*

$\mathcal{B}_2$ extends $\mathcal{B}_1$*, when $\mathcal{B}_1 = (E_1, \preceq_1)$ and $\mathcal{B}_2 = (E_2, \preceq_2)$ are p.o. sets, iff $E_1 \subseteq E_2$; $\preceq_1 = \preceq_2 \cap (E_1 \times E_1)$; and $\{e \colon \exists e_1 \in E_1 \,.\, e \preceq_2 e_1\} \subseteq E_1$.* ///

Fix some frame $\mathcal{F}$. What an observer at D knows is that some $\mathcal{B} \in D$-$\mathsf{runs}(\mathcal{F})$ occurred, since she observed some $\mathcal{B}$. She wants to consider what local runs are still possible at some source $D \subseteq \mathcal{CH}$. These are the members of D-$\mathsf{runs}(\mathcal{F})$ that are restrictions of executions that also restrict to $\mathcal{B}$.

Definition 4. *Let $C, D \subseteq \mathcal{CH}$ and $\mathcal{D} \in D$-runs.*

1. *A local run $\mathcal{B} \in C$-runs is* compatible with *$\mathcal{D}$ iff, for some $\mathcal{A} \in \mathsf{Exc}$, $\mathcal{A} \restriction C = \mathcal{B}$ and $\mathcal{A} \restriction D = \mathcal{D}$.*
2. *$J_{C \lhd D}(\mathcal{D}) = \{\mathcal{B} \in C$-$\mathsf{runs} \colon \mathcal{B}$ is compatible with $\mathcal{D}\}$.* ///

We use the letter J to indicate that these $\mathcal{B}$ can occur jointly with $\mathcal{D}$. The subscripts indicate that information would flow from C to D if $J_{C \lhd D}(\mathcal{D})$ fails to have suitable closure properties. The subscript D adjacent to the argument $\mathcal{D}$ is meant to remind that $\mathcal{D} \in D$-runs, as a kind of type-annotation; the left-most subscript C is a reminder of the type of the local runs in the result.

3.3 Blurs to Limit Information Flow

Generally speaking, when $J_{C \lhd D}(\mathcal{D})$ is "large" for all $\mathcal{D} \in D$-runs, then there is little flow from C to D. The observations at D leave open many possibilities for what could have happened at C. We can make precise what the observer at D cannot learn by considering *closure operators* on sets of local C-runs. We think of the observer's vision as blurred insofar as she cannot distinguish a local C-run from other members of a closed set. Thus, the relation of coarsening on closure operators represents the observer's loss of resolution as information flow decreases.

Generally speaking, a closure operator obeys three properties. Each set in *included* in its closure; closure is *idempotent*; and closure is *monotonic* with respect to the inclusion relation. We found that information flow respects the graph structure of frames when we strengthen the montonicity property somewhat [4]. We call operators that satisfy these strengthened conditions *blur operators*.

Definition 5. *A function ϕ on sets is a* blur operator *iff it satisfies:*

Inclusion: *For all sets S, $S \subseteq \phi(S)$;*
Idempotence: *ϕ is idempotent, i.e. for all sets S, $\phi(\phi(S)) = \phi(S)$; and*
Union: *ϕ commutes with unions: If $\{S_a\}_{a \in I}$ is a family indexed by I, then*

$$\phi(\bigcup_{a \in I} S_a) = \bigcup_{a \in I} \phi(S_a).$$

*S is ϕ-*blurred *iff ϕ is a blur operator and $S = \phi(S)$.* ///

Observe that $\bigcup_{a\in I}\phi(S_a)\subseteq\phi(\bigcup_{a\in I}S_a)$ is equivalent to monotonicity, so that the *union* property is effectively monotonicity plus a converse. The *union* property ensures that ϕ is determined by its action on singletons. Since $S=\bigcup_{a\in S}\{a\}$, $\phi(S)=\bigcup_{a\in S}\phi(\{a\})$.

Blur operators form a lattice under pointwise inclusion, which provides a way to compare the flow of information in different situations. Thus, ϕ allows at least as much information flow as ψ if $\phi(S)\subseteq\psi(S)$ for every S.

The EpiDB blur. In the case of EpiDB, we are interested in a blur ϕ on the local runs at channel 3, i.e. $C=\{3\}$. Since, by the *union* property, we only need to define $\phi(\{\mathcal{B}\})$ for singletons of a $\mathcal{B}\in C\text{-}\mathsf{runs}$, we must say which local runs $\mathcal{B}'$ should be indistinguishable from $\mathcal{B}$ for the observer on channels $1,2$, i.e. A_1. However, Section 2.3 already makes clear which $\mathcal{B}'$ this should be. Analyst A_1 has permissions defined in terms of $\mathsf{persons}(A_1)$, $\mathsf{diseases}(A_1)$, and $\mathsf{Int}(A_1)$.

Define $R_0(\mathcal{B})$ to be the set of disease records (p,d,t) delivered in $\mathcal{B}$ such that $p\in\mathsf{persons}(A_1)$, $d\in\mathsf{diseases}(A_1)$, and $t\in\mathsf{Int}(A_1)$. Define $R_1(\mathcal{B})$ to be the set of disease records (p_1,d,t_1) in $\mathcal{B}$ such that there is an encounter record $(\{p,p_1\},e)$ in $\mathcal{B}$ with t,e,t_1 successively ϵ-near. Then

$$\phi(\{\mathcal{B}\})=\{\mathcal{B}'\colon R_0(\mathcal{B})=R_0(\mathcal{B}')\text{ and }R_1(\mathcal{B})=R_1(\mathcal{B}')\}$$

We can also express this more operationally: $\phi(S)$ is closed under

1. permutations;
2. adding:
   (a) records submitted elsewhere in $\mathcal{B}$;
   (b) encounter records not connecting $R_0(\mathcal{B})$ to any disease record in $\mathcal{B}$;
   (c) disease records $r_d=(p,d,t)$ such that
      i. $p\notin\mathsf{persons}(A_1)$, $d\notin\mathsf{diseases}(A_1)$, or $t\notin\mathsf{Int}(A_1)$, and
      ii. r_d is not connected to $R_0(\mathcal{B})$ by an encounter record;
3. omitting records of the same kinds.

Limited flow. The blur notion suggests a *restricted information flow* notion, and moreover the latter respects the graph structure. Specifically, limiting what information flows to a *cut set* in the graph guarantees the same limit applies to observers beyond that cut set.

Definition 6. *Let* $\mathsf{obs},\mathsf{src}\subseteq\mathcal{CH}$ *and* $\phi\colon\mathcal{P}(\mathsf{src}\text{-}\mathsf{runs})\to\mathcal{P}(\mathsf{src}\text{-}\mathsf{runs})$.

$\mathcal{F}$ ϕ-limits src-to-obs flow *iff* ϕ *is a blur operator, and, for every* $\mathcal{B}\in\mathsf{obs}\text{-}\mathsf{runs}$ $J_{\mathsf{src}\lhd\mathsf{obs}}(\mathcal{B})$ *is* ϕ*-blurred.*

This notion respects the graph structure of the frame $\mathcal{F}$. First, since effectively information can flow in either direction over a channel, we consider the undirected graph $\mathsf{ungr}(\mathcal{F})=(V,E)$ where the vertices V are the locations, $V=\mathcal{LO}$, and where an undirected edge (ℓ_1,ℓ_2) exists iff, for some $c\in\mathcal{CH}$, $\mathsf{sndr}(c)=\ell_1$ and $\mathsf{rcpt}(c)=\ell_2$ or vice versa. Now, for $C_0,C_1,C_2\subseteq\mathcal{CH}$, let us say that C_1 is a *cut between* C_0 and C_1 iff, for every path p through $\mathsf{ungr}(\mathcal{F})$ that starts at a $c_0\in C_0$ and ends at a $c_2\in C_2$, p traverses some $c_1\in C_1$. Now:

Theorem 1 (Cut-Blur Principle, [4]). *Let* $\mathsf{src}, \mathsf{cut}, \mathsf{obs} \subseteq \mathcal{CH}$*, where* cut *is a cut between* src *and* obs *in* $\mathcal{F}$*.*

If $\mathcal{F}$ ϕ*-limits* src*-to-*cut *flow, then* $\mathcal{F}$ ϕ*-limits* src*-to-*obs *flow.*

There is also a two-frame version of the same idea. Here, $\mathcal{F}_2$ agrees with $\mathcal{F}_1$ on the portion of the graph that lies from src to cut, and on the LTS of those locations. As long as $\mathcal{F}_2$ does not exercise possibilities at cut that $\mathcal{F}_1$ does not, then ϕ-limited flow is preserved. We write $\mathcal{CH}_i, \mathcal{LO}_i, C\text{-}\mathsf{runs}_i$, etc. for the channels, locations, local runs etc. of $\mathcal{F}_i$.

Theorem 2 ([4]). *Let* $\mathsf{src}, \mathsf{cut} \subseteq \mathcal{CH}_1$ *in* $\mathcal{F}_1$*.*

Let $\mathcal{F}_2$ *be a frame, with* $\mathsf{src}, \mathsf{cut} \subseteq \mathcal{CH}_2$*, and such that, if* p *is any path in* $\mathsf{ungr}(\mathcal{F}_1)$ *starting at some* $c_0 \in \mathsf{src}$ *and traversing no arc in* cut*, and* p *reaches* $c \in \mathcal{CH}_1$*, then:*

1. $c \in \mathcal{CH}_2$, $\mathsf{sndr}_1(c) \in \mathcal{LO}_2$, *and* $\mathsf{rcpt}_1(c) \in \mathcal{LO}_2$*;*
2. $\mathsf{sndr}_2(c) = \mathsf{sndr}_1(c)$, *and* $\mathsf{rcpt}_2(c) = \mathsf{rcpt}_1(c)$*;*
3. $\mathsf{lts}_1(\mathsf{sndr}_1(c)) = \mathsf{lts}_2(\mathsf{sndr}_2(c))$ *and* $\mathsf{lts}_1(\mathsf{rcpt}_1(c)) = \mathsf{lts}_2(\mathsf{rcpt}_2(c))$.

Let $\mathsf{obs} \subseteq \mathcal{CH}_2$ *be such that* cut *is a cut between* src *and* obs *in* $\mathcal{F}_2$*. If* $\mathsf{cut}\text{-}\mathsf{runs}_2 \subseteq \mathsf{cut}\text{-}\mathsf{runs}_1$*, and* $\mathcal{F}_1$ ϕ*-limits* src*-to-*cut *flow, then* $\mathcal{F}_2$ ϕ*-limits* src*-to-*obs *flow.*

In fact, the cut-blur principle is a corollary of this; when we equate $\mathcal{F}_2 = \mathcal{F}_1$, the assumptions necessarily hold.

This principle is useful for "localizing" the enforcement of ϕ-limiting to the portion of the system lying between src and cut. It says that we can freely vary the structure of the remainder of the system, just so long as we do not force cut to engage in new local behaviors. For instance, if we consider $\mathsf{cut} = \{1, 2\}$ and $\mathsf{src} = \{3\}$ in either Fig. 1 or Fig. 4, it says that we can freely expand the node A_1 into multiple nodes and arcs, as long as cut remains a cut. The assumption that $\mathsf{cut}\text{-}\mathsf{runs}_2 \subseteq \mathsf{cut}\text{-}\mathsf{runs}_1$ is immediate here, since we assume that A_1 may attempt any sequence of communications anyway.

4 Questions and Answers

We would now like a corresponding way to specify functionality goals, i.e. *lower* bounds on information flow between a source and an observer. For instance, if A_1 is permitted to submit a query $q = (p, d, t)$ over channel 1, then A_1 really should be able to learn from the system what the answer is, as of the time of this interaction. Thus, the system is guaranteeing that a local run over channels $1, 2$ can always extend to one in which A_1 submits query q and receives a symbol or set S of records over channel 2. And this answer tells A_1 whether PR has submitted a nearby disease record, and, in the stream of records PR has submitted on channel 3, what other disease records are adjacent via encounter records.

Thus, the response is compatible with a set of local PR runs, and serves to notify A_1 that no other type of run remains possible. We will call a classification like this a *question about* a set of channels such as the PR's channel set $\{3\}$.

Definition 7. *A family of sets $\mathcal{Q}$ is a* question about *a set of channels $C \subseteq \mathcal{CH}$ in $\mathcal{F}$ iff $\bigcup \mathcal{Q} = C\text{-}\mathsf{runs}(\mathcal{F})$.*

In our example, we can regard each permitted query $q = (p, d, t)$ as determining a question $\mathcal{Q}$ about PR's channel 3. Namely, two $\mathcal{B}, \mathcal{B}' \in \{3\}\text{-}\mathsf{runs}$ belong to the same $X \in \mathcal{Q}$ iff either:

– in both $\mathcal{B}$ and $\mathcal{B}'$, p is not sick with d at t, or else
– in both p is sick, with the same sick acquaintances and the same timings.

We can regard an impermissible query as determining a question also, but it is the trivial, singleton family $\{\{3\}\text{-}\mathsf{runs}\}$. Thus, each query q determines a question $\mathcal{Q}_q$ about channel 3.

An observer at D may want to determine which member of this family $\mathcal{Q}$ obtains. That is, the observer would like to extend the current local run so that the system's behavior will determine an $A \in \mathcal{Q}$ that must have been found at C. This may require D to engage in certain events that "ask about" $\mathcal{Q}$, after which the system's behavior will lead to the information. Naturally, the events that pose the question must be within the power of the observer at D.

Definition 8. $\mathcal{F}$ answers $\mathcal{Q}$ for $D \subseteq \mathcal{CH}$ *iff (i) $\mathcal{Q}$ is a question about C in $\mathcal{F}$, and (ii), for every $\mathcal{D} \in D\text{-}\mathsf{runs}$, there is an extension $\mathcal{D}'$ of $\mathcal{D}$ and a family $\mathcal{R}$ of finite extensions of $\mathcal{D}'$ such that:*

1. *For all $\mathcal{A} \in \mathsf{Exc}$, if $\mathcal{A} \restriction C = \mathcal{D}$, then there exists an extension $\mathcal{A}'$ of $\mathcal{A}$ such that $\mathcal{A}' \restriction C = \mathcal{D}'$;*
2. *for every $\mathcal{E} \in \mathcal{R}$, there exists a $X \in \mathcal{Q}$ such that $J_{C \lhd D}(\mathcal{E}) \subseteq X$; and*
3. *for every extension $\mathcal{E}$ of $\mathcal{D}'$, there exists a $\mathcal{E}_0 \in \mathcal{R}$ such that either $\mathcal{E}$ extends $\mathcal{E}_0$ or $\mathcal{E}_0$ extends $\mathcal{E}$.*

The first of these clauses ensures that the observer can always request the system to answer $\mathcal{Q}$. The second ensures that an observation in $\mathcal{R}$ selects some answer to the question, although there may be more than one right answer. The second says that the observations that determine an answer *bar* the tree of all extensions of $\mathcal{D}'$, so that any sufficiently long extension will have selected an answer.

Evidently, EpiDB answers the question $\mathcal{Q}_q$ for each q. The extension $\mathcal{D}'$ to a local A_1-run $\mathcal{D}$ consists in waiting for an answer on channel 2 to a previous, unanswered question (if any), and then submitting q on channel 1. The family $\mathcal{R}$ is then the set of local runs in which $\mathcal{D}'$ is extended by a symbol or set of records.

Of course, if a frame ϕ-blurs flow from C, then an answerable question about C can never be more informative than a ϕ-blurred question:

Lemma 1. *Let $\mathcal{Q}$ be a question about C in $\mathcal{F}$. Suppose that $\mathcal{F}$ answers $\mathcal{Q}$ for $D \subseteq \mathcal{CH}$, and that $\mathcal{F}$ ϕ-limits C-to-D flow.*

Then there is a $\mathcal{Q}'$ such that $\mathcal{Q}$ is a coarsening of $\mathcal{Q}'$, $\mathcal{F}$ answers $\mathcal{Q}'$ for D, and for every $X \in \mathcal{Q}'$, X is ϕ-blurred.

*Indeed, $\mathcal{Q}'$ can be chosen so that a pair of D-*runs *that can receive the same answer in $\mathcal{Q}$ can receive the same answer in $\mathcal{Q}'$.*

Proof. For each choice of $\mathcal{D}'$ and $\mathcal{R}$, collect the sets $J_{C \lhd D}(R)$ for $R \in \mathcal{R}$; let $\mathcal{Q}'$ be the resulting collection. Since $\mathcal{F}$ ϕ-limits C-to-D flow, each $J_{C \lhd D}(R)$ is ϕ-blurred.

To preserve "can receive the same answer," coarsen that $\mathcal{Q}'$ by taking unions:

In particular say that $R, R' \in \mathcal{R}$ are $\mathcal{Q}$-*similar*, which we will write $R \sim_{\mathcal{Q}} R'$, if there is an $X \in \mathcal{Q}$ such that $J_{C \lhd D}(R) \subseteq X$ and $J_{C \lhd D}(R') \subseteq X$. Define

$$\mathcal{Q}'_{\mathcal{R}} = \{ \bigcup_{R' \sim_{\mathcal{Q}} R} J_{C \lhd D}(R') \colon R \in \mathcal{R} \}.$$

The *union* property of blurs ensures that the resulting sets are ϕ-blurred.

Now let $\mathcal{Q}'$ collect $\mathcal{Q}'_{\mathcal{R}}$ from each choice of $\mathcal{D}'$ and $\mathcal{R}$. $\square$

In our EpiDB example, the questions $\mathcal{Q}_q$ are already ϕ-blurred.

5 Refining EpiDB

Although the simple presentation of EpiDB in Figs. 1–2 makes it clear why it will meet its information flow goals—both upper bounds and lower bounds—they are very far from a reasonable implementation. A reasonable implementation should have a number of different properties:

- It should be implemented via a number of virtual machines, so that its components can be responsive under high loads;
- It should separate an index from the actual archive that stores the data, to allow fast retrieval despite large quantities of data;
- It should separate critical services such as authorization from more vulnerable components that must service potentially malicious connections from analysts and providers.

All of these considerations militate for breaking the component E in Fig. 1 into a collection of cooperating components that interact via message channels. This decomposition fits the frame model very naturally, since the connections among these components are easy to define statically.

Step 1: Separating Authorization. A natural thing to do first is to identify a distinct component that uses the credentials of A_1 and the query q to make an authorization decision. For instance, these credentials could be certificates used in a bilateral TLS handshake. The authorization service can emit a cryptographic token that will be consulted by components in later expansions. Fig. 3 shows the resulting frame graph. Now the state of AR reflects whether authorization has been requested by the current query, and if so, the value of the resulting token. The behavior of the system on its channels $1, 2, 3$ is actually unchanged: In particular, given a local run $\mathcal{D}$ on channels $1, 2$, the set of compatible local runs on channel 3, $J_{\{3\} \lhd \{1,2\}}(\mathcal{D})$ is the same for the two systems.

Since the information flow of the system is defined solely in terms of $J_{\cdot \lhd \cdot}(\cdot)$, any desired upper and lower bounds on flow are necessarily preserved.

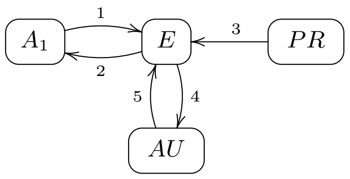

Fig. 3. EpiDB with authorization service separated

A refined EpiDB architecture. After several stages of refinement, we obtain a system of the form shown in Fig. 4. It breaks down the database into components with specialized responsibilities:

QC is a *query controller*. It accepts queries from A_1, passes requests to the index controller IC, which extracts records from the archive controller AC that are accumulated at QC. It returns the resulting sets to A_1.

IC is an *index controller*. It maintains an association between keys p_i naming people and a list of disease record numbers for those people. It has a similar association from people to encounter records. When given a person and a table name, it passes a list of record numbers to AC for retrieval.

AC is an *archive controller*. It maintains a store of records for each table, organized by record number.

IG is an *ingress controller*. It maintains the maximum record number used so far. It receives new records from the provider PR, assigns the next record number, and sends the record and number to AC. It notifies IC of the new association of this record number with the relevant p_is.

AU is the *authorization service*. QC contacts AU for each new query, obtaining a signed authorization token that accompanies QC's messages to IC. These tokens also appear in the system audit logs, if an audit subsystem is added.

The self-loop channels $8, 9$ allow QC and AC to signal certain internal events. The only other channel needing explanation is 6. At the beginning of processing any query, QC uses channel 6 to request the current maximum record number from AC, which maintains this. QC then limits all records retrieved to ones below this maximum. Hence, even when new records are being deposited by PR and IG concurrently, the query elicits consistent information reflecting the state of the database at the time of that maximum record number. Channel 12 is used only to propagate the maximum query time (shown as m in Fig. 2) to the ingress controller.

Again, the functional correctness criterion for this system is just that the same local runs should be possible on its two external interfaces, and with the same compatibility relations $J_{\{3\}\lhd\{1,2\}}(\mathcal{D})$.[1] The practical requirement for the

[1] By an interface, we just mean a set of channels, often but not necessarily near each other in the graph.

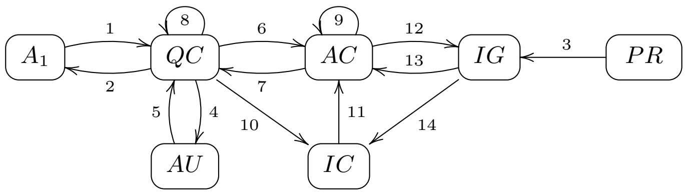

Fig. 4. Refined architecture for EpiDB

system designer to meet is that the index and archive controllers IC, AC should cooperate to maintain the database accurately, which is well understood.

The interface-preserving refinement principle. This refinement strategy is simple and easily formalized. When $\mathcal{F}_1$, $\mathcal{F}_2$ are frames, we write $J^i_{C \lhd D}(\cdot)$ for the compatibility function in $\mathcal{F}_i$.

Theorem 3. *Suppose that $\mathcal{F}_1$ and $\mathcal{F}_2$ are two frames, and $C, D \subseteq \mathcal{CH}_1 \cap \mathcal{CH}_2$. If D-$\mathsf{runs}_1 = D$-runs_2, and for all $\mathcal{D} \in D$-runs_i, $J^1_{C \lhd D}(\mathcal{D}) = J^2_{C \lhd D}(\mathcal{D})$, then:*

1. *$\mathcal{F}_1$ ϕ-limits C-to-D flow iff $\mathcal{F}_1$ ϕ-limits C-to-D flow;*
2. *$\mathcal{F}_1$ answers $\mathcal{Q}$ for D iff $\mathcal{F}_2$ answers $\mathcal{Q}$ for D.*

This follows directly from the forms of the definitions.

However, it is useful. For instance, it immediately follows that the properties of the system are preserved in case the system serves more than one analyst. In Fig. 5, we present an augmented system containing multiple analysts. However, since the behaviors on the interfaces $1, 2$ and 3 are unaffected, Thm. 3 immediately entails that the augmented system continues to meet its goals for A_1. By symmetry, it meets the same goals for the other A_i.

As another example, the system we have described has no audit mechanism built in. However, having designed the system and established its information flow properties, we can add nodes and channels to perform audit without changing the local runs and compatibility functions for the interfaces $1, 2$ and 3. This

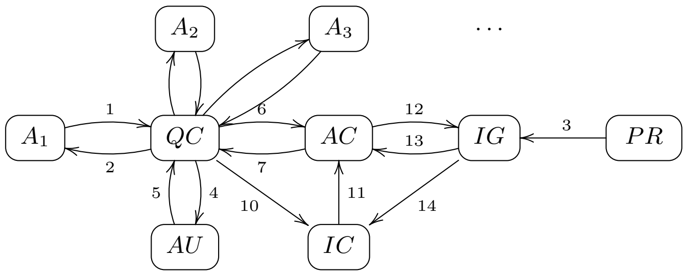

Fig. 5. EpiDB augmented with multiple analysts. Channels 8, 9 omitted as clutter

provides a clear argument for orthogonality of design that has sometimes eluded secure systems methodology.

6 Conclusion

We have discussed the frame model, and illustrated how to use it to establish *what*-declassification policies, or information flow upper bounds. The same ideas lead to a natural approach for showing lower bounds, i.e. that a system really answers questions which may be posed on one of these interfaces.

However, the frame model gives an abstraction of a possible system: How can one determine that an actual system displays the structure and behavior of a given frame as designed? In particular, two central items are needed. First, the active components of the actual system should correlate with the nodes of the frame. The behaviors of each component should conform to the LTS of the correlated node. Second, the message-passing activity of the system should occur along channels identified in the frame. There should be no other interactions, whether between components of the system or between components and the external world.

Similarly, to build a real system using a frame as specification, one needs, first, a way to build local programs that conform to an LTS specification, and various familiar ideas such as reactive programming and event-handling libraries appear helpful. In any case, the programming here is purely sequential and independent of any shared state interactions.

How then to establish, second, that the components interact with each other, and only with each other, as specified in the graph? This requires cryptographic support, both for secrecy to ensure that messages between components canot leak to the external world, and for authenticity to ensure that a component cannot receive a message off a channel unless its peer transmitted onto the channel. A protocol is needed also to ensure that message passing approximates the synchronous semantics the model uses.

Indeed, there is an additional role for cryptography, which is to provide attestation, i.e. digitally signed evidence that a node is genuine and under the control of the expected code. The Trusted Platform Modules were intended as an anchor for this sort of evidence, and user-level trusted execution environments (TEEs) such as Intel's Software Guard Extensions provide a simpler framework for achieving attestations [5]. TEEs provide symmetric cryptographic support to protect a thread and local memory, encrypting pages as they leave the processor's cache. Moreover, the processor provides digital signatures that attest to the code in control of the TEE. These attestations allow components to validate one another, to ensure that they are affiliated in the pattern stipulated in their model. The attestations also allow an external party to decide to believe this also, before making a decision as to whether to deliver data into the system, or accept it from the system. Thus, in addition to hardware support, we need to be able to use cryptographic protocols in the right way; another area in which A. W. Roscoe has also made his contributions.

Acknowledgments. I am grateful to Paul D. Rowe and John D. Ramsdell, with whom I discussed many of these ideas. In particular, John Ramsdell worked out the successive frame versions summarized in the figures.

References

1. Riccardo Focardi and Roberto Gorrieri. The compositional security checker: A tool for the verification of information flow security properties. *IEEE Transactions on Software Engineering*, 23(9), September 1997.
2. Riccardo Focardi and Roberto Gorrieri. Classification of security properties. In *Foundations of Security Analysis and Design*, pages 331–396. Springer, 2001.
3. Joseph A. Goguen and José Meseguer. Security policies and security models. In *IEEE Symposium on Security and Privacy*, 1982.
4. Joshua D. Guttman and Paul D. Rowe. A cut principle for information flow. In *IEEE Computer Security Foundations*. IEEE Computer Society Press, July 2015.
5. Intel. Intel Software Guard Extensions (Intel SGX). `https://software.intel.com/en-us/sgx`, 2016.
6. Jeremy Jacob. Security specifications. In *IEEE Symp. Security and Privacy*, pages 14–23. IEEE Computer Society, 1988.
7. Carroll Morgan. The shadow knows: Refinement of ignorance in sequential programs. In *Mathematics of program construction*, pages 359–378. Springer, 2006.
8. A. W. Roscoe. CSP and determinism in security modelling. In *IEEE Security and Privacy*, pages 114–127. IEEE, 1995.
9. A. W. Roscoe. *The Theory and Practice of Concurrency.* Prentice-Hall, 1997.
10. A. W. Roscoe and M. H. Goldsmith. What is intransitive noninterference? In *12th IEEE Computer Security Foundations Workshop*, pages 228–238. IEEE CS Press, June 1999.
11. A.W. Roscoe, J.C.P. Woodcock, and L. Wulf. Non-interference through determinism. *Journal of Computer Security*, pages 27–53, 1996.
12. John Rushby. *Noninterference, transitivity, and channel-control security policies.* SRI International, Computer Science Laboratory, 1992.
13. P. Y. A. Ryan. A CSP formulation of noninterference and unwinding. In *IEEE CSFW 3*, June 1990.
14. Andrei Sabelfeld and David Sands. Declassification: Dimensions and principles. *Journal of Computer Security*, 17(5):517–548, 2009.
15. David Sutherland. A model of information. In *9th National Computer Security Conference*. National Institute of Standards and Technology, 1986.