DEALING WITH ANONYMITY: A DESIGN PATTERN FOR PRIVACY-AWARE SYSTEMS

ALBERTO COEN PORISINI PIETRO COLOMBO SABRINA SICARI

Dipartimento di Informatica e Comunicazione
Università dell'Insubria
Via Mazzini, 5 - 21100 Varese, Italy
Email: {alberto.coenporisini, pietro.colombo, sabrina.sicari}@uninsubria.it

Nowadays the wide diffusion of applications that handle data referring to individuals requires the definition of ad hoc mechanisms that aim at protecting citizens privacy.
Anonymity is a fundamental requirement for privacy aware systems, which aims at preventing the identification of data owners starting from their data.
This paper proposes a design pattern for the definition of anonymity mechanisms built around a previously defined conceptual model for privacy policies.

1. Introduction

Nowadays privacy is a key issue and has received increasing attention from consumers, companies, researchers and legislators. Legislative acts, such as the European Union Directive[1] for personal data, the Health Insurance Portability and Accountability Act[2] for healthcare and the Gramm Leach Bliley Act[3] for financial institutions, require governments and enterprises to protect the privacy of their citizens and customers, respectively.

An important requirement for a privacy aware system consists in introducing mechanisms that aim at protecting the identity of the individuals whose data are handled by the system. Data handled by a system can be categorized into different classes. Among them, one class includes data that contain information concerning the private life, political or religious creed, behavior of individuals. Another class contains data that describes the identity of individuals (e.g., first name, family name, address, telephone, etc.).

In privacy aware systems only authorized users can view the existing relationship between data belonging to those two classes, while other users may be able to retrieve data belonging to one of the above mentioned classes without viewing the existing relationships.

For example, a hospital information system stores both health related and personal data of hospital patients. For diagnostic purposes a hospital doctor is allowed to access both types of data; while for other purposes such as epidemiology statistics only health related data should be accessed, that is it should not be possible to retrieve the identity of patients. This kind of problem is know as anonymity.

The aim of this work is to propose a domain independent solution scheme that drives the construction of anonymity assurance mechanisms.

More specifically, we propose a design pattern based on the general UML conceptual model introduced in,[4] which drives the definition of mechanisms that guarantee anonymity and which can be used in different application scenarios.

The paper is organized as follows: Section 2 shortly summarizes the most important features of the UML conceptual model proposed in;[4] Section 3 introduces the anonymity pattern; Section 4 presents the application of the pattern to a simple example; Section 5 discusses the related works; finally Section 6 draws some conclusions and discusses plans for future works.

2. The Privacy model

A privacy policy defines the way in which data referring to individuals can be collected, processed and diffused according to the rights that individuals are entitled to. In the following we summarize the UML model proposed in[4] that describes the main aspects of privacy according to the EU directive.[1]

A *PrivacyPolicy* is characterized by three classes: *User*, *Data* and *Action*.

User represents an actor either interested in processing data or involved by such a processing. Users are characterized by functions and roles. More specifically, *Function* represents the job performed by a user in an application domain, while *Role* character-

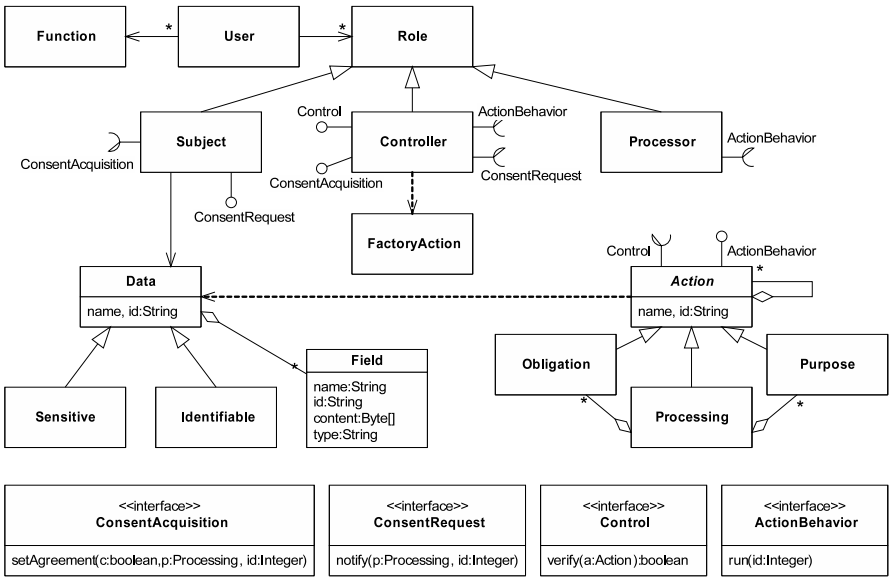

Fig. 1. Privacy Class Diagram

izes users with respect to privacy. *Role* is extended by three distinct classes to represent the different roles: *Subject*, which is anyone whose data are referred to, *Processor*, which is anyone who asks for processing data by performing some kind of action on them and *Controller*, which defines the allowed actions that can be performed by the processors.

Data represents the information referring to subjects that can be processed by processors. *Data* is extended by means of *Identifiable* data (e.g., family name, address, phone n.) and *Sensitive* data (e.g., health, religion). The former represents the information that can be used to uniquely identify subjects, while the latter represents information that deserves particular care and that should not be freely accessible. *Data* is a complex structure composed of basic information unit named *Field*. *Field* is characterized by the attribute *name* that identifies the contained information, which in turn is described by the attribute *content*.

Action represents any operation performed by *User* (usually *Processor*). *Action* has been defined using an abstract class and it is extended by *Obligation*, *Processing* and *Purpose*.

Purpose is the reason for which an authorized processor access data (e.g., marketing, customer satisfaction, evaluate the customer needs). *Processing* is any operation or set of operations which is performed upon data, whether or not by automatic means (such as collection, recording, etc.), while *Obligation* is a set of actions that the processor guarantees to perform, after the data have been processed.

Moreover, each action can be recursively composed of other actions. Since in a privacy aware scenario a processing can be executed under a purpose and an obligation, *Processing* specifies an aggregation relationship with *Purpose* and *Obligation*.

Figure 1 depicts the aforementioned classes along with their relationships. Moreover, classes interact among them exchanging information by means of interfaces.

3. Anonymity

This section illustrates the Anonymity pattern, that is a basic scheme for preventing the identification of individuals, starting from their sensitive data.

3.1. *Requirements*

Several requirements must be taken into account when defining anonymity assurance mechanisms.[5]

- Identity masking. Anonymity enabling mechanisms shall mask the identity of subjects.
- Usability. An anonymous data set shall be usable. Extreme solutions such as not releasing any data cannot be accepted. Moreover,

anonymity enabling mechanisms shall not alter the processing actions performed by a system.

- Performance. Anonymity enabling mechanisms shall minimally alter the overall system performances.

3.2. *Solution*

The proposed solution starts from the classification of data and users proposed in the conceptual model.

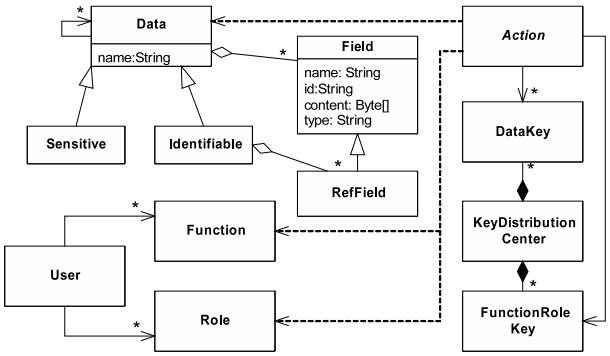

Fig. 2. Extensions of the conceptual model to support anonymity

Data structure In order to define anonymity, the data handled by a system need to be suitably structured. Data are composed of fields that, depending on their characteristics, are grouped into sensitive and identifiable subsets. Moreover, a data type may be characterized by a hierarchical structure composed of other data types possibly classified as sensitive or identifiable.

In order to keep the link between identifiable and sensitive data we introduce a reference field to the data structure used for identifiable data. This is done by means of class *RefField*, introduced in Figure 2, which extends class *Field* of the conceptual model, as shown in Figure 1. The inherited attributes of class *RefField* are used in the following way: the attribute *name* is set to the name of data to which it refers, while the attribute *content* is set to the value of the attribute *id* of the instance of *Data* to which it refers.

For example, let us consider the definition of a data structure composed of the fields "familyName", "city" and "disease". Fields "familyName" and "city" identify the data owner, while "disease" represents a sensitive information. As a consequence two different data types are defined. The former, named "Person", is composed of the identifiable fields, while the latter, named "Health", contains the sensitive one. Let us consider the following data sets: 1) "Smith", "Milan", "hemicranias"; 2) "Brown", "New York", "gastric ulcer". Therefore, the first triplet is represented by an instance of class *Identifiable* in which the attribute *name* is set to "Person", and attribute *id* is set to "data001", and by an instance of class *Sensitive* in which the attribute *name* is set to "Health" and attribute *id* is set to"data003".

Moreover, "data001" contains an instance of class *Field* characterized by the attribute *name* initialized to "familyName", the attribute *id* initialized to "field001", and the attribute *content* set to "Smith". It also contains a further *Field* characterized by the attribute *name* set to "city", the attribute *id* initialized to "field002", and the attribute *content* set to "Milan". Finally, "data003" contains an instance of *Field* characterized by the attribute *name* initialized to "Disease", the attribute *id* initialized to "field005", and the attribute *content* set to "hemicranias". In order to represent the link between identifiable data represented by "data001" and the sensitive data represented by "data003", "data001" contains an instance of *RefField* in which the attribute *name* is set to "Health", the attribute *id* is set to "ref001" and the attribute *content* is set to "data003". A graphical description of the structure of such data is shown by the Composite Structure Diagram of Figure 3.

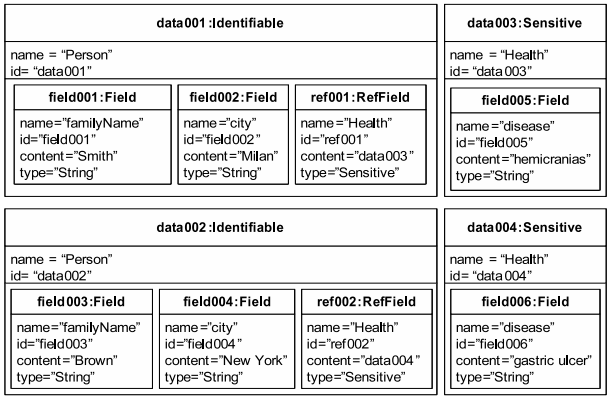

Fig. 3. The Composite Structure Diagram that describes the example

In order to prevent the identification of data owners starting from their sensitive data, instances of

Identifiable may own references to instances of *Identifiable* or *Sensitive*, while instances of *Sensitive* can own only references to instances of *Sensitive*. In other words, instances of *Sensitive* cannot own any reference to instances of *Identifiable*.

A second issue that must be taken into account concerns the possibility that starting from identifiable data one can access the asssociated sensitive data, by following the reference fields. However, the system should prevent non authorized users of the system to follow such references, that is there may be some users that can access identifiable data without being authorized to access sensitive data.

The way in which we prevent non authorized accesses to sensitive data is based on cryptography. Notice that at this level we do not need to choose any particular encryption technique (e.g., public key, symmetric key, etc.), since the needed extensions of the conceptual model are independent from encryption techniques.

Handling cryptography The way in which we introduce cryptography is based on three new classes (see Figure 2): *KeyDistributionCenter*, *DataKey* and *FunctionRoleKey*. The class *KeyDistributionCenter* manages the generation of the keys usable for encryption purposes. *KeyDistributionCenter* generates keys according to the restrictions imposed by the privacy policy. *FunctionRoleKey* represents the key associated with a specific pair *Function-Role*, while *DataKey* represents the key to encrypt the content of data fields.

In what follows we present the use of the previously introduced concepts for the definition of anonymity mechanisms.

Data encryption A key, named *DataKey*, is generated to encrypt the value of the attribute *content* of the reference fields that refer to instances of sensitive data. As an example, let us consider that for statistics purposes we need to know how many people living in Milan suffer from hemicranias. As described above, such data types are separately defined and a reference field, named "Health", is defined on "Person". Notice that the attribute *content* of "Health" is encrypted, and therefore it is not possible to access the sensitive data without knowing the key that is required to decrypt such a field. Notice that although the content of sensitive data is not encrypted, such data do not provide any reference to identifiable data.

In order to prevent data inference we assume that all the instances of data types with one (or more) reference field have a non null value associated with attibute *content*. In this way the fact that a person has a reference to a sensitive data element cannot be used to infer that there is some sensitive information in the system (e.g., health condition), since the sensitive data element can be empty.

Actions Data can be accessed only by means of actions (see Figure 2). Actions are expressly built to be executed by users that belong to a given function-role pair. In order to guarantee that actions once defined can be only executed by the authorized users, an authentication mechanism is introduced. More specifically, a key, represented by the class *FunctionRoleKey*, is generated and released to the authorized users.

FunctionRoleKey instances are handled by *KeyDistributionCenter*, which provides generation and secure communication mechanisms like the ones proposed by Kerberos.[6] Whenever a user-controller defines a new *Action*, two keys are generated. The former key is associated with the pair *Function-Processor* that is authorized to execute the action, while the latter with the pair *Function-Controller* that has to supervise the execution. Notice that the specification of the algorithm to be used for key generation, and of the communication protocol is out of the scope of this pattern.

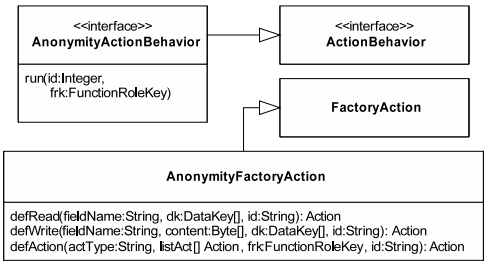

Fig. 4. Extensions of the conceptual model to support actions management

In order to support encryption a new class and a new interface are introduced (see Figure 4). The class, named *AnonymityFactoryAction*, extends class *FactoryAction*, while the interface, named *AnonymityActionBehavior*, extends the interface *Ac-*

tionBehavior.

AnonymityFactoryAction defines three methods: *defRead()*, *defWrite()* and *defAction()*.

defRead()/*defWrite()* are used to define basic actions that allow one to read/write data. Since data fields are encrypted, such actions use keys to access the data content.

The parameters of *defRead()*/*defWrite()* specify: the name of the data fields, *dataField*; the data type, *data*; the keys, *kd* that are required to decrypt/encrypt the content of the data fields, and the identifier of the action *id*. Such actions represent the basic constituent parts of complex actions that can be executed by authorized users. Notice that basic actions cannot be directly executed.

Controller defines executable (complex) actions by means of the method *defAction()*, whose parameters specify the involved basic actions and the users that are allowed to execute the action. In particular the parameters of *defAction()* specify: the action type, *typeId*, which expresses whether the action is a *Processing*, a *Purpose* or an *Obligation*; a list of predefined basic actions, *a[]*; the *FunctionRoleKey*, *frk* that identifies the authorized users, and the action identifier *id*.

For instance, let us suppose that a researcher who works in a health care institute is interested to know how many people living in Milan suffer from hemicranias. Moreover, suppose that data are organized by means of the structure described in Figure 3. Therefore it is necessary to access the fields "city" and "disease". The controller, in order to create such an action, invokes the method *defAction()* passing as paremeter an instance of class *FunctionRoleKey* associated with the pair Researcher-Processor that is authorized to execute the action once defined.

Actions can be executed by invoking the method *run()* (defined by *AnonymityActionBehavior*), providing the key *FunctionRoleKey*, and the *id* of *User*. Notice that users authentication can be carried out in different ways. For example, the first task of the method *run()* may check whether the function-role key provided by the user is the same key that was set at action definition time.

3.3. *Consequences*

The pattern has the following benefits.

- Privacy. The separation of sensitive data from identifiable data, and the adoption of encryption techniques make it more difficult to associate sensitive data with the identity of data owners.
- Minimal user involvement. The users are not required to modify their normal activities.

Notice that the pattern should be applied to only populated datasets, since it cannot guarantee anonymity if the population of the dataset is composed only by a single entry.

A not properly defined implementation of this pattern may suffer from the following weaknesses.

- Usability. A too high granularity level of encryption mechanisms can undermine the usefulness of data. As an example, in the case of database applications, if all the data entries are encrypted, the resultant dataset may be hardly used even by authorized users.
- Overhead and delay. The application of encryption mechanisms requires adequate computational resources. Hence, the overall system performances may worsen, and delays and/or overheads can be generated. In order to guarantee an adequate level of usability and privacy, it is necessary to balance the usage of encryption techniques.

4. Example

This section provides a simple example concerning the definition of a privacy policy for data management in the context of a small neurology department characterized by the functions of neurologist, employee and outpatient:

- Outpatients need to be medically assisted. They request to undergo a medical examination with the neurologist, and once examined they pay the fee
- Neurologists examine outpatients, access and modify their case histories, prescribe therapies or physical examinations.
- Employees perform bureaucratic activities such as registering outpatients, making appointments for medical examinations, preparing purchase orders and so on.

Data processing is regulated by a policy that specifies 1) who is allowed to process data, and 2)

what can be done with such data. The system manages different types of data:

- Patient case histories (sensitive data): detailed records on the background of a person under treatment.
- Identifiable data: identifiable data associated with patients such as first name, family name, address, telephone number, etc.
- Administrative data: the status of payments for medical examinations and treatments.

For example let us consider an outpatient who needs to contact the department to request an appointment for a medical examination. The following scenario sketches the involved actors and the actions:

- An employee makes the appointment for the medical examination;
- The outpatient goes to the appointment and is visited by the neurologist;
- The outpatient pays the fee at the payment office.

The main goal of this example is to show how it is possible to exploit the conceptual model and the anonymity pattern for modeling the following requirements:

- The system shall prevent the identification of individuals starting from their sensitive data
- The processing actions can be exclusively executed by authorized users of the department information system.

4.1. *Modeling privacy*

The actors involved in this example are represented by means of instances of classes *User*, *Function* and *Role*.

Employees are instances of *User* characterized by *Function* "Employee" and by the role *Processor*, since employees process data of the outpatients. Similarly, neurologists are characterized by *Function* "Neurologist" and the role *Processor*, while outpatients are the owners of data that will be processed by doctors and employees. Outpatients are characterized by the role *Subject* and no *Function* is associated with them.

Actions can be exclusively executed by authorized users of the information system of the neurology department. We assume the existence of a key distribution center and of a key management service that are able to create and distribute encryption keys to each pair function-role that operates the system.

Actions affect data of subjects, and thus in order to allow processing, the interested subjects have to grant their consent. In what follows we assume that the outpatients provided the explicit consent to access their data. Moreover, we assume the existence of the following actions:

- "Medical examination reservation": performed by employees to reserve medical examinations
- "Update case history": performed by the neurologist to modify the outpatient case history and the prescribed treatment
- "Pay the fee": performed by the employee to record the payment of the fee associated with a medical examination

For space reasons we do not consider the outpatients registration. Hence we assume that the outpatients that require to be examinated are already registered.

At definition time, actions were provided with the name of: 1) the data types and the fields that are accessed; 2) the keys that are needed to access encrypted information (i.e., the references to sensitive data and possibly encrypted further fields), 3) the keys of the pairs *Function-Role* that are allowed to execute them. A more detailed description of the functionalities of such actions is provided later on.

The data managed by the system concern: identifiable information of the outpatients, information on the reservation, the cost, the payment and the case histories of the outpatients (sensitive data).

According to the Anonymity pattern and the proposed conceptual model, in order to prevent the identification of individuals starting from their sensitive data, sensitive and identifiable data are separately modeled by means of the following data types:

- "Person": a data type characterized by fields like " first name", "family name", "birth date", "address", "telephone number", which identify an outpatient.
- "Medical examination": composed of fields like "data", "time", "place", "examination type", "examination report", which provide

information on the examination.

- "Price list": characterized by fields such as "examination type" and "price", which describe the price associated with each examination.
- "Case history": sensitive data composed of fields that keep track of the health state of outpatients and the prescribed treatments.
- "Payment information": composed of fields like "total" and "paid" that keep track of the payment of the examinations.

"Person" is an identifiable data type that stores references to instances of sensitive data such as "Medical examination". Moreover, "Medical examination" stores a reference to a sensitive data named "Case history". Instead "Payment information" and "Price list" are neither sensitive nor identifiable data.

We assume the existence of a key management service that generates and stores a new key whenever a new type of sensitive data is introduced. Moreover, similarly to the action management case, we assume that all the previously described data types were defined by an authorized *User* with the role of *Controller*. The generated keys are used to encrypt/decrypt the content of the reference fields to instances of the previously listed sensitive data.

The employee makes an appointment for the medical examination by invoking the action "Medical examination reservation". Such action is a complex processing composed of multiple data writing and data reading basic actions, which affect some of the fields of "Person", "Price list" and "Medical examination". More specifically, the action accesses the fields " first name", "family name", "address", "birth date", "telephone number" of "Person" and the fields "examination type" and "price" of "Price list". The action also defines a new instance of "Medical examination" and initializes its fields "data", "time", "place", "examination type", "examination description". The action is also provided with the keys that are required to encrypt the content of the reference field "Medical examination" of "Person" and of "Payment information", and "Case history" of "Medical examination". Moreover, it integrates the keys associated with the users that are authorized to execute the action, that is those characterized by the pair *Employee-Processor*.

At execution time the employee invokes the method *run()* of the action by specifying, his/her *id*, and his/her *FunctionRoleKey*, the *id* of the outpatient, the type and the description of the medical examination, data, time and place of the examination. The action creates a new instance of "Medical examination", characterized by a certain *id*. The value of *id* encrypted with the key of the sensitive data "Medical examination" is stored in the homonymous reference of the interested instance of "Person". The action creates an instance of "Payment information" and generates a reference to a "Case history" that is stored in "Medical examination". The first data type specifies the total amount due for the examination. Notice that the values of *id* and of the field "total" are initialized. The reference to "Case History" is used to store the results of the examination. Notice that all the other fields are "empty". The values of *id* are encrypted using the key of the sensitive data "Payment information" and "Case History" and the resulting values are stored in the homonymous reference fields of the interested instance of "Medical examination". At the end of the execution, the medical examination is booked.

The outpatient provides the *id* of the reservation to the neurologist. The doctor, after examining the outpatient, updates his/her case history by means of the action "Update case history". More specifically, the key associated with the couple *Neurologist-Processor* is integrated into the action.

Finally, the outpatient has to pay the fee for the medical examination. Hence he/she provides the *id* of the reservation to the employee. The employee records the payment by invoking the action "Pay the fee". Such action accesses the field "examination type" of "Medical examination" and the fields "examination type" and "price" of "Price list". The value of the field "examination type" of "Medical examination" is used to calculate the price associated with the examination. The resulting amount is used to update the field "paid" of the involved instance of "Payment information".

5. Related works

Anonymity, pseudonymity, unlinkability, informed consent and unobservability are some of the main features of privacy aware systems. Such features can be effectively modeled by means of design patterns, i.e., general reusable design schemes.

Many security patterns were defined to address enterprise, architectural and user-level security,[7–10] while, at present only few contributions, concerning the privacy domain, have been defined. Chung *et al.*[10] define privacy patterns for ubiquitous computing domain. Schummer[11] describes the privacy masquerade pattern, i.e., pattern that specifies how it is possible to prevent personal information from being improperly transmitted. Schumacher[12] describes two privacy patterns, named Pseudonymous Email and Protection against Cookies, respectively. The former specifies mechanisms for hiding the sender of an email message; while the latter describes how to control the cookies in a web browser. Romanosky et al.[13] introduce privacy pattern for online interactions, distinguishing between pattern for system architecture issues and pattern for end-user support. Hafiz[5] defines anonymity design patterns for various types of online communication systems, online data sharing, location monitoring, voting and electronic cash management.

All these contributions address specific application domain issues, while our solution is general and it can be applied to different contexts.

6. Conclusions

This paper proposed a general solution, based on the conceptual model for privacy policies presented in,[4] to implement anonymity. The model provides the conceptual foundations that are required to implement anonymity, such as the separation of sensitive from identifiable data, and the classification of roles and actions. The proposed solution represents a design pattern for anonymity and it consists in concepts and guidelines that drive the modeler towards the definition of privacy aware systems. The solution extends a part of the conceptual model by introducing new concepts that aim at supporting those encryption mechanisms that are necessary for the implementation of anonymity.

A simple example concerning a neurology department illustrated the application of the pattern. The example drives the reader through the classification of users and actions, and shows how it is possible to integrate the encryption mechanisms in order to define anonymity.

Future works concern the evaluation of the scalability of the proposed solution, i.e., we are interested in evaluating the application of the pattern to a case study of realistic complexity. Moreover, we are working on the definition of additional design patterns for privacy aware system. More specifically, at present we are completing the definition of a pattern that drives the acquisition of informed consent of subjects for handling their data.

References

1. Directive 95/46/EC of the European Parliament. Official Journal of the European Communities of 23 November 1995 No L. 281 p. 31.
2. http://www.hipaa.org.
3. http://www.glba.org.
4. A. Coen-Porisini, P. Colombo, S. Sicari and A. Trombetta, A conceptual model for privacy policies, in *Proc. of SEA*, (Cambridge (MS), USA, November, 2007).
5. M. Hafiz, A collection of privacy design patterns, in *Proc. of PLoP*, 2006.
6. http://web.mit.edu/Kerberos/.
7. M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann and P. Sommerlad, *Security Patterns: Integrating Security and Systems Engineering* (John Wiley & Sons, 2006).
8. B. Blakley, C. Heath *et al.*, *Technical Guide* .
9. C. Steel, R. Nagappan and R. Lai, *Core Security Patterns: Best Practices and Strategies for J2EE, Web Services, and Identity Management* (Prentice Hall, 2005).
10. E. S. Chung, J. I. Hong, J. Lin, M. K. Prabaker, J. A. Landay and A. L. Liu, Development and evaluation of emerging design patterns for ubiquitous computing, in *Proc. of Designing Interactive Systems*, (New York, NY, USA, 2004).
11. T. Schümmer, The Public Privacy–Patterns for Filtering Personal Information in Collaborative Systems, in *Proc. of Pattern Language (CHI 2004 workshop)*, 2004.
12. M. Schumacher, Security Patterns and Security Standards, in *Proc. of EuroPLoP*, 2002.
13. S. Romanosky, A. Acquisti, J. Hong, L. Cranor and B. Friedman, Privacy Patterns for Online Interactions, in *Proc. of PLoP*, 2006.