Managing Big Data with Information Flow Control

Thomas F. J.-M. Pasquier, Jatinder Singh and Jean Bacon
Computer Laboratory, University of Cambridge
Cambridge, United Kingdom
Email: firstname.lastname@cl.cam.ac.uk

Olivier Hermant
MINES ParisTech
Paris, France
Email: hermant@cri.ensmp.fr

***Abstract*—Concern about data leakage is holding back more widespread adoption of cloud computing by companies and public institutions alike. To address this, cloud tenants/applications are traditionally isolated in virtual machines or containers. But an emerging requirement is for cross-application sharing of data, for example, when cloud services form part of an IoT architecture. Information Flow Control (IFC) is ideally suited to achieving both isolation and data sharing as required. IFC enhances traditional Access Control by providing continuous, data-centric, cross-application, end-to-end control of data flows. However, large-scale data processing is a major requirement of cloud computing and is infeasible under standard IFC. We present a novel, enhanced IFC model that subsumes standard models. Our IFC model supports 'Big Data' processing, while retaining the simplicity of standard IFC and enabling more concise, accurate and maintainable expression of policy.**

Keywords*—*Information Flow Control; Data Management; Security;

I. Introduction

Concern about data leakage is holding back more widespread adoption of cloud computing by companies and public institutions. There is an increasing volume of applicable legislation and regulation [1], but ensuring and demonstrating compliance by cloud service providers and third parties is problematic. In recent work we have explored the use of Information Flow Control (IFC) for cloud and distributed computing, based on a proof-of-concept implementation (FlowK) of the standard IFC model as a basis for evaluation [2].

Based on this experience, we believe that the deployment of IFC to augment traditional authentication and authorisation has the potential to make a substantial contribution to the security of distributed and cloud systems, both through enforcement mechanisms and demonstration of compliance through audit. However, the use of IFC for large-scale data sharing and analytics is problematic using the standard IFC model. In this paper we present an enhanced IFC model which, while retaining the simplicity of expression and implementation of the standard model, easily extends to large scale.

Much work remains to be done, particularly when cloud services are incorporated as part of wide-scale distributed systems, as in the Internet of Things (IoT). In a cloud context, tenants/applications are traditionally isolated in virtual machines or containers. An emerging requirement is for cross-application sharing of data, particularly when cloud services are used for IoT. IFC is ideally suited to achieving both isolation and data sharing as required [3]. If IFC is incorporated into cloud service provision as part of PaaS or SaaS clouds, it can provide continuous, data-centric access control policy within and across applications, see §II.

Traditionally, (principal/role-specific) access control is applied before data can be accessed by an entity, after which no further control is exercised on where that data flows in the system. IFC augments access control so that data flows are monitored continuously, in context, to enforce more general policy. This is achieved by associating *labels* with data and the entities that process them. Labels comprise a number of *tags* that describe the nature and/or source of the data such as *healthcare, personal, government-information*, etc. Flows are permitted only if the labels match, see §III. The challenge in integrating multiple applications' access control policy with IFC is *to create labels that express policy accurately, concisely, naturally and efficiently, and in a way that makes policy changes easy to implement*. We believe the new tag design we present here contributes substantially to this goal.

We have previously investigated how labels can be used to enforce certain laws and regulations, such as "data originating in the EU must not leave its boundaries, except to certain Safe Harbors" [1], [4], [5]. A simple tag EU can be used for this purpose. But access control policies in general have great richness and complexity whereas IFC should be simple to express and enforce, to minimise runtime overhead while providing evidence (logs) for audit and demonstrating compliance. A major research question is "which aspects of access control policies need to be embodied in IFC tags for continuous, runtime, cross-application enforcement?" Essentially, problems arise when some software can access *all* data items of a certain kind, e.g., to perform analytics, generate statistics, anonymise or encrypt, as required for 'Big Data', and other software is restricted to access only *one such item*, see §II, §IV.

Here we propose two-component tags to represent the *concern* of data and a *specifier* for an item of that kind, for example $\langle medical, bob \rangle$ for Bob's medical record. We have implemented this in FlowK2 (**FlowK 2**-component). Such a tag model more closely reflects the policy maker's intent than previous models, and current atomic tags can easily be expressed in this way, such as $\langle location, EU \rangle$ or $\langle regulation, EU111 \rangle$. We have experience of collaboration in healthcare record management [6], [7] and monitoring [8] and take examples from this domain to demonstrate this approach.

The main contribution of this paper is this enhancement of standard IFC tags to make IFC feasible for 'Big Data' processing in the cloud and to allow better expression and maintenance of policy. The ability to express Conflict of Interest (CoI) is new in FlowK (§III-E) and CoI policy is more powerful, concisely expressed and capable of maintenance in FlowK2 (§V-C). We have presented a performance evaluation of FlowK elsewhere [2] and here focus on a comparison of the new tag model with standard IFC.

§II outlines IFC models and application level policy expression. §III presents the standard atomic tag model. §IV gives some motivating examples, showing how two-component tags solve problems of large-scale data processing under IFC. §V presents the two-component tag model, extending that of FlowK. §VI shows that the modifications can easily be implemented and give comparable or improved performance. §VII concludes.

II. Background and Related Work

IFC tags are metadata that 'stick' to data, allowing every flow to be controlled. Although sticky policies [9] are ostensibly similar, they are machine-readable, high-level policies applied at domain boundaries and protected cryptographically. Our aim is for brevity and simplicity, at fine granularity, in a kernel-level, runtime, policy-enforcement mechanism.

We first introduce IFC, with an overview of IFC implementations. We then highlight the importance of what IFC potentially delivers when offered as part of cloud service provision. Finally we consider the well established area of authorisation policy as a basis for establishing which aspects should be carried forward into IFC tags for runtime enforcement.

A. IFC Models & Implementations

Motivated by the deficiencies in standard security techniques, such as firewalls and access control mechanisms, in 1976, Denning [10] proposed a Mandatory Access Control (MAC) model to track and enforce rules on information flow in computer systems. In this model, entities are associated with security classes. The flow of information from an entity a to an entity b is allowed only if the security class of b (denoted $\underline{b}$) is equal to or higher than $\underline{a}$. This allows the *no-read up, no-write down* principle of Bell and LaPadula [11] to be implemented to enforce secrecy. By this means a traditional military classification *public, secret, top secret* can be implemented. A second security class can be associated with each entity to track and enforce integrity (quality of data) during *reading down* and *writing up*, as proposed by Biba [12]. A current example might distinguish information from a government website in the $.gov.uk$ domain from that from "Joe's Blog". Using this model allows one to control and monitor information flow to ensure data secrecy and integrity.

In 1997 Myers [13] introduced a decentralised IFC model (DIFC) that has inspired most later work. This model was designed to meet the changing needs of systems from global, static, hierarchical security levels to a more fluid system, able to capture the needs of different applications. Each entity has two labels: a *secrecy* label and an *integrity* label, to capture respectively the privacy/confidentiality of the data and the reliability of a source of data. Each label comprises a *set of tags*, each of which represents some security concern. Data is allowed to flow if the security label of the sender is a subset of the label of the receiver, and conversely for integrity.

We describe in §III the model we use in FlowK that follows this general idea. Details of implementations of IFC at language [14], language library and operating system level are given in [2], [15]. She et al. [16], use run-time dependency and information flow control to track information within and between services in a cloud service composition chain. Roy et al. [17] and Akoush et al. [18] propose to implement IFC in a MapReduce framework to prevent the computation provider leaking data from the data provider. Airavat [17] is limited in the expressiveness of the policy due to its reliance on SELinux, as acknowledged by the author "While [DIFC] would provide far greater flexibility [...] only prototype DIFC operating systems exist". MrLazy [18] only verifies IFC policy at sink-points, forcing the entire data-processing to be re-done or data discarded. Finally, Xie et al. [19] propose IFC to secure stream processing in the cloud, to prevent data disclosure between competing organisations, in a Chinese Wall policy. Our model, see §III, achieves this through Conflict of Interest groups. In this paper we focus on using IFC for flexible expression and enforcement of application level policy.

B. The need for IFC in Cloud Computing

IFC controls the use of data beyond applications' access control points, throughout its whole life cycle. Therefore users and applications can safely share and exchange data, if they are willing to trust the cloud provider as a policy-enforcing third-party. This is not unreasonable, given the existing contractual relationships between tenants and cloud providers and the role of regulators that operate in many jurisdictions. IFC allows the nature of the shared data usage to be defined precisely and enforced [20] (e.g. data labelled as medical can only flow to applications that have been authorised and labelled to manipulate medical data). The trust relationship is greatly simplified as only a common, well-defined party needs to be trusted (i.e. the cloud provider). Such an approach potentially leads to more data sharing and the emergence of cross-silo innovative applications.

Our IFC design is such that application instances need not be aware that their execution is controlled by IFC. We built a framework for web service provision [2] where an application manager is privileged to create a *security context* for each application instance, see §III. This involves creating tags and assigning labels to application instances.

IFC enforcement takes place on every system call and inter-machine data exchange. It is therefore a natural place for an audit log to be created; moreover, such a log is in terms of applications' data flows rather than low-level system concerns. Such an audit log can be used by cloud providers to demonstrate compliance with laws, regulations and contracts. It can also be used to show whether a claimed data leak did in fact occur. Such logs may be considered 'Big Data' and tools are needed to process them efficiently. Provenance systems already gather such logs, e.g. [18].

Authorisation policy tends to be application-specific. When data is used by more than one application, policy differences and conflicts can occur. When data is labelled for IFC, access policy is enforced across all entities that use the data. As data sharing becomes more pervasive, for example when cloud services are incorporated into IoT applications, system-wide, end-to-end policy enforcement will be crucial.

IFC does not have cybersecurity as its main focus but it strictly contains data flows and detects attempted illegal flows, so can contribute to early detection and/or damage limitation.

C. Role Based Access Control

RBAC is a mature, widely used access control scheme with a large literature and existing standards [21], [22].[1] Role definitions in RBAC tend to be functional in their scope, being

[1] http://csrc.nist.gov/groups/SNS/rbac/

application- or organisation-specific. Administrative roles are also included to capture the need to manage RBAC itself.

In work on RBAC by ourselves [7] and others [23], [24], parametrised roles were found to provide elegant expression of policy and avoid explosion in the number of roles required. For example, certain company software such as "Payroll" may need to access all employees' data whereas each employee can access only their own data record. To achieve this, a company either creates a role per employee e.g. *employee_smith* or parametrises a single role, for example *employee*(*smith*) etc. The Payroll software can then access *employee*(*), where * indicates all employees. In this paper we argue that the IFC label model needs similar refinement in order to carry forward to runtime such aspects of application policy, thus following the Principle of Least Privilege (PoLP).

Role parametrisation allows relationships between parameters and exclusions to be checked. For example the role *treating_doctor*(*doctorID, patientID*) allows the role to express the dynamic set of patients for each doctor for controlling access to records. An exclusion is expressed as a rule on a parameter value to indicate, for example, "all doctors can access my records except *doctorID*", see [7]. In §IV we argue that such checks are most appropriately carried out at authorisation points within the application. Environmental/context checks such as "A Pharmacist can only authorise the issuing of drugs while on duty in the Pharmacy" can be enforced continuously by IFC using appropriate label design.

III. Atomic Tag Model

IFC augments authorisation by enforcing dynamically that only permitted flows of information can occur, end-to-end, across applications. *Entities* to which IFC constraints are applied include cloud web applications [2], a web worker instance [18], a file, a database entry [25], etc. IFC is applied continuously, typically on every system call for an IFC-enabled OS. IFC policy should therefore be as simple as possible, to allow verification, human understanding and to minimise runtime overhead.

A. Enforcing Safe Flows via Labels

A tag within a label's set of tags represents a particular security concern for a category of data. In our IFC model two labels are associated with every entity A: a *secrecy label* $S(A)$ and an *integrity label* $I(A)$. The current state of these two labels (sets of tags) is the *security context* of an entity.

A flow of information from an entity A to an entity B, denoted $A \to B$, is allowed if the following rules are respected:

$$A \to B, \text{ iff } S(A) \precsim S(B) \wedge I(B) \precsim I(A) \qquad (1)$$

where $\precsim$ is any preorder (here it is mere inclusion $\subseteq$, it will be refined for our two-component tag model in §V). These checks are simple to understand and apply, involving only matching of the tags at the communication endpoints.

Consider the *read* and *write* functions of the Bell-LaPadula model [11] and the Biba model [12]. In the IFC world *read* is the equivalent of an incoming flow and *write* is the equivalent of an outgoing flow. The subrule concerning secrecy labels ensures that an entity only passes information to an entity that is allowed to receive it, thus enforcing the "no read up, no write down" policy of the Bell-LaPadula model. The subrule concerning integrity labels enforces quality of data during reading down and writing up, as proposed by Biba [12]. It is therefore possible to represent traditional security requirements as IFC constraints, although we use labels to represent more general security contexts, e.g., integrity can also indicate authority, such as to send an actuation command to a vehicle or home automation device.

Example – secrecy: Suppose a hospital patient Bob, on being discharged to his home, is issued with a heart monitor. Data from this device is stored in his home system and also flows to a process in the hospital's system, which carries out an analysis of his condition. Because Bob's health data is private, the heart monitor and data are labelled with $S = \{medical, bob\}$. In order to receive this data, the hospital process's S label must also include the tags $medical$ and bob.

Example – integrity: The hospital process is only allowed to receive data from a hospital-issued device and to achieve this is labelled $I = \{$*hospital-issued*$\}$. In order to send data to the hospital process, Bob's device and data must also be labelled $I = \{$*hospital-issued*$\}$. Suppose Bob's heart monitor is capable of remote actuation, e.g. to change the sampling rate if analysis detects a possible health problem. The device must only accept actuation commands from authorised sources, e.g. also labelled $I = \{$*hospital-issued*$\}$.

B. Creation of an Entity

We define $A \Rightarrow B$ as the operation of the entity A creating the entity B. We have the following rules for creation:

$$\text{if } A \Rightarrow B, \text{ then } S(B) := S(A) \text{ and } I(B) := I(A) \qquad (2)$$

That is, the created entity inherits the labels of its creator. Examples are creating a process in a Unix-style OS by fork and creating passive data such as files or messages.

C. Privileges for Managing Tags and Labels

Certain active entities (e.g. an application manager) have privileges that allow them to modify their labels. An entity has two sets of privileges for removing tags from its secrecy and integrity labels (P_S^- for S and P_I^- for I), and two sets for adding tags to these labels (P_S^+ for S and P_I^+ for I). That is, for an entity A to remove the tag $t_s \in S(A)$, it is necessary that $t_s \in P_S^-(A)$, similarly to add the tag t_i to the label $I(A)$ it is necessary that $t_i \in P_I^+(A)$.

For an entity A, a label $X(A)$ (where X is S or I) and a tag t, a change of the label is authorised if the following rule is respected:

$$\begin{array}{lcl} X(A) & := & X(A) \cup \{t\} \text{ if } t \in P_X^+(A) \quad \text{or} \\ X(A) & := & X(A) \setminus \{t\} \text{ if } t \in P_X^-(A) \end{array} \qquad (3)$$

For example, in order to receive information from an entity B, an entity A will need to set its labels (if it has the privilege) such that the flow constraints expressed by the tags associated with B are respected; i.e. such that the flow $B \to A$ respects the safe flow subrules in rule (1). We propose the following notation: for a process and its labels $(A, S, I) \rightsquigarrow (A, S', I')$ is the modification of the process labels following rule (3).

Only privileged processes can change their security context. We envisage application instances running within a constant security context and being unaware of IFC [2]. A privileged application manager creates these instances, and privileged processes are invoked transparently.

Example – declassification: Suppose the hospital that receives health monitoring data from its patients at home wishes to make this data available for research on the efficacy of home monitoring. Before releasing such a data set it must be anonymised, i.e. individual patients must not be identifiable from the data. A process that carries out the anonymisation must have the privilege to read the private data before anonymisation; i.e. the input data set and the process may be labelled $S = \{medical, private\}$. After anonymisation, the process must *reclassify* the data with label $S = \{medical, anonymised\}$. The anonymisation process must therefore have the privilege to remove the tag (declassify) $private$ from its S label and add the tag $anonymised$ before outputting the data.

In the case where data needs to become more carefully controlled and less widely available, e.g., after decryption, the reverse process of **reclassification** would involve, e.g. adding a tag $private$.

Example – endorsement: Endorsement usually involves adding a tag to an I label. For example, a process might receive data from the network, carry out a verification process then output the data with tag $valid\_data$. Such a process may be involved in data format conversion if non-standard data came from a remote source. A similar endorsement process can be used for many kinds of input data such as PhP scripts, downloaded software, indeed, any input amenable to a validation process.

In our hospital example, the patient may have received some treatment in another hospital or clinic and have a treatment record there, where the data format may differ. A process is charged with checking the patient's identity and verifying the data, including reformatting. The data is then output with tag $valid\_data$ and can be safely processed within the hospital domain.

Integrity tags may need to be removed after an anonymisation process, as the quality of data may have been degraded by the process. For example, if detailed information is removed, the data may no longer be proper to use as the source of an actuation command.

D. Creation and Privileges

On creation, labels are automatically inherited by a created entity from its creator (rule 2), but privileges are not. If the child is to be given privileges over its labels, they must be passed explicitly. We denote the flow generated by an entity A giving selected privileges $t_X^{\pm}$ to an entity B as $A \overset{t_X^{\pm}}{\hookrightarrow} B$ (for example allowing t to be removed from S, would be denoted $A \overset{t_S^-}{\hookrightarrow} B$). In order for a process to delegate a privilege to another process it must own this privilege itself. That is,

$$A \overset{t_X^{\pm}}{\hookrightarrow} B \text{ only if } t \in P_X^{\pm}(A) \tag{4}$$

E. Conflict-of-Interest (CoI)

A policy maker may need to specify a CoI (or Separation of Duty) between principals and/or roles [26], [27]. A CoI may arise when a principal could give professional advice to a number of competing companies. Separation of data access may be enforced by a Chinese Wall policy [27]

We believe CoI support in IFC is unique to FlowK. We define a set C of tags that represents some specified conflicting interests. In order for the configuration of an entity A to be valid with respect to C, rule (5) must be respected:

$$\left|\left(S(A) \cup I(A) \cup P_S^+(A) \cup P_I^+(A) \cup P_S^-(A) \cup P_I^-(A)\right) \cap C\right| \leq 1 \tag{5}$$

That is, an entity is non-conflicting in this context if the set of its potential tags (past, present and future) contains at most one element from the set of tags within the related CoI group. In detail, by potential tags we mean the tags in its current S and I labels and those tags that it has the privilege to add to $S(A)$ (i.e. $P_S^+(A)$) and to $I(A)$ (i.e. $P_I^+(A)$) or that it may have removed from $S(A)$ (i.e. $P_S^-(A)$) and from $I(A)$ (i.e. $P_I^-(A)$). CoI rules should be checked every time a privilege is granted.

Example – Conflict-of-Interest: A CoI can arise when data relating to competing companies is available in a system. In a hospital context, this might involve results of analyses of the usage and effects of drugs from competing pharmaceutical companies. The companies might agree to analysis only if their data is guaranteed to be isolated, i.e. not leaked to competitors.

The hospital may be participating in drug trials and want to ensure that information does not leak between trials: suppose a conflict is $C = \{Pfizer, GSK, Roche, ...\}$ and some data (e.g. files) are labelled $PfizerData[S = \{Pfizer\}, I = \emptyset]$ and $RocheData[S = \{Roche\}, I = \emptyset]$. The CoI described ensures that it is not possible for a single entity (e.g. an application instance) to have access to both $RocheData$ and $PfizerData$ either simultaneously or sequentially, i.e. enforcing that Roche-owned data and Pfizer-owned data are processed in isolation.

IV. Motivating Use Cases

In this section we motivate the use of two-component tags in FlowK2's label model. Our model is designed for a distributed system or cloud platform where there is likely to be a large amount of user data stored with persistent labels in files, databases, key-value stores, etc. In a company context, data records may relate to individual employees; in a public health context, data may represent the medical records of patients; in an educational context, data may relate to students, staff etc. Specifying and enforcing access to all, some specified subgroup or only one data record of a given type is a universal requirement, discussed in the literature on policy.

Suppose a principal is allowed access to a subset of records, e.g. doctors may be able to access only the records of the patients they are currently treating. Temporally separated processes are likely, i.e. to deal with one patient's records at a time. Each time, current authorisation policy is enforced and is translated into labels to ensure correct behaviour at runtime. Note that as a doctor's group of patients under treatment changes, a lookup of current patients at the authorisation point in the application will ensure that labels are created only for current patients, selected at runtime from the entire database.

The first use case below arises from the need of certain software to perform computations on all health records of a given type, whereas other software is authorised only to access records on behalf of a single individual. The second considers a system log containing records from all running applications relating to large numbers of principals. These problems are akin to the role proliferation that motivated role parametrisation in RBAC, as mentioned in §II.

We have solved these problems by making our tags more

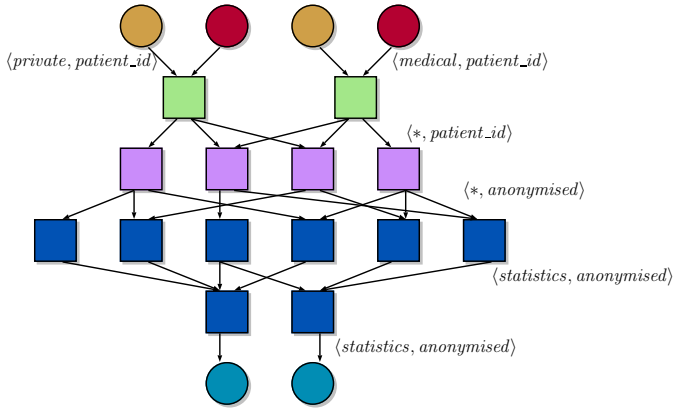


Figure 1: 'Big data' Directed Graph Computation (as in Dryad [28]) showing secrecy labels.

expressive to reflect the intended policy more accurately. Also, this new tag structure correctly represents the trust that should be placed in certain entities, according to PoLP. We use two-component tags, where a tag is of the form $\langle concern, specifier\rangle$. We use $\langle concern, *\rangle$ to indicate all tags of the specified concern and $\langle *, specifier\rangle$ for tags of all concerns with the given specifier. $\langle *, *\rangle$ then indicates all tags in some naming domain. In §V we present this label model more formally.

A. Data Analysis

In the healthcare domain, statistical analysis of the medical records of patients is needed for various purposes including public health, environmental concerns, clinical practice etc. We are concerned with the privacy and confidentiality of medical data and will therefore discuss the construction of the IFC secrecy label for a statistical analysis program.

In the standard atomic tag model a tag represents a single security concern. To express the idea of Bob's medical data, we would use two tags bob_data and $medical\_data$. The entity carrying out the statistical analysis of the medical data would then need to have not only the $medical\_data$ tag, but also a tag corresponding to every patient's data, for rule (1) to be satisfied. This makes the use of IFC infeasible for such purposes: (1) The performance implications are significant. We have shown [2] that the processing overhead induced by the number of tags in labels is not insignificant (especially at such a scale). (2) Enumerating "all" tags would be prone to error as the database state changes, with records being added and removed. (3) This entity would be over-privileged by the PoLP, being able to receive any data labelled only with the tag bob_data although our intention was for it only to be concerned with $medical\_data$. It would also be privileged to declassify, see §III-C, over all the patients' personal tags and trusted not to leak information about any patient. An alternative would be to define different tags for every category of user data such as $bob\_medical\_data$, bob_tax_data, bob_home_data and many more. The performance issues caused by huge numbers of tags would still hold. Also, an entity running on behalf of Bob could not simply have a label with a single tag bob_data but would have to enumerate all the particular types of Bob's data it could process.

The problems are solved by using two-component tags, where a tag is of the form $\langle concern, specifier\rangle$. In Fig. 1, we illustrate how such IFC tags can be used to constrain the flow of data within a 'big data' computation graph. The first computation phase aggregates medical and private information belonging to individual patients. The second phase anonymises these aggregates and the third generates statistical data from the anonymised data.

In detail, suppose each input record has an S label containing a tag such as $\langle medical, patient\_id\rangle$ or $\langle private, patient\_id\rangle$. We express that the first phase processing components can access all data belonging to a patient by including the tag $\langle *, patient\_id\rangle$ in their S labels, where $*$ indicates all records with specifier $patient\_id$. These components output equivalently labelled data (§III rule (2)) which is input to the second phase components (also labelled $\langle *, patient\_id\rangle$) for anonymisation. The anonymisers must change their security context (see §III-C) in order to output data labelled $\langle *, anonymised\rangle$.

The third phase processes (also labelled $\langle *, anonymised\rangle$) input the anonymised data and carry out statistical analysis. These processes must change their security context in order to output data labelled $\langle statistics, anonymised\rangle$. Finally, the anonymised statistical data is used, the results being output without need of a further security context change.

By specifying for each task the expected input and output *security context* and enforcing them in the operating system and across machines, we are able to guarantee that no data leakage can occur between concerns (e.g. from private to medical) or between specifiers (i.e. patients).

B. Log Audit

Consider a system log comprising labelled records that several active entities are able to access for different purposes. A process concerned with digital forensics requires access to the whole log to detect and investigate suspicious patterns of behaviour. Applications, e.g. cloud tenants, and individual users should be able to audit the use of their own data.

With the standard atomic tag model an entity performing audit over all log records would need access to all relevant concerns and specifiers: in the two-component tag model $\langle *, *\rangle$. It would need declassification privileges over all those data for the audit result to be readable as required, see §V. An entity performing audit over data logs for specified applications would need $\langle application\_name, *\rangle$ and again, the privilege to declassify over all specifiers. The tag needed by the entity performing audit on behalf of an individual over all their data (from any application) would be $\langle *, person\_name\rangle$. The entity would need the privilege to declassify the output, see §V.

In this section we have given the intuition and motivating examples for the two-component tag model of FlowK2. In the next section we present the formal notation, extending the standard atomic tag model.

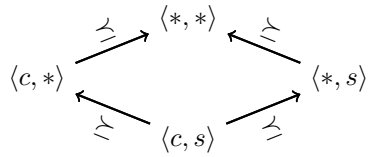
V. Two-Component Tag Model

A major aim for FlowK (our prototype cloud-IFC implementation using atomic tags [2]) was simplicity, for ease of understanding and verification, and efficient enforcement. This section presents the formal notation for the two-component tag model (implemented in FlowK2), as motivated in §IV. We will show that policy, particularly for large-scale data processing, is expressed more concisely using the FlowK2 tag model and is easier to maintain. Also, tag-checking overhead

is comparable for small-scale processing and is reduced for large-scale processing.

We propose to decompose a tag t into a pair $\langle c, s\rangle$ with c the concern of type $\mathscr{C}$ and s a specifier of type $\mathscr{S}$. For example, the pair $\langle medical, bob\rangle$ represent Bob's medical data. A statistical analysis over a set of patients' medical data is represented as $\langle medical, statistical\_analysis\rangle$ and anonymised medical records as $\langle medical, anonymised\rangle$.

As we saw in §IV, a major requirement is to be able to specify *all* data records of a certain kind without enumerating all possible tags, as required by current models. Therefore for any concern c and specifier s we establish the following subtyping relation:



That is, a tag $t = \langle c, s\rangle$ is a subtype of $t' = \langle c, *\rangle$ and $t'' = \langle *, s\rangle$ which are themselves subtypes of $t''' = \langle *, *\rangle$. For instance, $\langle medical, bob\rangle$ (Bob's medical data) is a subtype of $\langle medical, *\rangle$ (medical data) and a subtype of $\langle *, bob\rangle$ (Bob's data) which are each subtypes of $\langle *, *\rangle$ (all data in the current naming domain).

A. Changes to Flow Constraints

To adapt rule (1) from §III-A for the flow $A \rightarrow B$, we need only redefine the $\precsim$ binary relation between sets of tags X and Y as follows:

$$X \precsim Y \text{ iff } \forall t \in X\ \exists t' \in Y : t \preceq t' \tag{6}$$

Together with rule (1), this entails that a flow $A \rightarrow B$ is allowed if and only if for all secrecy tags of A there exists a supertype in the secrecy tags of B and that for all integrity tags of B there exists a supertype in the integrity tags of A.

Example – secrecy: The examples illustrating the atomic tag model given in §III, relate to hospital patients being monitored in their homes. We saw that the process receiving Bob's heart rate data, labelled with $S = \{medical, bob\}$, also needed to have the tags $medical$ and bob in its S label. This process would need a tag for every such patient ($S = \{medical, alice, bob, charlie, donald, etc....\}$) and the (large) set of patients' tags would need to be kept consistent with the current set of home-monitored patients. Instead, in FlowK2, we propose to label the process $S = \{\langle medical, *\rangle\}$. This expresses the intended policy concisely and accurately, without the need to maintain the current set of patients' tags. In the implementation, the tag matching overhead on every system call is linear with the number of tags [2]. This overhead is reduced by using the single 2D tag $\langle medical, *\rangle$.

Example – integrity: Consider a home control (domotic) system. An entity A labelled $I(A) = \{\langle actuator, *\rangle\}$ is able to send data (an actuation instruction) to an entity B labelled $I(B) = \{\langle actuator, alarm\rangle\}$ or an entity C labelled $I(C) = \{\langle actuator, light\rangle\}$. The entity A could represent the central domotic control system with B and C being actuators in the house. In a patient monitoring context the controller might be sending actuation commands to a variety of patient monitoring devices, e.g., to start, stop or change the monitoring intervals or thresholds. Since actuation affects the physical world, including people's health and safety, it is important that the authority of the actuation command is established.

B. Changes to Privileges

Rule (3) from §III becomes (where X is S or I):

$$\begin{array}{rcl} X(A) & := & X(A) \cup \{t\} \text{ if } \exists t' \in P_X^+(A) : t \preceq t' \quad \text{or} \\ X(A) & := & X(A) \setminus \{t\} \text{ if } \exists t' \in P_X^-(A) : t \preceq t' \end{array} \tag{7}$$

We also add special privileges noted $\langle c, \Delta\rangle$, $\langle \Delta, s\rangle$ and $\langle \Delta, \Delta\rangle$ that allow removal only of the tags $\langle c, *\rangle$, $\langle *, s\rangle$ and $\langle *, *\rangle$ respectively.

The privilege delegation rule (4), becomes:

$$A \overset{t_X^{\pm}}{\hookrightarrow} B \text{ only if } \exists t' \in P_X^{\pm}(A) : t \preceq t'$$

Example – declassification: A process A, with the privilege $P_S^-(A) = \{\langle medical, \Delta\rangle\}$ and the label $S(A) = \{\langle medical, *\rangle, \langle medical, anonymised\rangle\}$ is able to declassify to $S(A) = \{\langle medical, anonymised\rangle\}$, but does not have the privilege to remove $\langle medical, anonymised\rangle$. The use of Δ privileges therefore allows the trust placed in a certain entity to be precise and is particularly useful when specifying declassifier privileges. Without it, we would have had only $P_S^-(A) = \{\langle medical, *\rangle\}$ and no guarantee that the process would not declassify to $S(A) = \emptyset$ (also removing $\langle medical, anonymised\rangle$), thus allowing universal access to the anonymised data, rather than to medical research processes with the tag $\langle medical, anonymised\rangle$.

Examples – integrity label changes: A process A, with the privilege $P_S^-(A) = \{\langle actuator, \Delta\rangle\}$ and the label $I(A) = \{\langle actuator, *\rangle, \langle actuator, alarm\rangle\}$ is able to remove the tag $\langle actuator, *\rangle$ but not the tag $\langle actuator, alarm\rangle$. A process A, with the privilege $P_S^-(A) = \{\langle local, \Delta\rangle\}$ and the label $I(A) = \{\langle network, *\rangle, \langle local, *\rangle\}$ is able to remove the tag $\langle local, *\rangle$, after endorsing local input but not the tag $\langle network, *\rangle$.

C. Changes to Conflicts of Interest

There are now three types of policy we must express: constraints applied to whole tags, to concerns and to specifiers. We define three operations on a tag's pair, the projections π_1 in $\mathscr{C}$, π_2 in $\mathscr{S}$ and the identity function id:

$$\begin{array}{rcl|rcl} \pi_1 : \mathscr{C}\times\mathscr{S} & \rightarrow & \mathscr{C} & \pi_2 : \mathscr{C}\times\mathscr{S} & \rightarrow & \mathscr{S} \\ \pi_1(\langle c, s\rangle) & = & c & \pi_2(\langle c, s\rangle) & = & s \end{array} \tag{8}$$

We extend these operations to sets, such that:

$$\begin{array}{rcl|rcl} \pi_1 & : & \wp(\mathscr{C}\times\mathscr{S}) \rightarrow \wp(\mathscr{C}) & \pi_2 & : & \wp(\mathscr{C}\times\mathscr{S}) \rightarrow \wp(\mathscr{S}) \\ \pi_1(T) & = & \{\pi_1(t) \mid t \in T\} & \pi_2(T) & = & \{\pi_2(t) \mid t \in T\} \\ & = & \{c \mid \langle c, s\rangle \in T\} & & = & \{s \mid \langle c, s\rangle \in T\} \end{array} \tag{9}$$

For an entity A we note the union of its labels and privileges:

$$SU(A) = S(A) \cup I(A) \cup P_S^+(A) \cup P_S^-(A) \cup P_I^+(A) \cup P_I^-(A) \tag{10}$$

A conflict of interest is denoted $P_{CoI}(f, C)$ where f is π_1, π_2 or id and C is a set of conflicting tags. Rule 5 in §III-E becomes:

$$\forall P_{CoI}(f, C), |f(SU(A)) \cap C| \leq 1 \tag{11}$$

Here we note:

- $\{a, b, c\} \cap \{*\} = \{a, b, c\}$;
- $\{\langle a, b\rangle, \langle a, d\rangle, \langle c, d\rangle\} \cap \{\langle a, *\rangle\} = \{\langle a, b\rangle, \langle a, d\rangle\}$;
- $|\{*\}| = \infty$, $|\{a, *\}| = \infty$ and $|\{*, a\}| = \infty$.

The conflict of interest rule: $P_{CoI}(\pi_1, \{medical, private\})$ means an entity can handle the concern medical or private but not both. $P_{CoI}(id, \{\langle private, *\rangle\})$ means an entity can only ever manipulate the private data of a single user.

Example – conflict of interest: Consider the example used in §III-E on isolating application instances that access drug trial data for different companies. If a new company was to use the application, a new tag would need to be added to the CoI group. For a rule applying to a more rapidly changing set this could prove problematic.

Using the two-component model, we have application instances labelled: $[S = \{\langle drug, Roche\rangle\}, I = \emptyset]$, $[S = \{\langle drug, Pfizer\rangle\}, I = \emptyset]$ etc. and the CoI policy is expressed as: $P_{CoI}(id, \{\langle drug, *\rangle\})$. This is simple to read and understand (i.e. an application instance can manipulate information for only one specifier of concern *drug*) and this policy will not change over time as companies come and go.

D. Compatibility with Atomic Tags

Some tags function adequately as atomic tags, e.g., $\langle$*network-input*$\rangle$ may be included in an integrity label to indicate that input data should not be trusted. The tags $\langle$*EU-data*$\rangle$ and $\langle$*US-data*$\rangle$ can be used to enforce the geographical location of stored data to allow laws and regulations to be enforced. Such tags do not need to be two-component tags but can conveniently be expressed as such, e.g. $\langle input, network\rangle$ or $\langle location, EU\rangle$. Policy may sometimes be conveniently expressed for such tags using $*$, but if $*$ is never used, our two-component tag model degrades gracefully to what is effectively a conventional atomic tag model since both components must match for data to flow, as for two separate tags. Backwards compatibility with policies defined for atomic tags, could be achieved e.g. by a convention that the single tag should become a specifier of a null concern in a two-component tag.

E. Is More General Tag Complexity Needed?

A major research question (§I) is "which aspects of access control policies need to be embodied in IFC tags for continuous, runtime, cross-application enforcement?" It is possible to design tags that capture every aspect of e.g. parametrised RBAC with environmental constraints [7]. But for IFC to be deployed in practice it needs to be simple to understand and evaluate and to impose minimal overheads due to management and enforcement. Further work is needed on IFC label design for specific (cross-application) use cases, but our experience to date is that few tags are needed to capture legal/regulatory and compliance requirements at runtime [4], [29] and many checks need only be done at authorisation points in applications. In this work we have shown the need to generalise tags for a specific style of application that is central to cloud service provision, i.e. large-scale data analytics.

VI. Evaluation

Our IFC platform comprises a local OS IFC enforcement system [2] and a messaging middleware for inter-process communication [30]. Above these components it is possible to build a PaaS (and by extension SaaS) platform as discussed in [2], [3]. Performance evaluation of FlowK was carried out on a quad-core 2.2Ghz Intel i7 with 6GiB of RAM running Fedora 20 (kernel version 3.14). Our aim here is strictly to compare the impact on performance of the two-component tag model compared with atomic tags and not to evaluate a specific implementation, already available in [2].

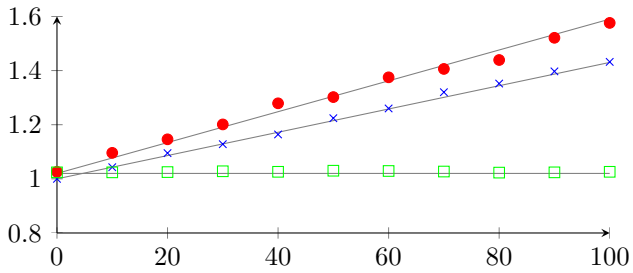


Figure 2: Performance comparison, normalised over FlowK with label size 0, for unmonitored and 0-100 tags: FlowK (cross), FlowK2 compatibility mode (circle), FlowK2 * policy (square).

Suppose a data analysis worker needs to be given access to the data belonging to several users (see §IV). We assume each user's data is uniquely labelled. To run under IFC, the number of tags needed by the analyser increases linearly with the number of users.

First, we run the atomic tag model's IFC constraints algorithm over an increasing number of secrecy tags: users would have data labelled $S = \{medical, alice\}$, $S = \{medical, bob\}$ etc. and the analysing process $S = \{medical, alice, bob, ...\}$. Secondly, we run the two-component tag model in atomic tag compatibility mode (see §V-D); i.e. the medical tag is translated as $\langle\emptyset, medical\rangle$ and personal tags as e.g. $\langle\emptyset, alice\rangle$ etc. Therefore the analysing process is labelled $S = \{\langle\emptyset, medical\rangle\langle\emptyset, alice\rangle, \langle\emptyset, bob\rangle, ...\}$. Finally, we use the full power of our proposed two-component tag model. That is, the data to be analysed is labelled $S = \{\langle medical, alice\rangle\}$ etc., and the analysing process $S = \{\langle medical, *\rangle\}$.

The normalised overhead measured for each of the above is shown in Fig. 2. We take an analysis process with a label size varying from 0 to 100 tags, the data label is taken from the tags present in the process's label. We repeat the operation 100,000 times and average the result. We see clearly that when using the $\{*\}$ specifier in policy, the performance is not dependent on the number of users. This is extremely important in a cloud environment where the number of tags required to analyse user data could scale to millions.

As discussed previously and presented in [2], there is a cost to security interposition that is not affected by the complexity of the policy to be enforced. The worst case complexity of the atomic tag policy algorithm is $\mathcal{O}(n)$. The worst case complexity of our two-component tag model is $\mathcal{O}(n^2)$. However, we have seen that in scenarios when label complexity has an impact on performance (i.e. the number of tags n is high), the two-component tag model will most likely reduce the number of tags necessary to express the policy.

In practice, IFC would be deactivated or data aggressively declassified in order to perform such analyses when using an atomic tag model. Indeed, the management of such a policy for an existing list of user-associated tags would rapidly become impossible for services with a large user base, without considering performance issues. Our proposed model allows such

operations to remain controlled by IFC with good performance, simple policy management, and with control over where the output of such analyses flows.

VII. Conclusion and Future Work

The feasibility of IFC has been demonstrated in practical systems: in the database domain [25], [31], for web-applications [32], for cloud-service composition [16], within operating systems [2], [33], across distributed systems [30], [34], for PaaS [3], for networks [35] and for hypervisors [36]. It has been shown that, with careful design, IFC does not require changes to cloud tenant applications [2], [17], but would require some engineering effort by the cloud provider.

The benefit of providing IFC as part of cloud services is that IFC supports both application isolation and data sharing between applications as required. Cross-application data sharing, as opposed to strong isolation, is of increasing importance, particularly when cloud services form part of IoT architectures. As clouds become akin to utilities that provide computing services, proof of compliance with regulations and contracts will become necessary. IFC naturally provides audit at a meaningful data-specific level, rather than via system logs.

Previous IFC implementations have followed the standard atomic tag model which is simple to express and enforce, but is infeasible for use in large-scale data analysis, where the number of tags required to enforce individuals' privacy scales with the number of entities to be processed. For large-scale cloud data processing, only limited SELinux-style IFC [17] or taint tracking has been attempted [18] to date. In this paper, we have shown that a simple extension of the standard IFC model to two-component tags allows IFC to be used for large-scale data analysis, with equivalent (or improved) performance and enhanced policy expression and maintenance.

Acknowledgement

This work was supported by UK Engineering and Physical Sciences Research Council, grant EP/K011510. We acknowledge the support of Microsoft through the Microsoft Cloud Computing Research Centre.

References

[1] C. J. Millard, Ed., *Cloud Computing Law*. OUP, 2013.

[2] T. F. J.-M. Pasquier, J. Bacon, and D. Eyers, "FlowK: Information Flow Control for the Cloud," in *6th International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2014.

[3] T. F. J.-M. Pasquier, J. Singh, and J. Bacon, "Information Flow Control for Strong Protection with Flexible Sharing in PaaS," in *IC2E, International Workshop on Future of PaaS*. IEEE, 2015.

[4] T. Pasquier and J. Powles, "Expressing and Enforcing Location Requirements in the Cloud using Information Flow Control," in *IC2E International Workshop on Legal and Technical Issues in Cloud Computing (Claw'15)*. IEEE, 2015.

[5] K. Hon, C. Millard, C. Reed, J. Singh, I. Walden, and J. Crowcroft, "Policy, Legal and Regulatory Implications of a Europe-Only Cloud," Queen Mary University of London, School of Law, Tech. Rep., 2014, accessed: 30th April 2015. [Online]. Available: http://papers.ssrn.com/sol3/Delivery.cfm/SSRN_ID2527951_code1577160.pdf

[6] T. Pasquier, B. Shand, and J. Bacon, "Information Flow Control for a Medical Web Portal," in *e-Society 2013*. IADIS, 2013.

[7] J. Bacon, K. Moody, and W. Yao, "A Model of OASIS Role-based Access Control and its Support for Active Security," *ACM Transactions on Information and System Security (TISSEC)*, vol. 5, no. 4, pp. 492–540, 2002.

[8] J. Singh and J. Bacon, "On Middleware for Emerging Health Services," *Journal of Internet Services and Applications*, vol. 5, no. 6, pp. 1–34, 2014.

[9] S. Pearson and M. C. Mont, "Sticky Policies: An Approach for Managing Privacy across Multiple Parties," *Computer*, vol. 44, July 2011.

[10] D. E. Denning, "A lattice model of secure information flow," *Commun. ACM*, vol. 19, no. 5, pp. 236–243, 1976.

[11] D. E. Bell and L. J. LaPadula, "Secure Computer Systems: Mathematical Foundations and Model," The MITRE Corp., Bedford MA, Tech. Rep. M74-244, 1973.

[12] K. J. Biba, "Integrity Considerations for Secure Computer Systems," MITRE Corp., Tech. Rep. ESD-TR 76-372, 1977.

[13] A. C. Myers and B. Liskov, "A Decentralized Model for Information Flow Control," in *17th Symposium on Operating Systems Principles (SOSP)*. ACM, 1997, pp. 129–142.

[14] A. Sabelfeld and A. Myers, "Language-based Information-Flow Security," *Journal on Selected Area in Communication*, vol. 21, no. 1, pp. 5–19, 2003.

[15] J. Bacon, D. Eyers, T. Pasquier, J. Singh, I. Papagiannis, and P. Pietzuch, "Information Flow Control for Secure Cloud Computing," *IEEE TNSM SI Cloud Service Management*, vol. 11, no. 1, pp. 76–89, 2014.

[16] W. She, I.-L. Yen, B. Thuraisingham, and S.-Y. Huang, "Rule-Based Run-Time Information Flow Control in Service Cloud," in *International Conference on Web Services (ICWS)*. IEEE, 2011, pp. 524–531.

[17] I. Roy, S. T. Setty, A. Kilzer, V. Shmatikov, and E. Witchel, "Airavat: Security and Privacy for MapReduce," in *7th Symposium on Networked System Design and Implementation*. USENIX, 2010, pp. 297–312.

[18] S. Akoush, L. Carata, R. Sohan, and A. Hopper, "MrLazy: Lazy Runtime Label Propagation for MapReduce," in *6th Workshop on Hot Topics in Cloud Computing (HotCloud)*. USENIX, 2014.

[19] X. Xie, I. Ray, R. Adaikkalavan, and R. Gamble, "Information Flow Control for Stream Processing in Clouds," in *Symposium on Access Control Models and Technologies (SACMAT'13)*. ACM, 2013, pp. 89–100.

[20] N. Kumar and R. Shyamasundar, "Realizing Purpose-Based Privacy Policies Succinctly via Information-Flow Labels," in *Big Data and Cloud Computing (BdCloud'14)*. IEEE, 2014, pp. 753–760.

[21] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *Computer*, vol. 29, no. 2, pp. 38–47, 1996.

[22] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed NIST Standard for Role-based Access Control," *ACM Trans. Inf. Syst. Secur.*, vol. 4, no. 3, pp. 224–274, 2001.

[23] L. Giuri and P. Iglio, "Role Templates for Content-Based Access Control," in *Workshop on Role-based Access Control*. ACM, 1997, pp. 153–159.

[24] E. Lupu and M. Sloman, "Reconciling Role Based Management and Role Based Access Control," in *Workshop on Role-based Access Control*. ACM, 1997, pp. 135–141.

[25] D. Schultz and B. Liskov, "IFDB: Decentralized Information Flow Control for Databases," in *European Conference on Computer Systems (Eurosys'13)*. ACM, 2013, pp. 43–56.

[26] R. Sandhu, "Separation of Duties in Computerized Information Systems," in *Database Security IV: Status and Prospects*, 1990.

[27] D. Brewer and M. Nash, "The Chinese Wall security policy," in *IEEE Symposium on Security and Privacy*, 1989, pp. 206–214.

[28] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," *SIGOPS Operating Systems Review*, vol. 41, no. 3, pp. 59–72, 2007.

[29] J. Singh, J. Bacon, J. Crowcroft, A. Madhavapeddy, T. Pasquier, W. K. Hon, and C. Millard, "Regional Clouds: Technical Considerations," University of Cambridge, Tech. Rep. UCAM-CL-TR-863, 2014, accessed: 30th April 2015. [Online]. Available: http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-863.pdf

[30] J. Singh, T. Pasquier, J. Bacon, and D. Eyers, "Integrating Middleware with Information Flow Control," in *International Conference on Cloud Engineering (IC2E)*. IEEE, 2015.

[31] D. Schoepe, D. Hedin, and A. Sabelfeld, "SeLINQ: Tracking Information Across Application-Database Boundaries," in *19th SIGPLAN Conference on Functional Programming*. ACM, 2014, pp. 25–38.

[32] T. F. J.-M. Pasquier, J. Bacon, and B. Shand, "FlowR: Aspect Oriented Programming for Information Flow Control in Ruby," in *13th International Conference on Modularity*. ACM, 2014.

[33] M. Krohn, A. Yip, M. Brodsky, N. Cliffer, M. F. Kaashoek, E. Kohler, and R. Morris, "Information Flow Control for Standard OS Abstractions," in *Symposium on Operating Systems Principles*. ACM, 2007, pp. 321–334.

[34] N. Zeldovich, S. Boyd-Wickizer, and D. Mazières, "Securing Distributed Systems with Information Flow Control," in *5th USENIX Symposium on Networked System Design and Implementation*, 2008, pp. 293–308.

[35] A. Alghothami and F. Kammuller, "Network Information Flow Control: Proof of Concept," in *Systems, Man, and Cybernetics (SMC'13)*. IEEE, 2013, pp. 2957–2962.

[36] Y. Mundada, A. Ramachandran, and N. Feamster, "Silverline: Data and network isolation for cloud services," in *HotCloud'11*. USENIX, 2011.