HAL
open science

Stream Processing with Secure Information Flow Constraints

Indrakshi Ray, Raman Adaikkalavan, Xing Xie, Rose Gamble

To cite this version:

Indrakshi Ray, Raman Adaikkalavan, Xing Xie, Rose Gamble. Stream Processing with Secure Information Flow Constraints. 29th IFIP Annual Conference on Data and Applications Security and Privacy (DBSEC), Jul 2015, Fairfax, VA, United States. pp.311-329, 10.1007/978-3-319-20810-7_22 . hal-01745816

HAL Id: hal-01745816

https://inria.hal.science/hal-01745816

Submitted on 28 Mar 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Stream Processing with Secure Information Flow Constraints

Indrakshi Ray[1], Raman Adaikkalavan[2], Xing Xie[1], and Rose Gamble[3]

[1] Computer Science, Colorado State University
{iray, xing}@cs.colostate.edu
[2] Computer and Information Sciences & Informatics, Indiana University South Bend
raman@cs.iusb.edu
[3] Computer Science, University of Tulsa
gamble@utulsa.edu

Abstract. In the near future, clouds will provide situational monitoring services such as health monitoring, stock market monitoring, shopping cart monitoring, and emergency control and threat management. Offering such services require securely processing data streams generated by multiple, possibly competing and/or complementing, organizations, such that there is no overt or covert leakage of sensitive information. We demonstrate how an information flow control model adapted from the Chinese Wall policy can be used to protect against sensitive data disclosure in data stream management system. We also develop a language based on Continuous Query Language that can be used to express information flow constraints in stream processing and provide its formal semantics.

1 Introduction

Data Stream Management Systems (DSMSs) [1, 5, 9, 8, 14, 6, 15] are needed for situation monitoring applications that collect high-speed data, run continuous queries to process them, and compute results on-the-fly to detect events of interest. Consider one potential situation monitoring application – collecting real-time streaming audit data to thwart various types of attacks in a cloud environment. Detecting such precursors to attacks may involve analyzing streaming audit data belonging to various, possibly competing and/or complementing, organizations. Sensitive information, such as, company policies and intellectual property, may be obtained by mining audit data and hence it must be protected from unauthorized disclosure and modification. Most research on secure data stream processing focus on providing access control to streaming data [18, 13, 21, 12, 19, 2, 3]. Controlling access is crucial, but it is also important to prevent illegal information flow through overt and covert channels. Towards this end, we emphasize how lattice-based information flow control models can be adapted and used for streaming data and provide a query language that supports this model.

In this work our main contribution is formalizing a new language based on CQL [6] for expressing continuous queries with information flow constraints.

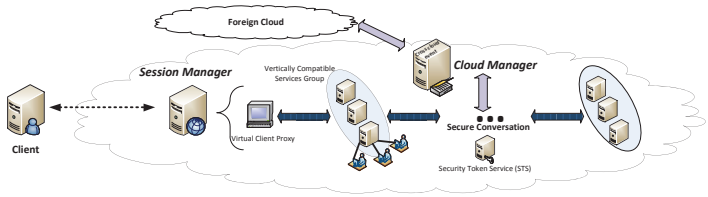

Fig. 1. Multi-Tier Architecture of a Cloud

Query processing, in light of information flow constraints, differs from CQL as the security level of user issuing the query impacts the responses returned. Our language allows for the specification of a set of partially ordered security levels that forms a lattice, where the ordering relation is referred to as the dominance relation, denoted by $<$. We provide the formal semantics of our language such that we can argue about query equivalences and optimization.

The rest of the paper is organized as follows. In Section 2, we present an architecture for processing continuous queries generated from the various tiers in the cloud. In Section 3, we present our information flow control model that formulates the rules for accessing data streams generated by various organizations in the cloud. In Section 4, we provide the formal semantics of our language. In Section 6, we conclude the paper with pointers to future work.

2 Example Application

We have a service that aims to prevent and detect attacks in real-time in the cloud. Such a service provides warning about various types of attacks, often involving multiple organizations. Figure 1 shows a multi tier architecture of the cloud adopted from [30]. Various types of auditing may take place in the cloud. The first level is the *company auditing tier*, not explicitly shown in Figure 1, that is represented by the users connected to some service. In this tier, the activities pertaining to an organization are analyzed in isolation. The next level is the *service auditing tier*, identified by shaded ellipses that contain sets of resources and services. Each shaded ellipse depicts vertically compatible services or resources; this implies the services or resources that can be functionally substituted for each other, possibly on demand. The *cloud auditing tier* is shown with connecting dark arrows, which depicts the internal communication within the cloud due to a service invocation chain.

Various types of audit streams must be captured to detect the different types of attacks that may take place in a cloud. The company auditing tier logs the activities of the various users in the organization. If the behavior of an authorized user does not follow his usual pattern, we can perform analysis to determine if the user's authentication information has been compromised. This tier is responsible for analyzing the audit streams of individual companies in isolation. Typically,

at this layer, the audit streams generated by a single company are analyzed. The service auditing tier logs information pertaining to the various companies who provide similar services. Session Manager at this tier can detect whether there is a denial-of-service attack targeted at a specific type of service. Session Manager analyzes audit streams generated from multiple competing organizations, so we need to protect against information leakage and corruption. In short, the Session Manager needs to analyze data from one or more companies belonging to the same COI class. The cloud auditing tier collects audit information pertaining to a service invocation chain and is able to detect the presence of man-in-the-middle attack. Cloud Manager is responsible for analyzing audit streams from multiple organizations associated with service invocation chains, but the organizations may not have conflict of interest. Thus, at this tier, the audit streams from the companies belonging to one or more CI classes are analyzed.

In order to detect and warn against these attacks, continuous queries must be executed on the streaming data belonging to various organizations. Queries must be processed such that there are no overt or covert leakage of information across competing organizations. We assign security levels to categorize the various classes of data that are being generated and collected at the various tiers. The security level of the data determines who can access and modify it. In the next section, we discuss how security levels are assigned to various classes of data.

Audit data generated by the services are sent to the DSMS. For this paper, we consider a centralized DSMS architecture. Compatible services are grouped and they interact based on client needs. Servers contain event detectors that monitor and detect occurrence of interesting events. The detectors sanitize and propagate the events to the data stream management system, which arrive at the stream source operator. This operator checks for the level of the incoming audit events and propagates them to the appropriate query processor's input queue. The query processor architecture is based on the replicated model, where there is a one-to-one correspondence between query processors and security levels. A query specified by a user at a particular level is executed by the query processor running at that level. Also a query processor at some level can only process data that it is authorized to view. After processing the query results are disseminated to authorized users via the output queues of queries. In addition to the query processors and stream source operator the data stream management system contains various other components (trusted and untrusted) as discussed in the implementation and experimental evaluation section.

3 Continuous Queries with Information Flow Constraints

Secure Information Flow Model

We provide an information flow model that is adapted from the lattice structure for Chinese Wall proposed by Sandhu [24]. We have a set of companies that provide services in the clouds that are partitioned into conflict of interest classes based on the type of services they provide. Companies providing the same type of

service are in direct competition with each other. Consequently, it is important to protect against disclosure of sensitive information to competing organizations. We begin by defining how the conflict of interest classes are represented.

Definition 1. [Conflict of Interest Class Representation:] *The set of companies providing service to the cloud are partitioned into a set of n conflict of interest classes, which we denote by COI_1, COI_2, ..., and COI_n. Each conflict of interest class is represented as COI_i, where $1 \leq i \leq n$. Conflict of interest class COI_i consists of a set of m_i companies, where $m_i \geq 1$, that is $COI_i = \{1, 2, 3, \ldots, m_i\}$.*

On the other hand, a set of companies, who are not in competition with each other, may provide complementing services in the cloud. A single company can provide some service, and sometimes multiple companies may together offer a set of services. In the following, we define the notion of complementing interest (CI) class and show how it is represented.

Definition 2. [Complementing Interest Class Representation:] *The set of companies providing complementing services is represented as an n-element vector $[i_1, i_2, \ldots, i_n]$, where $i_k \in COI_k \cup \{\perp\}$ and $1 \leq k \leq n$. $i_k = \perp$ signifies that the CI class does not contain services from any company in the conflict of interest class COI_k. $i_k \in COI_k$ indicates that the CI class contains services from the corresponding company in conflict of interest class COI_k. Our representation forbids multiple companies that are part of the same COI class from being assigned to the same complementing interest class.*

We next define the security structure of our model. Each data stream, as well as the individual tuples constituting it, is associated with a security level that captures its sensitivity. Security level associated with a data stream dictates which entities can access or modify it. Input data stream generated by an individual organization offering some service has a security level that captures the organizational information. Input streams may be processed by the DSMS to generate *derived streams*. Derived data streams may contain information about multiple companies, some of which are in the same COI class and others may belong to different COI classes. Before describing how to assign security levels to derived data streams, we show how security levels are represented.

Definition 3. [Security Level Representation:] *A security level is represented as an n-element vector $[i_1, i_2, \ldots, i_n]$, where $i_j \in COI_j \cup \{\perp\} \cup \{T\}$ and $1 \leq j \leq n$. $i_j = \perp$ signifies that the data stream does not contain information from any company in COI_j; $i_j = T$ signifies that the data stream contains information from two or more companies belonging to COI_j; $i_j \in COI_j$ denotes that the data stream contains information from the corresponding company in COI_j.*

Consider the case where we have 2 COI classes, namely, COI_1 and COI_2 as shown in Figure 2. COI_1 has two companies denoted by 1 and 2 and COI_2 has two companies denoted by A and B. The audit stream generated by Company

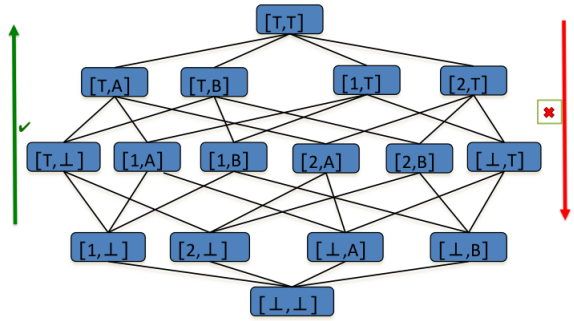

Fig. 2. Lattice-Based Information Flow for the Chinese Wall

2 in COI_1 has a security level of $[2, \perp]$. Similarly, the audit stream generated by Company B in COI_2 has a security level $[\perp, B]$. When audit streams generated from multiple companies are combined, the information contained in this derived stream has a higher security level. For example, audit stream having level $[1, B]$ contains information about Company 1 in COI_1 and Company B in COI_2. It is also possible for audit streams to have information from multiple companies belonging to the same COI class. For example, a security level of $[\perp, T]$ indicates that the data stream does not have information from any company in COI_1, but has information from both companies in COI_2. We also have a level $[\perp, \perp]$ which we call *public* and that has no company specific information. The level $[T, T]$ correspond to level *trusted* and it contains information pertaining to multiple companies in each COI class and can be only accessed by trusted entities. We next define dominance relation between security levels.

Definition 4. [Dominance Relation:] *Let* $\mathbf{L}$ *be the set of security levels,* L_1 *and* L_2 *be two security levels, where* $L_1, L_2 \in \mathbf{L}$*. We say security level* L_1 *is dominated by* L_2*, denoted by* $L_1 \preceq L_2$*, when the following conditions hold:* $(\forall i_k = 1, 2, \ldots, n)(L_1[i_k] = L_2[i_k] \vee L_1[i_k] = \perp \vee L_2[i_k] = T)$*. For any two levels,* L_p*,* L_q $\in \mathbf{L}$*, if neither* $L_p \preceq L_q$*, nor* $L_q \preceq L_p$*, we say that* L_p *and* L_q *are incomparable.*

The dominance relation is reflexive, antisymmetric, and transitive. In Figure 2 the level *public*, denoted by $[\perp, \perp]$, is dominated by all the other levels. Similarly, the level *trusted*, denoted by $[T, T]$, dominates all the other levels. Note that the dominance relation defines a lattice structure, where level *public* appears at the bottom and the level *trusted* appears at the top. Incomparable levels are not connected in this lattice structure. In our earlier example, level $[1, \perp]$ is dominated by $[1, B]$ and $[1, T]$. $[\perp, 2]$ is dominated by $[T, T]$. That is, $[1, \perp] \preceq [1, B]$ and $[1, \perp] \preceq [1, T]$. $[\perp, 2]$ and $[1, \perp]$ are incomparable.

Each data stream is associated with a security level. Each of the tuples constituting the data stream also has a security level assigned to it. Thus, the schema of the data stream has an attribute called level that captures the security level of tuples. The level attribute is generated automatically by the system and cannot be modified by the users. Note that, the security level of an individual tuple in a data stream is dominated by the level of the data stream. When a DSMS operation is executed on multiple input tuples, each having its own security level, an output tuple is produced. The security level of the output tuple is the least upper bound (LUB) of the security levels of the input tuples.

In our work, various types of queries are executed to detect security and performance problems. Each continuous query Q_i, submitted by a process, inherits the security level of the process. We require a query Q_i to obey the simple security property and the restricted $\star$-property of the Bell-Lapadula model [10].

1. Query Q_i with $L(Q_i) = C$ can read a data stream x only if $L(x) \preceq C$.
2. Query Q_i with $L(OP_i) = C$ can write a data stream x only if $L(x) = C$.

Note that, for our example, a process executing at level $[5, \perp, T]$ can read streams belonging to Company 5 in COI_1 and all companies in COI_3 and also streams derived from them. Thus, the process is trusted w.r.t. COI_3, but not w.r.t. the other COI classes. Our information flow model thus provides a finer granularity of trust than provided by the earlier models. Our goal is to allow information flow only from the dominated levels to the dominating ones. All other information flow, either overtly or covertly, should be disallowed by our architecture.

Continuous Queries for Motivating Example

Consider a simple application that tries to detect example denial-of-service attacks in the cloud. We have two conflict of interest classes denoted by COI_1 and COI_2. The constituent companies in each COI class is given by, $COI_1 = \{1, 2\}$ and $COI_2 = \{A, B, C\}$. Examples of security levels in our configuration are $[\perp, \perp]$ (public knowledge), $[T, T]$ (completely trusted), $[1, \perp]$ (data from 1), $[\perp, T]$ (trusted w.r.t. COI_2), $[1, B]$ (data from 1 and B), $[1, T]$ (data from 1 in COI_1 and trusted w.r.t. COI_2). Continuous queries are executed at various tiers to detect performance delays and possibly denial-of-service (DoS) attacks. In any given tier, different types of DoS attacks may occur – some involving the data belonging to single organizations, others involving data belonging to multiple organizations. Thus, a tier can have query processors at different levels, each of which executes queries on data that it is authorized to view and modify.

We consider a single data stream, called `MessageLog`, that contains the audit stream data associated with message events, such as `send` and `receive`. `MessageLog` is obtained from `SystemLog` by filtering the events related to sending and receiving the messages. Note that, `MessageLog` in reality may contain many other fields, but we only deal with those that are pertinent to this example. The various attributes in `MessageLog` are `serviceId`, `msgType`, `sender`, `receiver`, `timestamp`, `outcome`. `serviceId` is a unique identifier associated with each service; `msgType` gives the type of message which is either `send` or `receive`; `sender` (`receiver`) gives the id of the organization sending (receiving) the message; `timestamp` is the time when the event (`send` or `receive`) occurred; `outcome`

denotes `success` or `failure` of the event. In addition to these attributes, we have an attribute referred to as `level` that represents the security level of the tuple. The `level` attribute is assigned by the system and it cannot be modified by the user.

```
MessageLog(serviceId, msgType, sender, receiver, timestamp, outcome)
```

The queries are expressed using the CQL language [6]. We describe the various types of queries that can be executed at the various tiers.

Company Auditing Tier In the company auditing tier, companies have access only to their own audit records. We give some sample queries that are executed by *Company1* to detect performance delays and DoS attacks. All the queries are executed at level $[1, \perp]$.

Query 1 (Q1)

Company1 requests service from *CompanyB*. It is trying to check the times when such message could be successfully delivered.

```
SELECT timestamp FROM MessageLog WHERE msgType = "send" AND outcome = "success"
      AND receiver = "CompanyB"
```

Query 2 (Q2)

Company1 requests service from *CompanyB*. It is trying to check the times when such message could not be successfully delivered.

```
SELECT timestamp FROM MessageLog WHERE msgType = "send" AND outcome = "failure"
      AND receiver = "CompanyB"
```

Service Auditing Tier Service auditing tier receives log records from all the companies making use of some service. However, as the queries below demonstrate, not all the queries need to access all the data from the same COI class.

Query 3 (Q3): Level $[\perp, B]$

Log records received at the service auditing tier can be analyzed by the Session Manager to find out whether *CompanyB* is not available for some service.

```
SELECT timestamp FROM MessageLog WHERE msgType = "send" AND outcome = "failure"
      AND receiver = "CompanyB"
```

Query 4 (Q4): Level $[\perp, T]$

Session Manager may wish to find out whether all companies in COI_2 are target of some DoS attacks.

```
SELECT timestamp FROM MessageLog WHERE msgType = "send" AND outcome = "failure"
      AND receiver = "CompanyB" OR receiver = "CompanyA" OR receiver = "CompanyC"
```

Cloud Auditing Tier Cloud auditing tier gets log records pertaining to all the services. However, the various queries will have different types of security requirements.
Query 5 (Q5): Level $[1, B]$
Cloud Manager may want to look at all records pertaining to `serviceId 5` and measure the delays in order to detect possible man-in-the-middle attack. `serviceId 5` involves *Company1* from COI_1 and *CompanyB* from COI_2.

```
SELECT  MIN(timestamp), MAX(timestamp) FROM MessageLog [ROWS 100]
    WHERE outcome = "success" AND serviceId = "5"
```

Query 6 (Q6): Level $[1, B]$

Cloud Manager wants to find the delay encountered by *Company1* between sending the request and receiving the service from *CompanyB* for the last 100 tuples.

```
SELECT R.timestamp - S.timestamp  AS  delay
   FROM MessageLog R[Rows 100], MessageLog S[Rows 100]
   WHERE S.msgType = "send" AND S.outcome = "success" AND R.msgType = "receive" AND
   R.outcome = "success" AND R.receiver = "Company1" AND R.sender = "CompanyB" AND
   S.receiver = "CompanyB" AND S.sender = "Company1" AND S.serviceId = R.serviceId
```

Query 7 (Q7): Level $[T, T]$
Cloud Manager may want to find out the delay incurred in the different service invocation chains.

```
SELECT MIN(timestamp), MAX(timestamp) FROM MessageLog[ROWS 100]
      WHERE outcome = "success" GROUP BY serviceId
```

Challenges of Continuous Queries with Information Flow Constraints

In this section, we describe why existing DSMS are unsuitable for handling continuous queries in the cloud. The audit queries may be handled by a DSMS that is processing streaming data generated by the various organizations. The DSMS receives data from various stream sources. The stream shepherd system operator [6] receives the incoming data, cleans them, and converts them into a stream with a well-defined schema that can be acted upon by the DSMS. For processing streaming data, CQL has three types of operators, in addition to the shepherd operator: i) stream-to-relation (S2R), ii) relation-to-relation (R2R), and iii) relation-to-stream (R2S). The S2R operators (time, tuple, partitioned by sequential-window) convert input streams to relations. The R2R operators (select, project, join, aggregate) process tuples in the relations produced by the S2R operators and output relations. The R2S operators (istream, dstream, rstream) convert the relations produced by R2R operators back to streams.

In order to securely process queries, the basic requirement is to have an attribute that maintains the sensitivity level of a tuple and that cannot be tampered with by a user. We introduce an attribute termed *level* which is the security

level of the tuple and is system generated. It can be queried or used as part of select condition, but *cannot* be modified by the user. Schemas of all relations and streams in our secure DSMS must have this attribute. The results returned depend on the query level. Thus, when the queries Q_i and Q_j, which look identical, but issued at levels $[1, \perp]$ and $[T, \perp]$ respectively are executed, different results are returned.

- $Q_i([1, \perp])$: `SELECT COUNT FROM MessageLog [ROWS 100] WHERE msgType = "send" AND outcome = "success" AND receiver = "CompanyB"`
- $Q_j([T, \perp])$: `SELECT COUNT FROM MessageLog [ROWS 100] WHERE msgType = "send" AND outcome = "success" AND receiver = "CompanyB"`

The response to Q_i only involve information sent by *Company1* whereas the response to Q_j involve information sent by all companies in COI_1. One may argue that rewriting Q_j by adding an additional clause involving the security level, shown below as query Q'_j, may produce the same result as Q_i.

- $Q'_j([T, \perp])$: `SELECT COUNT FROM MessageLog [ROWS 100] WHERE msgType = "send" AND outcome = "success" AND receiver = "CompanyB" AND level =` $[1, \perp]$`"`

In other words, rewriting Q_j as Q'_j, does not give the same result as Q_i. Thus, post-processing Q_j cannot give us Q_i. Moreover, the window operator in Q_i should not know about the existence of the other types of tuples as that may constitute leakage of sensitive information.

In order to handle the above and other cases, one or more of the operators of types S2R, R2R, or R2S must be modified. In order to process Q_i, the R2R operator (count) which is at the second stage should receive 100 tuples that only have level $[1, \perp]$ from the window operator at the first stage. This cannot be achieved by using another R2R operator such as select (via a filtering condition as shown in Q'_j) in the second stage as the window is created in the first stage. Thus, the S2R operator in the first stage should create time varying relations with 100 tuples that are at level $[1, \perp]$ from the input stream and propagate it to the R2R operator in the second stage. This cannot be done by existing S2R operators. Moreover, in existing systems, an S2R operator is shared by queries accessing the same input stream to reduce resource usage. The input streams may have tuples belonging to different sensitivity levels. Consequently, allowing queries at different levels to read from the same S2R operator violates the information flow policies.

The final issue is the processing of the specified queries. The query processor should be able to execute the queries with different sensitivity levels without leaking information. Leakage can occur through overt or covert channels. Overt channels occur when a process reads or writes data at a different sensitivity level. Covert channels occur when a process at some sensitivity level manipulates the use of shared resources (such as CPU and memory) to pass information on to a process at a different sensitivity level. Current DSMS query processors are not equipped to provide protection against such illegal channels. For example, the scheduler in a DSMS is responsible for executing all the queries – by manipulating the time taken to execute certain queries, information can be passed from one security level to another. We need to redesign query processors in light of information flow constraints.

4 Formal Semantics of the Language

We begin with discussing the formal semantics of the language used to express streaming queries with information flow constraints. We present our language and semantics in a manner similar to that proposed by Arasu and Widom [7]. We have adapted the approach for our language, which we call SIF-CQL (Secure Information Flow Continuous Query Language). To make the paper self contained, we have used some definitions from [7].

In our secure DSMS, we have two types of data: *streams* and *relations*. Each stream or relation is associated with a schema consisting of a set of attributes as in the traditional relational model together with a system defined security attribute which we refer to as a *level*. We also assume the existence of a discrete and totally ordered time domain $\mathcal{T} = \{0, 1, \ldots\}$. An element in $\mathcal{T}$ is referred to as *time instant* or *instant*.

We can have two types of streams: *trusted* and *single level*. A trusted stream consists of inputs from various sources and is not associated with any specific security level, but consists of tuples each of which is associated with a security level. The assumption is that in a trusted stream the individual tuples are protected and information is not passed from one level to another. In contrast, we may have single level streams which are associated with a specific security level. A single level stream contains tuples, each of whose security level is dominated by the level of the stream. For example, a single level stream at $[1, \perp]$ level can contain tuples at $[\perp, \perp]$ or $[1, \perp]$ level. The formal definitions of these two types of streams appear below.

Definition 5. [Trusted Stream] *A* trusted stream *SS is not associated with any security level and consists of a possibly infinite bag of elements $< s, l, \tau >$, where $< s, l >$ is a tuple belonging to the schema of S, $\tau \in \mathcal{T}$ is a timestamp and $l \in \mathcal{L}$ is the security level of the tuple.*

Definition 6. [Single Level Stream] *A* single level stream *S is associated with a single security level L_S and consists of a possibly infinite bag of elements $< s, l, \tau >$, where $< s, l >$ is a tuple belonging to the schema of S, $\tau \in \mathcal{T}$ is a timestamp and l is the security level of the tuple such that $l \leq L_S$ and $L_S \in \mathcal{L}$.*

We have only single level relations (or just relations) that are associated with one security level.

Definition 7. [Single Level Relation] *A* single level relation *or a* relation *R is a mapping from $\mathcal{T} \times \mathcal{L}$ to a finite but unbounded bag of tuples belonging to the schema of R each of which is associated with a security level. A relation R is associated with a security level L_R which is the input to the mapping function. The security level of each tuple in R is dominated by $L_R \in \mathcal{L}$.*

A stream is a collection of timestamped tuples each of which is associated with a security level. The element $< s, l, \tau >$ signifies that tuple s arrives on S at time τ and is associated with the security level l. For a relation R, $R(\tau)$ denotes the bag of tuples, each associated with a security level, in the relation at time instant τ.

In secure DSMS, continuous queries submitted by users operate only on single level streams that have the same level as the query level. Continuous queries are composed from secure operators belonging to three classes: stream-to-relation (S2R), relation-to-relation (R2R), and relation-to-stream (R2S).

- A secure S2R operator takes an input stream, time instant τ, and security level l and produces a relation at level l. Each tuple of the relation is associated with a level that equals the level of the corresponding element in the stream as a stream can contain tuples at different levels. Note that, the level of each tuple of the relation is dominated by l. The bag of tuples in the output relation at time instant τ and security level l depends only on input stream elements with timestamps $\leq \tau$ and security level $\leq l$.
- A secure R2R operator takes one or more relations as input together with a security level l and produces another relation at level l as output. When the input has one relation, the level of each tuple in the output relation is same as the corresponding input tuples. When there is more than one input relation, the level of each output tuple is the least upper bound of the levels of the corresponding input tuples. Here again, the level of the output relation dominates the level of tuples in that relation. The bag of tuples in the output relation at time τ depends on the bag of tuples in the input relations at τ and their security levels.
- A secure R2S operator takes an input relation, timestamp τ, and a security level l, and produces a stream at level l. The security level of each element of the stream depends on the security level of the corresponding tuple in the relation. Note that, the security level of the elements of the stream is dominated by the level of the stream l. The elements of the stream with timestamp τ depend only on the relation tuples at time instants $\leq \tau$ and whose security level $\leq l$.

The trusted stream shepherd operator, a secure stream-to-stream operator (S2S-Op), takes as input a trusted stream and creates single level streams for each security level.

We define the following domains.

- *Time domain* ($\mathcal{T}$) is the domain of time instants. $\mathcal{T} = \{0, 1, \ldots\}$.
- *Security level domain* ($\mathcal{L}$) (defined in Section **??**) is the domain of security levels. We use $l_i \leq l_j$ to denote that l_j dominates l_i where $l_i, l_j \in \mathcal{L}$.
- *Tuple domain* ($\mathcal{TP}$) is the domain of tuples consisting of all but the security attribute.
- *Tuple multiset domain* (Σ) is the domain of finite but unbounded bag of tuples.
- *Relation domain* ($\mathcal{R}$) is the domain of functions that map time instants and security levels to bags of tuples. The security levels of the bags equal the input security level. Each tuple in the bag is also associated with a security level that is dominated by the input security level. We denote this as $R = \mathcal{T} \times \mathcal{L} \rightarrow \Sigma \times \mathcal{L}$.
- *Trusted stream domain* ($\mathcal{SS}$) is the domain of possibly infinite multisets over $\mathcal{TP} \times \mathcal{L} \times \mathcal{T}$.
- *Single level stream domain* ($\mathcal{S}$) is the domain of functions that map security levels to possibly infinite multisets over $\mathcal{TP} \times \mathcal{L} \times \mathcal{T}$. We denote this as $S = \mathcal{L} \rightarrow \mathcal{TP} \times \mathcal{L} \times \mathcal{T}$.
- *Relational operator domain* ($\mathcal{R}_{op}$) is the domain of functions that produce a bag of tuples from one or more bags of tuples and an input security level. Each input bag is associated with a security level which dominates the security level of tuples in the bag. The output bag is associated with a security level that is input to the operator. The security levels of the tuples in the output bag depend on the security levels of the corresponding input tuples. We formally denote this as: $\mathcal{R}_{op} = (\Sigma \times \mathcal{L})^n \times \mathcal{L} \rightarrow \Sigma \times \mathcal{L}$, where n represents the number of input bags.
- *Syntactic domains* are the domains associated with syntactic terms.
- *Relation lookup domain* ($RelLookUp$) is the domain of functions that map an identifier to its corresponding relation. We denote it as: $RelLookUp = Identifier \rightarrow \mathcal{R}$

– *Stream lookup domain* ($StrLookUp$) is the domain of functions that map an identifier to its corresponding stream. We denote it as: $StrLookUp = Identifier \rightarrow \mathcal{S}$

The abstract syntax for language is given below. Table 1 has the symbol descriptions and domains.

$$
\begin{array}{lll}
Q & ::= & Q_R \mid Q_S \\
Q_R & ::= & \texttt{RName} \mid \text{R2R-Op } (Q_R^1, \ldots, Q_R^n) \\
 & & \mid \text{S2R-Op } (Q_S) \\
Q_S & ::= & \texttt{SName} \mid \text{R2S-Op } (Q_R) \\
\texttt{SName} & ::= & \text{S2S-Op } (\texttt{SSName}) \\
\texttt{SSName} & ::= & \text{Id} \\
\texttt{RName} & ::= & \text{Id}
\end{array}
$$

Table 1. Symbol descriptions and domains

Symbol	**Description**	**Domain**
Q	Continuous Query (CQ) in SIF-CQL	$Query$
Q_R	CQ producing a relation	$RelQuery$
Q_S	CQ producing a stream	$StrQuery$
S2S-Op	Stream-to-Stream Operator	$S2SOp$
S2R-Op	Stream-to-Relation Operator	$S2ROp$
R2R-Op	Relation-to-Relation Operator	$R2ROp$
R2S-Op	Relation-to-Stream Operator	$R2SOp$
RName	Relation Name	$Identifier$
SSName	Stream Name	$Identifier$
Id	Identifier	$Identifier$

We are now ready to provide a denotational semantics [25], similar to the work in [7], for SIF Continuous Queries that are expressed using our language which we denote as $\mathcal{S}$. A denotational semantics is specified by a meaning function that we denote as $\mathcal{M}$. Function $\mathcal{M}$ applied to a continuous query Q, which we represent as $\mathcal{M}[\![Q]\!]$, takes as input the streams and relations referred to in Q together with a time instant τ and security level l, and produces an output consisting of a new relation or stream corresponding to the time instant τ. The security level of this output relation or stream is l and it consists of tuples or elements each of whose security level is dominated by l. Table 2 describes the meaning functions that we use.

We use *lambda calculus* [23] notations for defining our functions. The expression $\lambda x_1, \ldots, \lambda x_n.E$ represents a function that takes arguments $v_1, \ldots v_n$, and returns the result of evaluating E by replacing all free occurrences of x_i in E by v_i where $1 \leq i \leq n$. We now present the details of the meaning functions.

– $\mathcal{M}$: The function $\mathcal{M}[\![Q]\!]$ produced by $\mathcal{M}$ for query Q takes four parameters. The first two parameters are functions that map the relation or stream names in the query to the appropriate relations or streams. The third and fourth parameters are security level and time instant respectively. $\mathcal{M}[\![Q]\!](r, s, l, \tau)$ specifies the output produced by Q at time instant τ and security level l. $\mathcal{M}[\![Q]\!](r, s, l, \tau)$ invokes $\mathcal{M}_R[\![Q_R]\!](r, s, l, \tau)$ if $Q = Q_R$ produces a relation as an output and calls $\mathcal{M}_S[\![Q_S]\!](r, s, l, \tau)$ if $Q = Q_S$ produces a stream as an output.

Table 2. Meaning Functions

Query Part	Meaning	Signature
Q	$\mathcal{M}$	$Query \rightarrow (RelLookUp \times StrLookUp \times \mathcal{L} \times \mathcal{T} \rightarrow ((\Sigma \cup \mathcal{S}) \times \mathcal{L})$
Q_R	$\mathcal{M}_R$	$RelQuery \rightarrow (RelLookUp \times StrLookUp \times \mathcal{L} \times \mathcal{T} \rightarrow \Sigma \times \mathcal{L})$
Q_S	$\mathcal{M}_S$	$StrQuery \rightarrow (RelLookUp \times StrLookUp \times \mathcal{L} \times \mathcal{T} \rightarrow \mathcal{S} \times \mathcal{L})$
S2S-Op	$\mathcal{M}_{S2S}$	$S2SOP \rightarrow (\mathcal{SS} \times \mathcal{L} \rightarrow \mathcal{S} \times \mathcal{L})$
S2R-Op	$\mathcal{M}_{S2R}$	$S2ROp \rightarrow (\mathcal{S} \times \mathcal{L} \times \mathcal{T} \rightarrow \Sigma \times \mathcal{L})$
R2R-Op	$\mathcal{M}_{R2R}$	$R2ROp \rightarrow \mathcal{R}_{op}$
R2S-Op	$\mathcal{M}_{R2S}$	$R2SOp \rightarrow (\mathcal{R} \times \mathcal{L} \times \mathcal{T} \rightarrow \mathcal{S} \times \mathcal{L})$

$\mathcal{M}[\![Q_R]\!] = \lambda r.\lambda s.\lambda l.\lambda \tau.\mathcal{M}_R[\![Q_R]\!](r, s, l, \tau)$

$\mathcal{M}[\![Q_S]\!] = \lambda r.\lambda s.\lambda l.\lambda \tau.(\{\langle e, l', \tau\rangle : \langle e, l', \tau\rangle \in \mathcal{M}_S[\![Q_S]\!](r, s, l, \tau)\}, l)$

- $\mathcal{M}_R$: If Q_R is a query producing a relation, $\mathcal{M}_R[\![Q_R]\!](r, s, l, \tau)$ specifies the bag of tuples in the output relation at time τ that are dominated by level l. Parameters r and s signify the stream and relation lookup functions. If $Q_R = \texttt{RName}$, $\mathcal{M}_R[\![Q_R]\!](r, s, l, \tau)$ uses function r to look up the time-varying relation corresponding to `RName` and identifies the bag of tuples at time τ that are dominated by level l. The security level of the bag is l.
  $\mathcal{M}_R[\![\texttt{RName}]\!] = \lambda r.\lambda s.\lambda l.\lambda \tau.(\{\langle e, l'\rangle : \langle e, l'\rangle \in r(\texttt{RName})(\tau) \wedge l' \leq l\}, l)$

  $\mathcal{M}_R[\![\texttt{R2R-Op}(Q_R^1, \ldots, Q_R^n)]\!] = \lambda r.\lambda s.\lambda l.\lambda \tau.\mathcal{M}_{R2R}[\![\texttt{R2R-Op}]\!](\mathcal{M}_R[\![Q_R^1]\!](r, s, l, \tau), \ldots, \mathcal{M}_R[\![Q_R^n]\!](r, s, l, \tau), l)$

  $\mathcal{M}_R[\![\texttt{S2R-Op}(Q_S)]\!] = \lambda r.\lambda s.\lambda l.\lambda \tau.\mathcal{M}_{S2R}[\![\texttt{S2R-Op}]\!](\mathcal{M}_S[\![Q_S]\!](r, s, l, \tau), l, \tau)$

- $\mathcal{M}_S$: If Q_S is a query producing a stream, $\mathcal{M}_S[\![Q_S]\!](r, s, \tau, l)$ specifies the bag of stream elements in the output stream with timestamp $\leq \tau$ and security level $\leq l$. The security level of the output stream is l.
  $\mathcal{M}_S[\![\texttt{SName}]\!] = \lambda r.\lambda s.\lambda l.\lambda \tau.(\{\langle e, l', \tau'\rangle : \langle e, l', \tau'\rangle \in s(\texttt{SName}) \wedge \tau' \leq \tau \wedge l' \leq l\}, l)$

  $\mathcal{M}_S[\![\texttt{R2S-Op}(Q_R)]\!] = \lambda r.\lambda s.\lambda \tau.\lambda l.\mathcal{M}_{R2S}[\![\texttt{R2S-Op}]\!]((\lambda \tau'.\mathcal{M}_R[\![Q_R]\!](r, s, l, \tau')), l, \tau)$

4.1 Semantics for Example Operators

We present the abstract syntax for a few example operators. The abstract syntax for these operators presented in BNF like form appears below.

```
S2S-Op ::= StreamShepherd
R2S-Op ::= IStream | DStream | RStream
R2R-Op ::= SemiJoin(i, j) | Filter(i, v)
S2R-Op ::= Now Cond | Range(T) Cond
           | Row(N) Cond
Cond   ::= True | Filter(i, v)
```

- $\mathcal{M}_{S2S}$: We have only one stream-to-stream operator which is the `StreamShepherd`. The `StreamShepherd` operator takes a multilevel trusted stream and a security

level as its input and creates a single level stream at the corresponding security level as the output.
$\mathcal{M}_{S2S}[\![StreamShepherd]\!] = \lambda SS.\lambda l.(\{\langle e, l'\rangle : \langle e, l'\rangle \in SS \wedge (l' \leq l)\}, l)$

- $\mathcal{M}_{R2S}$: We have three relation-to-stream operators in SIF-CQL. The `IStream` operator takes a time-varying relation R, security level l, and a time instant τ and streams the new tuples inserted into R at time τ, that is, only those tuples that appear in $R(\tau)$ but not in $R(\tau - 1)$ whose security level is dominated by l. The `DStream` operator streams the tuples that were deleted from R at time τ, that is, tuples that appear in $R(\tau - 1)$ but not in $R(\tau)$. Here again only the tuples whose security level is dominated by l are reported. The `RStream` operator streams all tuples in $R(\tau)$ whose security levels are dominated by l. The security levels of the output streams in each case equals l.
$\mathcal{M}_{R2S}[\![\texttt{IStream}]\!] = \lambda R.\lambda\tau.\lambda l.(\{\langle e, l', \tau'\rangle : \tau' \leq \tau \wedge l' \leq l \wedge \langle e, l'\rangle \in R(\tau) \wedge \langle e, l'\rangle \notin R(\tau - 1)\}, l)$

$\mathcal{M}_{R2S}[\![\texttt{DStream}]\!] = \lambda R.\lambda l.\lambda\tau.(\{\langle e, l', \tau'\rangle : \tau' \leq \tau \wedge l' \leq l \wedge \langle e, l'\rangle \notin R(\tau) \wedge \langle e, l'\rangle \in R(\tau - 1)\}, l)$

$\mathcal{M}_{R2S}[\![\texttt{RStream}]\!] = \lambda R.\lambda l.\lambda\tau.(\{\langle e, l', \tau'\rangle : \tau' \leq \tau \wedge l' \leq l \wedge \langle e, l'\rangle \in R(\tau)\}, l)$

- $\mathcal{M}_{R2R}$: We do not give the semantics of all relational operators, but just present only two as examples. $\texttt{SemiJoin}(i, j)$ performs a semijoin on the ith attribute of its first input with the jth attribute of the second input, where both inputs are bags of tuples together with the security levels of the tuples. $\texttt{Filter}(i, v)$ returns all tuples from its input bag having value v in the ith attribute. The notation $e.i$ denotes the value in the ith attribute of a tuple e.
$\mathcal{M}_{R2R}[\![\texttt{SemiJoin}(i, v)]\!] = \lambda E_1.\lambda E_2.\lambda l.(\{\langle e_1, l_3\rangle :$
$\langle e_1, l_1\rangle \in E_1 \wedge (\exists\langle e_2, l_2\rangle \in E_2 \wedge l_1 \leq l \wedge l_2 \leq l \wedge e_1.i = e_2.j \wedge l_3 = lub(l_1, l_2))\}, l)$

$\mathcal{M}_{R2R}[\![\texttt{Filter}(i, v)]\!] = \lambda E.\lambda l.(\{\langle e, l'\rangle : \langle e, l'\rangle \in E \wedge l' \leq l \wedge e.i = v\}, l)$

- $\mathcal{M}_{S2R}$: We first consider three basic sliding window operators. All three operators take a stream S, a timestamp τ, and security level l as input and return a bag of tuples together with their security levels as output. The `Now` operator returns the tuples with timestamp τ and security level l' where $l' \leq l$. The `Range` operator specified with parameter T returns the tuples of S with timestamps in the range $[\tau - T, \tau]$ with security level l' where $l' \leq l$. The `Row` operator specified using an integer parameter N, returns the N most recent tuples of S with timestamps $\leq \tau$ and security level $\leq l$.
We then augment the basic operators with filtering conditions. The `Now` $\texttt{Filter}(i, v)$ operator returns the tuples with timestamp τ and security level l' where $l' \leq l$ such that the ith attribute of the tuple equals the value v. The other window operators are augmented with a condition in a similar manner.
$\mathcal{M}_{S2R}[\![\texttt{Now}]\!] = \lambda S.\lambda l.\lambda\tau.(\{\langle e, l'\rangle : \langle e, l', \tau\rangle \in S \wedge (l' \leq l)\}, l)$

$\mathcal{M}_{S2R}[\![\texttt{Now Filter}(i, v)]\!] = \lambda S.\lambda l.\lambda\tau.(\{\langle e, l'\rangle : \langle e, l', \tau, \rangle \in S \wedge (l' \leq l) \wedge (e.i = v)\}, l)$

$\mathcal{M}_{S2R}[\![\texttt{Range}(T)]\!] = \lambda S.\lambda\tau.\lambda l.(\{\langle e, l'\rangle : \langle e, l', \tau\rangle \in S \wedge (l' \leq l) \wedge (\texttt{max}(\tau - T, 0) \leq \tau' \leq \tau)\}, l)$

$$\mathcal{M}_{S2R}[\![\texttt{Range}(T)\ \texttt{Filter}(i,v)]\!] = \lambda S.\lambda\tau.\lambda l.(\{\langle e,l'\rangle : \langle e,l',\tau\rangle \in S \wedge (l' \leq l) \wedge(\texttt{max}(\tau - T, 0) \leq \tau' \leq \tau) \wedge (e.i = v)\}, l)$$

$$\mathcal{M}_{S2R}[\![\texttt{Row}(N)]\!] = \lambda S.\lambda\tau.\lambda l.(\{\langle e,l'\rangle : \langle e,l',\tau'\rangle \in S \wedge (l' \leq l) \wedge (\tau' \leq \tau) \wedge (N \geq |\{\langle e,l'',\tau''\rangle \in S : (\tau' \leq \tau'' \leq \tau) \wedge (l' \leq l) \wedge (l'' \leq l)\}\ |)\}, l)$$

$$\mathcal{M}_{S2R}[\![\texttt{Row}(N)\ \texttt{Filter}(u,v)]\!] = \lambda S.\lambda\tau.\lambda l.(\{\langle e,l'\rangle : \langle e,l',\tau'\rangle \in S \wedge(l' \leq l) \wedge (\tau' \leq \tau)\wedge (e.i = v) \wedge (N \geq \ \ |\ \{\langle e,l'',\tau''\rangle \in S : (e.i = v) \wedge (\tau' \leq \tau'' \leq \tau) \wedge (l' \leq l) \wedge (l'' \leq l)\}\ |)\}, l)$$

5 Related Work

DSMS Security: Most works on securing DSMSs [18, 13, 21, 12, 19, 2] focus on how role-based access control policies can be supported in stream processing. Punctuation-based enforcement of RBAC over data streams is proposed in [21]. Access control policies are transmitted every time using one or more security punctuations before the actual data tuple is transmitted. Query punctuations define the privileges for a CQ. Both punctuations are processed by a special filter operator (stream shield) that is part of the query plan. Punctuations have been used to enforce continuous access control for both data and queries [20, 22] where security restrictions can change while the continuous queries are being executed. While security punctuations are used for enforcing dynamic access control, our work in this paper is focused on preventing unauthorized access and illegal information flow. Secure shared continuous query processing is proposed in [2]. The authors present a three-stage framework to enforce access control without introducing special operators, rewriting query plans, or affecting QoS delivery mechanisms. Supporting role-based access control via query rewriting techniques is proposed in [13, 12]. To enforce access control policies, query plans are rewritten and policies are mapped to a set of map and filter operations. When a query is activated, the privileges of the query submitter are used to produce the resultant query plan. The architecture proposed in [18] uses a post-query filter to enforce stream level access control policies. The filter applies security policies after query processing but before a user receives the results from the DSMS. Designing DSMS taking into account multilevel security constraints has been addressed by researchers [3, 4] and the impact of information flow constraints on the performance has also been presented [31, 32].

Chinese Wall Policy and Cloud Computing: Wu et al. [29], show how the Chinese Wall policy, originally proposed by Brewer and Nash [11] and later refined by Sandhu [24] can be used for information flow control in cloud computing. The authors enforce the policies at the Infrastructure-as-a-Service layer and develop a prototype to demonstrate the feasibility of their approach. She et al. [26] provide a protocol for doing access validation during service compositions for ensuring information flow control. Each service is required to specify its information flow policy with respect to the other services in the service-chain. However, the authors shed little light on the structure of the policies themselves. Hung and Qui [17] also address COI issues using Chinese Wall policy. Each COI has a set of operations and each service does not perform more than one operation in the same COI class. However, history information is not maintained

which makes COI possible due to interleaved operation invocation by multiple services. Hsiao and Hwang [16] demonstrate how the Chinese Wall can be used in the context of workflows. Shen et al. [27] address COI issues in storage clouds to ensure isolation of data belonging to several companies. If a violation occurs because of data collaboration, the individual tenants have the right to approve or disapprove the violations. Tsai et al. [28] discusses how the Chinese Wall policy can be used to prevent competing organizations virtual machines to be placed on the same physical machine. Graph coloring is used for allocating virtual machines to physical machines such that the Chinese Wall policies are satisfied and better utilization of cloud resources is achieved. In an earlier work [32] we proposed an information flow control model suitable for cloud environments that was adapted from the Chinese Wall policy [24]. Our current work extends this by providing a formal semantics of the language used to express continuous queries with information flow constraints.

6 Conclusions and Future work

Data streams generated by various organizations in a cloud may need to be analyzed in real-time for detecting critical events of interest. Processing of such data streams should be done in a careful and controlled manner such that company sensitive information is not disclosed to competing organizations. We propose a secure information flow control model, adapted from the Chinese Wall policy, to be used for protecting sensitive company information. We formalized the language for expressing continuous queries using denotational semantics. Our future plans include doing secure processing over encrypted streams and investigating how information flow constraints can be maintained while processing such data.

References

1. D. J. Abadi, Y. Ahmad, M. Balazinska, U. Çetintemel, M. Cherniack, J. Hwang, W. Lindner, A. Maskey, A.Rasin, E.Ryvkina, N.Tatbul, Y.Xing, and S. B. Zdonik. The Design of the Borealis Stream Processing Engine. In *Proc. of the CIDR*, pages 277–289, 2005.
2. R. Adaikkalavan and T. Perez. Secure Shared Continuous Query Processing. In *Proc. of the ACM SAC (Data Streams Track)*, pages 1005–1011, Taiwan, Mar. 2011.
3. R. Adaikkalavan, I. Ray, and X. Xie. Multilevel Secure Data Stream Processing. In *Proc. of the DBSec*, pages 122–137, July 2011.
4. R. Adaikkalavan, X. Xie, and I. Ray. Multilevel Secure Data Stream Processing: Architecture and Implementation. *J. Comput. Secur.*, 20(5):547–581, 2012.
5. A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava, and J. Widom. STREAM: The Stanford Data Stream Management System. Technical Report 2004-20, Stanford InfoLab, 2004.
6. A. Arasu, S. Babu, and J. Widom. The CQL Continuous Query Language: Semantic Foundations and Query Execution. *VLDB Journal*, 15(2):121–142, 2006.
7. A. Arasu and J. Widom. A Denotational Semantics for Continuous Queries over Streams and Relations. *SIGMOD Record*, 33:6–11, September 2004.
8. B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proc. of the PODS*, pages 1–16, June 2002.

9. H. Balakrishnan, M. Balazinska, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, E. Galvez, J. Salz, M. Stonebraker, N. Tatbul, R. Tibbetts, and S. B. Zdonik. Retrospective on Aurora. *VLDB Journal: Special Issue on Data Stream Processing*, 13(4):370–383, 2004.
10. D. E. Bell and L. J. LaPadula. Secure Computer System: Unified Exposition and MULTICS Interpretation. Technical Report MTR-2997 Rev. 1 and ESD-TR-75-306, rev. 1, The MITRE Corporation, Bedford, MA 01730, Mar. 1976.
11. D. F. C. Brewer and M. J. Nash. The Chinese Wall Security Policy. In *Proc. of the IEEE S & P*, pages 206–214, May 1989.
12. J. Cao, B. Carminati, E. Ferrari, and K. Tan. Acstream: Enforcing access control over data streams. In *Proc. of the ICDE*, pages 1495–1498, 2009.
13. B. Carminati, E. Ferrari, and K. L. Tan. Enforcing Access Control over Data Streams. In *Proc. of the ACM SACMAT*, pages 21–30, 2007.
14. D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. B. Zdonik. Monitoring Streams - A New Class of Data Management Applications. In *Proc. of the VLDB*, pages 215–226, August 2002.
15. S. Chakravarthy and Q. Jiang. *Stream Data Processing: A Quality of Service Perspective Modeling, Scheduling, Load Shedding, and Complex Event Processing*, volume 36 of *Advances in Database Systems*. Springer, 2009.
16. Y.-C. Hsiao and G.-H. Hwang. Implementing the Chinese Wall Security Model in Workflow Management Systems. In *Proc. of the ISPA*, pages 574–581, Sept. 2010.
17. P. C. K. Hung and G.-S. Qiu. Specifying Conflict of Interest Assertions in WS-Policy with Chinese Wall Security Policy. *SIGecom Exchanges*, 4(1):11–19, 2003.
18. W. Lindner and J. Meier. Securing the Borealis Data Stream Engine. In *IDEAS*, pages 137–147, 2006.
19. R. V. Nehme, H. Lim, E. Bertino, and E. A. Rundensteiner. StreamShield: A Stream-Centric Approach towards Security and Privacy in Data Stream Environments. In *Proc. of the ACM SIGMOD*, pages 1027–1030, 2009.
20. R. V. Nehme, H.-S. Lim, and E. Bertino. Fence: Continuous access control enforcement in dynamic data stream environments. In *Proceedings of the Third ACM Conference on Data and Application Security and Privacy*, CODASPY '13, pages 243–254, New York, NY, USA, 2013. ACM.
21. R. V. Nehme, E. A. Rundensteiner, and E. Bertino. A Security Punctuation Framework for Enforcing Access Control on Streaming Data. In *Proc. of the ICDE*, pages 406–415, 2008.
22. R. V. Nehme, E. A. Rundensteiner, and E. Bertino. A security punctuation framework for enforcing access control on streaming data. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, ICDE '08, pages 406–415, Washington, DC, USA, 2008. IEEE Computer Society.
23. B. C. Pierce. *The Computer Science and Engineering Handbook*, chapter Foundational Calculi for Programming Languages, pages 2190–2207. CRC Press, 1997.
24. R. Sandhu. Lattice-Based Enforcement of Chinese Walls. *Computers & Security*, 11(8):753–763, 1992.
25. D. A. Schmidt. Programming Language Semantics. In A. B. Tucker, editor, *The Computer Science and Engineering Handbook*, pages 2237–2254. CRC Press, 1997.
26. W. She, I.-L. Yen, B. M. Thuraisingham, and E. Bertino. Security-Aware Service Composition with Fine-Grained Information Flow Control. *IEEE TSC*, 6(3):330–343, 2013.
27. Q. Shen, X. Yang, P. Sun, Y. Yang, and Z. Wu. Towards Data Isolation & Collaboration in Storage Cloud. In *Proc. of the APSCC*, pages 139–146, December 2011.

28. T. Tsai, Y. Chen, H. Huang, P. Huang, and K. Chou. A Practical Chinese Wall Security Model in Cloud Computing. In *Proc. of the APNOMS*, pages 1–4, September 2011.
29. R. Wu, G. Ahn, H. Hu, and M. Singhal. Information Flow Control in Cloud Computing. In *Proc. of the CollaborateCom*, pages 1–7, October 2010.
30. R. Xie and R. Gamble. A Tiered Strategy for Auditing in the Cloud. In *IEEE International Conference on Cloud Computing*, June 2012.
31. X. Xie, I. Ray, and R. Adaikkalavan. On the Efficient Processing of Multilevel Secure Continuous Queries. In *Proc. of Social Computing*, pages 417–422, 2013.
32. X. Xie, I. Ray, R. Adaikkalavan, and R. Gamble. Information Flow Control for Stream Processing in Clouds. In *Proc. of the ACM SACMAT*, pages 89–100, 2013.