Toward A Software Development Methodology for Anonymity Applications

Marzieh Ispareh
Dept. of Computer Engineering
University of Isfahan, Iran
ispareh@eng.ui.ac.ir

Behrouz Tork Ladani
Dept. of Computer Engineering
University of Isfahan, Iran
ladani@eng.ui.ac.ir

Shirin Shariat Panahi
Dept. of Entrepreneurship Management
University of Tehran, Iran
Shirin.panahi@ent.ut.ac.ir

Zahra Nasr Azadani
Dept. of Computer Engineering
University of Isfahan, Iran
znasrazadani@yahoo.com

Abstract

There are some software applications especially in privacy protection domain which they need anonymity support. Therefore lots of methods and protocols have been presented so far for providing this requirement. However no specific software development methodology has been yet provided for specification of anonymity requirements and consideration of anonymity as part of software design and implementation life cycle. In this paper we present a methodology for development of anonymity applications. The proposed methodology consists of three relevant phases named AnoModel, AnoUML, and AnoAPI. Requirement analysis and specification is based on AnoModel which is a conceptual model of anonymity. Also design and implementation phases are partially covered by AnoUML (which is an extension of UML for supporting anonymity design elements) and AnoAPI (which is a programming interface for implementing anonymity primitives) respectively. To show the applicability of the proposed methodology, two case studies of using it are presented.

Categories and Subject Descriptors

D.2.1 [SOFTWARE ENGINEERING]: Requirements/ Specifications -- *Methodologies*; K.4.1 [COMPUTERS AND SOCIETY]: Public Policy Issues -- *Privacy*.

Keywords

Anonymity, Software Development Methodology, Anonymity Application

1. Introduction

Nowadays methods for anonymity providing specially with the goal of privacy preservation are considered in applications such as e-voting, e-commerce, etc. However the features of required anonymity and also the required level of it are various in different applications. Therefore careful requirement analysis for determining precise anonymity properties which are suitable for a service is important. For example designing for complete anonymity is not the best choice for many applications. In a system with complete anonymity, tracking the entities is not possible while correct authentication and tracking is often essential for the sake of responsibility and accountability [3].

Beside the need for suitable requirement analysis and specification, having a systematic approach for considering anonymity requirements in different phases of application development is of high importance. This is while no serious and specific work has been reported in this regard.

Different software development methodologies contain different phases, but most of them use three general phases: requirement analysis, conceptual design, and implementation [4]. To cover the first phase of the methodology, we use our previous work in [5] which is a conceptual model of anonymity. We call it AnoModel here. AnoModel presents both a framework for definition of anonymity concepts and taxonomy of all service types for anonymity and their properties. So this model can be used for requirement analysis of an extended variety of anonymity applications.

For the second phase, AnoUML as an extension of UML for anonymity applications is introduced. This extension suits AnoModel, so requirement specifications resulted from the first phase can be modeled using AnoUML. AnoUML helps designers in simpler, more precise, and systematic design by focusing on anonymity properties based on AnoModel.

In the third phase, a programming interface for implementing anonymity primitives in communication layer is introduced. We call it AnoAPI. AnoAPI is in fact a partial set of software mechanisms which are used to implement design patterns specified in AnoUML phase. A prototype of AnoAPI has been implemented as a java package and contains basic techniques for supporting anonymity in communication layer. By adding this package in an anonymity application program and overriding its classes, one can implement the desired anonymity properties in his/her application.

The rest of paper is structured as follows. In section 2 we have a short survey on related works, then in section 3 a summary of our previous work in [5] is presented as AnoModel. Sections 4 and 5 present AnoUML and AnoAPI respectively. Then in section 6 two case studies on using the proposed methodology are presented. Finally in section 6 we conclude the paper.

2. Related Works

According to our researches, there is not so many especial methodology or methodology extension for development of anonymity applications. The only related work is a methodology for designing controlled anonymous applications [6]. In this work, the whole software requirements are classified in four categories of privacy, control, performance, and trust in which control and trust are two opposite types of requirements. The presented methodology consists of multiple models for privacy, performance and trust and uses several alternatives at each design step to partially avoid conflicts between requirements. Moreover, the final model foresees a mapping to control mechanisms in implementation phase.

A drawback of the methodology presented in [6] is that it has not considered all aspects of anonymity in the analysis phase. For example, each anonymity requirement can be just expressed as expression *Unlinkable(a,b)* in which *a* and *b* are actions or environment attributes. This way, designers can not analyze and specify anonymity requirements clearly.

Some other outspread tasks have been done on categorizing anonymity requirements. In [2], types of anonymity are just classified based on the information or entities which are anonymous. Also in [7] after defining some anonymity features, different types of anonymity has been categorized. Although this work is stronger than other related works, but no clear role for information or entities which become anonymous is defined in this taxonomy.

3. AnoModel

In this section we are going to summarize our conceptual model presented previously in [5]. We call it AnoModel here. AnoModel provides a framework for analysis and specification of anonymity properties (can be provided by an anonymity technique) or anonymity requirements (needed for an anonymity application).

To define anonymity, at first we introduce "Identification Information" or Idinfo in short: Idinfo is data or information that can be used to indicate the real identity of an entity or her messages precisely. Idinfo may belong to one entity or a group of entities. Based on the above definition, different instances of Idinfo in a system are Entity Idinfo and Message Idinfo. Different types of Entity Idinfo and Message Idinfo have been shown in figures 1 and 2 respectively. One can refer to [5] for detailed explanation of the taxonomies.

In each application, depending on application features, different anonymity requirements are needed. Usually in an application, multiple entities become anonymous from other entities, so we need multiple anonymity services. Each anonymity service is a set of activities which provide a valid combination of inaccessibility to some Idinfo instances from the viewpoint of other entities. So we can have two classes of anonymity service: Entity anonymity services, and Message anonymity services. Based on definition of Idinfo, each class of anonymity services can be divided into seven different anonymity types which are presented in Tables 1 and 2.

Two entities play the main role in each anonymity service: anonymous entity, and anonymity observer i.e. an entity which anonymity occurs from its viewpoint. Anonymous entities can have two specific properties: Authentication and Reply. Authentication means that despite an entity is anonymous, still it's possible to authenticate her and check her permissions for sending messages, access to data objects and so on. Reply means despite an entity is anonymous, but still it's possible to reply her messages.

Each anonymity service can be applied absolutely or conditionally. In the case of Absolute anonymity, entity becomes anonymous without any condition. On the other hand, in Conditional anonymity, the entity becomes anonymous condition to satisfying some constraints.

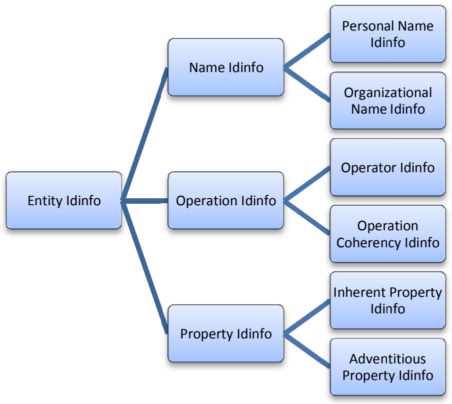


Fig. 1. Entity Idinfo types

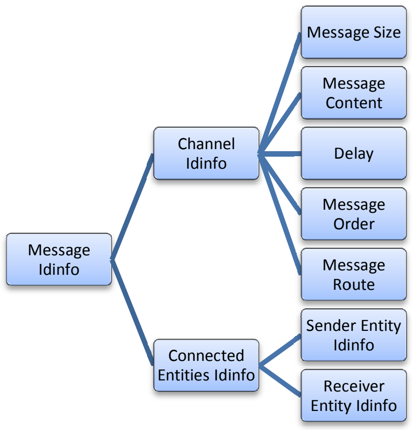


Fig. 2. Message Idinfo types

Three kinds of anonymity constraints could be defined:

- **Temporal anonymity constraints:** Anonymity is established or preserved based on some temporal conditions e.g. until when some special event is happen.
- **Spatial anonymity constraints:** Anonymity is established or preserved based on some spatial conditions e.g. an agent is anonymous in hosts with a special (authenticated) ID or IP address.
- **Committed anonymity constraints:** Anonymity is established or preserved so long as entity is faithful to some special rules e.g. while she does not received a special secret token or is obeyed to a certain rule.

Table1: Entity anonymity service types

Anonymity type	Name Idinfo	Property Idinfo	Operation Idinfo
Without anonymity	✓	✓	✓
EA1	✓	✓	×
EA2	✓	×	✓
EA3	✓	×	×
EA4	×	✓	✓
EA5	×	✓	×
EA6	×	×	✓
EA7 (Full anonymity)	×	×	×

Table 2: Message anonymity service types

Anonymity type	Connected entities Idinfos		Channel Idinfo
	Sender Entity Idinfo	Receiver Entity Idinfo	
Without anonymity	✓	✓	✓
MA1	✓	×	✓
MA2	×	✓	✓
MA3	×	×	✓
MA4	✓	✓	×
MA5	✓	×	×
MA6	×	✓	×
MA7 (Full anonymity)	×	×	×

So to specify the anonymity requirements of an application we can follow three following steps: (1) Specification of anonymity service types, (2) Specification of anonymity structure, i.e. factors involved in each required service, and (3) Specification of anonymity constraints, i.e. whether each anonymity service is conditional or not, and what are those constraints if there is any.

Anonymity requirements or properties could be specified as a tuple <EA, MA> where EA is the set of Entity Anonymity services that each of its members is a tuple as follows:

<AE, AO, AnoType, C, Authentication, Reply>

AE is the Anonymous Entity, AO is the Anonymity Observer entity, AnoType is the type of this anonymity service, C is a set of constraints; if it is empty, then the anonymity service is applied absolutely, and Authentication and Reply are Boolean parameters that indicate whether anonymous entity has those abilities or not.

MA defines the set of Message Anonymity services that each of its members is a tuple as follows:

<SE, RE, AO, AnoType, C >

In which SE and RE are communicated entities and other parameters are the same as EA.

4. AnoUML

In this section we present AnoUML which is an extension of UML compatible with AnoModel. AnoUML helps designers in simpler and more systematic design by focusing on anonymity services, types and the places which they must be deployed.

There are three mechanisms for extending UML [8]: Stereotypes which allow us to extend the vocabulary of the UML, Tagged values which allow us to extend the properties of a UML building block, and Constraints which allow us to extend the semantics of a UML building block by adding new rules, or modifying existing ones.

As we told earlier, each anonymity application has two classes of anonymity requirements:

- Entity anonymity
- Message anonymity

For explaining entity anonymity requirements in AnoUML, We choose the Use case diagrams. Use case diagrams show operations of each entity from viewpoint of entities outside the system. So anonymity of each entity from viewpoint of anonymity observers can be shown by these diagrams. On the other hand, message anonymity can be shown by Deployment diagrams in a subsystem. Each subsystem can contain every section of UML diagrams. In this case, all the features of subsystem are applied to all elements inside the subsystem. The reason for selecting Deployment diagrams is that these diagrams show the physical relations between different parts of the system. Besides high architectural level design capability of Deployment diagram, it's possible to show the types of anonymity for relations in system exactly as they must to be.

Stereotypes along with their constrains are shown in table 3. Tagged Values of each Stereotype with more complete explanations are presented in Appendix A. We describe Stereotypes of AnoUML in continue.

Anonymous Actor: This Stereotype is applied on an Actor in standard UML and it shows that this Actor has a task which must be done in an anonymous way. The AnoEntityType of this Stereotype defines the anonymity service types required for this Actor according to definitions of AnoModel.

Anonymous Observer: This Stereotype is applied on Actor in standard UML and it shows an observer entity. This Actor can have no real role in a system. The AnoObserverType of this Stereotype defines the anonymity service types required for this Actor according to definitions of AnoModel.

Anonymous Observing: This Stereotype is applied on Association relation in standard UML and it shows the relation between the Anonymous observer and a Use case which is operated anonymously from viewpoint of that observer. The EnAnoProperty of this Stereotype explains the anonymity features of this entity.

Sender Server: This Stereotype is applied on Nodes (Servers) and it defines a system which begins anonymous connection in Deployment diagram.

Sender Component: Deployed applications on sender servers which play roles in connection are Stereotypes named Sender component.

Receiver Server: This Stereotype is applied on Node (Server) and it defines a host which receives the request for anonymous connection in Deployment diagram.

Receiver component: Deployed applications on receiver servers which play roles in connection are Stereotypes with the name of receiver component.

Anonymizer System: In an anonymity providing system, normally there is a host for providing anonymous communication. We call the whole system between sender and receiver which play a role in providing communication anonymity as Anonymizer System. Anonymizer System is applied on Node as a Stereotype. The ServerType of this Stereotype defines type of server. For example it can be Mix Server, TTP Server (in conditional anonymity) and Authentication Server (in systems which need authentication mechanisms).

Mix System: This Stereotype is applied on Package in standard UML and in Deployment diagram it covers all responsible hosts for supporting the anonymity (Anonymizer Systems).

UsecaseView: This Stereotype is applied on Use case model. For each anonymity observer of an anonymous Actor, we consider one individual Use case diagram that specifies anonymity type of each Use case of anonymous Actor from viewpoint of that anonymity observer.

Table 3: AnoUML Stereotypes

Stereotype	Base class	Tagged value	Constraints
Anonymous Observing	Association	EnAnoProperty	Connects an anonymity observer to a Use case
Anonymity Observer	Actor	AnoObserverType	Is connected to a Use case via an Anonymous Observing link
Anonymous Actor	Actor	AnoEntityType	Initializes at least one Use case that is connected to an Anonymity Observer
Sender Server	Node	-	Contains at least one Server Component
Receiver Server	Node	-	Contains at least one Receiver Component
Anonymizer Server	Node	ServerType	-
Sender Component	component	-	-
Receiver Component	component	-	-
Anonymous Link	Subsystem	MeAnoProperty	All network layer communications of this subsystem has MeAnoProperty anonymity type
Mix System	Package	-	Contains only Anonymizer server
UseCaseView Diagram	Use case diagram	-	For each anonymity observer of an Anonymous Actor, we draw one UseCaseView that specify anonymity type of each Use case of anonymous Actor from viewpoint of that observer

5. AnoAPI

Anonymity properties especially in communication layer usually contain similar functionalities. By presenting these functionalities as individual units we can achieve basic anonymity primitives. Normally anonymity techniques in application layer are complex solutions for anonymity requirements and can support the required type of anonymity without mixing with other techniques. But primitives in communication layer are simpler and should mix with other techniques to achieve the ability for supporting anonymity requirements in application. In Appendix B various techniques of three mentioned classes of anonymity primitives and the type(s) of anonymity services which could be provided by primitives are listed.

We consider AnoAPI Java package to contain two base classes named Send/Receive for sending/receiving the messages. For each anonymity primitives in communication layer, an abstract class with required methods and attributes has been defined as an interface for that anonymity primitive. List of these classes are as follows:

- EncryptMessage for encrypting technique
- CacheMessage for request caching technique
- CompressMessage for compressing technique
- FilterMessage for filtering technique
- ImpersonateMessage for impersonating and pseudonym technique
- PadMessage for padding technique
- DelayMessage for adding some random delays and reordering the messages

Note that service layer techniques which can not be applied on each packet individually are designed as methods in Send/Receive classes. These methods are as follows:

- BroadCastPacket for broadcasting the messages in Send class.
- sendMultiPacket and receiveMultiPacket for multiplexing in Send/Receive class.
- sendBatchPacket for batch sending in Send class
- sendDummy for sending dummy messages in Send class.

6. Case studies

In this section, we show how to use the presented methodology for requirement analysis of an anonymity application. Also we show how to use AnoAPI for implementing an anonymity protocol.

6.1 Anonymous Electronic Payment System

Typically in an electronic payment system there are three main Actors which are seller, buyer and bank. Suppose that it's needed to satisfy the following anonymity requirements in an example system:

- No one can understand who is buying or selling by monitoring the exchanged messages during a transaction. Also no one can understand the real identity of seller during his/her transaction with bank.
- No one can be able to track buyer by relating together their activities. (Hiding operation coherency Idinfo's of buyer during his/her transaction).
- Anonymity of buyers should be such that cheating could be detectable in certain situations, e.g. when buyer does not pay on time. So the buyer anonymity is conditional.
- Seller must have the ability to reply anonymous buyers.
- Bank and seller must have the ability to authenticate anonymous buyers.

According to the above mentioned requirements we can specify the system requirements as tuple <EA, MA> in which:

MA = {< Buyer, Seller, global/local observer, MA4, null >,
< Buyer, Bank, global/local observer, MA4, null >,
< Seller, Bank, global/local observer, MA4, null >}

MA4 anonymity service type is needed to hide name Idinfo's (identity) of buyer from local/global observers during communication with seller and bank. This way, the identity of seller should be hidden in communication with bank as well.

EA also could be specified based on the mentioned requirements as follows:

EA = { <Buyer, Seller, EA2, {Paid on time}, True, True>,
<Buyer, global observer, EA2, null, False, False >,
<Buyer, local observer, EA2, null, False, False >,
<Buyer, Bank, EA2, null, True, False > }

As it has been shown, both name and operation Idinfo's of buyer should be preserved during her transaction from the viewpoint of all other entities. So the anonymity service type of buyer is EA2. It should be possible to authenticate the anonymous buyer by both seller and bank. Seller should be able to answer the anonymous buyer, and finally buyer will not remain anonymous for buyer if she doesn't pay on time. Note that it has been considered that bank has no direct communication with buyer, so it's not needed that bank be able to reply buyer.

To design the system by AnoUML we should notice that buyers need to be anonymous from sellers, bank and global/local observers. Therefore we need four Use case diagrams to represent the required anonymity for buyers operations i.e. Use case diagrams of buyer anonymity against seller, bank and global/local observer.

As an instance, we explain the Use case diagram of buyer anonymity against seller (Figure 3). Buyer in this system should perform four main tasks: Login to the system, Selection of the desired goods, Payment, and Logout from the system

In selection of goods, no authentication is needed. Also in all phases we need to reply to anonymous buyer because buyer should be able to see her own buying basket.

In payment phase if buyer could not be charged correctly and ontime, anonymity of buyer will be revoked and tracking buyer becomes possible. So in payment phase aononymity is applied in committed and conditional manner.

Deployment diagram as it is shown in figure 4 represents the required message anonymity type in connection between system elements. Because we have committed anonymity in some parts of our system, as a dsign option we could use a trusted third party in anonymizer system which we show it with TTP Server. Also because we need authentication mechanisms we consider authentication server.

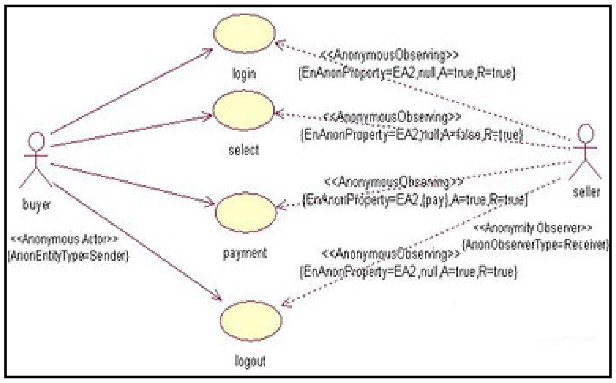
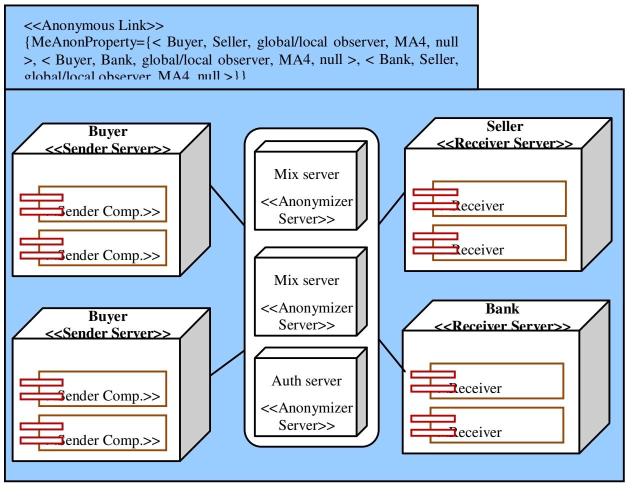


Fig. 3. Use Case Diagram of buyer against seller



<<Anonymous Link>>
{MeAnonProperty={< Buyer, Seller, global/local observer, MA4, null >, < Buyer, Bank, global/local observer, MA4, null >, < Bank, Seller, global/local observer, MA4, null >}}
Buyer <<Sender Server>>
<<Sender Comp.>>
<<Sender Comp.>>
Buyer <<Sender Server>>
<<Sender Comp.>>
<<Sender Comp.>>
Mix server <<Anonymizer Server>>
Mix server <<Anonymizer Server>>
Auth server <<Anonymizer Server>>
Seller <<Receiver Server>>
Receiver
Receiver
Bank <<Receiver Server>>
Receiver
Receiver

Fig. 4. Deployment diagram of E-payment system

6.2 Mix-Net protocol

David Chaum has suggested a protocol named Mix-Net for stablishing an anonymous channel in 1981 [11]. This protocol is used in different applications like sending anonymous e-mails or making anonymous connections in ISDN networks. Lots of anonymous protocols like Web mixe[9], ISDN-Mixes[12], Java Anon Proxy[10], Stop and Go Mixes[11], and Onion Routing[22] are based on Mix-Net Protrocol.
In this protocol, each mix has a public key which senders use to encrypt messages to that mix. The mix accumulates a batch of these encrypted messages, decrypts them, and delivers them to next receiver. Because a decrypted output message looks nothing like the original encrypted input message, and because the mix collects a batch of messages and then sends out the decrypted messages in a rearranged order, an observer cannot learn which incoming message corresponds to which outgoing message. We have analyzed the properties of this protocol using AnoModel in [5].
This protocol uses methods like batch sending, dummy message, adding random delays and so on to protect from traffic analysis. Flowchart of this protocol is shown in figure 5. In this flowchart gray blocks represent anonymity primitives. We have implemented this protocol using AnoAPI java package.

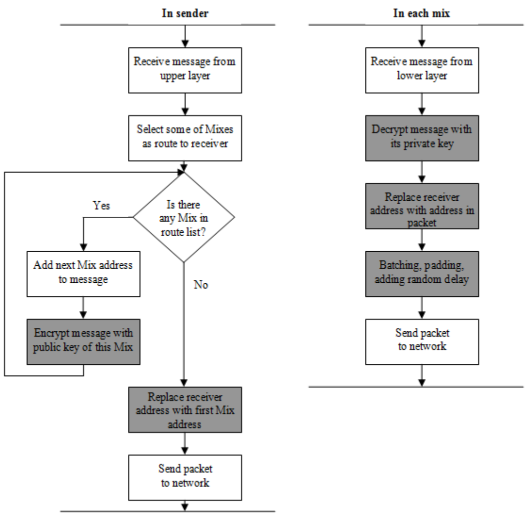
7. Conclusion

In this paper a methodology for developing anonymity applications was presented. The proposed methodology contains three parts: AnoModel, AnoUML, and AnoAPI to partially support requirement analysis, design and implementation of anonymity software development. Some of the advantages of the proposed methodology in developing anonymity software are as follows:

- Better understanding of anonymity concepts, properties and methods
- Better classification and comparison of anonymity services
- Software reuse with optional software components
- Reduction of software development time and cost
- Simpler software design because of separating anonymity components from other software logic

Also as future works, we plan to do the following activities:

- Formulating the methodology in a more formal manner
- Extending for more fine grained levels of anonymity
- Extending AnoUML to better concentrate on design aspects.
- Extending AnoAPI for supporting anonymity primitives in application layer.



In sender
Receive message from upper layer
Select some of Mixes as route to receiver
Is there any Mix in route list?
Yes
Add next Mix address to message
Encrypt message with public key of this Mix
No
Replace receiver address with first Mix address
Send packet to network
In each mix
Receive message from lower layer
Decrypt message with its private key
Replace receiver address with address in packet
Batching, padding, adding random delay
Send packet to network

Fig. 5. Flowchart of Mix Net protocol

8. References

[1] Qing Zhang, “A fair and anonymous E-Commerce schema”, PhD thesis, university of London, May 2007.

[2] D. Malkhi , “Lecture notes of Anonymity - Advanced Course in Computer and Network Security”, The Hebrew University, Jerusalem, May 2002.

[3] Software Engineering Institute, “Results of SEI Independent Research and Development Projects and Report on Emerging Technologies and Technology Trends”, Oct. 2004.

[4] Vincent NAESSENS, “A Methodology for anonymity control in electronic services using credentials”, PhD thesis, Katholieke Universiteit Leuven, June 2006.

[5] Marzieh Ispareh, Behrouz Tork Ladani, “A Conceptual Framework for Specification, Analysis, and Design of anonymity Services”, in Proceedings of the 2nd International Workshop on Privacy and Anonymity in the Information Society (PAIS’09), Saint-Petersburg (Russia), March 2009.

[6] V. Naessens, B. De Decker, “A Methodology for Designing Controlled Anonymous Applications”, In Proceedings of the 21th IFIP International Information Security Conference: Security and Privacy in Dynamic Environments, May 2006

[7] Bart De Win, et al., “Anonymity and Privacy in Electronic Services”, APES Project, Deliverable 2 - Requirement Study of different applications, 2001.

[8] David S. Rosenblum, “Advanced Analysis and Design Extension Mechanisms for UML”, Available from http://www.cs.ucl.ac.uk/staff/D.Rosenblum, 2008.

[9] O. Berthold, H. Federrath, and M. Kohntopp, “Project anonymity and unobservability in the internet”, In Computers Freedom and Privacy Conference 2000 (CFP 2000) Workshop on Freedom and Privacy by Design, April 2000.

[10] H. Federrath, “Jap: A Tool for Privacy in the Internet”, Available from http://anon.inf.tu-dresden.de/indexen.html.

[11] D. Kesdogan, J. Egner, and R. Buschkes,”Stop-and-go-Mixes Providing Probablilistic Anonymity in An Open System”, In Information Hiding, April 1998.

[12] A. P_tzmann, B. P_tzmann, and M. Waidner. Isdn-mixes, “Untraceable Communication with Very Small Bandwidth Overhead”, In "GI/ITG Conference: Communication in Distributed Systems", February 1991.

[13] M. Reed, P. Syverson, and D. Goldschlag, “Anonymous Connections and Onion Routing”, IEEE J. on Selected Areas in Communication. 1998.

Appendix A. AnoUML Tagged values

Tagged values	Constraints and description
EnAnoType	Specifies the Entity anonymity type. Based on AnoModel it can has one of the EA1 to EA7 values.
MeAnoType	Specifies Message anonymity type. Based on AnoModel it can has one of the MA1 to MA7 values.
EnAnoProperty	Specifies all anonymity properties of Entity as a set { EnAnoType , Authentication, Reply, C} in which EnAnoType shows Entity anonymity type. Other members are as AnoModel.
MeAnoProperty	Specifies all anonymity properties of Message as a set {Sender, Receiver, Observer, MeAnoProperty, C} in which Sender, Receiver, and Observer denotes anonymity of communication between sender and receiver from the viewpoint of observer. MeAnoProperty shows Message anonymity type and C is anonymity Constraints.
AnoObserverType	Specifies anonymity observer type. Based on AnoModel, it has one of the following values: Sender, Receiver, Local Observer, Global Observer, Members of anonymity provider system, or Users who have access to entities information.
AnoEntityType	Specifies Anonymous Actor type. Based on AnoModel it can has one of the following values: Sender, Receiver, or entity which its information is accessible
ServerType	Specifies Anonymizer System type. For example it can be TTP Server, Mix Server, Authentication Server, etc.

Appendix B: Anonymity primitives

Layer	Primitive name	Description	Anonymity service type
Com.	Padding	Changing the length of packet in route from receiver to sender	MA4 (Hiding messages length from viewpoint of network traffic observers).
	Dummy message	Generating and sending dummy messages	MA4 (Hiding messages delay from network traffic observers).
	Reordering messages	Reordering input messages before sending	MA4 (Hiding messages delay and mapping from network traffic observers).
	Batching	Storing messages and sending them in batch with variant delays.	MA4 (Hiding messages delay from network traffic observers).
	Adding random delays	Adding random delays in sending messages.	MA4 (Hiding messages delay from network traffic observers).
	Broadcasting messages	Broadcasting messages with special format such that only the real receiver can read it.	MA4 (Hiding receiver properties from network traffic observers).
	Caching messages	Caching and automatic answering to similar requests by anonymizer systems.	MA4 (Hiding messages delay and mapping from network traffic observers).
	Multiplexing	Sending multiple messages as one message by anonymizer system.	MA4 (Hiding messages delay, mapping and content from network traffic observers).
App.	Generalization	Replacing an attribute with some general ones while preserving the correctness of statistics.	Depending on the generalized information, can be one of EA4, EA3 or EA6 (Hiding owner properties or her name or both from others).
	Blind signature	Enabling an entity to sign on a message without knowing about the content of message.	Depending on the blinded information, can be one of EA4, EA3 or EA6 (Hiding sender properties or her name or both from the signer).
	Fair blind signature	A kind of blind signature with possibility of linking between original readable message and corresponding unreadable message.	Like blind signature, but provides conditional anonymity.
	Partially blind signature	A kind of blind signature, but some content of message may be readable for signer.	Like blind signature, but with partial anonymity of message content.
	Group signature	Signing a message without revealing the identity of signer.	EA4, EA3, EA6 (hiding signer information, name or both from sender).
	Zero knowledge proof	Proving the awareness of a secret to an entity without revealing that secret.	EA2 (Hiding entity properties from viewpoint of others).
Both	Encryption	Hides the mapping between input and output messages from others.	MA3 (Hiding sender and receiver information from observers)
	Filtering	Filtering and omitting the identification information from messages.	MA3 and EA1 (Hiding name and information of sender from receiver)
	Compression	Compressing messages before sending them.	MA4 (Hiding the message length and mapping between them from observers)
	Impersonation	Replacing and swapping identification information of different entities.	MA2 (Hiding sender properties from observers) EA4 (Hiding sender name from receiver) EA5 (hiding operation coherency from receiver) EA7 (hiding operation, name and properties of sender from receiver)
	Pseudonym	Using alias names for entities.	Like Impersonation