Privacy-Preserving Triggers for Pervasive Spaces

Jehan Wickramasuriya, Mahesh Datt, Bijit Hore, Stanislaw Jarecki,
Sharad Mehrotra & Nalini Venkatasubramanian

Department of Information & Computer Science
University of California, Irvine
Irvine, CA 92697-3425, USA
{jwickram,mahesh,bhore,stasio,sharad,nalini}@ics.uci.edu

Abstract. Providing pervasive services often necessitates gathering information about individuals that may be considered sensitive. Often, one is forced to make a difficult choice: either to risk loss of privacy or to let go of the benefits pervasive technology offers. Our conjecture is that such a choice is not always necessary. It is possible to design data collection strategies in such a way that services offered by pervasive environments do not come at the expense of individuals' privacy. Specifically, we consider a trigger-based pervasive environment in which end-user services are built using triggers over events detected through sensors. Privacy concerns in such a pervasive space are analogous to disclosure and misuse of event logs. We use cryptographic techniques to prevent loss of privacy, but that raises a fundamental challenge of evaluating triggers over the encrypted data representation. We address this challenge for a class of triggers that are powerful enough to realize multiple pervasive functionalities. We use the techniques proposed to develop a middleware framework for privacy-preserving pervasive spaces. We discuss a specific application, viz. privacy-preserving surveillance which we have built using the framework developed. We also perform experimental studies to assess the overheads and scalability of our proposed strategies.

1 Introduction

Emerging sensing, embedded computing, and networking technologies have created opportunities to blend computation with the physical world and its activities. The resulting pervasive environments offer numerous opportunities including customization (e.g. personalized advertising), automation (e.g. inventory tracking and control) and access control (e.g. biometric authentication, triggered surveillance). Building pervasive spaces requires collecting and logging information about the state of the environment, its users, and its resources. Such environmental data collected in the form of event logs is used to implement functionalities offered by pervasive spaces as well as to control access to resources.

In this paper, our interest is on human-centric pervasive spaces in which human subjects are part of the environment. In such spaces, the environmental data collected may include personalizing information about individuals immersed in the space. This naturally leads to concerns of privacy. For instance, information about a person's location used to customize the space could potentially be misused as evidence of his/her presence/absence at a given location and time (whether or not it is in their best interest). Such privacy concerns arise when users do not fully trust the pervasive environment. Measures to establish trust, such as explicit policies to prevent leakage or sharing of personalizing information may alleviate, but do not eliminate such concerns. For example, organizational level trust does not prevent perpetration at the human level in

the form of malicious administrators, operators, etc. Indeed, numerous studies have identified such privileged users as primary source of corporate data thefts.

A complete solution requires privacy-preserving data collection strategies that can support the functionality of pervasive spaces without violating the privacy of individuals. Such a design opens numerous research challenges, some of which will be addressed in this paper. Specifically, we develop mechanisms for privacy-preserving data collection in the context of a trigger-based pervasive space in which triggers defined over event logs are used to implement the policies and functionalities of the space. We observe that the trigger-based approach offers a general framework using which a wide variety of pervasive applications can be realized. Our privacy-preserving techniques can be used to support a large class of pervasive functionalities.

Privacy preservation in trigger-based pervasive environments requires mechanisms to generate and store events in secure logs as well as to evaluate triggers in such a way that personalizing information about individuals is not compromised. If the pervasive environment is entirely untrusted, privacy preservation will remain an elusive goal. In our approach, we assume the presence of tamper-proof sensing devices which can be programmed to generate a stream of events while at the same time hide the raw signals from which such events are generated. Such an assumption is not unreasonable given the advances in sensor technologies (e.g., see research on a number of low-cost cryptographic schemes for RFID technology [13, 12, 17, 26]).

For the purpose of privacy preservation, we differentiate between two types of triggers: *Instantaneous triggers* are triggers defined over specific events that can be evaluated independent of other events that may (or may not) have occurred in the environment. For instance, a trigger to implement an access decision on whether or not to allow a particular person into a physical space based on their credentials and/or time of day is an instantaneous trigger. Such a trigger requires determining if the event captured by the sensor satisfies the condition of the trigger. The privacy- preserving implementation of such a trigger is relatively straightforward. Either the trigger can be evaluated at the sensor (such as a camera or a RFID reader) where the event is captured, or alternatively, at a trusted hardware connected to the sensor through secure communication channels. Evaluating such triggers does not necessitate logging and storing events. Thus, if sensors (and/or trusted hardware) are tamper-proof, privacy of individuals involved in events is fully preserved.

In contrast to instantaneous triggers, *historical triggers* depend upon other events that may have occurred in the pervasive space. In general, their evaluation requires access to the log of prior events. Consider a scenario where employees for a corporation have associated printing quotas. Enforcing such quotas requires tracking resource usage (i.e., number of pages printed) by individuals. Such a log, while essential for policy realization could also be the source of privacy violation. For instance, using the log, the administrator could determine the printer usage of individuals even when users do not violate their quotas. A privacy-preserving implementation of historical triggers in an environment that is untrusted is significantly complex. One approach is that event logs be encrypted and the triggers be evaluated over the encrypted log representation. [1]. This raises a fundamental challenge of evaluating predicates/conditions over encrypted representations. Even though querying over encrypted data has been studied in a num-

[1] Note that an alternative solution could have been to hide the logs and policies within a secure perimeter which is not visible to any potential perpetrator. However, given the complexity of a generic policy implementing system and the need for administration (which requires human participation) we believe that the completely automated secure computation solution is not a viable alternative

ber of different contexts(discussed in Sec. 6), no solution to the general problem has emerged.

In this paper, we consider a limited class of historical triggers which we refer to as *counting triggers*. The (test) conditions in such triggers is of the form $\sum_{i=0}^{n} x_i = T$, where x_i is the transaction attribute and T is the threshold. Such triggers can be used for both discrete threshold detection (e.g. a person P_1 should not enter the third floor of building A more than five times) and cumulative threshold detection (e.g., an employee's daily travel costs should not exceed more than $100 per day total). We will show in Sec. 2 that many useful access control policies relevant to pervasive spaces can be modeled using a framework of this nature. Using secret-sharing techniques from applied cryptography, we devise protocols to test such conditions in a way that the user data is not accessible or viewable up until the time at which the condition (or set of conditions) are met. Our approach is especially useful in the context of implementing access control policies of the pervasive space. Using our approach, it is guaranteed that the adversary (i.e., people with access to the servers and logs of the pervasive space) does not know any additional information about individuals except what it can decipher from the knowledge of trigger execution. For instance, if a trigger is used to implement an access control policy, the adversary will learn nothing other than the fact that the policy has been violated.

Paper organization. The remainder of this paper is organized as follows. Section 2 describes our model for pervasive spaces, as well as some basic semantics for trigger definition. In Section 3, we present our system framework and detail our protocol for privacy-preserving trigger evaluation. Section 4 discusses further privacy considerations pertaining to our scheme and also describes how the basic scheme can be extended to support other types of triggers. Section 5 discusses implementation and performance as a result of integrating our techniques with PADoC, a framework for privacy-preserving video surveillance. In Section 6 we discuss related work, and conclude in Section 7.

2 A Model for Pervasive Spaces

In this section we define the various entities and terms used to model a pervasive space in our scheme. We envision a pervasive environment as a physical space embedded with a sensing infrastructure which is used to monitor its state. The data captured by the sensors creates an evolving representation of the physical world (observed world) over which the functionalities offered by the pervasive environment are implemented. We assume that the state of the physical world is stored in the form of an event log. *Triggers* are evaluated on this log data and corresponding actions taken, which may be reflected in the physical world. Fig. 3 illustrates the connection between these entities. We define the basic components of the pervasive environment below in further detail. We will use the running example of a hospital setting for illustration purposes.

Physical World. The physical world is modeled as a set of users, spaces and resources.

Users & Groups (U, G): U corresponds to the set of individuals who interact with the pervasive space. Each user belongs to one or more *groups*$\in G$. It should be noted that a user minimally belongs to at least one group (his/her own group). The set of groups have a partial order defined over them, denoted by $\mathcal{G} = (G, \preceq_G)$, which can also be represented by a hierarchy in the form of a *directed acyclic graph* (DAG) on the set of vertices G (Fig. 1). Examples of user groups in a hospital setting could be group of nurses, group of surgeons, outpatients, long-term patients etc. The leaves[2] of the corresponding DAG consists of the individuals, denoted by $i_j, j = 1 \ldots n$.

[2] vertices in a DAG which do not have any outgoing edges

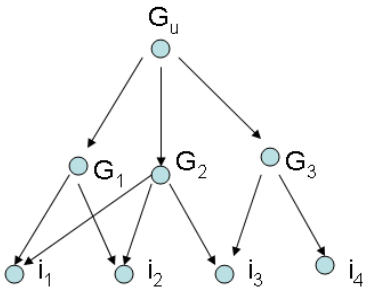

Fig. 1. Group Hierarchy.

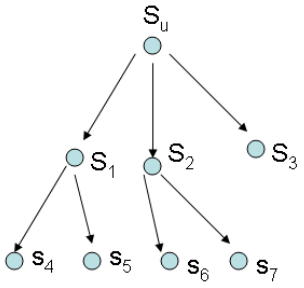

Fig. 2. Spatial Hierarchy.

Space (S): Space consists of a set of physical regions with a partial order defined on them and is denoted by the tuple: $\mathcal{S} = (S, \preceq_S)$. As above, this partial order can also be represented in the form of a hierarchy by a DAG as shown in Fig. 2, e.g. a floor in a hospital might be represented by an internal node in the DAG and all the rooms and wards on the floor by its children.

Resources (R): Resources are accessed within spaces and they form the subject of rules and functions of the pervasive space. Again drawing from our hospital example, the resources could be instruments and machines like the respirator, cat-scan machine, ultrasound machine etc. Again a hierarchy may be defined on the set of resources as well.

Sensing Infrastructure. The purpose of the sensing infrastructure is to monitor the physical world and provide a representation of its state. This can be achieved through a variety of mediums (e.g. video cameras, radio-frequency identification, motion detectors etc.). We do not assume anything about the underlying hardware for the purposes of the model, simply that the sensing infrastructure has the appropriate means to provide desired information about the physical world.

Observed World. The observed world consists of the information collected by the sensing infrastructure in the form of *events* involving users, that are eventually logged and stored. Events are described more formally below.

Event (e): Events are represented as a 5-tuple $e = <u, s, t, r, a>$ where u corresponds to an user, s denotes a region, t denotes a time instant (or interval) and r denotes a resource. a specifies the category of the activity corresponding to an event. The event e mentioned above, is interpreted as a user u has performed activity a, at (during) time instant (interval) t, in space s, with resource r. Activities in general form an ontology and are domain dependent. Due to the fact that we deal with space and resources in our domain, we will limit the activity associated with any event to one element from the set $\{entry, exit, usage\}$ (for simplicity). For example, <`Bob, Operating_Room_2, 3:20PM, ECG, exit`> denotes the event where Bob exited Operating_Room_2 at 3:20PM with an ECG unit. We can also have events of the type <`Alice, Operating_Room_3, 4:20PM, NULL, entry`> which corresponds to an entry event without an associated resource. The event <`John, Room_2, 1:00PM-1:30PM, MRI, usage`> denotes John used the MRI unit, in Room_2 from 1:00PM to 1:30PM. Entry and exit events need not have a resource associated with them, but those of the type "usage" need to have an associated resource.

Following event generation, the trigger-relevant information is committed to a *log*. A generic log format is utilized to log the sequence of events.

$$D_j = LSN \mid ts \mid C_{ID} \mid R_{ID} \mid O_L \mid ACTIVITY$$

Here, LSN is the log sequence number, ts the time-stamp associated with the entry, C_{ID} the credential (e.g RFID) associated with the subject, R_{ID} the region identifier (i.e. the area of physical space in question), O_L the object list or associated set of objects the subject may be accessing (typically these are inventory items which may also be tagged with identification), and finally $ACTIVITY$ type (i.e. entry/exit/usage).

Triggers. Finally, we define the concept of triggers which are defined over the log data gathered from the activity in the physical world. Triggers are modeled as *condition-action* pairs[3]. The condition component of the trigger can be represented as a predicate to be evaluated on the log data. The action component could comprise of any sort of action, such as charging an individual's account, raising an alarm, issuing a warning and so on. Actions are domain dependent and used to represent pervasive functionalities. The trigger *fires* (i.e. action taken) only when the condition is true. In this paper we are interested in count-based triggers. The notion of this class of triggers is formalized below, starting with a few basic definitions.

Basic elements & sets: Basic elements of the system are the entities of the pervasive space, like individuals, resources, spatial regions etc. In our representation, we have 5 classes of basic elements: *individual, space, time, resource* and *activity*. A universal set is defined for each of the above class of elements, denoted $\mathscr{I}, \mathscr{S}, \mathscr{T}, \mathscr{R}$ and $\mathscr{A}$[4] respectively. Elements of $\mathscr{I}, \mathscr{S}, \mathscr{T}, \mathscr{R}$ are denoted by i, s, t, r and a(possibly with subscripts) respectively. A *basic set* is well defined for each of the above classes and denotes any subset of the corresponding universal set. We use notations $\mathcal{I}, \mathcal{S}, \mathcal{T}, \mathcal{R}$ and $\mathcal{A}$ (with possible subscripts) to denote basic sets.

Basic triggers: Basic triggers have simple semantics and are implemented as in the standard *condition-action (CA)* model mentioned above. On occurrence of an event, the trigger condition (predicate) is evaluated and if it is true, the corresponding action is taken. The condition component of a basic trigger consists of a 6-tuple $<\mathcal{I}, \mathcal{S}, \mathcal{T}, \mathcal{R}, \delta, \mathcal{A}>$ where $\mathcal{I}, \mathcal{S}, \mathcal{T}, \mathcal{R}, \mathcal{A}$ are basic sets from the classes *individual, space, time, resource* and *activity* respectively[5] and δ is the positive integer denoting the threshold for the trigger. The condition of the basic trigger specified above, is true if and only if the following predicate P is true:

$$P : |\{e | e = (u, s, t, r, a)\}| == \delta \text{ where } e \text{ is an event}$$
$$u \in \mathcal{I}, s \in \mathcal{S}, t \in \mathcal{T}, r \in \mathcal{R}, a \in \mathcal{A}$$

Example: Consider the basic trigger, with reference to Figs. 1 & 2:
$tr = <G_1, s_4, (7am - 7pm), \{r_1, r_2\}, 5, \text{“}usage\text{”}>$[6] This trigger for instance, needs to detect the condition when, the number of times members of group G_1, in region s_4, between the times $7am$ and $7pm$, have used resources r_1 or r_2 a total of 5 times.

[3] A condition-action trigger model is similar in power to a more typical event-condition-action (ECA) model since complex events can be transformed into corresponding condition classes over the event log. Our model can be viewed as a ECA model in which events correspond to simple events captured (5 tuple format discussed) by the sensing environment

[4] $\mathscr{A} = \{\text{“}entry\text{”}, \text{“}exit\text{”}, \text{“}usage\text{”}\}$

[5] We have referred to the hierarchy on class of individuals as the “group hierarchy” earlier. The leaves of this hierarchy, denoted by a DAG ($\mathcal{G}$), correspond to individuals in $\mathscr{I}$. We will use the label $label(n)$ of a node $n \in \mathcal{G}$ to denote the basic set of leaves (individuals) rooted at n in $\mathcal{G}$

[6] this is a more concise notation for the basic trigger $<\{i_1, i_2\}, \{s_4\}, \{(7am - 7pm)\}, \{r_1, r_2\}, 5, \{\text{“}usage\text{”}\}>$

High-level trigger specification: Many functionalities and rules for a pervasive space are applicable to many groups of individuals, across multiple regions, over multiple time periods for a multitude of resources. Such functionalities are enabled by a set of basic triggers that can be represented concisely in the form of a *high-level* trigger. Consider the example below.

Example: Consider the rule: "If a doctor or a member of the medical staff issues more than 10 ampules (each) of morphine or codeine from the pharmacy in a single day, then flag that activity". This rule may be realized by implementing the following set of basic triggers, where there are m doctors and n medical staff members (a total of $2(m+n)$ triggers):

$<d_1$, Pharmacy, (12AM-11:59PM), morphine, 10, "exit"$>$
$<d_1$, Pharmacy, (12AM-11:59PM), codeine, 10, "exit"$>$
:
$<d_m$, Pharmacy, (12AM-11:59PM), morphine, 10, "exit"$>$
$<d_m$, Pharmacy, (12AM-11:59PM), codeine, 10, "exit"$>$
$<ms_1$, Pharmacy, (12AM-11:59PM), morphine, 10, "exit"$>$
$<ms_1$, Pharmacy, (12AM-11:59PM), codeine, 10, "exit"$>$
:
$<ms_n$, Pharmacy, (12AM-11:59PM), morphine, 10, "exit"$>$
$<ms_n$, Pharmacy, (12AM-11:59PM), codeine, 10, "exit"$>$

We define a high-level trigger as the ordered set of 6-tuples $<\mathfrak{I}, \mathfrak{S}, \mathfrak{T}, \mathfrak{R}, \{\delta\}, \mathfrak{A}>$ where the elements $\mathfrak{I}, \mathfrak{S}, \mathfrak{T}, \mathfrak{R}$, and $\mathfrak{A}$ of the tuple, each represent a "set of basic sets" from the classes *individual, space, time, resource* and *activity* respectively. The set $\{\delta\}$ is a singleton set of the positive integer denoting the threshold. Such a high-level trigger denotes the set of basic triggers which are the elements of the 6-way cartesian product of the sets $\mathfrak{I}, \mathfrak{S}, \mathfrak{T}, \mathfrak{R}, \{\delta\}$ and $\mathfrak{A}$. For example, in case of the above rule, the complete set of basic triggers can be defined by the single high-level trigger:

$<$ DOCTORS* $\cup$ MED_STAFF*, {{Pharmacy}}, {{12AM-11:59PM}}, {{morphine}, {codeine}}, {10}, {{"exit"}}$>$

where DOCTORS$^* = \{\{d_1\}, \ldots, \{d_m\}\}$ is a set of basic sets on the group of doctors and MED_STAFF$^* = \{\{ms_1\}, \ldots, \{ms_n\}\}$ is a similar set on the group of medical staff members. Therefore, high-level triggers provide a concise representation and an easy method of generating basic triggers that implement many useful functionalities in a pervasive space.

So far we have introduced the basic entities associated with the pervasive space, their definitions and formalizations. These will be used to describe our system and its operational characteristics in subsequent sections. To reiterate, the central goal of this paper is to design a system that can capture events and evaluate triggers efficiently, in a privacy-preserving manner. Though we will restrict our trigger-semantics to that of count-based predicates, this class of triggers is able to capture many functionalities of interest in a pervasive environment as we have seen in this section.

3 A Framework for Privacy-Preserving Pervasive Environments

In this section we describe our framework for privacy-preserving data collection in trigger based pervasive environments. In our approach, events generated by the sensing

infrastructure are encrypted, and triggers are evaluated over the encrypted data. For the purpose of encryption, we require users to carry keys which serve as credentials and help the system link them to the events they generate via sensors. A user also has group keys for groups it is a part of (i.e. each member of the group has a copy of the group key). This (private) key is stored securely with the individual (e.g. embedded in a tag or card).

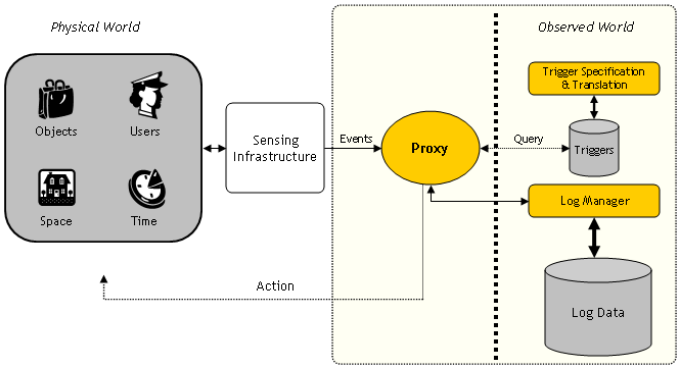

Fig. 3. Framework for privacy-preserving predicate evaluation in pervasive spaces.

3.1 System Framework

The system framework is illustrated in Fig. 3 which highlights the major components. The primary components besides the sensing infrastructure consists of a proxy, trigger manager and a log manager whose functionalities and roles are described below.

- **Proxy:** The proxy is a trusted computational unit that sits between the sensing infrastructure and the untrusted pervasive environment. The task of the proxy is to encrypt the events generated by the sensing infrastructure prior to those events being logged. Furthermore, it participates in protocols with the other components of the untrusted pervasive environment (viz. trigger and log managers) to store and evaluate triggers over the encrypted representation [7]. Depending upon the nature of the sensing environment, a proxy may be a sensor itself (if it has sufficient computational power). Alternatively, it may be a dedicated piece of secure hardware connected to a set of sensors to enable secure logging and trigger evaluation based on sensor-generated events.
- **Trigger Manager:** The trigger manager is responsible for handling all triggers defined for the pervasive space. It enables users (e.g. administrators of the space) to specify policies in the form of high-level triggers (as discussed in Sec. 2). Such triggers are translated by the trigger manager into specific basic triggers on events. Information about these basic triggers is maintained in a trigger table, which is discussed in detail later in this section.
- **Log Manager:** The log manager is responsible for interacting with the proxy in performing secure log entry generation and storage, as well as trigger evaluation.

[7] The proxy is also responsible for evaluation of instantaneous triggers on the incoming event stream

The framework supports the following operations; (a) *trigger specification:* performed by administrators and entered via the trigger manager which parses and translates them before storing the final form in the trigger table, (b) *event processing:* events raised by the sensing infrastructure are parsed and processed by the proxy which initiates a protocol with the trigger/log managers, (c) *trigger processing:* this consists of two parts, trigger evaluation and execution (where an action is invoked as a result of the trigger). The proxy and the log manager work together in transforming (into its encrypted representation) and storing (encrypted representation with meta-data) the incoming event stream. Triggers are then evaluated over this representation.

In the remainder of this section, we describe details of the data representation maintained by the various components of the system and the protocols used for logging and trigger evaluation. We begin by first describing an approach to evaluating triggers over encrypted logs.

3.2 Trigger Evaluation Over Encrypted Data

We use an adaptation of Shamir's threshold scheme [23] for evaluating triggers over encrypted data (see Sec. 6 for details). Recall that in our system we have restricted ourselves to counting triggers, in which conditions test, if the count of events of a specific type is above a certain threshold. For instance, a trigger <MED_STAFF$_1$, OP_THEATRE, 7am-7pm, NULL, 5, "entry"> will detect the state when MED_STAFF$_1$ has entered the operating theater between 7am to 7pm a total of 5 times. To enable evaluation of such triggers , along with the encrypted log for an event E, we also store a *secret share* which is of the form (x, y) where $y = f(x)$, f is a polynomial function of degree $\delta - 1$, x is a random value (point) from the domain of f, and δ is the threshold associated with the trigger. For instance, in case of the trigger above, $\delta = 5$, and f will be a degree 4 polynomial. Note that a particular event E may invoke multiple triggers. In such a case, the log for E will store a secret share for each of the triggers invoked (the exact representation of the log and how it is generated will become clear later).

When δ events that meet the conditions of the trigger have occurred, the log manager knows of δ secret shares using which it can determine the polynomial function f. More specifically, the log manager can compute the value of f at $x = 0$. This value is chosen (while constructing the logs) to be the key used to encrypt the log record. Hence, once the threshold for the trigger is met, the log manager can determine the secret key, and hence decrypt the cipher-text containing the details of the event that lead to the realization of the trigger condition. Notice that in the above strategy, while the trigger threshold is not met, the log manager does not have enough information to determine the value of $f(0)$ and hence does not have access to the key used to encrypt the log record.

Our framework utilizes the aforementioned approach for evaluating triggers over encrypted log records. In the following three sub-sections, we provide details of the data structures maintained by the various components, and the protocols used for event storage and trigger evaluation.

3.3 Data Representation

The server-side contains the transformed event data gathered by the sensing infrastructure. All data here is stored in encrypted form. Fig. 4 shows a snapshot of how the data is organized. The two major components are the transaction log and trigger table. The transaction log simply stores the events in an encrypted representation (with respective

meta-data) and the trigger table contains the attributes of the basic triggers which are being enforced (in plain-text). The structure for the trigger table is straightforward, given the trigger model specified in section 2. For instance, consider the trigger table in Fig. 4. For example, row T_1 in the figure denotes the trigger that fires when the condition: the number of times some member of group G_1, from region R_2, between times 8:00AM and 10:00AM have taken out resources A or D, equals 10. The first time an event occurs, that leads to a change of the state of a trigger condition, a new "category" of transactions needs to be initiated in order to log any subsequent events that might eventually lead to realization of the condition-predicate. We call each new sequence of transactions like this, a "session". Note that the occurrence of an event may affect more than one trigger in the trigger table and therefore those different sessions will be initiated (updated) in the transaction log.

With each trigger T in the trigger table, we associate a session key whenever an event corresponding to the trigger occurs. When the first event corresponding to the event occurs, a new session key is generated. Session keys are used to encrypt the log data corresponding to an event that affects that trigger. These keys are stored in a separate table called the *session-key table*, whose schema is of the form, *(Trigger-id, session-key, valid bit, random noise string)*. The rows of the session-key table are stored encrypted under the group-key of the group of the trigger (each basic trigger is associated with a unique group). Note that we store the session key for a trigger and the trigger (attributes) itself in two separate tables to prevent the trigger manager from learning the correspondence between the key and the trigger. Preventing the trigger manager from learning such a correspondence is important to avoid disclosure due to linkage of events with log updates. The session key is random key drawn from $\mathbb{Z}_p$ (where p is a large prime number and $\mathbb{Z}_p$ denotes the corresponding prime-field). The row of the session-key table, when decrypted, allows one to determine which trigger the session-key corresponds to using the *Trigger-id*. The *valid bit* if set to "1", signifies that there is an active session for the trigger in the log table (that is it has at least one event that affects the trigger condition). The *random-noise* attribute is simply a constant length noise-string appended every time this row needs to be written back on the server. We will elaborate on this representation and disclosure due to linkage in the context of an example later on.

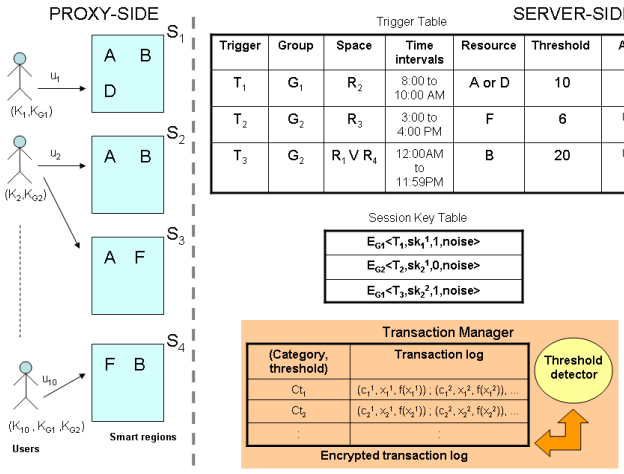

Fig. 4. Data representation.

The transaction log scheme is of the form, *(category, encrypted_tuples)*, where the category of each row is unique and corresponds to a distinct session of transactions. For every active trigger there is a unique session. Once a particular threshold has been met, a new session is set up for the next iteration of the trigger (if it is recurring). Therefore, a new category must be generated. *Category* is nothing but an unique label for the session generated using the hash function H, for example:

$$Category(Trigger_i) = H(\text{session-key}(Trigger_i))$$

The *encrypted_tuples* are simply the event related data and meta-data that has been encrypted by the proxy using the session key and will be described in detail in the forthcoming section.

3.4 A Protocol for Log Generation

When a user u generates an event, the proxy contacts the server and gets the complete *Trigger-table* and the *Session-key table*. The set of triggers that are affected by the event are identified from the *Trigger-table* and the remaining triggers are discarded. Now to get the corresponding session keys, the proxy attempts to decrypt each row of the session-key table using keys of each of the groups the individual belongs to. Assuming no two group keys are the same, the proxy is able to decrypt only those rows which correspond to the individual's groups. Therefore, in at most one scan of the trigger and session-key tables, the proxy is able to determine the set of triggers which have been affected by the event and also decrypt the corresponding rows in the session-key table. For a trigger, if the corresponding row in the session-key table has the valid-bit set to "0", a new session-key sk is generated and the bit is set to "1". Else if the the bit is already "1", it signifies that the session is active, and the current key is used for encryption. The algorithm depicted in Fig. 5 describes the steps carried out by the proxy.

For each trigger affected by event e, the associated proxy constructs the 4-tuple called *log-tuple*:(*category-tag*, c, x, $f(x)$), where the entities of the tuple are described below.

- *category-tag*: as the name suggests, it is a tag for the category to which the event belongs. There is a unique category tag for each session, given by = H(sk) where H is a cryptographically strong hash function such that H: $\{0,1\}^* \rightarrow \{0,1\}^l$. sk is the session key for the event that is obtained from the *session-key table* on the server.
- x: a random point taken from the domain of the function f i.e. $\mathbb{Z}_p$. For every event in a session, x needs to be a distinct point. p is a large prime with $\|p\| \simeq l$
- $f(x)$: the value of f at x, where f is a degree '$\delta-1$' polynomial where δ is the threshold in the trigger condition. Polynomial f is computed as follows: $f = \Sigma_{i=0}^{i=\delta-1} a_i x^i \mod p$, where a_i are the coefficients which are elements of $\mathbb{Z}_p$ and a_0 is the secret session key (that needs to be hidden till δ events have occurred). The pairs $(x, f(x))$ are also called the "random shares" of the polynomial f. Coefficients, $a_i \forall i = 1 \ldots \delta - 1$ are computed as $a_i = H'(sk, i)$, where H' is a cryptographically strong hash function, such that H: $\{0,1\}^* \rightarrow y \in \mathbb{Z}_p$.
- c: Enc_{a_0}(e), encryption of event e under a_0 (the session key).

Note that the only information learned by the server about any two log-tuples is whether or not they belong to the same category (i.e. affect the same trigger or not).

Producing Log Data on Event Generation
BEGIN
1. **On Generation of** $Event$, $e = <u, s, t, r, a>$ **Do**
2. $Groups(u) \leftarrow$ groups to which u belongs
3. retrieve *Trigger-table* from server $\rightarrow TT$
4. retrieve *Session-key table* from server $\rightarrow ST$
5. $Trig \leftarrow \{t | t \in TT, e$ affects $t\}$
6. $Keys \leftarrow \{k | k \in ST, k.$Trigger-id $\in Trig\}$
7. /* Assume rows in $Trig$ correspond to rows in $Keys$ */
8. let $n \leftarrow \mid Trig \mid$ /* Size of $Keys$ is also n */
9. **For** $i = 1$ to n **Do**
10. **If** $Keys[i].$*valid-bit* $== 1$ **Then**
11. $sk \leftarrow Keys[i].$*session-key*
12. randomly pick $x \in domain(f)$ and compute $f(x)$
13. $log \leftarrow < H(sk), E_{sk}(m), x, f(x) >$
14 /* m is event-data, $H(sk)$ is the category tag */
15. send log to server
16. **Else** /* create new session key */
17. $sk \leftarrow Trig[i].Group.key() \oplus$ random-pad
18. Generate polynomial f of degree $d = Trig[i].Threshold - 1$
19. randomly pick $x \in domain(f)$ and compute $f(x)$
20. $log \leftarrow < H(sk), E_{sk}(m), x, f(x) >$
21. send log to server
22. **End If**
23. **End For**
END

Fig. 5. Protocol for log generation.

Also, the cryptographically strong hash functions ensure that two log-tuples corresponding to the same trigger but belonging to different sessions, cannot be linked to each other in any manner by looking at their representations. This ensures that an adversary on the server-side cannot determine the identity of the triggers corresponding to the active sessions in the log (i.e. sessions for which the trigger condition has not been met).

3.5 Trigger Evaluation

The log-manager receives the 4-tuples from the proxy and stores them indexed on the category. That is with each category tag, it stores the 3-tuples (c, x, $f(x)$). For every transaction, the log manager incrementally interpolates the new share (i.e. the $(d+1)^{th}$ for the session) and constructs the unique d-degree polynomial that interpolate this shares (efficient implementation of this repeated interpolation is discussed in Sec. 5). It returns to the proxy the value of the polynomial computed at $x = 0$, which is value of the coefficient a_0 for the polynomial. It is almost impossible, unless the log-manager receives all the δ shares (corresponding to a trigger with threshold δ), that a server-side adversary will be able to derive the true value of a_0, which is the secret session key. Therefore the returned value will correctly tell the proxy when the number of events affecting the trigger has reached the threshold number δ in the trigger condition[8]. At

[8] It should be noted that depending on the setup, it is not always necessary to send the entire cipher-text back to the proxy unless it explicitly needs to be revealed upon violation

this point the state of the trigger is potentially disclosed to the server. The diagram in Fig. 6 shows the complete protocol that is carried out between the proxy and the server.

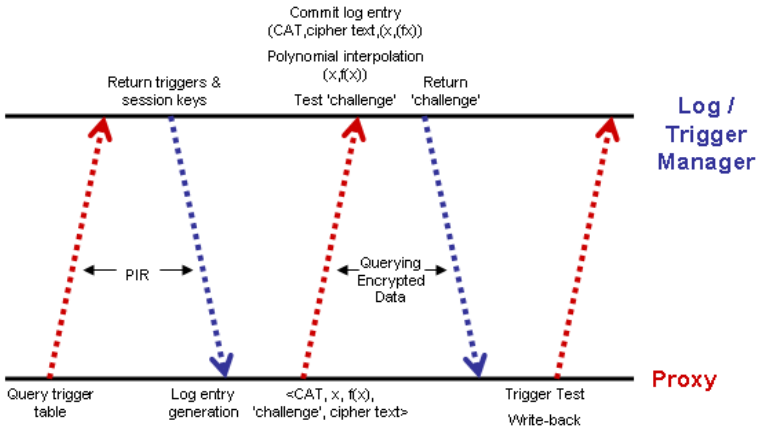

Fig. 6. Proxy-server protocol for trigger evaluation.

The last step of the trigger-evaluation protocol consists of a possible write-back operation on the server. There are two cases when a write-back must happen: First when a trigger condition has been met, the corresponding session needs to be "closed". This requires the proxy to set the valid-bit to "0" for the corresponding entry (row) in the session-key table. The proxy sets the the bit to "0", replaces the noise string by a new random noise string and writes back the encrypted row (encrypted under the group key corresponding to the trigger) into the session-key table on the server. Second, when a session is activated, i.e. a trigger is retrieved which has the valid-bit set to "0", a new secret session key is generated (by randomly picking a point from $\mathbb{Z}_p$ and the valid-bit is set to "1". This new session-key information has to be written back into the session-key table as well. Since both these write operations are visible to the server, we need to disassociate the session-key information from the triggers they correspond to, else information regarding the status of a trigger might be disclosed to the adversary who monitors these write operations. Therefore our trigger-table contains no session-key information and is a static one. Write operations are made only in the session-key table.

4 Generalizing & Extending the Scheme

In this section we discuss some issues related to balancing information disclosure and efficiency in the log-generation and trigger-evaluation phases of the protocol shown in Fig. 6 of section 3.

4.1 Trading Privacy for Efficiency

As noted in Sec. 3, there are a number of privacy issues that arise during the protocol for log generation. Specifically, this occurs during the phase where the proxy queries the

of the relevant trigger. In our basic protocol, all that is required is that the log manager interpolate the polynomial on each iteration, and attempt to decrypt a 'challenge' that the proxy transmits and return that to the proxy. If it has been decrypted correctly (i.e. not garbage) the proxy will know that the trigger has been violated and can take appropriate action.

trigger table on the server side to retrieve the relevant triggers for evaluation against incoming events. In our system, since the triggers contain identifying information, retrieving exact sets of triggers affected by an event, can potentially disclose the identity of the individual involved as well as the nature of the event. The problem we face in this situation then becomes similar to that of private information retrieval (PIR) [6, 5, 15, 4]. In PIR the data belongs to the server (i.e. trigger table) and the individual (here proxy acting on behalf of the individual) issues the query. The goal is to retrieve the solution set with minimum overhead of communication without revealing the query to the server. The technique we outlined in the basic version of the protocol was that of sending the entire trigger table each time the protocol was invoked for an incoming event. This has been theoretically shown to be the only optimal technique available for single-server PIR [6], if absolute privacy is to be guaranteed. Though this solution has a high overhead in terms of communication and proxy-side processing, our experimental results show that even this technique performs fairly reasonably, given the rate of event generation we expect in our target application.

There are a number of fairly simple relaxations to this privacy-optimal solution which lets one trade-off some privacy for efficiency (if required). For instance, we can reduce the communication and processing overheads by only retrieving the set of triggers associated with a region (or set of regions) the proxy is responsible for. Secondly, we can easily employ caching at the proxies such that querying on each event is not necessary. This requires some (trusted) storage on the proxy server but may be a reasonable assumption based on the trust requirements of the application.

One necessary (though not sufficient) condition is that the size of the retrieved set should be bounded from below so as to ensure a minimum level of anonymity regarding the identity of the individual involved in an event. Anonymity is not the only concern in our application, in general any information disclosure regarding the nature of the triggers being queried has to be minimized. Techniques for achieving k-anonymity [19] and other means of disclosure control in databases [10, 2] can perhaps be adapted to balance disclosure and efficiency in querying the trigger table, but these issues are not explored further in this paper and they form the topics of our future work.

4.2 Generalizing Basic Triggers

In this paper we presented a solution for evaluating count-based triggers in a privacy-preserving manner. However, this work can be extended to include other types of triggers. A logical next-step is the evaluation of *cumulative* thresholds. Consider an example where we wish to enforce a spending quota for office personnel without having to expose the exact amount spent corresponding to each individual unless they exceed the quota. In this case, each event can possibly advance the trigger by variable amount (rather than one unit like that of the basic triggers discussed previously).

We can extend Shamir's basic threshold scheme to cumulative thresholds as was done in work on probabilistic escrow of financial transactions [11]. While the scheme generalizes to the case where each event can increment the count by more than a single unit, the detection of the condition satisfaction becomes probabilistic in nature. That is, there is a small chance that the condition will be wrongly evaluated to be true even when the threshold has not been reached and vice-versa. The probability of this disclosure is proportional to the difference between the true count and the threshold, which makes it favorable for implementation in our scheme. The authors conceptually split the threshold δ into d parts. Whenever the user performs a transaction for some value $t \leq \frac{T}{d}$ (higher values need to be subdivided into pieces of at most $\frac{T}{d}$ value) and

submits the corresponding event to the server, the user must also reveal one share of his/her encryption key with probability exactly equal to $\frac{d}{T}$ * t. If the probability of submitting a share is set in this way, then regardless of the size t of the individual transactions that make up a cumulative amount A, the expected number of shares generated by a user who generates n = $\frac{A}{t}$ transactions will be n($\frac{d}{T}$ * t) = $\frac{A}{T}$*d, which is independent of t.

Approaches to support more general triggers that can detect more complicated conditions (as in SQL queries) over the log data could be developed by adapting recent research for querying encrypted data [9, 10, 2]. Such techniques have been proposed in the context of database outsourcing, and typically require some processing and/or maintenance of meta-data as well as log processing on the proxy-side. In our current setup since neither the proxy nor the individuals are assumed to have significant storage or computational power available to them, these techniques are not implementable in their current form.

A limitation of our work is that triggers are associated with only one group. Checking predicates/conditions belonging to multiple groups requires additional techniques. Currently there are no straight-forward practical solutions that exist unless we can assume that group keys are somehow available to the proxy (during trigger processing) even though no individual from that group is involved in the event. Such an assumption will severely degrade the security of the system and therefore this avenue is not explored further in this work. Deriving solutions to multi-group triggers remains a direction for our future work[9].

5 Implementation & Performance

In this section, we provide details of how our techniques for privacy-preserving trigger evaluation were utilized in the context of a fully functional video surveillance application for pervasive spaces. We further conduced experiments to study; (a) the power of our model in expressing the desired functionality (in the form of triggers) for a specific environment (i.e. video surveillance) and (b) the scalability of our approach to this application under varying conditions. The general techniques provided in this paper can be utilized for many pervasive applications, however the goal of our experimentation was to explore whether the performance of these techniques would be acceptable for our chosen application scenario. We first describe our PADoC framework for video surveillance and show how we integrated our techniques on privacy-preserving trigger evaluation.

5.1 PADoC: Privacy-Preserving Video Surveillance

In the PADoC framework [28] we adopted a novel approach to combining localization sensors (in particular RFID [10] technology) with traditional video surveillance as a first step toward demonstrating frameworks that collect minimal user information in pervasive spaces and try to preserve the privacy of users as much as possible.

[9] work on multi-party function evaluation [8] might shed light on this problem, but designing practical implementable solutions for these protocols is non-trivial and moreover they make the assumption that parties are always available over a communication channel. This assumption clearly cannot be made in a pervasive scenario such as ours

[10] A RFID (Radio-Frequency IDentification) tag is a tiny, relatively inexpensive device capable of transmitting a piece of static information across a distance. RFID tags are currently in use for mediating access to various regions, however it does not provide enough information to pinpoint the object being tracked within that space.

In this system, access control policies specify the access rights of individuals to different regions and resources of the monitored space (effectively, instantaneous triggers according to the terminology defined in Sec. 2). These policies are defined using a XACML-based (eXtensible Access Control Markup Language) infrastructure. Policy violations are used to trigger the video surveillance subsystem. Video manipulation techniques such as masking and obfuscation are used to preserve the privacy of authorized subjects when the surveillance system is turned on. This system is fully functional and operates in real-time at a target resolution of 320x240 at a framerate of 25fps.

The key concept in PADoC was that the video information was "filtered" according to access control policies such that the privacy of users was not violated in the final, outgoing product. However, the trigger logs being generated by the sensors (e.g. trace of RFID-based activity) was still vulnerable and hence a threat to users' privacy. Our scheme for privacy-preserving trigger evaluation allows us to specify and enforce historical triggers in the context of this application setting. That is, we are able to use sensor-generated event data to enforce triggers that depend on prior event data and feed this information back into the video processing subsystem where it is realized in the form of masking on the real-time video stream. The implementation details of the video subsystem are beyond the scope of this paper, but details can be found in [28, 27].

5.2 Enhancing PADoC with Privacy-Preserving Trigger Evaluation

PADoC inherently utilizes a trigger-based approach to enforcing access control policies. We use our encryption-based representation to construct log entries from the incoming events reported by the RFID and motion sensors. We introduced our notion of a proxy into the PADoC architecture at the point at which the sensor data is collected. The information provided by these sensors is parsed into an access control request and passed to the proxy where the protocol with the trigger/log manager is initiated. High-level policies are specified by administrators (via a web-based interface), and then processed by the trigger manager who translates them into individual triggers and stores them in the trigger table.

Once the proxy discovers that a trigger has been violated, it signals the video subsystem with the particular user (or group) that is the subject of the trigger. The video subsystem will then appropriately mask the individual(s) with the appropriate filter (Fig. 7) (depending on privacy preferences) which is then reflected in the outgoing video stream.

PADoC is a video-based application, therefore it was important to study the impact of our trigger evaluation techniques to see how significant an overhead would be applied to the end-application. What we ideally want is for the decision to be reflected in the video as quickly as possible (i.e. in the form of masking).

The PADoC system is being deployed in the new CAL-IT^2 building at the University of California, Irvine. The entire building has been instrumented with video cameras and various sensors at the entry and exit points. We demonstrate the general class of policies that we are looking to implement in the context of this building (in a privacy-preserving manner).

- **Limiting access to spaces:** These include triggers such as "Person A is allowed to enter Room X between the hours of 3-5PM" or "The Robotics group is allowed access to the server room on their own floor".
- **Protection of resources:** These types of policies allow control over inventory items. For example, "Members of group A are not allowed to bring/take K laptops into/from the conference room".

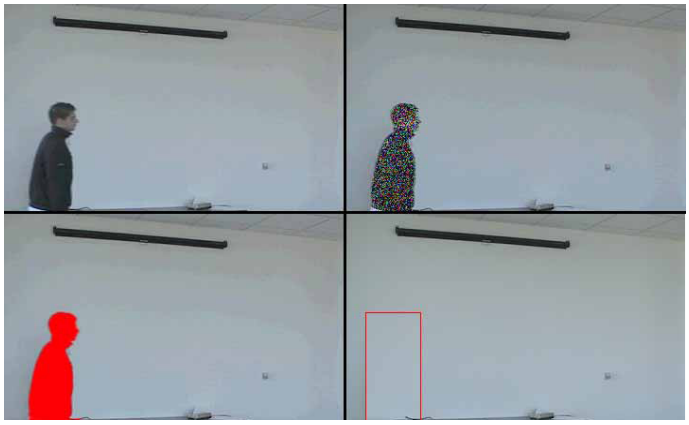

Fig. 7. Privacy-protecting masking techniques for video utilized by our system. They represent different levels of user privacy. A frame of the original video is shown (top-left), followed by a noise/blur filter (top-right). A pixel-coloring approach is shown (bottom-left) followed by a bounding-box based technique (bottom-right) which hides details such as gender, race or even dimensions of the person in question.

- **Monitoring/flagging of suspicious activity:** For example, "Any individual who is present in the lobby area more than 10 times in a 24-hour period should be flagged".
- **Studying human interactions:** Policies can be devised for cross-party interaction. For example, "Raise a flag if the number of (spatial) interactions between the Middleware and A.I. groups exceed 25 in a given week". Now this is really a composite trigger similar to that defined for monitoring/flagging. Such a trigger will be translated to a union of two counting triggers which detect entries by each group into the other's space.

5.3 Experimental Evaluation

In investigating the feasibility of using our trigger-based approach in the context of the video surveillance application, we performed some experiments on the scalability of our techniques. These experiments were done to evaluate the overheads involved in performing both the (a) retrieval of triggers from the trigger database, and (b) overhead of the log generation/evaluation process. We simulated the incoming event stream by a poisson distribution of randomly generated events. These events were similar to those described in Sec. 2, with the various fields (region, time etc.) varied in a random fashion. The input parameters were; (1) the number of groups, (2) the number of triggers, (3) the number of triggers per event (T/E) - intuitively this is on average how many triggers are affected by any particular event. For example if an individual user is a member of multiple groups, then each event generated by that user may affect multiple triggers. This metric captured this functionality. (4) the total number of events for each iteration of the experiment and finally (5) the maximum definable threshold (effectively the size of the polynomial).

We utilized the Java Cryptographic API (SDK 1.4.2) for implementing the cryptographic primitives in our protocol. Blowfish was utilized for encryption using the default key size of 16 bytes. A six digit prime number was used for modulo arithmetic. All experiments were performed in a LAN environment on Pentium IV 2.8 Ghz

PCs with 1GB of RAM running the Windows XP operating system. Communication between the proxy, trigger and log managers were implemented via Java RMI.

Our experiments simulated event generation by a poisson distribution with parameter λ. We measured the mean event processing times when the rate at which events are generated was varied. In order to simulate different delays for event generation, events were generated, queued and then retrieved by another process in a random fashion. One process simulated the generation of events. This was achieved by randomly generating events based on various parameters defined by our model (regions, time etc.). Once the event is retrieved by the proxy it initiates the protocol with the server. The server keeps these requests in a queue and processes them in FIFO order. So when the events are produced at a faster rate than the server's processing rate, the total waiting time adds up for the events that are in the queue. But when the events are produced at a slower rate than the server's processing rate, then the queue is empty most of the time, resulting in almost constant processing delay (which is equivalent to the server's interpolation time for a log message).

In our implementation, the log manager requires interpolating the value of a new polynomial (of degree one more than than the previous) at $x = 0$ for every iteration (i.e. on arrival of a new share). Using the basic Lagrange interpolation formula, the log manager needs to do $O(n^2)$ arithmetic operations in each iteration if there are n total shares, but this can be reduced to $O(n)$ operations by maintaining some intermediate data in the form of a divided-difference table. We implement the techniques suggested by Knuth [14] and describe it in the appendix.

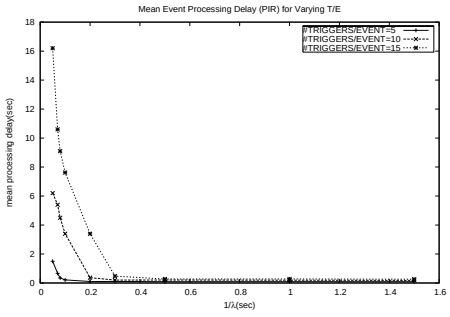

Fig. 8.

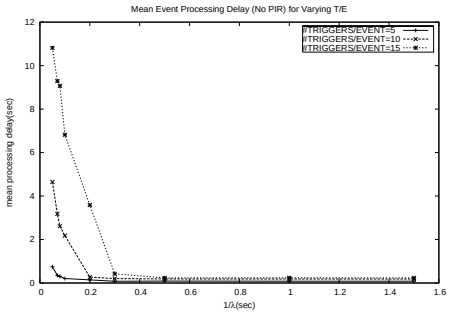

Fig. 9.

Fig. 8 and 9 show the mean processing delay for different T/E values. The parameters for these experiments were; number of events = 100, size of trigger DB = 100, number of groups = 10, maximum threshold = 15. It should be noted that the delay overhead is less in the absence of PIR (in Fig. 9) compared to the case with it which intuitively makes sense as the proxy has to receive the entire trigger database for processing each event.

Fig. 10 shows the case where the delay overhead is affected by the size of the trigger database (with PIR being utilized). This is an extreme case where all triggers are downloaded from the database and hence processing scales up with more data to be communicated. Finally, Fig. 11 shows a comparison of the various PIR approaches we can adopt for the transfer of triggers to the proxy. The tradeoff here is the level of privacy one may possibly give up by (for example) accessing the trigger every time an event comes in for processing. The second set of results used identical parameters to the first set except for the fact that the number of groups was reduced to 10

One common theme from all the presented experiments was that the processing delay becomes almost constant once the average rate at which events being generated crosses a threshold. From our comprehensive experimentation we estimated τ to be

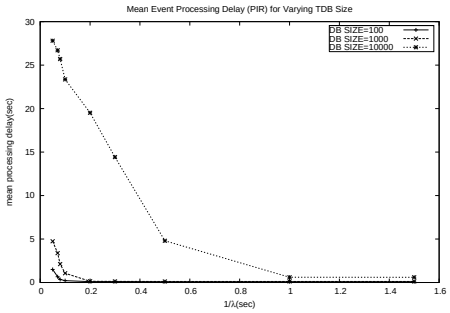

Fig. 10.

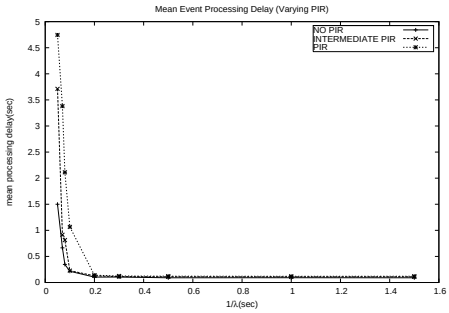

Fig. 11.

around 200ms-1sec. Given the human nature of the type of environment we are targeting we feel these types of delays are reasonable even for large pervasive systems.
.

6 Related Work

There is a large body of work in the area of systems for pervasive computing environments [18]. An example of a system designed inherently to enable pervasive computing is GAIA [21]. GAIA is a component-based middleware operating system that provides a generic infrastructure and set of common interfaces that assist in constructing pervasive computing environments. GAIA's security infrastructure offers a dynamic, context-aware security architecture that allows it to utilize various types of authentication devices. It also provides the ability to create and enforce access control policies for different configurations of the smart space (e.g. support for hierarchical, group-based access control). Other systems such as CRICKET [20], tackled the problem of localization for embedded sensor and mobile context-aware applications.

The importance of privacy to pervasive computing environments for system designers and users has been raised in [7]. Langheinrich [16] stated the danger current pervasive systems research faces in continuing on without considering privacy as an inherent part of system design. Research on privacy for pervasive computing environments has been looked at from multiple angles. For example, the MIST system [3] combines hop-to-hop routing based on handles with limited public-key cryptography to preserve privacy from eavesdroppers and traffic analyzers. Privacy concerns evolving from the interactions of users has been studied in the context of trust negotiation [22]. In that work the focus was on protecting the privacy of individuals involved in online negotiation. Our previous work on the PADoC system [28, 27] sought to address issues in preserving the privacy of users in the context of real-time video surveillance. Here the focus was on protecting the privacy of users appearing in video surveillance feeds, via the information provided by various sensors paired with access control policies defined for the users. To the best of our knowledge the work presented in this paper is the first of its kind that proposes solutions for tackling privacy issues between users and the pervasive environment at the data management layer.

Our solution is based on trusted-third party computation via a proxy. The use of trusted-third parties for privacy-preserving evaluation of value and multi-source predicates was hinted at in [1]. We envision that the functionality of this third-party can be pushed to an individual sensor (or sensors) as the hardware become more and more powerful. Current research in the area of RFID sensors have proposed low-cost cryptographic techniques [13, 12, 17, 26] that facilitate tamper-proof processing at the level of the sensor itself. Eventually, increased hardware support for privacy-preserving

schemes will further assist in relaxing the requirements on the server and allow even for the evaluation of even more powerful triggers.

The strategy for evaluating triggers without access to logs in plaintext form can be viewed as a direct adaptation of the well known Shamir's threshold scheme [23] for the purpose of trigger evaluation. Shamir's $\lambda-$threshold scheme is a method of sharing a key K among a set of participants (size λ) in such a way that λ participants can compute the value of K together, but no group of less than or equal to $K-1$ participants can do so. In Shamir's strategy, the value of K is chosen by a special participant who distributes secret shares (similar to our strategy) to each participants. The K participants can pool all shares in order to compute the value of K as a function of the shares they collectively hold. Privacy-preserving query evaluation for relational data [9, 10, 2], and text [24] has also been studied in recent research

7 Conclusions

In this paper, we address the privacy challenges that arise in pervasive spaces when environmental data (captured via sensing infrastructures) is collected and stored. Such data is often logged to implement various functionalities such as access control and resource management in pervasive spaces. If logged data contains information about users in the space, a privacy violation may occur. We propose a trigger-based middleware architecture for pervasive spaces; using which a wide range of pervasive applications can be realized. In such an architecture, privacy preservation is achieved by encrypting the logs. This however, raises a fundamental challenge - that of evaluating triggers over encrypted data. We develop techniques for privacy-preserving trigger evaluation for a special class of triggers (i.e. *counting triggers*), using which a variety of pervasive functionalities can be built.

We integrated our scheme into PADoC, a fully functional video surveillance system for pervasive spaces that we have developed in our previous work. Using the integrated framework, we were able to enforce a wide variety of policies that arise in video surveillance applications. Furthermore, through experimentation, we demonstrated that the overheads caused by evaluating triggers over encrypted logs are within acceptable ranges for application deployment. Finally, we also provide a systematic approach to tradeoff privacy for reduced overheads. We view our work as the first step towards a grandiose goal of realizing privacy-preserving pervasive environments. Our work has opened up numerous challenges. Some examples of our future work include techniques to improve the performance and scalability of our approach and extending our approach to deal with more general classes of triggers, as well as triggers over multiple groups

References

1. Agarwal, G., Bawa, M., Ganesan, P., Garcia-Molina, H., Kenthapadi, K., Mishra, N., Motwani, R., Srivastava, U., Thomas, D., Widom, J., and Xu, Y. Vision Paper: Enabling Privacy for the Paranoids. In *30th Conference on Very Large Databases (VLDB 2004)* (2004).
2. Agrawal, R., Kiernan, J., Srikant, R., and Xu, Y. Order preserving encryption for numeric data. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data* (2004), ACM Press, pp. 563–574.
3. Al-Muhtadi, J., Campbell, R., Kapadia, A., Mickunas, M. D., and Yi, S. Routing through the mist: Privacy preserving communication in ubiquitous computing environments. In *ICDCS '02: Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)* (2002), IEEE Computer Society, p. 74.

4. Castner, A. Survey of single database private information retrieval systems. Tech. rep., University of Maryland, 2002.
5. Chor, B., and Gilboa, N. Computationally private information retrieval. In *ACM Symposium on Theory of Computing* (2000), vol. 32.
6. Chor, B., Kushilevitz, E., Goldreich, O., and Sudan, M. Private information retrieval. In *36th Symposium on Foundations of Computer Science* (1998), vol. 45.
7. et. al, R. C. Towards Security and Privacy for Pervasive Computing. In *International Symposium on Software Security (ISSS 2002)* (2002).
8. Goldreich, O. Secure Multi-Party Computation. Tech. rep., available from http://www.wisdom.weizmann.ac.il/ oded/pp.html, 2002.
9. Hacigumus, H., Iyer, B., and Mehrotra, S. Efficient execution of aggregation queries over encrypted databases. In *International Conference on Database Systems for Advanced Applications (DASFAA)* (2004).
10. Hore, B., Mehrotra, S., and Tsudik, G. A privacy-preserving index for range queries. In *International Conference on Very Large Databases (VLDB)* (2004).
11. Jarecki, S., and Shmatikov, V. Probabilistic escrow of financial transactions with cummulative threshold disclosure. In *Financial Cryptography (to appear)* (2005).
12. Juels, A. Yoking-Proofs for RFID tags. In *First International Workshop on Pervasive Computing and Communication Security* (2004).
13. Juels, A., and Pappu, R. Squeling Euros: Privacy Protection in RFID-Enabled bank notes. In *Financial Cryptography* (2003), vol. LNCS 2742, Springer-Verlag, pp. 103–121.
14. Knuth, D. E. *The Art of Computer Programming, Volume 2/Seminumerical Algorithms*. Addison-Wesley Publishing Company, 1968.
15. Kushilevitz, E., and Ostrovsky, R. Replication is not needed: Single database computationally private information retrieval(extended abstract). In *Proceedings of 38th IEEE symposium of Foundations of Computer Science* (1997), pp. 364–373.
16. M. Langheinrich. Privacy by Design: Principles of Privacy-Aware Ubiquitous Systems. Presented at ACM UbiComp 2001, Atlanta, GA 2001.
17. M. Ohkubo, K. S., and Kinoshita, S. A Cryptographic Approach to a Privacy Friendly Tag. Tech. rep., NTT Laboratories, 2003.
18. MIT Project Oxygen. Pervasive Human-Centered Computing. http://oxygen.lcs.mit.edu/.
19. P., S., and L., S. Protecting Privacy when Disclosing Information: k-Anonymity and Its Enforcement through Generalization and Suppression. Tech. rep., As technical report, SRI International, 1998.
20. Priyantha, N. B., Chakraborty, A., and Balakrishnan, H. The cricket location-support system. In *Mobile Computing and Networking* (2000), pp. 32–43.
21. Roman, M., and Campbell, R. H. Providing middleware support for active space applications. In *ACM/IFIP/USENIX International Middleware Conference (Middleware 2003)* (2003).
22. Seamons, K. E., Winslett, M., Yu, T., Yu, L., and Jarvis, R. Protecting Privacy during On-line Trust Negotiation. In *2nd Workshop on Privacy Enhancing Technologies* (2002).
23. Shamir, A. How to share a secret. *Commun. ACM 22*, 11 (1979), 612–613.
24. Song, D., Wagner, D., and Perrig, A. Practical Techniques for Searches on Encrypted Data. In *2000 IEEE Symposium on Research in Security and Privacy* (2000).
25. Stinson, D. R. *Cryptography Theory and Practice*. CRC Press, 1995.
26. Weis, S. A., Sarma, S. E., Rivest, R. L., and Engels, D. W. Security and Privacy Aspects of Low-Cost Radio Frequency Identification Systems. In *Security in Pervasive Computing* (2004), vol. 2802 of *Lecture Notes in Computer Science*, pp. 201–212.
27. Wickramasuriya, J., Alhazzazi, M., Datt, M., Mehrotra, S., and Venkatasubramanian, N. Privacy Protecting Video Surveillance. In *Real-Time Imaging IX* (2005).
28. Wickramasuriya, J., Datt, M., Mehrotra, S., and Venkatasubramanian, N. Privacy Protecting Data Collection in Media Spaces. In *ACM Multimedia 2004* (2004).

APPENDIX

Fast Polynomial Interpolation

We describe our implementation of the polynomial interpolation algorithm on the server-side (log manager). We implemented the Lagrange's polynomial interpolation algorithm for determining the polynomial given, a set of shares (i.e. set of (x, y) pairs). In Shamir's secret sharing scheme, the number to be hidden is some $K \in \mathbb{Z}_p$ where p is a large prime number. The goal is that: if δ individuals come together, only then the secret can be determined and no group of $\delta - 1$ or less individuals can determine the secret. To achieve this, Shamir uses the well known property of polynomials: There exists exactly one polynomial $f(x)$ of degree $\delta - 1$ that passes through a given set of δ distinct points $(x_i, f(x_i)), i = 1 \ldots \delta$ in the plain. Therefore to share a secret K, a polynomial of degree $\delta - 1$ is generated as follows (we reproduce the presentation from chapter 11 of Stinson's book [25]).

$$f(x) = K + \sum_{j=1}^{\delta-1} a_j x^j \bmod p$$

Therefore $K = f(0)$. Now given δ points on the polynomial, the unique polynomial passing through these points can be computed as follows:

$$f(x) = \sum_{j=1}^{\delta} \prod_{1 \le k \le \delta, k \ne j} \frac{x - x_{i_k}}{x_{i_j} - x_{i_k}}$$

It is easy to see that this polynomial passes through all the δ points $(x_i, f(x_i))$. But since the secret to be computed is only the value $K = f(0)$, the computation can be simplified using the following formula.

$$K = \sum_{j=1}^{\delta} \prod_{1 \le k \le \delta, k \ne j} \frac{-x_{i_k}}{x_{i_j} - x_{i_k}}$$

Fast Incremental Interpolation

In our implementation, the log manager requires to interpolate the value of a new polynomial (of degree one more than than the previous) at $x = 0$ in every iteration (i.e. on arrival of a new share). Using the basic Lagrange interpolation formula, the log manager needs to do $O(n^2)$ arithmetic operations in each iteration if there are n shares, but this can be reduced to $O(n)$ operations by maintaining some intermediate data in the form of a divided-difference table as described in section 4.6.4 (pp.429) in Knuth [14]. We implement this incremental algorithm suggested in Knuth [14].

If the polynomial computed using n shares is denoted $u^{[n]}(x)$, observe that $u^{[n]}(x) - u^{[n-1]}(x) = 0$ at $x = x_0, \ldots, x_{n-1}$. Therefore $u^{[n]}(x) - u^{[n-1]}(x)$ is a polynomial of degree $\le n$ and a multiple of $(x-x_0)(x-x_1)\ldots(x-x_{n-1})$, therefore $u^{[n]}(x)$ can be written as $u^{[n]}(x) = \alpha_n(x-x_0)\ldots(x-x_{n-1}) + u^{[n-1]}(x)$ where α_n is a constant. Therefore one simply needs to compute the constant α_n in each iteration, and as shown in [14] the value of these coefficients can be read off a divided difference table. We show a divided difference table when 4 points on the curve are given, $(x_0, y_0), (x_1, y_1), (x_2, y_2), (x_3, y_3)$:

$$
\begin{array}{lll}
y_0 & & \\
\quad \frac{y_1-y_0}{x_1-x_0} = y_1' & & \\
y_1 & \frac{y_2'-y_1'}{x_2-x_0} = y_2'' & \\
\quad \frac{y_2-y_1}{x_2-x_1} = y_2' & & \frac{y_3''-y_2''}{x_3-x_0} = y_3''' \\
y_2 & \frac{y_3'-y_2'}{x_3-x_1} = y_3'' & \\
\quad \frac{y_3-y_2}{x_3-x_2} = y_3' & & \\
y_3 & &
\end{array}
$$

The value of $\alpha_{\mathbf{0}} = y_0$, $\alpha_{\mathbf{1}} = y_1'$, $\alpha_{\mathbf{2}} = y_2''$ and $\alpha_{\mathbf{3}} = y_3'''$. Therefore in our implementation $O(\delta_{max}^2)$ space is allocated for the divided difference table in each session, where δ_{max} is the maximum threshold for any trigger. The n^{th} iteration requires only $O(n)$ computation to compute the new divided differences. We store the value of $u^{[n-1]}(0)$ and compute the other term, $\alpha_n(-x_0)(-x_1)\ldots(-x_{n-1})$ using one more multiplication (since the product $(-x_0)(-x_1)\ldots(-x_{n-1})$ can also be stored from the previous iteration) once α_n has been computed. Hence the new value $u^{[n]}(0)$ can be computed in $O(n)$ steps by the log manager.