University of Crete
School of Sciences and Engineering
Computer Science Department

Identity management infrastructure for the digital world

Vassilis Poursalidis

Master’s thesis

Heraklion, November 2005
Identity management infrastructure for the digital world

Vassilis Poursalidis

A thesis submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Author:

Vassilis Poursalidis, Computer Science Department

Supervisor committee:

Christos Nikolau, Professor, Supervisor

Manolis Marazakis, Adjunct Lecturer, Member

Dimitris Plexousakis, Associate Professor, Member

Approved by:

Dimitris Plexousakis, Associate Professor
Chairman of the Graduate Studies Committee

Heraklion, November 2005
Abstract

In today’s Information Systems, including applications, platforms and services, users are requested to present credentials with local significance, in order to be authenticated and gain access to internal functionality. This way every user is required to have a different login-password combination for each online service, not to mention different credentials for different roles within a service. As a result users tend to choose poor passwords that are easy to remember, or even repeat the same login-password information in several services. This poses serious security threats to service providers and a severe privacy risk for end-users.

The solution is to shift to digital identity management systems. Such a system will issue a digital identity for every user and will be able to control the full life cycle of these identities, from creation to termination. Another aspect of such a system is the single sign-on mechanism, whereby a single action of user authentication and authorization can permit a user to access multiple services without the need to execute any local authentication procedure. We can think of the digital identity as the identity card for the digital world, but with privacy concerns in order to eliminate any threats posed by the nature of the Internet.

We started by evaluating existing identity management implementations and to propose a solution of our own. Our Identity Management Infrastructure (IMI) differs from similar approaches, by targeting a global scale deployment and problems that arise from such a goal. Another difference is that our technique sets the end-user as the sole holder of his identity information. This prevents the existence of a single point where multiple digital identities are held, which could become a target for potential attackers. The benefits (as seen from our approach) are improved security, accountability, reduced administration costs and privacy protection.

In this thesis we explore the opportunity of shifting to a digital identity management infrastructure, while preserving the anonymity currently experienced by users. Our goal is to provide accountability to digital identity holders, while allowing the user to remain anonymous and give service providers and end-users strong guarantees about the security aspects of our approach. The design and development of our infrastructure was mainly driven by these contradicting factors, but the ease of deployment was also among our considerations.

Finally, we consider the security risks involved in our approach. The model used identified the amount of information being exposed, in various attacks towards the components of our infrastructure. The results where crucial in order to identify the weak links of our approach and to provide us with directions for future work.
Περίληψη

Τα σημερινά Πληροφοριακά Συστήματα, όπως εφαρμογές, πλατφόρμες και υπηρεσίες, υποχρεώνουν τους χρήστες τους να παρουσιάζουν διαπιστευτήρια, για να έχουν πρόσβαση στις εισωτερικές λειτουργίες κάθε συστήματος. Τα διαπιστευτήρια αυτά περιορίζονται στην αρχή που τα εξέδωσε, και αναγνωρίζονται μόνο τοπικά. Με αυτόν τον τρόπο κάθε χρήστης πρέπει να έχει έναν διαφορετικό συνδυασμό ψευδονύμου-κωδικού πρόσβασης για κάθε υπηρεσία, καθώς και διαφορετικά διαπιστευτήρια για τους διαφορετικούς ρόλους που μπορεί να διαδραματίσει ο χρήστης σε μια υπηρεσία. Το αποτέλεσμα είναι ότι οι χρήστες τείνουν να διαλέγουν απλούς κωδικούς πρόσβασης που είναι εύκολοι στην απομνημόνευση, ή ακόμη να επαναλαμβάνουν τον ίδιο συνδυασμό ψευδονύμου-κωδικού πρόσβασης σε πολλά συστήματα. Έτσι οι υπηρεσίες έχουν να αντιμετωπίσουν σοβαρές απειλές ασφάλειας, ενώ οι τελικοί χρήστες αντιμετωπίζουν την απειλή παραβίασης των προσωπικών τους δεδομένων.

Η λύση είναι να μεταβούμε σε συστήματα διαχείρισης ψηφιακών ταυτοτήτων. Τα συστήματα αυτά εκδίδουν μια ψηφιακή ταυτότητα για κάθε χρήστη και είναι ικανά να ελέγξουν τον πλήρη κύκλο ζωής αυτών των ταυτοτήτων, από την δημιουργία μέχρι τον τερματισμό. Μια ακόμη δυνατότητα ενός τέτοιου συστήματος είναι ο μηχανισμός μοναδικής εισόδου, όπου μια και μόνο πράξη για πιστοποίηση και εξουσιοδότηση επιτρέπει τον χρήστη να έχει πρόσβαση σε όλες τις υπηρεσίες, αποφεύγοντας την εκτέλεση μιας τοπικής διαδικασίας εισόδου. Μπορούμε να θεωρήσουμε την ψηφιακή ταυτότητα σαν το ανάλογο της πραγματικής ταυτότητας για τον ψηφιακό κόσμο, με ιδιαίτερο ενδιαφέρον στην εξάλειψη της πιθανότητας παραβίασης των προσωπικών δεδομένων του χρήστη, πρόβλημα εγγενές από την φύση του Διαδικτύου.

Εξελικνήσαμε με την επισκόπηση των υλοποιήσεων συστημάτων διαχείρισης ψηφιακών ταυτοτήτων και προχωρήσαμε στην δημιουργία μιας δικής μας λύσης. Η δική μας Υποδομή Διαχείρισης Ταυτοτήτων (ΥΔΤ) διαφοροποιείται από παρόμοιες προσεγγίσεις μιας και είχαμε ως στόχο να αναπτυχθεί σε παγκόσμια κλίμακα και να αντιμετωπίσει προβλήματα που ανακύπτουν από αυτόν τον στόχο. Μια ακόμη διαφορά είναι ότι ορίσαμε τον τελικό χρήστη ως τον μόνο κάτοχο της πληροφορίας που σχετίζεται με την ψηφιακή του ταυτότητα. Σκοπός μας ήταν να αποτρέψουμε την ύπαρξη ενός σημείου όπου κρατούνται πληροφορίες για πολλαπλές ψηφιακές ταυτότητες, σημείο που μπορεί να γίνει στόχος από κακόβουλους χρήστες. Τα πλεονεκτήματα της προσέγγισής μας είναι βελτιωμένη ασφάλεια, υπευθυνότητα, μειωμένο κόστος διαχείρισης και προστασία των προσωπικών δεδομένων των χρηστών.

Σε αυτήν την εργασία αναζητούμε την δυνατότητα αξιοποίησης ενός συστήματος διαχείρισης ψηφιακών ταυτοτήτων, ενώ παράλληλα να διατηρείται η ανωνυμία των χρηστών, σύμφωνα με την παρούσα κατάσταση. Ο στόχος μας είναι να παρέχουμε υπευθυνότητα στους κατόχους ψηφιακών ταυτοτήτων, ενώ παράλληλα διαφυλάσσουμε την ανωνυμία τους και παρέχουμε ισχυρές εγγυήσεις πάνω στην ασφάλεια του συστήματός μας, τόσο στους τελικούς...
χρήστες όσο και στις διαθέσιμες υπηρεσίες. Ο σχεδιασμός και η ανάπτυξη της υποδομής έγινε με βάση τα παραπάνω αντικριτικά κριτήρια, αλλά έχοντας υπ’ όψιν και την ευκολία της ενσωμάτωσης της λύσης μας.

Τέλος, μελετάμε τους κινδύνους ασφαλείας που υπάρχουν στην προσέγγισή μας. Τα μοντέλα που χρησιμοποιήσαμε ανέδειξαν την πληροφορία που εκθέτεται σε κάθε τύπο επίθεσης στα δομικά στοιχεία της υποδομής μας. Τα αποτελέσματα ήταν αποφασιστικά για να αναγνωρίσουμε τις αδυναμίες της προσέγγισής μας και για να προβλέψουμε κατευθύνσεις για μελλοντική εργασία.
# Table of Contents

1. Introduction .................................................................................................................. 15
2. Related work .................................................................................................................. 17
   2.1 Identities in the digital world .................................................................................. 17
   2.2 Similar approaches .................................................................................................. 17
      2.2.1 Credential repositories .................................................................................. 18
      2.2.2 .NET Passport .............................................................................................. 18
      2.2.3 Shibboleth .................................................................................................... 20
      2.2.4 WS-Federation ............................................................................................ 22
      2.2.5 Liberty Alliance project .............................................................................. 25
   2.3 Technology ............................................................................................................... 26
3. Infrastructure specifications ............................................................................................ 27
   3.1 Architecture description ......................................................................................... 27
   3.2 Digital identity management framework .................................................................. 29
      3.2.1 Authorities aggregation catalogue ............................................................... 32
      3.2.2 Identity issue authority ................................................................................. 35
   3.3 Service mediator ..................................................................................................... 38
   3.4 End-entity middleware ........................................................................................... 40
   3.5 Communication protocols ...................................................................................... 45
      3.5.1 Authority lookup ........................................................................................... 47
      3.5.2 Identity creation ............................................................................................ 48
      3.5.3 Identity validation .......................................................................................... 49
      3.5.4 Service login ................................................................................................. 50
      3.5.5 Identity authentication .................................................................................. 52
      3.5.6 Authority validation ....................................................................................... 53
   3.6 Discussion ................................................................................................................ 55
4. Implementation ............................................................................................................... 59
   4.1 Custom packages ..................................................................................................... 59
      4.1.1 Software package "imi.util" ........................................................................... 59
      4.1.2 Software package "imi.util.xml" ................................................................... 61
      4.1.3 Software package "imi.util.crypto" ............................................................... 63
      4.1.4 Software package "imi.trust.util" ................................................................. 65
      4.1.5 Software package "imi.security" ................................................................... 66
      4.1.6 Software package "imi.network.message" ................................................... 73
      4.1.7 Software package "imi.trust.protocol" ......................................................... 74
      4.1.8 Software package "imi.db" ............................................................................ 79
      4.1.9 Software package "imi.db.queries" ............................................................... 79
   4.2 Software components .............................................................................................. 80
      4.2.1 Software package "imi.trust.digidentity" ......................................................... 80
      4.2.2 Software package "imi.trust.service" ............................................................. 84
      4.2.3 Software package "imi.trust.middleware" ..................................................... 85
      4.2.4 Software package "imi.trust.digidentity.protocol" ....................................... 88
      4.2.5 Software package "imi.trust.service.protocol" ............................................ 89
   4.3 Design decisions ..................................................................................................... 89
5. Evaluation of implementation ......................................................................................... 91
   5.1 Attacks to our system .............................................................................................. 91
      5.1.1 Attack to a service provider .......................................................................... 92
      5.1.2 Attack to an AAC ......................................................................................... 93
      5.1.3 Attack to an IIA ............................................................................................ 94
      5.1.4 Attack to an end-entity middleware ............................................................. 95
      5.1.5 Combined attack to a service provider and an IIA ....................................... 98
      5.1.6 Man-in-the-middle attack ............................................................................. 99
   5.2 Measurements ......................................................................................................... 100
   5.3 Potential applications ............................................................................................. 102
6. Conclusions and future work ......................................................................................... 105
List of Figures

Figure 1: Steps followed in order to determine if the principal behind the User Agent can access a resource held by the Service Provider ...............................................................22
Figure 2: Messages exchanged to gain access to remote resources is WS-Federation ..........25
Figure 3: Sequence of interactions to perform a single sign-on and authenticate secondary identities without using another set of credentials or end-entity interaction .................29
Figure 4: Example of the responsibility and coverage domain of an authority in the digital identity management hierarchy .....................................................................................33
Figure 5: Schematic representation of the information inside a digital identity (A) and the nested information encapsulated in the personal information block (B) .........................36
Figure 6: An example showing how the client mediator communicates with the end-entity middleware in order to establish a virtual communication channel .........................42
Figure 7: Steps for establishing a virtual channel between the service client and core service functionality. The sequence of the steps is given by the numbers preceding the description 43
Figure 8: Representation of the steps followed when creating the end-entity mediator creates a new secondary identity .................................................................43
Figure 9: Representation of the steps followed when validating a secondary identity ........44
Figure 10: Anonymous (A) and Eponymous (B) channel establishment procedure ..........46
Figure 11: Messages exchanged in order to receive an authority identity .........................48
Figure 12: Messages exchanged in order to create a digital identity ................................49
Figure 13: Messages exchanged in order for an individual to validate his digital identity ...50
Figure 14: Messages exchanged in order for an individual to gain access to a service .......52
Figure 15: Messages exchanged in order for a service mediator to authenticate the user requesting access to the service functionality ..........................................................53
Figure 16: Two authority chains. The left chain is connected from root to leaf, while the right one is not included in the hierarchy of the DIM framework ........................................54
Figure 17: Messages exchanged in order to verify the legitimacy of an authority ..........55
Figure 18: Example XML document used for data transfer ..............................................62
Figure 19: The head portion of the XML document in Figure 18 ......................................62
Figure 20: The XSD document used to validate the XML in Figure 18 ...............................63
Figure 21: The steps for creating (A) and validating (B) digital signatures .......................64
Figure 22: Example XML document representing a digital signature ...............................65
Figure 23: Example XML document representing an authority ........................................65
Figure 24: Example XML document representing an authority identity .........................67
Figure 25: Example XML document representing a user identity ....................................69
Figure 26: Example XML document containing the personal information of a user ......69
Figure 27: Example XML document containing the private information of a user ..........70
Figure 28: Example XML document representing an issuer certificate ............................71
Figure 29: Example XML document representing a service certificate. The “issuer” and “timestamp” tag details were omitted as they are similar with the ones in Figure 28 ....72
Figure 30: Example XML document representing a status certificate. The “issuer” and “timestamp” tag details were omitted as they are similar with the ones in Figure 28 ....72
Figure 31: Communication using unsecured channel .....................................................74
Figure 32: Communication protocols hierarchy .............................................................75
Figure 33: Payload communication for an Anonymous (A) and an Eponymous (B) connection .................................................................76
Figure 34: XML document passed from AnonymousServer to MessageHelper ..................77
Figure 35: XML document passed from AnonymousClient to MessageHelper used to initiate an Anonymous connection .................................................................77
Figure 36: XML document passed from AnonymousClient to MessageHelper used to conclude the challenge-response protocol and exchange the first request .....................77
Figure 37: XML document passed from EponymousServer to MessageHelper ...............78
Figure 38: XML document passed from EponymousClient to MessageHelper used to initiate an Eponymous connection .................................................................78
Figure 39: XML document passed from EponymousClient to MessageHelper used to conclude the challenge-response protocol and exchange the first request ...........................................78
Figure 40: Layered software element architecture ...........................................................................79
Figure 41: Example XML document containing the private information of the user as presented by the end-entity middleware ...............................................................................83
Figure 42: This document contains the digital identity of the user along with his identity key ........................................................................................................................................83
Figure 43: The credentials information to be validated by the IssuerAnonymous ..................84
Figure 44: Challenge request sent by the IssuerAnonymous to authenticate the individual ..84
Figure 45: Challenge response returned by the end-entity middleware ..................................84
Figure 46: Credential information sent by the end-entity middleware to the service mediator ..................................................................................................................................................85
Figure 47: Form presented by the end-entity middleware in order to create a new digital identity for the individual ..................................................................................................................................................88
Figure 48: Form requesting the individual to enter his password in order to activate his digital identity and the information held by the end-entity middleware ..................................................................................................................................................88
Figure 49: Form requesting the individual to select a secondary identity that will be used for the current session of service access ..................................................................................................................................................88
Figure 50: Example mapping of services and secondary identities ..............................................88
Figure 51: Sequence of interaction to perform authentication based on an implant ..........96
Figure 52: Sequence of interactions to illustrate the man-in-the-middle attack ......................100
List of Tables

Table 1: Notations used in this thesis ................................................................. 26
Table 1: Notations used for cryptography ............................................................ 45
Table 2: The available interfaces and a brief list of properties ............................. 46
Table 3: Comparison of the approaches in identity management systems .............. 57
Table 4: Notations used in this chapter ............................................................... 59
Table 5: A description of the methods provided by Globals software element .......... 61
Table 6: The name of the software element and a description of the task performed .... 80
Table 7: Attack scenarios and a brief description on the outcome of the attack .......... 92
Table 8: Overhead added by our approach, as compared to current solutions .......... 102
1 Introduction

“It’s recently become clear that the software for managing user identity and authentication is one of the key building blocks of the emerging Internet operating system”

- Tim O’Reilly

In today’s Information Systems, including applications, platforms and online services, users are required to hold multiple credentials in order to digitally identify themselves to different services, or in different roles. The identification process allows services to authenticate users, granting them access to internal service functionality thus providing a personalized experience. But these credentials have local significance, meaning that they are only eligible for the originating service.

The problem of multiple credentials is expected to become even more challenging for end-users before it gets easier. This foresight is partly based on the growth of online services, and partly from the increasing need to utilize remote resources and services. Beyond the basic problem of users having trouble remembering multiple username and password combinations, current technology presents security risks and administrative costs to service providers. In fact, current technology drives most users into creating accounts using poor and easy to remember passwords, or even to use the same login-password combination across multiple services. On the other hand, the security conscious user will choose different passwords for every different account. The accounts held by the first group of users will be easily breached, posing security threats to service providers and exposing the privacy of account holders. The second group will increase administrative costs, due to forgotten passwords.

The inability of existing solutions to seamlessly authenticate users has drawn attention to digital identity management systems. Such systems address the problem of multiple credentials by incorporating a Single Sign-On (SSO) mechanism. The mechanism is a single point where users perform a login procedure once. If the outcome is successful a set of assertions is returned, which can be presented to services and gain immediate access. The assertions certify the identity of the holder to the service, preventing it from issuing custom credentials. Besides the SSO mechanism, identity management systems also control the full life cycle of digital identities, from creation to termination. This assures services about the information they receive concerning digital identities, while users are held accountable for their actions. By accountable we mean that the user can be traced back from their digital identity, in case of abuse. But accountability must respect the privacy of users, by preventing the linkage of actions to physical persons if digital identities are to be used legally.
facilitate this identity management systems allow users to create as many pseudonyms as they like, using a single digital identity. This eliminates the possibility of linking certain actions to the digital identity that executed them, without the express permission from the authority that issued the digital identity of the user.

Our digital identity management system is a multiple pseudonym identity system, shifting the control and creation of the plethora of pseudonyms to end-users. In fact pseudonyms are secondary identities, derived from an identifier (master identifier) that was supplied from the authority that issued the digital identity. The advantage of our scheme is twofold. On the one hand users are able to preserve their anonymity, as experienced today. This is achieved by allowing the user to control his secondary identities and the way in which they are used. He is free to create as many secondary identities as he pleases and present multiple identities in a single service. On the other hand services are certain that secondary identities belong to a physical person, avoiding the custom development of authentication methods. Also, services are certain that if legal issues arise with one of the secondary identities the master identifier and the actual user can be traced back, with the help of the authority that issued his digital identity. Our approach also protects the privacy of the user, by preventing the existence of a single point where multiple digital identities are held. This single point could become a target for potential attackers and result in mass identity exposure. Concluding, we review the primary contributions of our thesis:

- A survey of current trends in Digital Identity Management
- The design and development of an infrastructure for large scale digital identity management
- The implementation of a privacy enhanced identity management solution, that sets the end-user the sole holder of his identity information
- A roadmap for future research on digital identity management systems

The rest of this thesis is organized in the following way: Chapter 2 clarifies the terminology used, presents background information and assisting technologies used. Chapter 3 describes our infrastructure specification, its components and communication patterns between them, to provide the desired functionality. Chapter 4 gives insights on our implementation, raging from message formats to protocol structure. Chapter 5 evaluates our approach by examining the damage and the amount of information exposed in a number of attack scenarios. Finally chapter 6 presents directions for future work.
2 Related work

Our research topic concerns the better understanding of digital identity management systems and the development of an infrastructure to address specific shortcomings that arise and may negatively affect their wider deployment. In our attempt to investigate the field we had to familiarize ourselves with current advancements in digital identity management, privacy centric technologies and applied cryptography.

2.1 Identities in the digital world

Identity is a unique piece of information associated with an entity. Identity itself is simply a collection of characteristics which are either inherent or are assigned by another [27]. In the digital world, a digital identity comprises electronic records that represent network principals, including people, machines, and services [25], [34], [35]. But to be able to create, maintain and use digital identities the deployment of a digital identity management system is required. This infrastructure will make identities operational in the process of authentication and in mapping of identifiers to the information needed for identification and authorization [27], [29]. The functionality described sets the basis for single sign-on solutions, where a system attempts to capture identification and authentication information once, and provide it to services accessed by a user automatically. The objective of single sign-on systems is to reduce the number of different authenticators a user must have/know, and to reduce the frequency with which the user must provide those authenticators to systems.

But given a unique set of credentials that can identify us presents major privacy threats. We can overcome the problem with the use of pseudonyms, where one has the ability to prove a consistent identity without revealing oneself, instead using a pseudonym. Pseudonymity is a state which combines many of the advantages of having a known identity with the advantages of anonymity. The main difference between anonymity and pseudonymity is that while in anonymity the identity is not known, in pseudonymity, there exists a separate persistent “virtual” identity but it cannot be linked to a physical person [25], [32]. The unique digital identity issued is simply used to create multiple and dependable secondary identities that can be used in different services, preserving the users privacy but still holding him accountable [36].

2.2 Similar approaches

The most active field in identity management systems is identity federation. The idea is that multiple organizations form a federation and authentication tokens from one organization...
in the circle are considered valid to the remaining of the organizations [28], [30], [33]. The user of such a system is able to authenticate across multiple information systems or even organizations, using a single set of credentials. Research to the field is mainly driven by cooperating organizations requiring to access resources on joint projects.

In the next sections we will review five approaches and try to identify the limitations and advantages of each one. Every system (except the Liberty Alliance project in section 2.2.5) uses a different technique to face the problem, setting its own goals and assumptions. This enumeration of systems will clarify the need for further study in the field and define the problem that our approach will try to overcome.

2.2.1 Credential repositories

To redeem users from having to remember multiple login names and passwords, credential repositories [1] provide the means to manage and use these accounts through a single “wallet”. Every user has his personal wallet, where he registers his accounts information in a per service basis. A request from the user to access a service will be intercepted by his wallet, in order to check if specific credentials exist for the service in question. If so the information is retrieved and the login procedure is executed automatically. Otherwise, the user must first register to this service and add the custom set of credentials to his wallet. Subsequent request to use the service again will be able to automatically login the user, providing him direct access to the functionality available.

As we can see credential repositories try to provide a short-term solution to the problem of multiple accounts. The approach they take is very successful, as there is no need to apply changes to service providers or depend on newly created authorities. In fact, their only requirement is special purpose software running on the user’s computer system. Their simplicity also benefits the security offered. The user is the sole holder of his credential information and responsible for keeping it private. The disadvantage is that accounts remain unbound from their physical holders, giving no incentives for good behaviour. The results are poor confidence, when performing online business transactions, and consent to anonymous misuse and abuse of service functionality.

2.2.2 .NET Passport

Most likely the largest public single sign-on network in existence is Microsoft’s .NET Passport [2], [3], [4]. The .NET Passport is important not because of using advanced technological solutions, but for the reason that the company is in position to drive most people in registering for an account. In fact by requiring all Microsoft online services to use .NET Passport accounts the service became widely known and is considered the most
prevalent single sign-on implementation. To be precise the service is not a single sign-on solution, but rather a single credentials solutions. This means that a user will create an account to the .NET Passport service, and use these credentials to individually login to any service in the participant's network. The procedure followed to login to a service includes a redirect to the .NET Passport web site, the completion of the login procedure and finally the redirection of the user agent to the service that initiated the request. Through this procedure the service is also notified about the outcome of the login process, so as to deny or allow access to the user. The .NET Passport assigns a unique pseudo-random 64 bit number to every account created and every service in the participant's network use this number to refer to the user. This way no personal information is exchanged between the .NET Passport and the services in the network, but the service is free to use the 64 bit number and assign personal information given by the user specifically to the service.

From the user's point of view there is a clear benefit: he only needs to remember a single login and password in order to login to every service in the participant's network of the .NET Passport system. On the hand of businesses these is also a clear benefit, as there is an out of the box solution to access an enormous database of possible customers, providing a richer experience by not requiring a specific set of credentials. Finally the biggest advantage of the .NET Password is that it requires no custom software installed on the end-users device. It can work with any web browser that supports cookies, after all the target group of the .NET Passport is Internet users accessing services via the web.

We will now examine various drawbacks imposed by the incorporation of such a solution. For the record the .NET Passport system was released in 1999 and has proven quite insecure in its first years of operation [5], but we will not focus on implementation flaws but rather on inherent design problems. The first and major concert is that a single authority operates and is charge of the credentials and the huge amount of personal information. This makes the system an attractive target, not only to gain credential information but also to issue denial of service attacks. Another problem is the assignment of a unique identifier to every user. The problem is twofold: the user is not free to register in the same service twice, using different profiles, and secondly services can collude and exchange private user information without his consent. Another problem is the absence of secure communication during the entire session where the user remains logged in. This way an active attacker can gain the session information in transit and impersonate a user, achieving full access to the victim's account. Finally to become a member of the .NET Passport network Microsoft charges a set-up and annual fee and requires custom software installed on the Web servers of the service providers. The fees are mainly because the service provider is given a large database of potential customers and the custom software because the .NET Passport depends on the awareness of service providers to provide the desired functionality.
Many changes happened since the first launch of the service and most of them focus on security problems presented by the absence of custom end user software. In this direction the service was enhanced to prevent the use of third party cookies [6], incorporate the SSL protocol [7] during the login procedure and the refinement of information exchanged between the .NET Passport and the service providers. The final step was taken to prevent cross site scripting attacks [8] that led to information exposure between service providers. The big change to drive the .NET Passport to its fully potential was the incorporation of a client authentication module in Windows XP that is aware of the participant’s network. This way a local account in Windows XP can be integrated with a .NET Passport account and a login to the local account will trigger the login procedure of the Passport account. Next any application, including the Web browser, can use the API provided by the module to authenticate the user to service providers in the participant’s network. This step allows the .NET Passport to support non-Web authentication mechanisms (just like Windows Messenger [9]) and present a truly sign-on solution, as credentials will not be re-entered.

Concluding we could say that this solution, from a security perspective, is worse than the credential repositories mainly for the reason that there is a single authority in charge of a huge amount of identifiable and personal information. Furthermore the use of a unique identifier per user poses threats in the privacy of users, as services can collude and exchange personal information for their common set of identifiers. The user and the .NET Passport system are unable to prevent or even detect such acts. This is why the Passport must allow the use of multiple identifiers to different or the same service, allowing the user to present the same identifier to different services when he wants to join the benefits of both services under a single identifier.

2.2.3 Shibboleth

Shibboleth [10] is an architecture that enables organizations to build single sign-on environments that allow users to access web-based resources using a single set of credentials. The research project is being developed by the Internet2 [12] middleware group using open standards (such as SAML [11]) and targeting the demand for inter-organizational access to networked services. Broadly speaking, the Shibboleth architecture defines a set of interactions between an identity provider and a service provider to facilitate web browser single sign-on and attribute exchange.

In the Shibboleth model, the organization is responsible for authenticating the user - that is, for checking that the credentials the user presents are correct (typically with a username/password combination). The organization is also responsible for providing information about the user; for example the department or project the user works in. This information is called attribute information and will be used to decide whether or not to give
access to certain resources. The organization is called the Identity Provider. The decision to authorize access to information is the responsibility of the owner of the resource, and is based on the user's attribute information. Attribute information can be as simple as “member of certain department” or as complex as “member of project team who has signed up to the project terms and conditions”. The provider of the resource is called the Service Provider. Since only attribute information is exchanged between the Identity Provider and the Service Provider, and not user identifiable information, the privacy of the user is preserved. The following sequence diagram [Figure 1] illustrates the set of required and optional interaction (dashed lines) between the transacting parties. The additional WAYF Service stands for “Where Are You From?” and is a centralized mechanism for interactively determining a principal's Identity Provider. In fact is a point where the user selects from a list his organization.

Let us assume that two organizations A and B collaborate on a project. What we want is for users of both organizations to have access on resources relevant to the project no matter the organization they reside. The two organizations must form a “circle of trust”, meaning that the Service Provider of organization A must accept assertions from the Identity Provider of organization B regarding the joint project and vice versa. Of course this is just an example, so numerous circles of trust can coexist in a single Service Provider and can be much wider (two or more Identity Providers in the same circle). Every circle of trust has its own criteria for organizations that want to join it, and defined levels of trust for access to the set of services. So a user with permission to access the project resources will be able to login to the local Identity Provider and use the attributes assigned to him to access project related resources on the remote Service Provider. If a user from the same organization is not working on the joint project and tries to access the same resource, the assertion will not contain the needed information and the Service Provider will deny him access.

The Shibboleth architecture we discussed tries primary to cope with problems faced by education institutions and their partners (the Internet2 consortium consist of universities). Among other the problem space includes access to off campus digital library resources, online education courseware, co-taught lessons Web sites and research/collaboration Web sites. Currently the solution practices try to utilize IP-based access, proxy servers, shared credentials and finally the creation of additional accounts for every user eligible to access the online resources. A replace to all these is for universities to incorporate the Shibboleth architecture and establish circles of trust, allowing access to resources based on assertions from Identity Providers of remote universities. This way access is based on role attributes, is available from everywhere on the Web and reduces dramatically administration costs. The same practices can be applied to different organization that cooperate and need to exchange resources.
Just like the .NET Passport, Shibboleth requires no custom software installed on the client’s machine, simply a standards Web browser is sufficient to utilize the underlying architecture. The burden is transferred to local system administrators, to provide the needed attributes, and to remote resource managers, to control access to resources based on the attributes of each user. The advantages of Shibboleth are that it provides a clean single sign-on solution, while preserving the privacy of its users. The only drawback is that requires the different domains to enter a circle of trust, where assertions from a foreign domain are trusted, and access to resources will be control by the presented attribute values. Unlike the .NET Passport the Shibboleth architecture was not designed to be used as a global scale single sign-on solution. The focus was on organizations and institutions that need to collaborate, without having the huge administration and strict solution imposed by today’s implementations. Next we will see the WS-Federation and Liberty Alliance project, which present similarities with Shibboleth but where designed for use in a broader environment, such as public Internet services.

2.2.4 WS-Federation

IBM, Microsoft, RSA Security, BEA Systems and VeriSign are working on a set of specifications known as the WS-Roadmap or WS-*. WS-Security was the first white paper they published, followed by WS-Policy, WS-Federation and WS-Trust, with more to come. Their goal is to build a set of security models for Web Services [13] in order to provide a complete support framework towards security, access control and federated identity.
management. Our interest focuses on the WS-Federation ([14], [15]) specifications, where it defines the mechanisms to allow different realms or domains to federate by allowing and brokering trust of identities, attributes and authentication between participating Web Services.

The simple truth behind the above technical definition is that WS-Federation specifies the required software elements and their interactions in order to establish a model and a framework for identity federation. The ultimate goal of WS-Federation is to provide a single sign-on environment, where end-users authenticate once to local Identity Providers, and are provided with security tokens that can be used to access remote resources. The specification goes a step further by providing anonymity and preserving the privacy of end-users. In fact the security tokens presented by the user agent, when accessing a resource, contain no personal or globally identifiable information. The security tokens only verify that the request is eligible or not to access the in question resource (access credentials). If a unique identifier is required, in order for a service to provide personalized experience to the end-user, then a unique pseudonym with local significance is used (local identifier). The privacy of the end-user is preserved through the Attribute Service, while his anonymity with the Pseudonym Service. Both services receive and alter security tokens, allowing the end-user presenting them to access the desired resource and preventing the resource from inferring information about the requestor.

To further clarify the details of WS-Federation specifications we will discuss the interactions shown in [Figure 2]. The diagram presents two organizations (separated by the dotted line) with the need to federate their local identities and gain access to cross-organization resources, using WS-Federation. Just like in Shibboleth the two organizations must first enter a circle of trust, meaning that security tokens issued from one organization’s Identity Provider will be considered valid to the rest organizations in the circle. This is illustrated with the arrow named Trust, which defines a mutual trust for credentials issued by both organizations. The first step towards utilizing the WS-Federation framework is for the Requestor to complete a local authentication procedure using a set of credentials (1a). The credentials will be verified by the Identity Provider (IP/STS) and an identity token will be returned, bearing evidence on the outcome of the authentication. Since the token information is considered a security token, the Identity Provider also plays the role of a Security Token Service (STS). The role of a STS is to make assertions on evidence that it trusts, to whoever trust it. In our case the IP/STS provides identity tokens to end-users that could successfully complete the authentication procedure (trusted credentials); returning an assertion that will be trusted in the circles of trust the organization belongs. Next the IP/STS will contact the Attribute and Pseudonym Service (1b), so as to append in the assertion the attributes associated with the Requestor. This step will enhance the information contained in the security token so as to include informative name-value pairs. These pairs will be used to
control access on remote resources and provide non-identifiable information about end-users. Continuing the procedure the Requestor must contact the IP/STS of the remote organization (2). The idea is that local Resources will only accept assertions made by local IP/STS. The local IP/STS is responsible for maintaining the organizations that are considered trusted and providing assertions of its own to Requestors originating from trusted organizations. This way there is a single point where trusted organizations are held and maintained, while local Resources will transparently treat every Requestor like originating from within the organization. But to issue local assertions the IP/STS must contact the Attribute and Pseudonym Service (step 3), and acquire access control information about the Requestor. For example the IP/STS may lookup for attributes associated with a joined project, in order to issue local assertions. But once the Requestor has been given these assertions, he is free to present them to any Resource on the remote organization and gain access (4). Of course, to gain access, certain attributes must exist in the assertions presented. The final step we review is optional and is executed if the Resource needs to acquire identifiable information about the user (step 5). This includes personal information and or a unique identifier for accounting purposes. To execute this step first the Requestor must authorize it, and secondly the IP/STS of the organization, the Resource belongs to, must allow it (1c).

From the review it is clear that WS-Federation outweighs the approach taken by Shibboleth, providing a richer system for identity federation and targeting a wider application domain. But still both approaches have a lot in common. For example both approaches require no custom software installed on the client side and use attributes to control access to resources. The main differences include the pseudonym service and the fact that Resources are simpler as they provide access only to local credentials.

The WS-Federation was published on July 2003 and still remains a specification document, with no working implementation. On October 2004 IBM became a board member of the Liberty Alliance project, but stated that will continue to support the WS-Roadmap initiative. It is believed that this turn will convergence the specifications provided by both Liberty Alliance and WS-Federation.
2.2.5 Liberty Alliance project

The last approach we will examine is provided by the Liberty Alliance organization, a consortium of more than 150 companies, non-profit and government organizations. Its purpose is to develop open standards in federated identity management that supports all current and emerging network devices. Liberty has released a two step specifications in order to achieve its goals [16], [17], [18]. The first step tries to set the basis for federated identity management, given the current technologies deployed worldwide. The specifications define a solution similar to the one discussed in Shibboleth project, with the exception that project Liberty allows users to have pseudonyms when accessing a service, besides the use of attributes. The ultimate goal of phase one is to provide a simple to deploy yet concrete solution for federated identity management. The second phase extends the work of the first and provides a solution very similar to the framework presented in WS-Federation [19], [20]. The similarity is noticed to both interactions between components and components responsibilities.

The members of Liberty Alliance are very active, providing implementations based on the proposed standards and publishing information about new directions and best practices in identity managements. It is believed that an implementation using Liberty specifications will be used in mobile markets, to provide a seamless experience to online services via the user’s mobile phone.
2.3 Technology

Our work takes advantage of current approaches and technologies in applied cryptography. We make use of both symmetric and asymmetric ciphering techniques, while utilizing digital signatures and certificates [21], [26], [31]. Our goal is to provide secure communication channels to all transacting parties and verify the integrity of information stored locally by the components of our infrastructure. To facilitate this we use an open source cryptography API developed by BouncyCastle [22], a group of Internet users trying to overcome limitations presented by current cryptography implementations in the Java programming language. The following table presents the notations we use when referring to cryptography and to cipher keys for our protocol implementations [24].

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_{A,B}$</td>
<td>Secret key shared by A and B</td>
</tr>
<tr>
<td>$K_A^+$</td>
<td>Public key of A</td>
</tr>
<tr>
<td>$K_A^-$</td>
<td>Private key of A</td>
</tr>
<tr>
<td>$R$</td>
<td>Random number</td>
</tr>
</tbody>
</table>

Table 1: Notations used in this thesis

Another emerging technology we used is XML. This mark-up language is considered the de facto standard for information exchange between software components. We used it to define the structure of messages transmitted and the information held in local repositories by each component. More details can be seen in Chapter 4.
3 Infrastructure specifications

This chapter discusses the issues and implementation details followed by our identity management infrastructure. It provides a step-by-step discussion of the semantic components of the system that were developed along with the communication patterns between these components.

3.1 Architecture description

Broadly speaking, our infrastructure defines a set of entities and the interaction between them to provide a global scale single sign-on system. In that sense different modules of software provide only part of the desired functionality to make available a robust and distributed infrastructure for identity management. These modules can be grouped into a digital identity management framework, the service mediator and the end-entity middleware. The first is a framework that supports the management and authentication of identities in the digital world. Besides creating and revoking identities, the framework provides to services proof of the legitimacy of the end-entities.

A service based on the entity's credentials can allow or deny access to the functionality provided, but in either case it is assured of communicating to an eligible end-entity. The service mediator handles all details about the authentication of the entity's identity, outsourcing the burden of accounting to the digital identity management framework. The outsourcing also benefits the service in terms of mapping an identity to a real world entity, since the digital identity management framework assures the existence of a real world entity behind every digital identity. Given this strong validation scheme the end-entity must be protected from privacy or anonymity violations.

The end-entity middleware preserves the user's privacy by creating different secondary identities for each service and its anonymity by creating different secondary identities for the same service. By creating secondary identities that cannot be correlated and at the same time can be validated by the digital identity framework, the middleware achieves both strong validation and end-entity protection. On top of this middleware tailor-made software can be executed providing specific service functionality. The content of a secondary identity is just a unique identifier, a partial identity containing the number of this secondary identity and the issuer of the identity. By shifting the creation and management of secondary identities to the end-entity middleware, the user is responsible for the information that is shared amongst services and of the extent of information given. The main advantage from moving this information to the edges of our infrastructure is the absence of a single repository containing a list of user profiles. In this way users are protected from exposure of sensitive information and every user is responsible for protecting their profile.
We have described the general functionality provided by the software modules that enable the existence of a global scale single sign-on system, but more important than their internal workings is the strict communication implementation that each module supports. To avoid information exposure from security hazards and attempts to breach the global scope of the system every software module handles only parts of the information. Only through secure communication and a well-defined protocol can this information be combined to serve the purposes of validating and authenticating an end-entity. The next figure [Figure 3] presents the basic steps for performing single sign-on and subsequently using the digital identity management framework to authenticate users to services. While most of the details about the exchange of messages are hidden, for clarity reasons, the basic idea remains the same: we use services without the requirement of creating a separate account for each service.

All the steps are self-explanatory but we would like to focus on the first and fifth message. Notice that the line of the first message is dashed, denoting that user interaction is required in order to assemble the message. In our case the user needs to type a password in order to validate their digital identity to the framework. This step can be seen as proof of possession, since only the user can unlock their digital identity using the password; and the password is the only information the user has to remember. From that point on minimal, and in most cases, no user interaction is required independent of the number of services accessed. The fifth message is a challenge created by the framework that only the middleware can access and provide a response. In this way the user is authenticated as being the true holder of the secondary identity presented in step 2, and the validation is achieved without user interaction. An additional reason for this step is to protect the end-user from impersonation and “man-in-the-middle-attacks”. Keep in mind that we assume a human acting behind the end-entity middleware, but every step can be performed by a software agent acting on users’ behalf.

The rest of this chapter provides details about the implementation and the communication between the different software modules of our infrastructure. The implementation description will cover the information available to every software module and the interfaces available to communicate and to combine this information. The communication and information exchange between parties will be discussed in a separate section. This section will provide evidence about the focus of our work in terms of security and the expandability provided by our low-level communications library.
1. An entity performs a single sign-on to the identity framework
2. Presents a secondary identity to create an account to the service
3. Try to validate the secondary identity to authenticate the entity
4. Challenge request to verify the end-entity
5. Challenge request forward
6. Challenge response from the end-entity
7. Challenge response forward
8. Notification about the legitimacy of end-entity
9. Notification about granting or denying end-entity request

Figure 3: Sequence of interactions to perform a single sign-on and authenticate secondary identities without using another set of credentials or end-entity interaction

3.2 Digital identity management framework

The name not only reveals the purpose of this framework, but also the distributed nature of our proposal for digital identity management. We have already given the definition of a Digital Identity Management (DIM) system, but in this section we will focus on the technical aspects and set the requirements that such a system must meet. Furthermore we will try to explain the reasons we think our DIM framework performs better than related work in the field and finally provide inside information of our framework components. As we have already discussed in the section where we surveyed today’s DIM systems, most of them employ centralized or federated solutions. Our goal is to produce a hybrid solution that merges the advantages of both approaches, while eliminating the disadvantages imposed by either approach. But before going through the details we will step back and discuss the necessity of such a framework.

The major reasons to deploy a DIM framework is to facilitate users and services in their every day needs for accessing personal or privileged information. From the user’s perspective, they expect to perform a login procedure once and to access seamlessly every service they wish. This means that no further credentials will be used or issued by the service; in contrast the service will query the DIM framework to verify the authenticity of the user. So the user benefits from the existence of a single sign-on framework, while the service
is relieved from the burden of integrating a custom made credentials system. But in order for the DIM framework to work, both users and services must have specific purpose software running on their computer systems. This software handles all communication with the framework, creating a single sign-on system that operates across different administration domains, but at the same time is treated as being a locally deployed identification module. However the most important challenges of such a framework are the resilience to attacks and the minimization of information exposure in the case of a successful security breach. The first challenge is met by carefully designing the interfaces available by the authorities of the framework. In this direction numerous tests and scenarios have to be addressed to further investigate the tolerance of the framework to different attack patterns and intrusion attempts. The second challenge is more demanding, as it requires breaking down information and making it unusable, unless several entities cooperate. The outcome must be the creation of a session statement that users can present, in order to be validated by service providers. To better understand the demands of a DIM framework we will provide an evolutionary description on the field of identity management systems and conclude with our proposal.

One could think of a centralized approach, where a single authority or pool of authorities is responsible for creating and managing digital identities. The main advantage of this system is that it operates under a single administration domain, so harder security measures can be taken to minimize the possibility of a successful attack. Services also benefit as they have to trust a single system for providing legitimate user information. As for the users, they are given a set of credentials in order to activate their account on the system. This way no sensitive information is handled by users, as they are considered the weak link when it comes to security. On the other hand this approach defines a single point of failure, meaning that if it is compromised, all the information that is stored will be exposed to the adversary. This information includes user credentials, secondary identities and the services that are being used and possibly private information acquired from the user, in order to create the digital identity. And all these will become available with a single security breach! Moreover this system can act as a bottleneck or have enormous operational cost, if we consider the widespread use of it. There are also privacy considerations for a single system vouching digital identities for large number of users.

Another approach would be to define several independent identity management authorities. This way the amount of information available to every authority will decrease and continue on decreasing as the number of authorities grows. The advantage here is that every authority has a smaller number of users to support and less information in the case of security breach. The second advantage is the ability of the user to select the authority they want to handle their digital identity information. This will drive the authorities to be competitive and apply stronger security policies, further benefiting the user. The drawback here is that services must have knowledge of all the available independent authorities, as
users credentials can originate from any of the available authorities. Another problem is that some authorities may misbehave, issuing false credentials and creating problems for the services. If we compare this approach with the one in the previous paragraph we have limited the privacy considerations and avoided having a single point of failure, but now the service has to know all existing identity providers and avoid using credentials from illegitimate authorities.

A quick solution to the validation problem would be to define a global catalogue, where all legitimate authorities are listed. This way services can check the authority’s validity, before granting access to users holding credentials from that authority. The right solution would be to define a tree hierarchy with every non-leaf node acting as a catalogue and the leaf nodes being the identity providers. This way the administration burden is broken down only to a small set of children nodes, while validation of authorities is trivial by traversing the tree to its root. With this approach services are required to know only the root node of the tree and are able to verify the validity of any authority by simply examining if they belong to the tree hierarchy (more information about the hierarchy created and how validation is performed are given in the next section). If we review the approach we discussed in the previous paragraph with the modification presented here, we see that we have eliminated the drawbacks caused by our shift towards having independent identity providers. Indeed, there is no need to overload the service with information about the available identity providers, since only the root catalogue needs to be well-known. As for malicious authorities, they are excluded by examining if they belong to the hierarchy defined by the root node.

Our approach goes a step further, by completely transferring the management of identities and relative information to the user. This way the identity provider is left with all the information needed to verify the authenticity of the user and perform the single sign-on. All the information about secondary identities and the mapping of identities to services is held and managed by the user. This way the user is the sole responsible entity for their security and how their personal information is being used (privacy concerns). The advantages of this approach are that even if a catalogue or an identity provider authority is attacked successfully, no information can be gained on how the digital identities were used. This information is located in the users computer system, thus we avoid defining the authority providers as points of failures. Performing a final comparison between the different approaches we presented, we can see clearly that not only the disadvantages of the previous approaches were eliminated but even more their advantages where strengthened. Our argument is that users are not the weak links when it comes to security, rather the centralized approaches are, with black box security policies. It is centralized or distributed approaches that handle information for a number of users that suffer from intruders, since the ratio of “gain to labour” is much greater when compared with solutions where every user preserves their personal identity information. The only way for a digital identity management
framework to be secure and at the same time trusted (even by its users) is to be publicly available and open source. The only disadvantage in our approach is that specialized software must be downloaded and run on the user’s computer system, but this is a small price to pay for security. More details on how information is handled by the user are presented in section 3.4, where the end-entity middleware is described.

Before referring to an example DIM hierarchy we review the components that make up the DIM framework and the names used to point to them. First of all we have the catalogues holding information about the authorities that belong to the hierarchy. The full name we have assigned to every catalogue is “authorities’ aggregation catalogues”. For convenience we will refer to them using the acronym AAC. The special AAC of our framework is the root node of the tree. We will call this catalogue Global AAC, to be able to easily refer to it. The last convention is about the identity providers, which are called identity issue authorities or AAI in short. Finally, the term authority will be used when a node in the tree hierarchy of the DIM framework can be either an AAC or an IIA. Let us now examine [Figure 4], where a sample hierarchy is shown. The figure shows a typical tree with the root node being the Global AAC, all intermediate nodes being AACs and leaf nodes being IIAs. The root node is the authority that is considered to be well known. This means that all other authorities have the details of the Global AAC embedded in their implementation. This information helps define a start point for authority trustworthiness, and subsequent authorities are trusted only if a path exists between the authority in question and the Global AAC. The rest of the AACs in the tree simply expand it and partition the administrative domain of every authority. The leaves of every hierarchy are IIAs, preventing the expansion of the hierarchy and performing the identity management operation.

3.2.1 Authorities aggregation catalogue

AACs fulfil the need to keep the hierarchy protected from malicious authorities and establish independent administration domains that can opaquely expand the hierarchy. The first is achieved by defining a list of child authorities for every AAC. This way when engaging in communication with an unknown party, the parent authority can be contacted to verify the party’s trustworthiness. This procedure can be repeated for the parent authority as well, until a trusted or a well-known AAC is reached. To establish independent administration domains every AAC is limited to know only the list of direct children authorities, ignoring the presence of a possible sub-tree defined by its children. This way every AAC defines a tree of descendant authorities, but the absence of knowledge about them limits the responsibility domain to the direct children. Instead the burden is moved to the selection and constant evaluation of this relative small list of children authorities. The selection procedure is essential, since the trust from parent to child authorities is implicit. This type of trust is
unconditional, as blind confidence is transferred from the parent to its children. Parent authorities must vouch for the behaviour of their children. A schematic example is shown in [Figure 4] where the responsibility domain of the AAC #1 consists of its children authorities, while the corresponding coverage domain includes every descendant authority.

![Figure 4: Example of the responsibility and coverage domain of an authority in the digital identity management hierarchy](image)

The next step is the definition of the information that identifies an authority, which we will call authority identity. The attributes we considered efficient are a distinct name, the host name where the instance is running and finally the public key to be used when contacting the authority. The distinct name is a unique character sequence for every authority in the framework. It is used to easily identify an authority inside the hierarchy and avoid having cumbersome serial numbers. The host name provides the network name of the computer system where the instance is running. This information is vital when authorities’ information is exchanged between software components, as the party informed about an unknown authority can contact it to verify its legitimacy. The final piece of information is the public key of the authority. This key will be used when initiating a communication channel, in order to secure the messages transmitted. The public key also acts as proof of identity, preventing impersonation. Indeed, only the holder of the public key can decipher our messages and provide the appropriate responses.

Until now we have referred to authorities in general, without making any assumption about the type of the authority. This is because both AACs and IIAs share the characteristics described above. Their differences are the interfaces offered and their location in the
hierarchy. We will focus on the interfaces of the AAC, postponing the survey of IIAs to the next section.

Every AAC offers three interfaces, the first two are public for everyone to connect and exploit the provided functionality while the third is private, and authorization is required. The authority validation is one of the public and most important interfaces. It provides functionality for checking if authorities belong to the DIM hierarchy or masquerade as being legitimate. The general idea is that every authority reports their parent authority, and this claim can be verified by asking the parent about its children. If the parent fails to acknowledge the child then the authority is illegitimate and further communication is blocked. A practical example includes the service mediator requesting to validate an IIA. The IIA reports its parent authority and that AAC is contacted to verify this claim. As a result a trust link is established between the IIA and the AAC we queried. But this may not be enough, since the legitimacy of the AAC may not be guaranteed. To solve this, during the validation procedure the AAC returns the authority identity of its parent. The validation can now be repeated for the AAC, until a trusted authority is reached. The only hypothesis needed to be made, for the validation process to work, is the establishment of a well-known AAC, which in our case is the Global AAC. This way the worst case scenario will be not to trust any of the intermediate authorities and reach the Global AAC, which is considered by default trusted.

The second public interface handles authorities' lookup. This is a specially designed interface to facilitate the creation of digital identities. As we will see in the next section the user is free to connect to any IIA and present some personal information to create a digital identity. The problem is how to retrieve the contact information of an IIA. We decided to implement a lookup interface, where the requestor will present the distinct name of an authority and receive full contact information for that authority. This way the end-entity middleware can connect to any AAC and retrieve detailed information for any of the authority's children. This interface was created to help us with the implementation requirements of our infrastructure and is not designed to be supported outside the context of demonstration.

The last interface available is private and only authorized users can connect. The functionality provided is mainly for the administration and maintenance of the responsibility domain. The actions that can be performed are the listing of available children authorities, the addition of a new authority in the list and finally the deletion of a child authority. Only the add action is a bit complicated, since the information created has to be transported to the computer system that will host the authority. At the same time the information contained must be self-explanatory as the authority must be aware of its location in the hierarchy.

Concluding the description of AACs we would like to point out the distribution of the administration domain and the minimization of the set of responsible authorities. This leads to multiple and independent domains that are easy to manage and achieve higher quality (e.g.
more trustworthy authorities). The second point is the authority validation protocol, that imposes strict constraints on how authorities are checked. This recursive algorithm is able to exclude illegitimate authorities preserving the enclosure defined by the hierarchy of legal authorities.

### 3.2.2 Identity issue authority

The IIA is considered a Trusted Third Party (TTP) working with the service provider and the users, in order to vouch for digital identities held by the latter. Its primary role is to supply digital identities to individuals and perform the single sign-on process. This process will activate the individuals’ digital identity and enable its usage to different services without the use of another set of credentials.

First of all a digital identity has to be issued for every individual. Since our system was built for demonstration purpose, no strategy was imposed on how identities will be issued. Instead a special public service was implemented that entities can use and create as many digital identities they wish. The main reason of our selection was the minimization of administration and maintenance burden, as compared with the case of applying a realistic policy. Another difficulty is the definition of a specific policy for licensing identities, since the strategy imposed will be defined by the context in which our infrastructure will be used (public sector, institute or cross-corporation domain). Finally, such a policy is beyond technology and as such, it is beyond the scope of this thesis.

To better understand the internal workings of this special purpose interface we must first examine the building blocks of a digital identity. From [Figure 5A] we see that a digital identity contains information about the holder of the identity (master identifier, public key and personal information), the authority that issued it and finally a digital signature. The public key and personal information is the information presented by the user to the IIA when creating a new digital identity, while the master identifier is assigned by the IIA and is used to uniquely identify the end-entity. The IIA information block contains details about the authority that issued the digital identity and how to contact it. In particular it consists of the authority's distinct name (host name where the instance is running and its public key). As we have mentioned in the previous section, this information is sufficient to identify, locate and securely communicate with any authority in the DIM framework. The digital signature block is created by the issuing IIA and is used as non-repudiation proof as well as for integrity check of the digital identity.

[Figure 5B] displays the information encapsulated in the personal information block. It consists of the private information and the password supplied by the individual for which the digital identity is created. Both the password and the private information are not placed inside the digital identity in the plain text format, as provided by the individual. Rather the private
information is encrypted using a secret key known only to the IIA. This way not even the holder of the digital identity has access to this information and no intelligible processing can be performed. In the case of the password we need to be able to make comparisons with a given password and the one stored inside the digital identity. For this reason the password is passed through a one-way secure hash function and the resulting output is stored. To perform the comparison, the given password is passed through the hash function and the output is tested against the stored one. Keep in mind that it is computational infeasible to retrieve the plain text password given its hashed counterpart.

Summarizing, the digital identity is the passport of the individual for the digital world. This digital document contains user identifiable information, but every piece of private and sensitive information is protected using cryptographic methods. As we will see in subsequent sections, the master identifier is the most important and widely used block, as this information is used to spawn secondary identities.

![Diagram of Digital Identity and Personal Information](image)

**Figure 5:** Schematic representation of the information inside a digital identity (A) and the nested information encapsulated in the personal information block (B)

To create a digital identity, an individual is expected to present some personal information. This information will be processed by the IIA and the resulting digital identity will be returned to the individual. Along with the digital identity, a secret key is exchanged between the IIA and the individual. We will call this, the identity key, as it is used to encrypt secondary identities. The encryption prevents different digital identities from being associated and at the same time the IIA can validate its authenticity. We will further examine the usage of the identity key in section 3.4, where the process for creating secondary identities is presented. Let us refocus on the digital identity and the personal information submitted. The individual needs to provide his public key, a hard-to-guess password and some private
information. In our demonstration we have defined the private information to include the individual’s full name, country and prefecture of residence. Once again this is not a hard constraint, as new attributes can be attached without affecting the rest of our infrastructure.

The questions that arise from processing private information is how this information is used to create the digital identity, the measures taken to protect the personal data inside the identity and how this information is stored locally by the IIA. For the first concern we must notice that private data are only used to create a unique and opaque number for every individual requesting an identity. This number is the master identifier we mentioned earlier. For the creation of the master identifier the private information is passed through a one-way secure hash function, resulting in a unique identifiable number that can be used by the individual without exposing any private information. We have already answered the second question, as the embedded private information is first encrypted using a secret key known only to the IIA. The last concern addresses the issue of private information stored locally by the issuing authority. The storage is required only in the case of identity abuse, where legal issues may arise. But once again the information is encrypted before sent to a repository, and only the IIA has access to its contents.

Along with the special purpose public interface for creating digital identities two more public interfaces exist. The first performs the single sign-on of the individual, allowing him to use services without needing any other form of credentials. The interface tries to bind the holder of the digital identity with the digital identity itself. To succeed, individuals are required to present their digital identity along with the password that was chosen when creating it. The digital identity represents something they have, while the password something they know. In fact the password is only known to the holder of the digital identity, which prevents other individuals from completing the validation procedure even if they get a hold of the digital identity. After completing the validation procedure a mapping scheme is exchanged between the IIA and the end-entity middleware, which in fact is a secret key. We call this key validation key as it is used to verify and protect individuals. To achieve this IIA encrypts information with the validation key that only the individual can decrypt and respond to it. We will focus on the details with an example later in this section.

The second public interface is used by the service provider (in particular by the service mediator) to authenticate an individual requesting access to privileged information. Before initiating a request to a service, individuals must connect to the IIA that issued their digital identities and execute the identity validation procedure. Afterwards individuals can be authenticated to services simply by providing a secondary identity and the IIA contact information. The service mediator will connect to the IIA and present the identity of the individual. After specific protocol details that are examined later, the IIA responds informing the service mediator only about the legitimacy of the secondary identity. The key aspect of this approach is the ability for an individual to be uniquely identified (proof of legitimacy) and
at the same time retaining his anonymity (preserve of privacy). In addition the user is protected from impersonation as he is the only one able to respond to messages encrypted with the validation key.

We return to [Figure 3] to provide supplementary information about the usage of the validation key and the messages exchanged when authenticating a secondary identity. In the first step of the figure the end-entity middleware connects to the validation interface (first interface) of the IIA and performs a single sign-on. During this procedure a validation key is exchanged and shared between both parties. This key is used to encrypt challenges created by the IIA in order to verify that the individual that presented the secondary identity is really the one that performed the sign-on process. So the service mediator connects to the authentication interface (second interface) of the IIA and forwards the secondary identity supplied by the individual (step 3). The IIA locates the validation key and creates a challenge request encrypted with that key. The service mediator knows nothing about the information contained in the challenge, and just forwards the message to the end-entity middleware (step 5). Keep in mind that only the end-entity middleware has access to the encrypted information and can provide the appropriate response. The response created by the middleware is passed to the service mediator, which forwards it to the IIA. The final step is for the IIA to examine the response and notify the service mediator about the success or failure of the individuals’ authentication process (steps 8).

Finally the IIA has a private interface where authorized users connect to perform administration and maintenance tasks. At present the main tasks are: listing of available digital identities, view details about a certain identity, delete and revoke an identity. Both delete and revoke are performed in the case of identity theft or if the user wishes to destroy his digital identity and all information associated with it. The listing is performed to facilitate locating a specific identity, and provides no identifiable information. Finally the action that displays the digital identity details is executed only in the case of identity abuse or arising legal issues. In a real deployment of our system the special public interface for creating digital identities must be embodied inside this private interface, to ensure the legitimacy of the users and the mapping of digital identities to real-world entities.

### 3.3 Service mediator

The service mediator is one of the two software components that are not part of the DIM framework. Their development was driven by our effort to provide a complete solution and investigate all aspects of a fully deployed identity management infrastructure. So the service mediator was designed to handle the validation of entity identities. Its purpose is to facilitate services regarding digital identity matters, providing an abstract layer to core service functionality. This way services are able to utilize the advantages of a DIM framework,
preserving their internal structure and logic intact. In fact, the service mediator acts like a local identification module, performing the necessary actions in order to validate the legitimacy of individuals. The functionality it provides includes the receipt of secondary identities from individuals and the validation of this information through the DIM framework. Finally, information about the outcome of the validation procedure is forwarded to the core module of the service, as specific actions must be taken depending on the context of the service.

The first step that will signal the success or failure of our DIM framework is its acceptance from the service providers. This means that the service mediator needs to be appealing, in order to be incorporated to as many services as possible. The strong incentive towards the wide spread acceptance of the service mediator is the warranties given by the DIM framework about the quality of the credentials supplied. Namely, all credential information presented by the end-entity middleware will be mapped to a digital identity and thus to an individual. This way services have knowledge of the individuals they serve and can apply a certain policy based on user behaviour. For example, in the case of service abuse the IIA along with the service mediator can prevent future service access to the individual. On the other hand, verification of individuals in today's services is very poor, as no mapping exists between the local account created and the individual behind it. Other incentives include the outsourcing of account management, the ease of service mediator incorporation and consumer satisfaction from the absence of a different set of credentials for each service.

To meet the above goals the service mediator must be able to communicate with both the DIM framework and end-entity middleware. The end-entity middleware will try to establish a connection with the service, by sending the necessary credential information. The connection request will be processed by the service mediator, and the credentials passed will be used to contact the DIM framework and validate the individual's legitimacy. The credentials supplied contain an identification number (secondary identity) and the IIA the issued the digital identity of the individual. The next step is to contact the IIA and verify the credentials validity, using the authentication interface we presented in section 3.2.2. The outcome of this procedure will determine if the user will be granted access to the service or communication will be stopped. The details of the information exchanged during the validation procedure are shown in Figure 3, and a thorough description of the every step was given in section 3.2.2. The only addition we want to make is the final message, where the end-entity middleware is notified about the outcome of its request. Actually the validation procedure performed by the service mediator is slightly more complicated, and falls down to the above description when the IIA in question is trusted. If the IIA that issued the individual's identity is not trusted, the service mediator starts the process of trying to find a path between a trusted authority and the target IIA. This process uses the hierarchical structure of the DIM framework in order to traverse the tree upwards and head for the root.
In every AAC encountered the authority validation interface is used to check the legitimacy of the authority that claimed to be its child. Further details are presented in section 3.2.1, wherein a complete description of the authority validation interface is given.

In practice, the service mediator defines a single public interface for the end-entity middleware, in order for an individual to gain access to the service. This interface exploits the functionality provided by the DIM framework, working with both IIAs and AACs, obtaining accurate and valuable information about the individual and its identity provider. The final step is to pass this information to the service, completing the service login procedure without the use of another set of credentials. As a side effect of the service login procedure performed, a secure channel is established between the service mediator and the end-entity middleware. This channel is also handed over to the service to be used for further communication between the two parties.

The statement that the service mediator acts as a local identification module is not far from being true. Hypothetically, the service mediator can be thought of as a verification function, where we input credential information and get notified about the information's legitimacy. From this point of view, the DIM framework will be the local database that holds the necessary records to verify that the credentials presented are valid. The above is evidence that the service mediator will not be hard to deploy, while changes to core service functionality will be minimal.

Closing the description of the service mediator we would like to point out the two different types of services we envisioned during our work. The first type consists of services where users create their own accounts, presenting personal information. This type includes most of the services available in the Internet; examples are online stores, portals, free e-mail providers, forums, IM tools and so on. The second type of services is those that perform validation based on a predefined account list. Besides requiring individuals to present valid credential, they must find those credentials in the members account list. This means that individuals have to present their digital identity along with some paper proof, in order for their digital credentials to be added in the members list. Such services include corporate or institutional services, where one has to be an employee or somehow affiliated in order to gain access.

### 3.4 End-entity middleware

The second software component that is not part of the DIM framework is the end-entity middleware. The main task of this component is to control and handle how digital identity information is published to services. To achieve this, the middleware has an internal storage mechanism, where the digital identity and secondary identities information is stored. The secondary identities are not just piled in a list, but organized according to the services in
which they are used. This way, when a connection to a specific service is required, the secondary identity can be retrieved and used. But the mapping scheme also allows multiple identities to be used in one service, and furthermore the same identity to be used across different services. The first permits the user to create multiple and independent profiles in a single service. These profiles cannot be correlated by the service provider, giving the user the flexibility of creating independent accounts at will. The second allows the integration of several services under a single account. This way information between these services can be exchanged, in order to help the user in his everyday tasks. This can be performed in the case of a car rental and airline company. For example, when booking a flight the information can be exchanged with the car rental company, which in turn can make an offer about renting a car for the same period. Although the middleware supports identity-service agglomerations, it is advised to perform them only when trusting the services participating in the agglomeration and their policies state exactly the type of information exchanged.

Besides the internal store, functionality is provided to activate this information and exchange it along the rightful services. The activation of the digital identity is performed to the authority that issued it, namely the IIA information field inside the digital identity, [Figure 5A]. Therefore, the end-entity connects to the identity validation interface of the IIA and provides the password chosen during the creation of the identity. The result is the exchange of session information that activates all information in the internal store. The next move is to use the service login interface to login to service providers and utilize the functionality provided.

Except from using provided functionality, the end-entity middleware establishes an abstract application development environment, to be used by the client application at the user's computer system. The client applications are the client-side tools used to connect to services and exploit their functionality. The only constrain is that no direct connections are allowed between the client application and the service provider, if the application wishes to utilize the middleware functionality. To facilitate this, special purpose software needs to be installed in every application, the client mediator. The client mediator is used to divert all communication of the application to the service provider through the end-entity middleware. This way all information will pass through the middleware ensuring the usage of the digital identity and the DIM framework. In [Figure 6] we see a diagram displaying how software components are situated on computer systems and how messages flow between the various components. In our example we want to establish the virtual channel defined by the dashed line using the middleware. For this reason the client mediator diverts all communication traffic from the application to the end-entity middleware. If this is the first connection to the service then a connection with the service mediator must be established and afterwards this will be passed to the service core. Further communication will be directly passed from the middleware to the service core. As for the responses provided by the service, they must
follow the opposite direction. The group consisting of the client application, the client mediator and the service core are specific to the context of the application. Multiple groups like this can be serviced in parallel by the end-entity middleware (multithreading implementation).

As one can picture we cannot give implementation details on the three software components we just mentioned. Both client application and service core are context and provider specific. As for the client mediator, it depends on how the client application was implemented. We can only provide best practice guidelines for the client mediator. In fact the only guidelines are that the client mediator has to redirect all communication to the middleware and upon connection initialisation the service details must be given to the end-entity middleware.

![Diagram showing communication between components](image)

**Figure 6:** An example showing how the client mediator communicates with the end-entity middleware in order to establish a virtual communication channel

A full scale example on how to establish a virtual channel between the client application and the service core is shown in [Figure 7]. In the figure every transaction is given an increasing number to easily follow the sequence of events. The first event happens upon client application initialisation along with the client mediator, where a connect request is send. This request contains all details on how to contact the service (hostname, port, public key). In the case that the end-entity middleware is activated for the first time the identity validation procedure is executed. The next event is to execute the service login procedure, presenting a secondary identity. After the authentication procedure a response is created stating the acceptance of the credentials presented. The final event is to notify the client mediator about the success of the whole procedure. From this point on normal application traffic can be sent to the service core through the virtual channel that was established. If one event fails the negative feedback is given to all intermediate components in order to display proper error messages.
We will now focus on the only thing we have not covered so far, how secondary identities are created and validated. In section 3.2.2 we mentioned both the identity key and the master identifier. The first is exchanged between the IIA and the end-entity middleware during the digital identity creation, while the second is derived from the personal information presented by the user. The procedure to create a secondary identity starts with the collocation of the master identifier with a sequence of random bits. This creates the secondary identifier, which must be unique if we are creating a new secondary identity. Next the secondary identifier is passed through a cipher function using the identity key as the encryption key. The outcome is the secondary identity, which can be used to services without privacy concerns as they are opaque to service providers. The procedure is shown in [Figure 8], where the white key represents the identity key.

![Diagram of identity creation process]

**Figure 7:** Steps for establishing a virtual channel between the service client and core service functionality. The sequence of the steps is given by the numbers preceding the description

A secondary identity created by the end-entity middleware will eventually reach the IIA. The task of the IIA is to check the validity of this secondary identity, namely if it matches to the master identifier of the user. The reason we require this matching is because the IIA does not collect information about the secondary identities of the user. So, if a user tries to repudiate that he is the holder of a certain account, the secondary identity can be deciphered, extracting the user’s master identifier. The procedure followed when validating...
an identity is shown in [Figure 9]. As we can see first the secondary identity is deciphered, revealing the secondary identifier. Next we cut off the 128 most significant bits of the secondary identifier and compare them with the master identifier stored by the IIA. The result of the comparison will determine the outcome of the validation procedure.

![Diagram of validation steps]

**Figure 9:** Representation of the steps followed when validating a secondary identity

So far we have presented details about the internal working of the end-entity middleware and specified out-ports and in-ports for establishing connectivity with other components. Now we will step backwards and explain the reason behind all these additions. If we review the design decisions of our infrastructure we will see the statement that users must be given the responsibility of managing the dissemination of secondary identities to services. To achieve this, the end-entity middleware must be running in every user's computer system. But things get a bit difficult when putting all together, since the client application is constrained to talk to the service core only through the end-entity middleware. The solution is given by the client mediator, which diverts all communication of the client application to the end-entity middleware and vice versa. As we can see the burden is again shifted to the service providers, they have to produce enabled service core and client applications in order to fully utilize digital identities held by individuals. On the other hand users are expected to have a copy of the end-entity middleware operating locally. This acceptance is the second metric that will signal the acceptance or rejection of our DIM framework. The only burden implied from the use of the end-entity middleware is a set of questions about identity management actions, which is more then acceptable compared with the benefits implied.
3.5 Communication protocols

One of the most important properties of our infrastructure is the establishment of secure channels between transacting parties. Each channel creation involves a series of messages exchanged to achieve confidentiality, message integrity and provide authentication. The endmost goal is to establish a channel between the parties, giving sufficient guarantees to each party about the identity of the other. The channel establishment procedure starts by exchanging messages using asymmetric cryptography, in order to share a secret key among the two parties. Afterwards, both parties switch to symmetric cryptography, as it is equally strong but demands less computational power. Using cryptography (either symmetric or asymmetric) for every message transmitted satisfies the confidentiality goal. Cryptography also satisfies message integrity, as the contents of encrypted messages cannot be accessed or altered by third parties. Finally, authentication is achieved during the use of asymmetric cryptography. Indeed, using the public key of the target entity ensures that only it is able to decipher the message and provide the proper response. Before we go into details, the next table repeats the notation and abbreviations we use for cryptography and was first presented in section 2.3.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_{A,B}$</td>
<td>Secret key shared by A and B</td>
</tr>
<tr>
<td>$K_A^+$</td>
<td>Public key of A</td>
</tr>
<tr>
<td>$K_A^-$</td>
<td>Private key of A</td>
</tr>
<tr>
<td>$R$</td>
<td>Random number</td>
</tr>
</tbody>
</table>

Table 2: Notations used for cryptography

A secure communication channel is created between two software instances that need to exchange information. The requester connects to one of the available interfaces of the other party and initiates the channel establishment procedure. This establishment procedure is the same regardless of the different functionality provided by every interface. In fact, the first messages remain the same, and only after the establishment of symmetric cryptography every interface is differentiated from each other. There are two connection types that define the two different channel establishment procedures: Anonymous and Eponymous. Anonymous connections require only the party that accepted the connections to be authenticated, as shown in [Figure 10A]. In this example the Party #2 uses the public key of Party #1 in order to start the establishment procedure and authenticate that he is talking with the indented recipient. After these messages the secure channel is established and a secret key is shared, this way every specific interface only focuses on the custom messages needed to achieve the desired functionality. Eponymous connections are different as they
require both parties to be authenticated as being legitimate. To achieve this, the public key of both parties is used in the establishment procedure as shown in [Figure 10B]. In both procedures the Request symbol is an abstract message containing the details about the reason behind this connection establishment.

![Diagram](image)

**Figure 10:** Anonymous (A) and Eponymous (B) channel establishment procedure

In the previous sections we have presented the software components that make up the identity management infrastructure. We have also presented the interfaces available in every component. In the next table we summarize this information, including the connection type used by every interface. Moreover we present the software component where the interface is available (receiver) and the component that connects to the interface in order to use the provided functionality (requester). In the next sections we will present details about our decision on the type of connection of every interface.

<table>
<thead>
<tr>
<th>Interface name</th>
<th>Connection type</th>
<th>Receiver</th>
<th>Requester</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authority lookup</td>
<td>Anonymous</td>
<td>AAC</td>
<td>End-entity mediator</td>
</tr>
<tr>
<td>Authority validation</td>
<td>Anonymous</td>
<td>AAC</td>
<td>Service mediator</td>
</tr>
<tr>
<td>Identity creation</td>
<td>Anonymous</td>
<td>IIA</td>
<td>End-entity mediator</td>
</tr>
<tr>
<td>Identity validation</td>
<td>Eponymous</td>
<td>IIA</td>
<td>End-entity mediator</td>
</tr>
<tr>
<td>Identity authentication</td>
<td>Anonymous</td>
<td>IIA</td>
<td>Service mediator</td>
</tr>
<tr>
<td>Service login</td>
<td>Anonymous</td>
<td>Service mediator</td>
<td>End-entity mediator</td>
</tr>
</tbody>
</table>

**Table 3:** The available interfaces and a brief list of properties

Until now we intentionally avoided to mention how every requester or receiver gets a hold of the public key that belongs to the intended target entity. Now we can use [Table 3] in order to refer to every interface and explain how public keys are exchanged. The first interface is the authority lookup, and is used by an end-entity middleware in order to find an IIA and create a digital identity. The procedure starts at the Global AAC which is asked to return one of its children. The next step is to ask the authority returned by the Global AAC to return a child of its own, and this procedure is repeated until an IIA is reached. If we exclude
the first step we can see that we have all the available information needed in order to execute the lookup procedure in every intermediate authority. So the end-entity middleware needs to know a priori the identity information of the Global AAC.

Afterwards, the end-entity middleware will connect to the IIA and execute the identity creation procedure. Because the identity creation is executed right after the authority lookup procedure the end-entity middleware knows the public key of the IIA, so no additional information is required. The identity validation and identity authentication procedures also do not require a priori knowledge on public keys. The end-entity middleware locates the public key of the IIA from the digital identity, and the IIA from the record containing the individual’s details. This record was created when the digital identity was issued. For the identity authentication procedure the service mediator needs to know the public key of the IIA. This information is passed by the user during the execution of the service login procedure.

The service login procedure is initiated by the end-entity middleware, so the public key of the service needs to be known. This information will be presented by the client mediator when initiating a connection with the service. Finally, we have the authority validation procedure with tries to validate a given authority. This procedure is somehow the opposite of the authority lookup we just reviewed. The service mediator traverses the tree upwards in order to examine if the authority in question is part of the DIM framework. Again, all information on intermediate authorities is known and only the information of the Global AAC must be known a priori.

Concluding, we see that the Global AAC is the only well-known authority needed by our infrastructure in order for every procedure to work properly. The identity of this authority must be included by default in the implementation of the end-entity middleware and the service mediator components.

### 3.5.1 Authority lookup

The authority lookup is a straightforward procedure and provides an authority’s identity, given its distinct name. This interface is available to every AAC and is used for retrieving the identity of the catalogue’s child authorities. This way, a software instance can connect to AAC #1 of [Figure 4] and retrieve the identity of the authority named AAC #2. Only the end-entity middleware uses this interface, in order to locate an IIA and create a digital identity for every individual. As we can see the interface defines no constrains on who can use it, this means that the connection type is Anonymous and only the AAC has to be authenticated of being legitimate. The exact messages transmitted are shown in [Figure 11], where the first three messages are dashed in order to denote common functionality among Anonymous connection types. The only differences are the parameter passed on the third message and the fourth message. The Lookup parameter is simply the distinct name of the
authority identity, while the AuthID is an XML document containing the authority’s identity information. In our example, Party #1 is considered to be any AAC in the DIM hierarchy while the Party #2 is any end-entity middleware that is bootstrapped for the first time and a digital identity needs to be created for the individual. We have mentioned in section 3.2.1 that the lookup procedure is a special designed protocol, as in our demonstration environment we wanted to avoid our manual interference in the process of identity creation. This interface will be disabled in a real-world deployment, where a policy will be applied on how digital identities are created.

![Diagram of message exchange](image)

*Figure 11: Messages exchanged in order to receive an authority identity*

### 3.5.2 Identity creation

This protocol involves an IIA and the end-entity middleware, where the latter wants to create a digital identity for the individual it represents. The end-entity middleware is responsible for collecting personal information from the user and presenting it to an IIA. Using this information the IIA will issue a digital identity for the individual. Along with the digital identity a secret key is returned, which will be used to encrypt secondary identities to ensure user privacy. It is clear that digital identities created this way are not very dependable since multiple digital identities can be created and even false information can be given by the individual. But, as we have mentioned this is a custom interface created to facilitate our development effort and will be replaced by a policy on how digital identities are created. In [Figure 12] we see the information exchanged between the two parties, resulting in the creation of a digital identity. From the messages we see that the identity creation procedure is an Anonymous connection type, requiring the addition of a single message. The personal information parameter in the third message contains sample personal information, and can
be extended to meet specific needs. The fourth message contains the digital identity of the individual, based on the personal information provided and the identity key.

**Figure 12:** Messages exchanged in order to create a digital identity

### 3.5.3 Identity validation

Digital identities where created to be mobile, meaning that they are not bound to a single workstation or Internet service provider. This poses some serious threats, as an adversary could get a hold of the digital identity and use it as its own. To prevent this, the user must certify that he is the true holder of a digital identity. For this reason the digital identities are disabled by default and a certain procedure must be followed to activate them, using something that the holder is or knows. To utilize something that the user is requires the deployment of specific hardware, like biometric readers or implant readers, while the second approach needs the user to memorize a password and present it before starting the validation procedure. In our demonstration environment we decided to follow the second approach, as it is easier to implement and examine its properties. At the same time it provides out of the box deployment for potential applications, as it does not depend on the existence of custom hardware. Furthermore, we can switch from one approach to the other, or use both, without affecting the rest of the infrastructure.

The validation procedure is available to every IIA and is used by the end-entity middleware to validate the digital identity of the individual holding it. The connection established between the two parties is Eponymous, meaning that only individuals holding identities from a certain IIA can connect to the validation interface it provides. This way both parties are certain of corresponding with the indented recipient, while preventing unknown entities from completing the establishment process. The individual will be required to perform this procedure once, and afterwards he will be validated to subsequent service logins, without
the use of another set of credentials. In practice, the individual performs a single sign-on to that authority and all compatible services will connect to that IIA in order to verify the individual. This will allow him to access the functionality provided by the service, without requiring custom credentials. The exact description of the procedure is given in section 3.2.2.

The next figure provides information about the messages exchanged between the end-entity middleware (Party #2) and the IIA (Party #1) in order to validate the digital identity of the individual and perform the single sign-on. The changes applied by this protocol are the parameter of the first and third message, as well as the entire fourth message. The first message parameter, masterID, is the master identifier of the individual and is used to locate the public key of the individual in order to provide the proper response. The third message parameter is the password given by the user, which must match the one presented during the creation of the identity. Finally, the fourth message is a certificate returned by the IIA containing the session key that will be used to authenticate the user, preventing the use of another set of credentials. Again the dashed lines denote common functionality among Eponymous connections, while solid lines represent extra messages needed by the specific implementation.

![Diagram](image)

**Figure 13:** Messages exchanged in order for an individual to validate his digital identity

### 3.5.4 Service login

Services that have employed the service mediator will be considered enabled, as they are aware of the DIM framework and can communicate with both the end-entity middleware and the DIM framework in order to authenticate users. The end-entity middleware will connect to the only interface available by the service mediator, in order for individuals to perform the service login procedure and gain access to service functionality. The service mediator will complete the validation process by executing the identity authentication
procedure contacting the IIA that issued the digital identity for the individual. This way the service mediator is able to validate the individual without using another set of credentials. Since the validation process is shifted, the connection type of the service login procedure can be Anonymous, preserving the user’s privacy. In deed, if the user had a single public key using it to connect to Eponymous connections types, the individual would be identifiable by this unique key, even if he used different secondary identities to those connections. One approach is the one we followed, having an Anonymous connection type and shifting the validation to the identity authentication procedure. The second solution would be to utilize a different public key for every secondary identity, but this approach is far more complicated and time consuming.

The figure below presents the messages exchanged between the end-entity middleware (Party #2) and the service mediator (Party #1) in order to complete the service login procedure. Once again the first three messages are part of the common functionality shared among Anonymous connections. The third message is send by the end-entity middleware and provides to the service mediator the required credentials, in order to contact the IIA and authenticate the individual. The credentials contain a secondary identity and the authority identity of the IIA that issued the digital identity, held by the individual. The fourth message is a challenge, to which the individual must respond in order to successfully complete the service login procedure. As we can see, the challenge number and the service information are not accessible by the service mediator as they are encrypted using the session key known only to the individual and the IIA. We show in the previous section that the key is exchanged during the execution of the identity validation procedure. The service information contains the host name of the service mediator as seen by the IIA, and as we will examine in forthcoming section it prevents man-in-the-middle attacks. The fifth message provides the response to the challenge, which again cannot be accessed by the service mediator. At last, depending on the outcome of the identity validation procedure access to the service functionality is allowed or denied. Information on the outcome is returned in the sixth message, inside a service certificate document.
3.5.5 Identity authentication

This procedure is closely bound with the service login process. In fact, if we examine [Figure 7] we will see that the service mediator contacts the IIA, that is responsible for the individual’s digital identity, in order to decide if the user is legitimate or not. It is an assistant interface provided by every IIA, where the service mediator can connect to authenticate individuals. Since there are no restriction on who can use the interface the connection type followed is Anonymous, where only the IIA needs to be validated of being legitimate. The exact messages transmitted are shown in [Figure 15] and as we can see they are very similar to the messages exchanged in [Figure 14]. Actually, both connection types are Anonymous, so the first three messages are the same. The fourth and fifth messages cannot be handled by the service mediator and are just passed from the IIA to the end-entity middleware and back again. The exact details are shown in [Figure 3], where messages four and seven represent the fourth and fifth messages in [Figure 15], while the messages five and six represent the fourth and fifth messages in [Figure 14]. The only differences are the parameters passed on the third and sixth message. The issuer credentials, in the third message, contain information about the secondary identity that the individual has selected to present and the host name where the end-entity middleware is running. Using this information the IIA will locate the necessary details, in order to create the challenge messages and verify that the secondary identity is derived from the master identifier. The certificate returned in the sixth message contains information about the result of the authentication process. In addition, it contains the identity information of the parent AAC to which the IIA is a child. This way the service mediator can perform the authority validation procedure, in case that the IIA in question is not trusted.
3.5.6 Authority validation

The most common task performed by the AAC is the validation of authorities, which includes tests to verify that the authority exists, is part of the hierarchy and knows the party claiming to be its child. To better understand every step of the validation procedure we will discuss the example presented in [Figure 16]. As we can see the example is based on the hierarchy of [Figure 4], where the valid path from the Global AAC to the IIA #1 is isolated. Next to it we added two authorities (AAC #7 and IIA #3) that are not enclosed in the hierarchy of our DIM framework. In the scenario we will consider Alice and Bob, two users that hold digital identities from IIA #1 and IIA #3, respectively. Both Alice and Bob want to gain access to services, but in our case Bob is not legitimate, his digital identity was not issued from an IIA that belongs in the hierarchy. The only assumption taken concerns the authority information of the Global AAC, which is presumed to be known to the service.

In our first case study we assume that Alice wants to use a service, so it performs the service login procedure. In its turn the service mediator will contact the IIA #1 and execute the identity authentication procedure, in order to verify the individual’s legitimacy. Along with the verification, the authority will return the identity of its parent. This way, if the service mediator has no knowledge about the IIA #1 the parent authority will be contacted to ensure that the IIA #1 is listed among its children. The next step is to contact AAC #4 and verify the legitimacy of the IIA #1, using the authority validation procedure. The successful outcome of the authority validation establishes a link between the IIA #1 and the AAC #4. The response will be positive, establishing a relationship between the two authorities and signalling the need to verify the legitimacy of AAC #4. During the execution of the validation of the IIA #1,
the AAC #4 returns its parent information to further examine the legitimacy claimed. The validation procedure will be repeated until a known or the Global AAC is reached. In our example the next step of the algorithm will reach the Global AAC of the hierarchy, which is considered to be well-known, establishing a relationship link between the AAC #4 and the Global AAC. This way the validation procedure ends successfully, since a link has been found between a known authority and the authority in question (direct or indirect), while all claims for child authorities have been confirmed. As for Bob the procedure will fail when the Global AAC is asked if a child named AAC #7 is among its children. The response will be negative and the chain broken, reporting the error and stopping communication with every transacting party. In the case of Alice, all nodes that where contacted will be added to the known authorities list, in order to prevent querying the entire chain again and again, as no changes will be made and every link will be re-established. But periodically the list of known authorities will be emptied, in order to re-examine the hierarchy and embody any changes.

The description we gave above presents details on how to use the authority validation protocol, in order to verify that an authority is enclosed in the hierarchy of the DIM framework. This way the service mediator can trust an unknown IIA and grant access to individuals presenting credentials that originated from that authority. In practice, every step along the path involves the execution of a simple validation protocol, which verifies a claim made by an authority about being among the children of another authority. So, the service mediator will contact an AAC and present the authority identity that claimed to belong among its children. The AAC will check the information given to the list of child authorities, and return a status message along with its own parent. This way the claim is verified, and the procedure will continue depending on the knowledge the service mediator has about the AAC.

Figure 16: Two authority chains. The left chain is connected from root to leaf, while the right one is not included in the hierarchy of the DIM framework.
in question. If the AAC is known then the procedure is terminated, otherwise the AAC has to be verified.

The authority validation interface is available to every AAC and allows everyone to connect in order to verify if an authority is part of the DIM framework. In [Figure 17] we see the exact messages exchanges and the information they contain. As we can see the connection type is Anonymous, validating only the identity of Party #1 which in our case is the AAC. The dashed lines represent the common functionality shared among Anonymous connection types, while the solid one specific implementation. The specific characteristic of the procedure are the parameter passed in the third message and the entire fourth message. The parameter is the authority that claimed to be a child of the AAC we are connected and is being presented in order to verify its legitimacy. The fourth message returns the outcome, which is a report about the claim and the parent identity of the AAC. The parent identity will be used if the report is affirmative about the claim and the current AAC is not known to the service mediator.

![Figure 17: Messages exchanged in order to verify the legitimacy of an authority](image)

### 3.6 Discussion

We have already mentioned that our approach tries to provide a global single sign-on infrastructure, where users are the only holders of information relative to secondary identities and how they are used in services. In fact the DIM framework and the service mediator have limited information about the individual. The DIM framework and especially the identity providers hold the necessary information in order to validate users and verify their secondary identities. They have no knowledge (or discard it) about secondary identities or the services that are being used at. Furthermore the knowledge of every service provider is limited to secondary identities that the user has decided to present. This information dissemination
prevents adversaries from attacking a single point and gaining information about more than one users. We will focus on attacks to our system in section 5.1, where we will see that the guarantees provided by our infrastructure overcome the ones given by today's single sign-on systems. The next advantage of our system is the tree of AACS above the available identity providers. This tree allows a service mediator to verify the legitimacy of an IIA simply by checking if it belongs in the DIM hierarchy defined by the tree. The goal is to prevent services from having to verify every IIA independently reducing the a priori knowledge to the Global AAC.

We have overviewed several identity management systems and now we compare them with our identity management infrastructure (IMI). To avoid repeating information about similar system we have aggregated the Shibboleth, Liberty Alliance and WS-* projects under the federated identity management approach (Federate IM). The only centralized identity management approach is Passport .NET which we will refer to as centralized identity management approach (Centralized IM). As we can see in [Table 4] we use six different traits to compare the different solutions. The first field can assume two values: Single or Multiple. Single means that users will have a unique identifier presenting them to different services, and multiple that users can use different identifiers to different services, preserving their privacy. The second field also assumes the Single and Multiple values, but this time the trait examines if multiple identifiers can be used in a single service. The third field refers to where the identity information is stored. The information can be secondary identities and mapping of the identities to services. As we can see we have the centralized approach where everything is kept in a single server or pool of server. The distributed approach break down the information in sets and every identity provider stores a single set of identity information. The final approach is where every individual keeps his identity information solely to his computer system. The “user aware” field refers to whether or not custom software must be running to the individual's computer system. The next trait examines how the service providers know or find out about the legitimacy of the identity provider. By point-to-point we mean that every service must have a list of known identity providers, and only digital identities by those providers will be considered valid. The single point implies that all credentials presented to services will be originated from a single identity provider. The hierarchy defines a tree of authorities, where every authority vouches the authenticity of its children. This way only the root needs to be known, in order to verify the legitimacy of every other authority enclosed in the hierarchy. The final trait defines if the user performs a single sign-on or the same credentials are used repeatedly in every service in order to gain access.

Our comparison did not focus on general aspects concerning identity management but on the different traits of every approach. In fact many traits where left out, since all three systems or two of them share common characteristics.
<table>
<thead>
<tr>
<th>Traits</th>
<th>Federated IM</th>
<th>Centralized IM</th>
<th>IMI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identity to services</td>
<td>Multiple</td>
<td>Single</td>
<td>Multiple</td>
</tr>
<tr>
<td>Identities to service</td>
<td>Single</td>
<td>Single</td>
<td>Multiple</td>
</tr>
<tr>
<td>Storage of ID info</td>
<td>Distributed</td>
<td>Centralized</td>
<td>Individual</td>
</tr>
<tr>
<td>User aware</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Verify ID provider</td>
<td>Point-to-point</td>
<td>Single point</td>
<td>Hierarchy</td>
</tr>
<tr>
<td>Sign-on</td>
<td>Single</td>
<td>Unique credentials</td>
<td>Single</td>
</tr>
</tbody>
</table>

**Table 4:** Comparison of the approaches in identity management systems
4 Implementation

In the context of the thesis a series of software libraries were developed, in order to provide a demonstration platform for our global scale single sign-on system. The implementation process started from creating a set of custom libraries that provide common or abstract functionality. These libraries where designed with a broader scope then the one our thesis wanted to achieve. Next the software components where built, using the custom libraries and fulfilling the specific functionality needs. The sections 4.1 and 4.2 will cover the details behind every library, while the final section will provide information on design decisions taken during the development course.

For the following sections the term package or library will be used to indicate a collection of software that is semantically connected, and the term software element to an individual class file performing certain computations. The name of every element is given with capital first letter, to easily distinguish them in the text.

4.1 Custom packages

During the implementation process a number of custom libraries were developed. These libraries were built to provide a standard interface for common functionality and to formalize the development effort. In the following sections we describe the aim of every library and the software elements it consists of.

Before we go into details of the various protocols, [Table 5] summarizes the notation and abbreviations we use.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_{A,B}$</td>
<td>Secret key shared by A and B</td>
</tr>
<tr>
<td>$K_A^+$</td>
<td>Public key of A</td>
</tr>
<tr>
<td>$K_A^-$</td>
<td>Private key of A</td>
</tr>
<tr>
<td>$R$</td>
<td>Random number</td>
</tr>
</tbody>
</table>

Table 5: Notations used in this chapter

4.1.1 Software package "imi.util"

This library aggregates general purpose utility software. The software elements of the library are widely used throughout the thesis, providing a programmable interface for common functionality. The most important software element of the library is named Base64 and provides methods for encoding and decoding 64-byte encoded characters. The first
method performs the encoding, taking as input binary data and outputting an array of printable characters. The other method performs the opposite procedure, retrieving the binary data from 64-byte encoded characters.

The Debug software element is a wrapper for debug messages. The messages do not relate to the thesis, but are useful for the development and debugging process. The debugger is controlled by allowing or denying debug messages to be printed, and by setting the detail of the information to be printed. So every debug message is not directly forwarded to the console output; but instead is handed over to this wrapper with a number indicating the level of detail of the current message. Then, depending on the internal state of the debugger the message will be ignored or printed to the console output.

The ErrorMessage software element wraps an error code and a description. Its purpose it to provide a programmable interface for handling error messages, especially between two communicating parties. When a computational error occurs, only one of the parties knows it, which means that the other party must be notified. To assist this notification an instance of ErrorMessage is created and transported to the other party. Upon receipt the receiver can reestablish the instance of the ErrorMessage gaining access to the enclosed information. To achieve the transportation the internal details of the ErrorMessage are extracted to an XML document. After the transport the XML document is used to reconstruct the ErrorMessage. The transportation resembles object transfer with the use of XML, where the instance of an object is transported over a channel and reestablished to the other side. The difference with XML is that on the receiver side a different programming language can be used, providing interoperability and programming language independence.

The Globals provide methods that have global significance to the thesis. Besides providing programmable interface for common tasks most of the methods are used for retrieving information that can be altered during the course of the thesis. [Table 6] shows the list of available methods. As we can see only the generateKeyPair, generateSecretKey and getPublicKey provide common functionality, while all other methods return information that can be used to change the characteristics of the infrastructure. For example if the port number used by the public interface of the AAC software component was to be changed, a single modification to the number returned by the getCatalogPort method will be sufficient. Not to mention a change in the encryption algorithms or the size of the keys used.

A common practice is the initialization of a software instance from files containing parameter-value pairs. The InitParams is the software element responsible for reading the contents of such files and providing a programmable interface for accessing the value of any parameter.
4.1.2 Software package "imi.util.xml"

As above this is a utility library, but contains software only for XML document manipulation. The software elements in the library are the Copier and the Validator. The Copier is used for extracting parts from an XML document, given the end tag and a node to start from. The Validator is the mostly used software element in all libraries, since every piece of information is transported or stored inside XML documents. The purpose of the Validator is to check the conformance of an XML towards a target XSD document. To demonstrate the usage of these two elements we will utilize the XML document in [Figure 18]. Let us assume that we want to extract the entire head tag (including any sub-tags). This functionality is provided by the Copier element, since it is not supported by the standard library API. What we need to do is set the end tag to “head” and provide a node pointing to “<head>” tag. This will return the entire sub-tree of tags under the head tag as shown in [Figure 19].

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>generateKeyPair</td>
<td>Returns a pair of public and private keys, hiding the provider used for creating them and the key size</td>
</tr>
<tr>
<td>generateSecretKey</td>
<td>Returns a secret key, again hiding the provider used for creating it and the key size</td>
</tr>
<tr>
<td>getCatalogPort</td>
<td>Returns the port used by the Catalog instance to receive incoming requests</td>
</tr>
<tr>
<td>getIssuerPort</td>
<td>Returns the port used by the Issuer instance to receive incoming requests</td>
</tr>
<tr>
<td>getGlobalHostname</td>
<td>Returns the hostname where the Global Catalog instance is running</td>
</tr>
<tr>
<td>getGlobalPublic</td>
<td>Returns the public key of the Global Catalog, since it considered to be well-known</td>
</tr>
<tr>
<td>getPublicKey</td>
<td>The method gets an encoded public key and returns an object representing the public key</td>
</tr>
<tr>
<td>getRenewSpan</td>
<td>Since there is no use of certificate revocation lists (CRL) a renew span must be set in order to check the current validity of the trusted authorities</td>
</tr>
<tr>
<td>getAsymmetricalAlgorithm</td>
<td>Returns the algorithm to use when encrypting or decrypting asymmetrical cipher data</td>
</tr>
<tr>
<td>getSymmetricalAlgorithm</td>
<td>Returns the algorithm to use when encrypting or decrypting symmetrical cipher data</td>
</tr>
</tbody>
</table>

Table 6: A description of the methods provided by Globals software element
Figure 18: Example XML document used for data transfer

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<message xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="_protocol/DataTrans.xsd">
   <head>
      <protocol version="1.0">DataTrans</protocol>
      <encoding>Base64</encoding>
      <encryption>RSA/ECB/OAEPPadding</encryption>
   </head>
   <body>MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAnLiafv56u+</body>
</message>
```

Figure 19: The head portion of the XML document in Figure 18

As we have mentioned the Validator will be used to check the conformance of the XML document. The first step is to check if the XML document is well-formed, in other words if the structure follows the XML syntax rules. After that it will check if the XML document is valid, namely if the structure conforms to an XSD document. The schema to be used during validation is stated inside the XML document, which in our example is “DataTrans.xsd”. The XSD document in question is shown in [Figure 20].
4.1.3 Software package "imi.util.crypto"

The only software element in this library is named DigitalSignature. Its purpose is to create and validate digital signatures. The creation process uses a message of arbitrary length and an encryption key to produce a signature of fixed length, depending on key size. The signature is tightly coupled with the message and is used to verify that the message has not been forged. This verification is performed by the validation process that uses the message, the digital signature and a decryption key to check the authenticity of the message. In addition to verification the digital signature provides non-repudiation proof, since it could only be created by the holder of this key pair that created the message. [Figure 21] shows a representation of the procedures we mentioned, with (A) being the creation and (B) the validation process. So before transmitting an important message the sender will use its
private key to create a digital signature for the message [Figure 21A]. The recipient will receive the message and the signature. The sender’s public key is available to the public domain, so the recipient can easily retrieve it (e.g. directory of public keys). Having all the information available the recipient can execute the validation process to verify the authenticity of the message [Figure 21B]. An example of a digital signature produced by the creation process is shown in [Figure 22], where besides the signature itself there are some metadata describing the exact algorithm used for hashing and cipher functions. Moreover the signature data are binary and are encoded using base-64 scheme to remain printable.

**Figure 21:** The steps for creating (A) and validating (B) digital signatures
4.1.4 Software package "imi.trust.util"

In contrast to previous libraries, the software elements of this library are more focused on our thesis. In details they provide a programmable interface for representing authorities and creating lists for aggregation and management of authorities. An authority is defined by three characteristics: distinct name, host name and public key. The distinct name is the unique identifier of the authority, mainly used to locate it. The host name is the network name, where the instance of the authority is running. Finally the public key is the encryption key to use when contacting the instance of the authority or the decryption key when validating a signature created by that authority. The software element representing an authority besides encapsulating the above characteristics provides methods for saving and loading an authority in encrypted form to permanent storage. The saving procedure creates an XML representation of the authority, and then it encrypts and encodes it disabling anyone from accessing or altering its contents. The load procedure reverses the previous process to retrieve the authority details and create a new instance; afterwards any of the characteristics can be accessed. Both save and load procedures are used to avoid tampering the saved form of an authority in the permanent storage, providing protection from unauthorized access. The details of an authority can also be sent between two communicating parties. To achieve this, methods for extracting the authority details in XML format and retrieving the authority instance from such a format are available. An example XML document containing authority details is shown in [Figure 23]. The distinct and host name are both alphanumeric, while the public key is stored in the standard X509 format and encoded using base-64 scheme in order to be printable. The name of this software element is Authority.

```xml
<authority xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:noNamespaceSchemaLocation="_protocol/digidentity/AuthorityDetails.xsd">
  <dn>Greece</dn>
  <host>ident.gov.gr</host>
  <public_key encoding="Base64" format="X509">MIIBIjANBgkqhkiG9w0BAQEF
</authority>
```

**Figure 23:** Example XML document representing an authority
For convenience all authorities are kept in lists. The basic software element that defines the functionality of a list containing authorities is named AuthoritiesList. This software element can not be instantiated since it defines common functionality rather than a strict implementation. The final element of the library is named TrustedAuthorities and it provides a specific implementation of an authorities list, designed for holding authorities that are considered trusted. The AuthoritiesList provide methods for checking the existence of an authority in the list, retrieving an authority and saving the list to permanent storage. The first two methods are straightforward using the distinct name of an authority in order to locate it. The store method exploits the save procedure of every authority to retrieve it in encrypted form before transferring it to permanent storage. Building on top of the AuthoritiesList the TrustedAuthorities provide methods for adding or removing an authority from the list as well as loading a list of authorities from permanent storage. The first two methods (add and remove) are used for altering the contents of the list given an authority or a distinct name respectively. The third method loads one by one the authority's encrypted form and hands it out to the load method of the authority element to retrieve its instance.

4.1.5 Software package "imi.security"


The DigitalIdentity defines the abstract functionality of every digital identity (namely user and authority identities). Its purpose is to handle information about an entity that uniquely identifies it, just like real-world identities. The basic entities of a digital identity are the characteristics of the issuing authority and the digital signature. The characteristics of an authority were discussed in section 4.1.4, while the insides of a digital signature in section 4.1.3. So the software element provides methods for verifying the digital signature and retrieving the details of the authority that issued the identity. Keep in mind that the digital signature is created by the issuing authority, meaning that the verification has to be performed using that authority's public key.

One of the specific implementations of a digital identity is the AuthorityIdentity, specifying identity information for authorities. The complementary functionality includes the characteristics of the authority that holds the identity, and methods for retrieving them. In addition it provides concrete implementation, for the abstract functionality inherited by the DigitalIdentity. An example of the XML representation of an authority identity is shown in [Figure 24]. As we can see we have the holder and issuer authority details encapsulated in
the “authority_identity” tag and the digital signature, at the end of the document, which can be used to verify the integrity of this digital statement.

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<certificate xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="_protocol/AuthorityCertificate.xsd">
  <authority_identity>
    <holder>
      <dn>Greece</dn>
      <host>ident.gov.gr</host>
      <public_key encoding="Base64" format="X509">MIIBIjA</public_key>
    </holder>
    <issuer>
      <dn>Global</dn>
      <host>ident.int</host>
      <public_key encoding="Base64" format="X509">MIIBIjA</public_key>
    </issuer>
  </authority_identity>
  <signature>
    <encoding>Base64</encoding>
    <encryption>RSA/ECB/OAEPPadding</encryption>
    <hashing>MD5</hashing>
    <data>MIIBiANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAnLiafv56u</data>
  </signature>
</certificate>
```

**Figure 24:** Example XML document representing an authority identity

The UserIdentity is another specific implementation of a digital identity, but is significantly different than authority identities. Its target is not to uniquely identify a piece of software but a human being. For this reason the holder is defined by his personal information, a public key and a binary string representing the unique serial number of the identity. This binary string will be referred to as master identifier. Again, all abstract functionality inherited by DigitalIdentity is implemented, to achieve the explicit goals of a user identity. To better describe the internal workings of a user identity we will examine the example in [Figure 25]. As we can see the issuer and signature parts of the XML document are identical to both authority and user identities, as they are inherited from the DigitalSignature element. The holder information (information under the “holder” tag of our example) contains the public key of the holder along with the master identifier, appointed by the IIA of this digital identity. The master identifier is a sequence of 128-bits and is created by hashing the private information of the user with the MD5 message digest algorithm. This procedure creates a different binary string for every user, under the constraint that the private information provided by every user is unique; an assumption we can consider true for convenience. The “personal” tag contains the personal information we mentioned earlier. In fact, it is an encapsulated XML message encoded using base-64 scheme to avoid escaping it. An example of the personal information, in decoded form, is shown in [Figure 26].
The document contains the private information of the user and his password. In fact the plain password is not stored, because it would be exposed to an adversary. In contrast the password is hashed using the MD5 message digest algorithm to produce a binary string representation. This representation is then base-64 encoded and placed inside the personal user information. The main advantages are that it is computationally infeasible for an adversary to reconstruct the password given the hash value, and at the same time we can check if the password given by the user matches the one stored inside the user identity. The procedure is similar to the validation of a digital identity, namely the password given by the user is hashed and then the value is compared with the hash value inside the identity. The usage and purpose of the password are described in section 3.2.2, where the identity validation procedure to an IIA is described. The private information under the “private” tag is another encapsulated XML document, containing the private and sensitive user information. As we can see it is encrypted and encoded to make it unintelligible to everyone. [Figure 27] shows an example of a plain text XML document containing the private information of a user. This information is supplied by the user upon creation of the identity and is mounted inside the identity contents as proof of his real-world identity. As we said the IIA encrypts the information using a secret key known only to him, in order to prevent unauthorized access to the sensitive information enclosed. Finally, the user identity is the first element in the libraries making use of the database connectivity package described in section 4.1.8. The store method uses the database package to connect to a database instance and save the identity in a relational table. This method is only used by the IIA component, for optimizing the performance of identity management.
Figure 25: Example XML document representing a user identity

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<certificate xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="_protocol/UserCertificate.xsd">
    <user_identity>
        <personal>
            <encoding>Base64</encoding>
            <data>MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAnLiafv56u</data>
        </personal>
        <holder>
            <masterid encoding="Base64">MIIBIjANBgkqh</masterid>
            <public_key encoding="Base64" format="X509">MIIBIjA</public_key>
        </holder>
        <issuer>
            <dn>Greece</dn>
            <host>ident.gov.gr</host>
            <public_key encoding="Base64" format="X509">MIIBIjA</public_key>
        </issuer>
        <signature>
            <encoding>Base64</encoding>
            <encryption>RSA/ECB/OAEPPadding</encryption>
            <hashing>MD5</hashing>
            <data>MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMICQgKCAQEAoLiafv56u</data>
        </signature>
    </user_identity>
</certificate>
```

Figure 26: Example XML document containing the personal information of a user

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<personal xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="_protocol/PersonalInfo.xsd">
    <private>
        <encoding>Base64</encoding>
        <encryption>AES/ECB/PKCS5Padding</encryption>
        <data>MIIBIjANBgkqhkiG9w0BAQFEAAOCAQ8AMICQgKCAQEAoLiafv56u</data>
    </private>
    <password>
        <encoding>Base64</encoding>
        <hashing>MD5</hashing>
        <data>MIIBIjANBgkqhkiG9w0BAQFEAAOCAQ8AMICQgKCAQEAoLiafv56u</data>
    </password>
</personal>
```
We will now proceed to DigitalCertificates. Certificates are statements issued as proof of performing a certain procedure, while providing valuable information to the recipient about the outcome of their request. The difference with digital identities is that certificates are used during the communication of two parties and are discarded when the communication is over. For this reason certificates are not important statements and they are not digitally signed. The basic entities of a certificate are a timestamp, the characteristics of the issuing authority and the host name of the recipient of this certificate. The timestamp is a date and time field indicating the instance in time when the certificate was issued, with second precision. The characteristics of an authority are those discussed in section 4.1.4. In addition it contains methods for retrieving the entities just described.

IssuerCertificate, ServiceCertificate and StatusCertificate are software elements that specialize and extend the functionality of a DigitalCertificate. The first two are created by the IIA component and delivered to an end-entity middleware or a service mediator respectively. The third is created by the service mediator and delivered to the end-entity middleware. The IssuerCertificate signals the successful completion of the identity validation procedure, performed by an individual, in order to activate his digital identity. As we have seen in section 3.2.2, during this procedure a secret key is exchanged; allowing the user to respond to challenge requests, in order to be validated to service logins without the need of another set of credentials. The key is created by the IIA component and placed inside the certificate, so the transfer of the certificate also facilitates the key exchange. An example XML representation of such a certificate is shown in [Figure 28]. In the example the “identity” tag holds the master identifier of the user that initiated the login procedure, encoded using base-64 scheme.

The ServiceCertificate is created upon successful authentication of a secondary identity. The service mediator will execute the identity authentication procedure to the IIA that issued the identity held by the individual. The IIA will examine the information presented and create a service certificate to notify the service mediator about the outcome of the request. For more information about the internal workings of the authentication process and how secondary identities work refer to section 3.2.2. [Figure 29] is an example of a service certificate. The extensions implemented are the incorporation of the secondary identity to the

Figure 27: Example XML document containing the private information of a user

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<private xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="_protocol/PrivateInfo.xsd">
  <firstname>Vassilis</firstname>
  <lastname>Poursalidis</lastname>
  <country>Greece</country>
  <prefecture>Heraklion</prefecture>
</private>
```
certificate in the “identity” tag and the parent authority information of the IIA. The user’s secondary identity is encoded and encrypted to make it unintelligible to the service mediator and prevent him from extracting the master identifier that was used to create it. At the same time it can be used to uniquely identify the individual. The parent authority information will be used by the service mediator, in order to verify the legitimacy of the IIA.

StatusCertificate is the final software element implementing a ServiceCertificate. Status certificates are created by the service mediator and are forwarded to the user that requested to use the service. Its purpose is to inform him about the status of his request. The only extension implemented is the incorporation of the secondary identity to the certificate in the “identity” tag. [Figure 30] presents an example status certificate.

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<certificate xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="_protocol/IssuerCertificate.xsd">
    <holder>
        <host>147.52.98.181</host>
        <identity>
            <encoding>Base64</encoding>
            <data>MIIBIjANBgkqh</data>
        </identity>
        <session_key algorithm="AES">MIIBIjANBgkqh</session_key>
    </holder>
    <issuer>
        <dn>Greece</dn>
        <host>ident.gov.gr</host>
        <public_key encoding="Base64" format="X509">MIIBIjANBgkqh</public_key>
    </issuer>
    <timestamp>
        <date>
            <year>2004</year>
            <month>6</month>
            <day>30</day>
        </date>
        <time>
            <hour>13</hour>
            <minute>43</minute>
            <second>32</second>
        </time>
    </timestamp>
</certificate>
```

**Figure 28:** Example XML document representing an issuer certificate
Figure 29: Example XML document representing a service certificate. The “issuer” and “timestamp” tag details were omitted as they are similar with the ones in Figure 28.

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<certificate xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="_protocol/ServiceCertificate.xsd">
    <holder>
        <host>147.52.98.180</host>
        <identity>
            <encoding>Base64</encoding>
            <encryption>AES/ECB/PKCS5Padding</encryption>
            <data>MIIBJANBgkqh</data>
        </identity>
    </holder>
    <issuer>...
    </issuer>
    <parent>
        <dn>Global</dn>
        <host>ident.int</host>
        <public_key encoding="Base64" format="X509">MIIBJANBgkqh</public_key>
    </parent>
    <timestamp>...
    </timestamp>
</certificate>
```

We did not mention that all software elements in this library provide methods for extracting their internal information to XML documents, and retrieving an instance of the

Figure 30: Example XML document representing a status certificate. The “issuer” and “timestamp” tag details were omitted as they are similar with the ones in Figure 28.

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<certificate xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="_protocol/StatusCertificate.xsd">
    <holder>
        <host>147.52.98.181</host>
        <identity>
            <encoding>Base64</encoding>
            <encryption>AES/ECB/PKCS5Padding</encryption>
            <data>MIIBJANBgkqh</data>
        </identity>
    </holder>
    <issuer>...
    </issuer>
    <timestamp>...
    </timestamp>
</certificate>
```
corresponding software element from its XML representation. The examples presented in [Figure 24], [Figure 25], [Figure 28], [Figure 29] and [Figure 30] are such representations. In DigitalIdentities and DigitalCertificates these methods are abstract, and are implemented independently by each sub element we have seen. This provides the necessary functionality for saving or sending any digital statement to permanent storage or through the network. On the other hand it allows us to restore the instance of a statement from such a representation.

4.1.6 Software package "imi.network.message"

This library provides functionality for message transmission, containing a single software element named MessageHelper. The helper is the final step that information goes through, before being transmitted over unsecured channels. Its purpose is to make the information unintelligible to an adversary and prevent tampering it. To achieve this, the helper makes use of strong cryptography; encrypting every piece of information before handing it over to transmission layer. In [Figure 31] we see an example, where the underlying communication network is unsecured. The helper transforms plain text to cipher text and forwards it for transmission. While the cipher text is in transit the adversary can read or modify it. Since encryption is tamper resistant the receiver will immediately detect that modifications have been made to the cipher text and cease communication. Moreover strong cryptography makes the cipher text useless to the adversary since he cannot extract any intelligible information from it. In the absence of the helper the information would have been transmitted in plain text, allowing the adversary to compromise the privacy of the communicating parties and even add or alter the information in transit.
The message helper can be thought of as an encryption and decryption layer on top of the transport and underneath the application layer. Its purpose is to safeguard the communication between two parties, while providing an abstract plain text communication system. This way all the complexity details of message encryption and decryption are hidden from the application, preserving the desired security guarantees. From the sender’s perspective the helper encrypts and wraps every piece of information; making it unintelligible to any one, except the intended recipient. On the other hand the recipient is able to reverse the process unwrapping and decrypting the incoming message, retrieving the enclosed information. The design is such that allows various encryption schemes to be used, while switching from asymmetrical to symmetrical cipher techniques is trivial. The example in Figure 18 presented the wrapped message returned by the message helper. As we can see the information is encrypted and encoded inside the “body” tags, whereas the rest of the information is metadata. The metadata describe the algorithms used for the encryption and encoding process. Once the message is delivered to the recipient the metadata will guide the message helper on how to decode and decrypt it; rebuilding the initial information.

4.1.7 Software package "imi.trust.protocol"

As we have mentioned in section 3.5 every communication establishment between two parties involves a series of challenge-response messages. These messages forego payload communication and are used to authenticate that either one or both parties are the intended
recipients. With payload communication we mean the messages transmitted by the application layer in order for the two parties to exchange information. The authentication is achieved through public-key cryptography, where only the holder of a private key can decipher a message encrypted with the corresponding public key. Together with authentication, a secret key is exchanged between the two parties to encipher future traffic in secure fashion. To accumulate this common functionality of message exchange and provide a programmable interface a hierarchy of software elements was created. This hierarchy was built on top of MessageHelper software element that we discussed in the previous section, and is illustrated in [Figure 32]. In the figure we see a tree structure where every node represents a software element and the parent-child edge a specialization in functionality (from general to specific).

So both ServerSide and ClientSide software elements extend the functionality of the message helper providing specific implementation needed for server and client side connections respectively. In addition they offer type safety for forthcoming implementations on either client or server side. We must clarify that we have two parties communicating without distinguishing them to client and server, at least not in the classical meaning. We name client the party that initiated the connection and server the party that accepted it. This way we can refer to every party depending on connection type, and not based on whether or not it provides some kind of service.

![Figure 32: Communication protocols hierarchy](image)

We mentioned earlier that we need to authenticate either one or both communicating parties. If we need to authenticate only one party, it means that the client that initiated the connection has to make sure that he is exchanging messages with the intended server. The server needs no knowledge about the client that connected. We call this type of connection Anonymous, since the client remains unauthenticated. In addition, when we need to authenticate both parties the client must also be authenticated by the server. This type of connection is called Eponymous, since both parties know the identity of one another. The leaf nodes of [Figure 32] are the software elements implementing the details just discussed. The
first two extend the ServerSide software element and are named AnonymousServer and EponymousServer, while AnonymousClient and EponymousClient extend the ClientSide software element. As one can imagine the AnonymousClient and AnonymousServer are complementary and are used by each party to exchange the challenge-response messages creating an Anonymous connection type. In the same way, EponymousClient and EponymousServer hide the details of an Eponymous connection type, creating a secured channel to be used by the application. To better understand the exact details of the software elements in question we created a graphical representation of the messages exchanged in [Figure 33]. The first part of the figure is an Anonymous connection with Party #1 being the server and Party #2 the client. The same goes for the second part of the figure, only this time we have an Eponymous connection. As we can see the main difference between the two communication types is during the first message of the Eponymous connection. The Party #2 has to send a unique identifier pinpointing to Party #1 how to locate his public key. The Party #1 is assumed to have a list of public keys indexed by this unique identifier, so he can locate the public key of Party #2 and send the second message encrypted with that key. The Request symbol denotes an abstract message exchanged between the communicating parties that must be implemented by specific connection implementations.

![Figure 33: Payload communication for an Anonymous (A) and an Eponymous (B) connection](image)

The AnonymousServer provides two methods, the first starts the session by consuming the first message from the client and sending back the response; while the second method consumes the second message and returns an abstract Request. The Request can be any XML document containing information about the specific service call and will be defined and parsed by subsequent software elements that implement specific protocol details. On the other hand, AnonymousClient is a bit more complicated. It has to generate a random challenge number and the secret key that will be used for the rest of the communication session. Analogous it provides two methods, one for sending the first message and collecting the response and the second sends the abstract Request to the server. Examples of the XML documents passed from AnonymousServer and AnonymousClient to MessageHelper are
shown in the next figures. [Figure 34] is the message sent from the AnonymousServer, while [Figure 35] and [Figure 36] are the messages sent by AnonymousClient in the respective sequence.

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<challenge xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:noNamespaceSchemaLocation="_protocol/message/ChallengeSync.xsd">
       <challenge_response>77622461</challenge_response>
</challenge>
```

**Figure 34:** XML document passed from AnonymousServer to MessageHelper

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<challenge xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:noNamespaceSchemaLocation="_protocol/message/ChallengeCommence.xsd">
       <challenge_number>77622461</challenge_number>
       <shared_key algorithm="AES">MIIBIjANBgkqh</shared_key>
</challenge>
```

**Figure 35:** XML document passed from AnonymousClient to MessageHelper used to initiate an Anonymous connection

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<challenge xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:noNamespaceSchemaLocation="_protocol/message/ChallengeAck.xsd">
       <service_type>lookup</service_type>
       <message encoding="Base64">MIIBIjANBgkqh</message>
</challenge>
```

**Figure 36:** XML document passed from AnonymousClient to MessageHelper used to conclude the challenge-response protocol and exchange the first request

Next we have the EponymousServer and EponymousClient pair. Eponymous and Anonymous server side implementation have similar methods. The extra functionality is an abstract method for retrieving the public key of the client, based on an identifier. The exact implementation of matching an identifier to a public key will be provided by subsequent software elements. The client side also provides two methods that perform similar functionality with the Anonymous connection type. The main differences in these software elements are the exact message details, the creation of the secret shared key by the server and the creation of challenge numbers by both the server and the client. The next three figures are the messages exchanged. [Figure 37] shows an example of the message send by the server to the client including the challenge number sent by the user, the newly created server challenge and a secret key to be used upon completion of the challenge-response
protocol. The next two figures ([Figure 38] and [Figure 39]) are the initial message send by
the client and the message containing the abstract Request message.

Figure 37: XML document passed from EponymousServer to MessageHelper

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<challenge xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="_protocol/message/ChallengeAccept.xsd">
  <challenge_number>83442915</challenge_number>
  <challenge_response>77622461</challenge_response>
  <shared_key algorithm="AES">MIIBIjANBgkqh</shared_key>
</challenge>
```

Figure 38: XML document passed from EponymousClient to MessageHelper used to initiate
an Eponymous connection

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<challenge xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="_protocol/message/ChallengeStart.xsd">
  <challenge_number>77622461</challenge_number>
  <identifier encoding="Base64">MIIBIjANBgkqh</identifier>
</challenge>
```

Figure 39: XML document passed from EponymousClient to MessageHelper used to conclude
the challenge-response protocol and exchange the first request

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<challenge xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="_protocol/message/ChallengeAccept.xsd">
  <challenge_response>83442915</challenge_response>
  <message encoding="Base64">MIIBIjANBgkqh</message>
</challenge>
```

To summarize the concepts of the current section we could say that we have built a
full set of layers on top of the transmission layer. This provides an abstraction and a
programmable interface to any application, in order to avoid both coping with the details of
message encryption and the primary stages of connection establishment (challenge-response
messages). In [Figure 40] we see that applications has no direct knowledge of the
transmission layer and can only be accessed through the MessageHelper or though one of the
client or server software elements. So an application can use the AnonymousClient layer to
establish an anonymous connection type, which is secured by the MessageHelper layer. When
the process of payload messages is completed, the application will stop using the
AnonymousClient layer. From that point on the functionality of MessageHelper will handle the
rest of traffic, since the authentication has finished and a secret key has been shared for fast
and secured communication.
4.1.8 Software package "imi.db"

This library consists of four software elements, namely: Connector, OracleConnector, InsertQuery and SelectQuery. The Connector is an abstract database connection that defines the functionality to be supported by an implementation for a particular database system. Basically it serves to group (and provide type safety for) all database connectors. The OracleConnector is such an implementation, using specific drivers to connect to the Oracle DBMS, and providing specialized functionality. The InsertQuery encapsulates the details of executing insert, delete and update queries. Although it defines all the required functionality it remains abstract, since no query is defined for execution. The query string will be set by sub-implementations, fulfilling explicit needs. Similarly the SelectQuery element is abstract and is used for executing select queries.

The purpose of the database library is to create a programmable interface for managing database connections and executing queries, and at the same time provide a database connection independent from the underlying DBMS implementation.

4.1.9 Software package "imi.db.queries"

To avoid incorporating database queries into software elements the queries library was introduced. Its purpose it to aggregate software elements that wrap database queries and provide abstract interface for ease of use. The next table lists the name of the software elements in the library along with a description of their usage. Keep in mind that certain
implementations override the standard interface provided by the InsertQuery and SelectQuery to satisfy casual needs.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UserIdentityDelete</td>
<td>An Issuer instantiates this query to delete a user identity that was previously issued</td>
</tr>
<tr>
<td>UserIdentityInsert</td>
<td>Given the details, a new user identity is stored for future reference by the Issuer</td>
</tr>
<tr>
<td>UserIdentityList</td>
<td>This query is executed by the private interface of the Issuer to retrieve a list of the available user identities</td>
</tr>
<tr>
<td>UserIdentityLogin</td>
<td>After completing the login procedure the Issuer stores relative information for future reference</td>
</tr>
<tr>
<td>UserIdentityRevoke</td>
<td>An Issuer instantiates this query to revoke a user identity that was previously issued</td>
</tr>
<tr>
<td>UserIdentitySelect</td>
<td>Selects the user identity given its master id, used when the Issuer is performing a login request</td>
</tr>
<tr>
<td>UserIdentitySelectAuth</td>
<td>Selects the user identity given its IP address, used when the Issuer is asked to authenticate a secondary identity</td>
</tr>
</tbody>
</table>

Table 7: The name of the software element and a description of the task performed

4.2 Software components

In this section we will present the software libraries that use the functionality of the custom libraries and provide concrete implementation of the software components that our infrastructure consist of. The first library we will examine is the “imi.trust.digidentity”, which contains software elements that implement the IIA and ACC software components of the DIM hierarchy. Next we have the “imi.trust.service” library which provides the service mediator component along with an example service, to demonstrate the functionality provided by the tools we have built. The final software component is the end-entity middleware, and the software elements that comprise the implementation are included in the “imi.trust.middleware” library. The libraries we have just mentioned provide the server-side functionality supported by the procedures we have defined in section 3.5. To also facilitate and provide a programmable interface for the client-side part of the procedures we have implemented two additional libraries. The “imi.trust.digidentity.protocol” library provides a programmable interface for every public interface available by the DIM hierarchy. Similarly the “imi.trust.mediator.protocol” defines the client-side interface for the service login procedure.

4.2.1 Software package "imi.trust.digidentity"

Nearly half of the software elements in this library provide functionality for the AAC component, while the rest of them are used by the IIA. In fact, both AAC and IIA share
common functionality as both are authorities inside the DIM hierarchy provide publicly available functionality as well as private interfaces for administration purposes. Their differences are in the exact procedures supported and in the fact that AACs handle authorities, while the IIAs handle digital identities. We will analyse the software elements of this library by examining how flow control passes through them, in a way to also realize their internal workings.

The instance of the AAC software component starts with the Catalog software element. The tasks performed are to verify information from external sources, initialise the software instance, start the listener for public connection and finally start the private interface. The external information sources are files stored in the computers file system and contain a public-private key pair, the authority's identity information and its parent and child authorities. All the information is either encrypted or digitally signed in order to prevent tampering, so the verification either deciphers the data or validates the available digital signature. If any of the checks fail the operator of the instance is informed and execution is stopped. Even the key pair is encrypted and requires a password to retrieve and validate the contents of the store where they are kept. This prevents unauthorized modifications to be made to the key store, which is the starting point of the verification process. Looking closer to the information contained in the files we see that the key pair is used for establishing secure channels through asymmetric cryptography, and to decipher encrypted files. The authority identity is used by the software instance to identify itself, while the parent and child authorities identify the position of the AAC in the DIM hierarchy. As we can see the initialisation is performed in parallel with the verification task.

The next step of the Catalog is to start the CatalogListener element, a thread that accepts incoming network connections and hands them over to the CatalogHelper. The goal of this element is to render the Catalog to a multithreaded environment, possible of handling multiple concurrent requests. The details of the connection are handled by the CatalogHelper, which extends the AnonymousServer element and provides the proper responses for both public interfaces available by the AAC. If the request is an authority lookup then the child authorities list and a child authority will be returned, given its distinct name. Otherwise a validation of the given authority will be conducted and an error message or the parent authority will be returned.

The final task is to commence the command line interface. This private interface is used by administrators to perform maintenance operations in order to alter the structure of the DIM hierarchy. The available commands include adding new authorities, listing the available child authorities and deleting a child authority. The extra action that needs to be taken in the case of an authority addition is the transfer of the files created to the computer system where the instance of this new authority will run. The files are protected, so even an insecure channel can be used.
What we forgot to mention is where the parent and child authorities are stored when read from their respective files. We could have used the TrustedAuthorities element, but in order to verify every piece of information read from the file system we needed the authority identity information to be coupled with the respective authority. This forced us in implementing another list, built on top of the functionality provided by the AuthoritiesList and adding the special purpose functionality. We named this software element CatalogAuthorities as only the DIM framework would use it.

The general recipe described for the AAC remains the same for the IIA software component as well. The execution starts from the Issuer element and its first concern is to check the validity of the information stored in the file system. The only difference here is the absence of the child authorities list. Again, the initialisation if performed during the verification but in the case of the IIA a connection to the RDBM is established in order to gain access to its digital identities. If everything is successful the Issuer starts the IssuerListener and the command line interface. The IssuerListener serves the same purpose as the CatalogListener, only this time it hands information to the IssuerHelper. The command line interface shares the goal of being used for performing maintenance operations, but supports completely different commands. Administrators can revoke, delete and even list the available digital identities.

The resemblance of the two components ends to the IssuerHelper, as it extends the ServerSide element in order to handle both Anonymous and Eponymous connections. The goal of the IssuerHelper is to consume the first message of the connection and decide if it is an Anonymous connection that must be handled by the IssuerAnonymous element, or the IssuerEponymous element should be called. These software elements provide the functionality needed by the three public interfaces available. The IssuerEponymous handles the identity validation procedure by extending the EponymousServer element. The first thing the implementation of the IssuerEponymous must specify is the retrieval of the public key of the individual based on his master identifier. This is because the functionality is not provided by the EponymousServer and must be specified by subsequent implementations. The next step is to verify the password presented by the user with the one stored locally in the database. Finally, depending on the outcome of the comparison an error message or a certificate is returned informing the end-entity middleware about the sign-in details.

The IssuerAnonymous element handles identity creation and identity authentication requests, extending the functionality of the AnonymousServer. In the case of identity creation the end-entity middleware sends the private information of the individual in exchange for a digital identity and an identity key. [Figure 41] is an example of the private information of a user requesting a digital identity, while [Figure 42] the response by the IssuerAnonymous. The second anonymous interface is used by the service mediator to authenticate the credentials an individual has presented. The first message received during the authentication
process by the IssuerAnonymous element is the credentials that the user presented. [Figure 43] is an example, where the host tag contains the IP address of the individual that presented these credentials and the data tag his encrypted secondary identity. The user information will be retrieved through the host tag and used to check the matching of the user's secondary identity to his master identifier. But to be certain about the legitimacy of the user, the IssuerAnonymous creates and sends a challenge number with the IP address of the service that requested the execution of this authentication process, [Figure 44]. This information is encrypted using the session key exchanged with the end-entity middleware during the identity validation process, in order to remain unintelligible to the service mediator. The next message received contains the challenge response provided by the user. The response is presented in [Figure 45] after being decrypted using again the session key. A comparison among those two numbers will determine if an error message or a certificate will be send to the service mediator.

Figure 41: Example XML document containing the private information of the user as presented by the end-entity middleware

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<private xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="_protocol/digidentity/PrivateData.xsd">
    <firstname>Vassilis</firstname>
    <lastname>Poursalidis</lastname>
    <country>Greece</country>
    <prefecture>Heraklion</prefecture>
    <password>qwerty</password>
    <public_key encoding="Base64" format="X509">MIIBIjANBgkqh</public_key>
</private>
```

Figure 42: This document contains the digital identity of the user along with his identity key

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<identity_creation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="_protocol/digidentity/UserIdentityDetails.xsd">
    <user_identity encoding="Base64">MIIBIjANBgkqh</user_identity>
    <identity_key algorithm="AES">MIIBIjANBgkqh</identity_key>
</identity_creation>
```
4.2.2 Software package "imi.trust.service"

The service library provides an implementation of the service mediator, along with a programmable interface to create new services that are aware of the existence of our infrastructure. Our approach establishes an entry point for incoming request, and spawns new service instances upon successful validation of the credentials presented by individuals. The legitimacy of the credentials is verified through the service mediator.

The running instance of the service starts with the Initiator software element. The element starts by checking the information that is stored in the computer’s file system, namely the public-private key pair that will be used for asymmetric cryptography and the list of trusted authorities. The trusted authorities list works like a cache, where recently visited authorities are held in order to avoid re-contacting them to verify their legitimacy. The functionality of the list is provided by the TrustedAuthorities software element we presented in section 4.1.4. Afterwards the Initiator waits for incoming connections and hands them over...
to the ServiceMediator. The ServiceMediator extends the functionality available by the AnonymousServer, in order to implement the service login protocol we show in section 3.5.4. To complete its purpose the software element also provides methods to perform the identity authentication to the IIA and the authority validation protocol, if the IIA is now known. The identity authentication is a simple procedure where the credentials of the individual are verified using the respective IIA. An example of the credentials submitted by the individual is shown in [Figure 46]. In contrast, the authority validation process is more demanding as the DIM hierarchy has to be traversed in order to examine if the IIA presented by the individual is part of the legitimate structure. During this process every intermediate authority visited is added to a temporary list. Upon successful validation of the legitimacy of the chain, the complete list is inserted to the TrustedAuthorities. The goal is to have a cache of trusted authorities and prevent future traverses. For example, we mentioned earlier that the authority validation procedure is executed only if the IIA is not known. The known authorities are those included in the TrustedAuthorities list. This way, if the IIA was previously visited and found to be legitimate, future authority validation procedures could be avoided.

The last step is to initiate the service specific functionality and present the user credentials. For type safety an AbstractService software element was created, enforcing all specific service implementation to extend it. It provides abstract methods to establish the entry point of the service along with methods for executing common functionality. Our final extension to this library was the SampleService software element. As the name denotes it extends the AbstractService and provides sample service functionality. The task performed is the dispatch of the “Hello World!” message to the individual.

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<credentials xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="_protocol/digidentity/CredentialsData.xsd">
    <issuer>
        <dn>Global</dn>
        <host>ident.int</host>
        <public_key encoding="Base64" format="X509">MIIBIjANBgkqh</public_key>
    </issuer>
    <identity>
        <encoding>Base64</encoding>
        <encryption>AES/ECB/PKCS5Padding</encryption>
        <data>MIIBIjANBgkqh</data>
    </identity>
</credentials>
```

**Figure 46:** Credential information send by the end-entity middleware to the service mediator

### 4.2.3 Software package "imi.trust.middleware"
The software elements, that comprise this library, provide the functionality of the end-entity middleware component. The execution starts from the Client element, where the internal state of the component is initialized by loading information from the file system. The required external sources include a key store, a digital identity, an IIA identity and the secondary identities store. The first source holds the public-private key pair used for asymmetric cryptography, and the identity key of the individual used to encipher his secondary identifiers. The second and third sources are respectively the individual's identity in XML document format and the issuing IIA. Finally, the fourth source keeps encrypted the individual’s secondary identities, along with their mapping to services. All loaded information is either encrypted or digitally signed; this ensures the integrity of the information and protects it from unauthorized modifications. But there is the case where the middleware component is executed for the first time, and no information is available in the file system. Then, the private information of the user is collected and presented to an IIA, in order for the middleware to retrieve a legitimate digital identity. The last step is to save all relevant information to the file system, allowing its usage from future instantiations of the component. This action also sets the internal state of the middleware, reaching the same execution point as the initialization process.

The Client element’s functionality concludes with the execution of the identity validation procedure and the establishment of a local listener for incoming connections. The first action activates the digital identity of the individual, along with all the information about his secondary identities. Now, all resources needed to access remote services are available and held by two software elements, namely SessionStore and IdentityStore. The SessionStore contains the digital identity of the user along with the identity and session keys, while the IdentityStore retains secondary identities and the services that are being used at. The IdentityStore also provides various methods for retrieving the secondary identities, along with the functionality to create new and unassociated identities. To further facilitate users, secondary identities are associated with aliases chosen by them. This is extremely valuable towards the ease of use and distinction of secondary identities. This association is achieved by defining the SecondaryIdentity element, which stores a secondary identity with the respective alias. Keep in mind that aliases are used locally for convenience, and are never sent to the service provider.

The local listener is used to serve applications in the individual’s computer system. In practice, the requests from applications are accepted by the Client, but are handed to the ConnectionHandler element. The ConnectionHandler provides the functionality needed to execute the service login procedure and establish a secure channel between the end-entity middleware and the service provided. So, the request of the application for remote service access is fulfill and the channel is ready for service specific communication. All details about authentication or the creation of the secure communication channel are hidden from the
application. Every single connection for remote service access is managed by a different instance of the ConnectionHandler element and every instance remains active throughout the needs of the application.

The crucial point of the end-entity mediator is the amount of information needed by the user in order to accomplish its tasks. The interactions required must be sufficient, in order to protect the user from unauthorized access or privacy violations, but at the same time minimal, to prevent unnecessary user interference. There are only three forms that require input from the user. The first is shown in [Figure 47], where the user presents his personal information, in order for the end-entity middleware to create a digital identity. Since the identity creation procedure is for demonstration purposes, this form is not required by the end-entity middleware and will be removed in a non-demonstration environment. In this case the user will have to manually supply the digital identity information to the end-entity middleware. The next form is shown in [Figure 48], where the user is required to enter his single sign-on password. This form will be presented only every time the user initializes the end-entity middleware, in order to activate his digital identity. The last form is presented in [Figure 49], where the user has to select the secondary identity to be used. The form appears every time a user requests to access a remote service, in order to provide the secondary identity to present for this session. As we can see the information is divided in three pieces: create a new identity, select an existing identity and federate a secondary identity. The first option creates a new secondary identity with the given alias and associates it with the current service provider. The second option provides a list of secondary identities that are already associated with the current service, and the user is requested to select one of them based on their alias. Finally we can perform identity federation, meaning that the user can select among all available secondary identities, including those used in other service providers. For this reason the first drop down box list all available service providers and the second the digital identities associated with the provider selected. In all cases the secondary identity selected or created is associated with the current service provider and is used to execute the service login procedure.
Figure 47: Form presented by the end-entity middleware in order to create a new digital identity for the individual

![Validate digital identity](image)

Figure 48: Form requesting the individual to enter his password in order to activate his digital identity and the information held by the end-entity middleware

![Select secondary identity](image)

Figure 49: Form requesting the individual to select a secondary identity that will be used for the current session of service access

4.2.4 Software package "imi.trust.digidentity.protocol"

This library provides a programmable interface for the client-side part of the procedures offered by the DIM framework. For example if the end-entity middleware wants to execute the identity validation procedure it will instantiate the IssuerUserValidate software element providing only the proper information. The outcome will be received without the need to go into the details of establishing an Eponymous connection, managing message encapsulation or details about the format of the messages exchanged. Of course the IssuerUserValidate depends on the functionality provided by the EponymousClient software element for the establishment of an Eponymous connection, specifying the details needed by the procedure in question. The remaining software elements of the library extend the
functionality provided by the AnonymousClient to supply implementations for the rest of the procedures. Namely, the CatalogRequest for authority lookup and validate request, the IssuerRequest for identity creation and the IssuerUserAuthenticate to authenticate secondary identities.

4.2.5 Software package "imi.trust.service.protocol"

The only public interface not included in the DIM framework is the service login, and is offered by the service mediator. Following the rational of the previous section the only software element of this library offers the client-side implementation of the service login procedure. The name of the element is ServiceUserAuthenticate and is used by the end-entity middleware to establish a connection with the service mediator, accessing the provided functionality.

4.3 Design decisions

Our first dilemma was to decide how authority identity information will be handled by the service mediator. As we have mentioned the mediator needs to know about the authorities that belong in the DIM framework, in an effort to block digital identities originated from illegitimate IIAs. This information is gathered through the authority validation process, where a valid path must exist between the authority being checked and an established member of the DIM hierarchy. During the process the authorities encountered are stored locally, to avoid revalidating them in future requests. Problems arise when an authority that was previously a member of the DIM framework leaves the hierarchy. Such actions make all descendant authorities illegitimate, and requests from individuals holding digital identities from those authorities should be rejected. As we can see the service mediator must somehow be informed about the changes in the DIM hierarchy. One solution would be to use authority revocation lists, where upon successful validation of an authority the service mediator registers to the revocation notification list of this authority. If a deletion of a branch occurs, then all members of the notification list receive this change, and prevent further interactions with that branch. The scheme resembles the certificate revocation lists (CRLs) [23], where a list of certificates which have been revoked, are no longer valid, and should not be relied upon by any system component. The second solution requires the local list held by the service mediator to act like a cache. Authorities added to this list will be discarded after a certain time span and revalidation must be executing. This scheme is much simpler, but leaves a period where invalid credentials will be considered legitimate. The service mediator will allow access to the holders of the credentials for an extra period equal to the time span of the revalidation. If we exclude the caching this solution is similar to the online certificate
status protocol (OCSP) [37], proposed as an alternative approach to CRLs. The second approach was deployed in our identity management infrastructure.

The second design decision was to implement our own certificate documents, rather than using version 3 of the X.509 standard [38]. The disadvantage of the X.509 certificates is their binary format, when compared with our self-explained certificates in XML format. In fact, we wanted to avoid conforming to a binary and complex standard during the implementation of our demonstration platform. This decision also prevented us from using the SSL [7] protocol to establish secure channels between two parties. But even so, our design allows us to incorporate any of these standards in the case of real world deployment or the need for further study. The incorporation will signal changes only to a small set of software elements in the custom libraries, preventing further modifications. Concluding the discussion on standards, we are aware of two recommendations proposed by the W3C that could be easily integrated to our infrastructure. The XMLEncryption [39] defines the specification and metadata regarding representation of encrypted data in XML format, while the XMLSignature [40] for digital signatures.
5 Evaluation of implementation

In this chapter we will focus on the security guarantees provided by our infrastructure. It is not realistic to assume that an adversary will not be able to get information held by the components of our system. The goal is to prevent this information from being intelligible to the attacker and minimize the risk of mass identity exposure. We will examine a complete list of attacks, their consequences and countermeasures that can be taken to further strengthen the overall system security. Next, we will present a series of measurements we conducted to the various procedures defined, in order to review the overhead implied by our approach and point out considerations for improvement. Finally, we provide a list of potential application where our infrastructure could be used to optimise the management of local accounts and user experience.

5.1 Attacks to our system

The most powerful aspect of our system is the guarantees it provides to both the service providers and the individuals. The procedures we have defined in the previous chapters are robust and manage to prevent an adversary from deceiving the components available in our infrastructure. The only feasible attacks are the ones against the operating environment where the components instances are running. This way the information held by the component is exposed to the adversary, creating some interesting scenarios about the usage of the gained information. But the main value of our approach is not to avoid information exposure, but to prevent mass identity attacks. This means preventing information about more that one users to be exposed with a single security breach. In the next sections we will examine such attacks along with an interesting man-in-the-middle attack.

For the suspicious reader we will go through a quick survey of the procedures we have implemented and the restrictions embodied from their enforcement. First of all the authority validation protocol prevents unauthorized authorities from claiming that they belong in the DIM hierarchy. Every authority that does not belong in this clique will be easily spotted, and credentials originated from it will be rejected by the service mediator. The identity validation prevents an adversary from using the digital identity of the individual without knowing the password associated with that identity. The validation also prevents adversaries from using the secondary identities of a user, as they are useless without the completion of the identity authentication process. Indeed, the authentication process requires knowledge established only by the successful execution of the identity validation procedure. Finally, the service login process allows the service mediator to certify the legitimacy of the user before granting to him access to service functionality. Keep in mind that many aspects like impersonation and
eavesdropping are eliminated with the use of public key cryptography and cryptography in general.

Before we continue we would like to distinguish the attacks in two categories: passive and active. Passive are the attacks where the component in question is compromised, but only the information available can be exploited. Active attacks are much more demanding but more powerful, since the adversary can impersonate the instance of the component attacked, providing valid responses to incoming requests. The benefits of an active attack increase as the duration of the attack increases, as more requests will be handled, resulting in greater information exposure to the adversary. The problem is that an attack is the exception to normal component execution, so performing them for long periods is somehow impractical.

We will examine these statements in every scenario independently. The following table [Table 8] summarises the various attack patterns we will examine and a brief description on how our infrastructure behaves.

<table>
<thead>
<tr>
<th>Attacks</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Provider</td>
<td>Good behavior, since it results in same information exposure as experiences in today infrastructure</td>
</tr>
</tbody>
</table>
| AAC              | • Active: Affects only service providers, as they accept dummy digital identities  
                   • Passive: No information gain                                           |
| IIA              | • Active: Impersonate the users that where intercepted during the attack   
                   • Passive: No exploitable information gain                                |
| End-entity       | • Active: Full identity information exposure                               
                   • Passive: The mapping between secondary identities and the services being used is exposed |
| Man-in-the-middle| Our approach avoids this type of attack                                      |
| Combined to a service provider and IIA | Only profile agglomeration |

Table 8: Attack scenarios and a brief description on the outcome of the attack

5.1.1 Attack to a service provider

In this type of attack we suppose that the adversary managed to gain access to the information stored by the service provider. The attacker now holds all the information that was published by the individuals to the service and custom information attached to every profile by the service. The information each individual has presented includes his digital identities and possible some personal information about himself. As for the custom data, they are relative to service needs and cannot be used to identify the individual. To the attacker every record about a secondary identity can only be viewed as a different user, regardless if an individual has presented several secondary identities to that service. This is because secondary identities cannot be correlated without the knowledge of the identity key.
Furthermore, there is no (computationally feasible) way for the attacker to inference the
digital identity that was used to create this secondary identity. This ensures that no
information is released regarding the individual’s real-world identity, since his profile to the
service cannot be linked to him. If we imagine the same attack in a system that does not
used our infrastructure the results would be the same, implying that our infrastructure
performs well to this type of attacks.

We will now assume that the attacker also managed to play the role of the service
provider, for a certain period of time (active attack). During this period every individual that
connected to the infiltrated service mediator gave the adversary the advantage of knowing
the IIA that was used to create his digital identity. This information was made available
through the execution of the service login procedure, where the individual is required to
present the IIA that must be contacted in order to validate his secondary identity. The
authority used for validation is also the one that issued the individual’s digital identity. This
exposure does not disclose any information about the identity of the individuals, but reveals
to the adversary the IIAs that are responsible for this set of users. We will discuss this
information gain later, where combined attacks are described.

5.1.2 Attack to an AAC

A passive attack to an AAC would reveal no valuable information to the adversary, as
the child authorities of every AAC are publicly available through the authority lookup interface
they provide. Only a successful active attack would allow the adversary to interfere with the
DIM hierarchy, altering its structure and exploiting his “achievement”. The attacker is able to
delete certain branches from the hierarchy or even add authorities to the hierarchy that are
under his control. By deleting a branch, the attacker denies service access to individuals
holding digital identities that were issued by IIAs belonging to that branch. This is because
the service mediator can no longer validate the legitimacy of these authorities, as the
traverse fails when passing through the attacked AAC. With the second action an adversary
can perform the insertion of a new IIA to the hierarchy. This dummy IIA will be under his
control, creating illegitimate digital identities that are considered valid by our infrastructure.
In practice, the service mediator is misinformed by the attacked AAC that the dummy IIA is
part of the hierarchy. This way, the digital identities created are considered legal and access
is allowed to the functionality provided by services. The result is that the attacker can abuse
the services that trust the DIM hierarchy causing many problems. And all this are completely
anonymous and untraceable, as the digital identities are fake, leaving no evidence about the
adversary. We mentioned the addition of a single authority, but the adversary is freely to
create complete branches containing both AACs and IIAs, and even imitate the structure that
preexisted. Once again all these authorities are under his control vouching for authorities and identities as needed.

As we can see this type of attack does not expose information about individuals. In the case of node deletion a group of individuals experience inconvenience, as access to services is denied. This type of attack only affects service providers, since they provide access to dummy users that can misbehave leaving no trace information. Looking again to the type of the attack we see that it can only last for a small period of time, as it will be spotted and terminated. Moreover, the digital identities created will be useless when the hierarchy resumes its legitimate execution structure. One very useful solution would be to create special predator software components that continuous traverse the hierarchy seeking for changes in the structure and reporting them, in order to take immediate countermeasures. Again, there are no drawbacks for our proposed infrastructure related to this type of attack, as current service providers face the problem of providing access to dummy users.

5.1.3 Attack to an IIA

Let us first consider a passive attack, where the adversary manages to retrieve all the information held by the IIA for the individuals appointed responsible. The most valuable parts of the information received include the master identifier, the identity key and the session key of every user. The rest of the information is either useless or in a non-exploitable form. For example the password is not stored in plain text, but is first hashed and the result is stored to the database. The master identifier and the identity key can help the attacker create secondary identities the same way that individual do. This also allows the attacker to present secondary identities that can be validated to match the master identifier of the individual, during the execution of an identity validation procedure. But to be able to use these secondary identities, the adversary must hold a valid session key. The session key will allow him to complete the service login protocol and gain access to services, impersonating legitimate individuals. We assumed that the information available to the attacker includes the session keys of every individual, but the question is if they are valid. In deed, the validity of these keys lasts for a limited period of time and for that period only the adversary is able to illegally gain access to service providers. The key problem the attacker has to face is the absence of knowledge about the secondary identities the individuals have created and to what services they where used. This prevents the attacker from gaining access to accounts created by individuals and confines him in creating new accounts simply abusing the stolen digital identities. Once this period is over the information available to the adversary is useless. Summarizing, we would like to mention that the privacy of individuals was not affected and the gain of the adversary was a short period of identity misuse and service abuse.
We will now examine the details of an active attack to an IIA. During this attack the adversary is supposed to mimic the functionality provided by the infiltrated IIA and is able to respond to incoming requests. This way the attacker could easily gain access to information about individuals and authenticate himself to every service provider he wishes. When the attacker assumes control of the IIA he will get hold of the information we discussed previously in the passive attack. The accessory information will be received when an individual uses one of the available interfaces, which are under the control of the adversary. For example if an individual tries to execute an identity validation process, then the password of that individual will fall in the adversary's hands. In addition if an identity authentication procedure is executed by a service mediator, then both the secondary identity of the individual and the service being used at are exposed to the adversary. Utilizing this information the adversary can now connect to services and access accounts created by the individual as being his own. This is the first type of attack, in our discussion, that manages to violate the individual's privacy and breach his security. At the same time the adversary is able to create dummy digital identities and use them to access service functionality anonymously. If we examine closer the attack we will see that the information gained is proportionally to the period of time the adversary remains in control of the IIA. In fact, a bigger time span is translated in more users connecting to the IIA, and as a consequence in more information exposure to the adversary. The only problem of the adversary is that he will be quickly spotted and terminated, seizing communication between him and the individuals. Once again the damage caused by the adversary is limited by the way information is scattered among the components and the time available to the adversary to perform the attack.

We avoided mentioning the private user information held by the IIA for every digital identity created. This information is encrypted using a secret key known only to the IIA. The key itself is kept encrypted using a password to prevent unauthorized access. The key is decrypted and revealed only when a digital identity is created or legal issues arise and the user information must be retrieved. This way, even a successful breach to the IIA will not expose the key and thus the private user information.

5.1.4 Attack to an end-entity middleware

In the case of a passive attack, the adversary manages to receive the information about the individual that is held by the end-entity middleware. The information includes the secondary identities of the individual, the mapping between secondary identities and services and finally his digital identity. The mapping reveals to the adversary the services the individual has accessed and the secondary identities used in each service, while the digital identity reveals the IIA that issued this digital identity and the master identifier of the individual holding it. The first remark is that the adversary cannot access the accounts
created by the individual to services. To achieve this, the adversary needs to execute the identity validation procedure, namely he needs the password of the individual. Nevertheless, the adversary holds a complete list of services and the secondary identities used in every service. This information is very sensitive as it can be exchanged with services resulting in profile agglomeration and violation of user's privacy. Our example mapping in [Figure 50] shows that the first two secondary identities are presented as different users to Service #1. If the adversary revealed his knowledge on the mapping, the unique holder would be revealed to the service provider.

To prevent the adversary from gaining this information through a passive attack, we could apply encryption techniques and make useless the data stored by the end-entity middleware. Only the running instance of the end-entity middleware would hold the key to decipher the information and make use it. By securing the mapping scheme, a passive attack to an individual reveals only his digital identity, which is useless to the adversary.

![Figure 50: Example mapping of services and secondary identities](image)

To fully attack the individual the adversary must perform an active attack and gain control of the individual's computer system. This way he is able to receive the mapping information, along with the password of the individual. Now the attacker has all the information he needs in order to completely impersonate his victim. Therefore he can copy the mapping information of secondary identities to his own end-entity middleware and perform the identity validation procedure using the stolen password. From that point on the attacker is free to use any of the accounts of the legitimate user, resulting in total security exposure and violation of the individual's identity and personal information. We conclude that this is the only feasible attack to our system that is not limited by information or time.

We claim that there is no way to prevent such an attack. Namely, stop an adversary that can get control of an individual's computer system from stealing the individual's identity. This argument is irrelevant from the infrastructure deployed or how information is scattered between the components of the system. We will give two examples in order to support our claim. The first regards the current situation where no single sign-on system exists. When an individual wishes to login to a personal account he connects to that service and presents his...
credentials. This information will be logged by the adversary, and then used to access the users account. The exposure in this situation is per service resulting in less privileged information from reaching the attacker. The second example assumes the existence of a federated identity system, where the identity information of individuals is kept to their identity providers. Again, the user must present some credentials to his identity provider in order to activate his personal account. This information will also be logged by the adversary, resulting in full identity information exposure.

The examples saw us that there is no way to prevent identity theft, if the individual’s computer system is compromised. The only difference is that in single sign-on systems the adversary gains credentials that have global significance, when compared with the exposure of local credentials in the first example. Only if we avoid the use of passwords we can surely protect the digital identity of the individual. But this requires the authentication process to rely on something that the users are, i.e. implants, biometric readers. In [Figure 51] we see an example usage of an implant that authenticates the user. The implant support cryptographic functionality in order to decipher and encipher messages and has adequate storage capacity to be able to store secret keys. We assume that during the creation of the digital identity a secret key was shared between the IIA and the user. The user uploaded his unique copy of this secret key to his implant. This way the key can be used by the IIA to encrypt a challenge message that only the implant can decrypt and provide response. This way the IIA is certain that the legitimate user is behind the end-entity middleware, granting him access and completing the single sign-on procedure.
Figure 51: Sequence of interaction to perform authentication based on an implant

Concluding this section we would like to mention that an attack to the end-entity middleware, either passive or active, has the inherent disadvantage of compromising the digital identity of a single individual. In other words, the effort that the adversary must devote, in order to breach the computer system of an individual, does not counterbalance the gain from the identity theft.

5.1.5 Combined attack to a service provider and an IIA

In this section we will examine a combined attack to a service provider and an IIA. We assume that we have managed to retrieve the information held by an IIA and performed our short term attack. Now we want to investigate if there is anything else we can gain from this list of credentials. This drives us in attacking a service provider, retrieving the list of the secondary identities available and their corresponding profiles. The final step is to perform the identity validation process for every secondary identity. The validation process includes the decipherment of the secondary identity, using the identity key, and the comparison of the output with the master identifier. If the process is successful then a semantic link is established between them, as both belong to the same individual. If it fails then the secondary identity belongs to another individual, but we are not able to determine if his digital identity was created by a second IIA or we must keep testing the remaining
credentials in our list. The only gain of the adversary would be the discovery of two or more secondary identities that belong to the same individual. As we can see the combination of information retrieved by the IIA and the service provider reveals small parts of the mapping scheme held by the individual.

The only problem is the effort needed to attain this extra information. In fact, if we consider having M credentials and N secondary identities an exhaustive search would require MxN validations. The number of successful hits depends on the volume of available IIAs, the popularity of the service and the popularity of the IIA we attacked. The popularity translates to the likelihood of a certain user in using both the IIA and the service provider in question. While at first an attacker could think of breaching very popular services and IIAs, he will soon realize the infeasibility of performing validation on the vast amount of available information. As we can see the computational effort and the small ratio of matches outweighs information gain making this type of attack impractical.

5.1.6 Man-in-the-middle attack

The last attack we are aware of is the man-in-the-middle attack. In this attack a malicious service provider tries to use the individual's credentials in order to gain access to other services, impersonating him. In the next figure we see how a malicious service can repeat messages back and forth to the end-entity middleware and the legitimate service, gaining access to the legitimate service. The malicious service is presented to the individual like an ordinary service, providing useful functionality. The problem is that it uses the individual's credentials in order to gain access to other services without the user's consent. The legitimate service mediator cannot separate the end-entity middleware from the malicious service since the latter provides the adequate responses. The solution is to inform the individual about the presence of the legitimate service provider, in the group of transacting parties. For this reason the fourth message contains the service provider as seen by the viewpoint of the IIA. We have mentioned in previous chapters that the information contained in the fourth message is encrypted and only the end-entity middleware can access it. This way the end-entity middleware will be informed about the presence of another service provider, and the information will not be altered by intermediate nodes. From there the procedure is straightforward, as the individual will realize that he is connected in a different service provider from the one seen by the IIA. The result will be to stop communication and alert the user about the event. The fake message repeats by the malicious service provider are the dashed lines in the figure.
5.2 Measurements

As we have seen a number of transactions occur between the components of our system. In this section we will focus on the additional overhead imposed by our approach to the end-users and the service providers. We let aside the overhead of deploying and maintaining the DIM framework since it is beyond the scope of our work. Moreover we will exclude from our discussion the demonstration protocols that where created simply to facilitate the development process, meaning the authority lookup and identity creation protocols. These protocols are part of the overhead of maintaining the DIM framework and are again beyond our scope. Finally we will exclude from our measurements the fixed cost of establishing a secure communications channel using strong cryptography practices. This cost is fixed with every communication channel establishment and can be added later on. In contrast we will examine the additional messages the end-entity middleware and the service mediator exchange with the DIM framework and between them, in order to provide the functionality discussed.

The end-entity middleware starts by executing a one time single sign-on to the identity framework. The single sign-on is performed during the bootstrap of the component and retains information that is valid throughout its execution, or until a single sign-off is performed. This information can be used to perform service logins, using secondary identities, without negotiating again with the IIA. In fact, the identity validation protocol adds a start-up overhead for establishing a mutual secret key that will be used to validate the end-entity to subsequent service logins (for details refer to section 3.2.2). The second protocol, the end-entity middleware participates in, is the service login. This protocol is executed every time the user wishes to gain access to an online service. In fact this service is a replacement
to current service login procedures, and provides a transparent solution to validate end-users. The overhead added is both the use of strong cryptography, to establish a secure virtual channel for communication, and the additional four messages to verify the legitimacy of the user (section 3.5.4). The overhead is again small, as only four additional messages are needed; and strong cryptography costs will be ignored for the time being. If we look closer at the messages exchanged, we will realise that two of them are the replacement of sending specific credentials to login to the service, plus the session information upon success (messages 4 and 6 respectively in [Figure 14]). This means that in fact the two messages in question where needed prior to our proposed infrastructure, so they can be excluded from the overhead imposed by our approach.

The second component that we will examine is the service mediator, which accepts service login requests and makes use of the authority validation and identity authentication interfaces in the DIM framework. We covered the service login protocol in the previous paragraph and estimated that it introduced the overhead of exchanging two messages. The next protocol we will examine is the identity authentication, which is a replacement of the internal authentication mechanism used by services to validate end-users. The overhead it introduces not only includes the establishment of a secure communication channel, with the IIA, but also the exchange of messages to perform the authentication (section 3.5.5). This whole procedure is a result of outsourcing user authentication; and offspring’s an additional delay for every first attempt of a user to access the service. We finally examine the authority validation protocol, which is used to verify that a certain IIA or AAC belongs to the hierarchy of legitimate authorities (section 3.5.6). The additional overhead imposed by the protocol includes the establishment of a secure communication channel and two additional messages to verify the legitimacy of the presented authority. Moreover the protocol is not executed only once, but depending on the state of the service mediator can vary from zero times to the maximum length of the DIM hierarchy tree (maximum length minus one to be strict). But with the internal cache maintained by the service mediator this cost can be reduced. For more details on the protocol internals please refer to section 3.5.6.

Concluding the estimation of the overhead our approach has, when compared to the way current systems work, we see that the real burden is for the service provider. In fact the end-user only has to perform a negligible single sign-on procedure and an additional two messages per service login. In contrast the service mediator has to authenticate the digital identity of the user and, depending on the mediator’s internal state, validate a number of authorities as being legitimate [Table 9]. But since the internal state of the service mediator will be refreshed by the newly visited authorities, the usage of the authority validation protocol will be reduced; removing the overhead that it imposes. The only constant overhead is the two messages from performing the user login procedure and the cost of the identity authentication protocol.
Table 9: Overhead added by our approach, as compared to current solutions

<table>
<thead>
<tr>
<th>Protocols</th>
<th>End-entity middleware</th>
<th>Service mediator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identity validation</td>
<td>Per component start-up</td>
<td>-</td>
</tr>
<tr>
<td>Service login</td>
<td>Two messages</td>
<td>Two messages</td>
</tr>
<tr>
<td>Identity authentication</td>
<td>-</td>
<td>Per session</td>
</tr>
<tr>
<td>Authority validation</td>
<td>-</td>
<td>Per session (multiplied by zero to the length of the DIM hierarchy tree times)</td>
</tr>
</tbody>
</table>

Until now, we have hidden that the user application and the service core functionality experience a small additional overhead, as both components have to make use of the end-entity middleware and the service mediator respectively. So instead of making direct connections, their requests are diverted, to transparently make use of the functionality provided by the DIM framework. In fact all requests go through the end-entity middleware and the service mediator respectively, that act as proxies. But this overhead is relatively small, compared to the overhead we just discussed. In addition this type of overhead can be avoided by incorporating the functionality provided in the core application, eliminating the cost of diverting the communication needs.

At the beginning of this section we chose to ignore any overhead added by the use of cryptography and the start-up overhead of establishing a secure communications channel. The true reason for ignoring this type of overhead was that our main focus in this thesis is security, which cannot be achieved without communicating through a secure manner. This means that the cost mentioned is a necessary evil, being paid by current critical online services. We see that any digital identity infrastructure is a critical application, to avoid exposing the privacy of individuals. With this in mind we argue that the cost of communicating securely is not an overhead added by our solution, but the sole way to deploy such an infrastructure; hence we choose to ignore it.

5.3 Potential applications

We have built our identity management infrastructure having in mind a global scale deployment of the DIM framework, vouching for digital identities that are mapped to real-world entities. In fact, we saw the digital identity as the analogous of an identity card for the digital world. The vision of a full scale development puzzled us on the security aspects and drove us in a design to prevent mass identity exposure. This is why we believe that our system is ready to be used as a single sign-on framework with global scope. We believe that current Certificate Authorities ([41], [42], [43]) can assume the role of our framework authorities (either AAC or IIA), and issue digital identities for end-users. Of course proper controls and identity verification policies have to be applied in order to avoid forgery. In this
scenario the only small change has to be the Global AAC, which can be replaced by a list of known authorities that can spawn different DIM trees that can be verified individually. This example resembles the way certificates work for Web sites, e-mail addresses and software packages. For example the owner of a domain name can request from a Certificate Authority to issue a certificate that will be used to identify the legitimacy of the web site and establish a secure communications channel. In our case the Certificate Authority will issue a digital identity for a user that will be presented to service providers and automatically gain access to service functionality.

While we where focusing on a global scale deployment, we show that our infrastructure can operate inside organizations with a large number of users and intra-domain services. Our identity management infrastructure can act as a replacement to specific credentials and custom authentication methods, used by each legacy service in the organization. This way every department in the organization will have its own IIA and issue digital identities to its users. The single Global AAC will be operated by a central administration office and its obligation includes the definition of the hierarchy tree. Finally, depending on service needs, it can allow any user in the organization to connect or have a predefined list of secondary identities that have access. Have in mind that services can exist on any department of the organization, but the DIM hierarchy makes possible for users from one department to access services on another. This way no second set of credentials are needed, and disabling the digital identity of a user prevents him from using any of his secondary identities. Our infrastructure not only simplifies administration costs but also takes the role of a directory service [44], which commonly exists in organizations and allows users to transparently access remote resources. The next step for organizations will be to unify their identity management infrastructures. This means that two co-operating organizations can verify as legitimate authorities and digital identities that originate from one another. In fact the only action that needs to be taken from both organizations is to consider as legitimate the two Global AACs.

We acknowledge that deploying our infrastructure is a demanding task and requires some changes to legacy systems. On the other hand we believe that through this chapter we have proven the hard guaranties our approach offers and the small amount of overhead it introduces to end-users and service providers. This gives as an optimistic scope to our proposed solution, and in the end the guarantees of our infrastructure will overcome the deployment difficulties.
6 Conclusions and future work

We will conclude our thesis providing a summary of the problem user's face when accessing online resources and the solution we have proposed. Furthermore, we will provide directions for future work that address both open problems in the field and optimizations to the current implementation.

6.1 Summary

The vast volume of online services has given rise to the need for deployment of more sophisticated identity management methods. Today, users are responsible for holding multiple credentials with local significance. The credentials are issued by each service independently, in order to authenticate the user and provide a personalized experience. Every user is responsible for maintaining his list of credentials, using a different login-password combination for each need and choosing hard to guess passwords for each account. But most users find the situation inconvenient, using the same credentials in multiple services or choosing very poor and easy to remember passwords. The remaining users following the guidelines tend to forget their passwords, frequently applying for password resets. The problem is expected to get worse, as the number of online services will grow requiring more credentials to be held by end-users.

At the beginning, we described various implementations and proposed standards attempting to provide a solution to the problem. The most simple but instantly applicable solution is the use of credentials repositories. These repositories hold locally the credentials of a user and use them automatically when needed. The second approach, presented a centralized solution, where a single authority is in charge of handling and vouching digital identities for all Internet users. Finally, the remaining solutions propose a federated identity management approach. Federated identity management fails to create an open and global single sign-on system, as every service must be aware and trust a large number of identity providers.

Subsequently, we described a method for identity management that is able to address both problems of anonymity and accountability, in a global scale. Our digital identity management system gives its users the freedom to use their digital identity in any way they want. This means that multiple secondary identities can be created and used in various services. On the other hand, services are guaranteed that in case of account abuse the anonymous end-user will be revealed. The service will report the secondary identity in order to extract the master identifier and trace the physical person holding it. The functionality we described is provided by the Digital Identity Management (DIM) framework. The framework is a tree of authorities, with the leave nodes acting as identity providers. The remaining
authorities are the intermediate nodes in order to formulate the tree. The tree structure is used both to prevent malicious nodes from claiming that they are legitimate and to break down the administration burden of every identity provider.

Another important aspect of our approach is that it preserves the privacy of its users. The user is in full control of how his identity information is used and published in services. Setting the user as the sole holder of his identity information also prevents the existence of a single authority, where identity information for numerous users is held. The elimination of such points prevents attackers from gaining mass identity information with a single breach, making attacks to our systems less tempting. The final privacy-enhanced property of our system is that it allows individuals to be authenticated by services without requiring them to present personal information. Today, most services request personal information from their users, in order to identify the person behind the account. Our approach eliminates such tactics, as the service is assured about the physical person from his digital identity.

Deploying the presented infrastructure raises questions about the security quality offered. To facilitate this, a series of attack patterns were considered and studied. We started by assuming that an attacker could gain the information held by an entity and moved to allow him to imitate the functionality of the entity. Afterwards, we assumed various combined attacks and presented the only one that provided an additional information gain for the adversary. Our security model concluded with the man-in-the-middle-attack, proving that our system was designed to spot and avoid such attacks. We analyzed the results and attributes of every attack and found that our system behaves better than related work, while preventing mass exposure of information from an attack to a single entity.

6.2 Future work

The current state of our IMI provides a complete solution in the field of identity management systems. However, we thought of a number of extensions that can be made in order to fine tune and enhance its performance. In this direction we foresee the incorporation of available standards, fine tune the proposed protocols (the information being cached and how it gets invalidated) and survey identity theft and single sign-off scenarios.

The first direction for future work would be to avoid having a single hierarchy for the management of digital identities. The single root node poses threats for the acceptance of a global scale identity management infrastructure. Instead, several independent hierarchies should be supporter with every hierarchy applying its own specific policies on how authorities are included, or even how digital credentials are supplied (the first potential application presented in section 5.3). We imagine a deployment analogous to the way digital CAs work, where autonomous root CAs define their own practices on how digital certificates are issued and security policies applied. Then, it is up to the service providers to establish relationships
with each autonomous hierarchy, trusting the root nodes and of course the digital identities their IIA issue. We believe that the relationships will be self-regulated by the trustworthiness and accuracy of credentials issued.

Our next concern focuses on the heavy load IIAs experience from the continuous identity authentication requests. In practice, every request from an individual to access a service must be authenticated by the IIA that issued his digital identity. This may cause serious problems and performance degradation to IIAs. We could overcome this problem if individuals associated a different public key with every secondary identity. This public key would be handed to the service mediator upon the first service login request. The service mediator would use the identity validation procedure to validate the user and store locally the public key, along with the secondary identity. Subsequent request to login to the service would exploit the public key, preventing the service mediator from initiating the identity validation procedure. Periodically the service should re-examine the validity of the individual to assure his current digital identity status. This approach would greatly benefit IIAs, sacrificing end-entity computational cycles. An open issue is the blind time the service mediator experiences between revalidating the end-users identity status. In fact the identity may have been revoked, but the user is allowed to access the service functionality.

The service mediators also experience a blind time period when caching information about known authorities, in order to avoid re-executing the authority validation protocol. During that period an AAC can be removed from the hierarchy, but remain trusted to numerous service providers. One solution to avoid both blind time periods is for every IIA and AAC to hold a list of interested parties. For example an AAC would hold a list of service mediators that recently executed the authority validation protocol. This way if the AAC becomes aware about a change in its direct children authorities it will notify the service mediators in its list. This way the cache information held by the service mediator will be considered stale and the authority validation protocol would have to be re-executed. Once the cache information about an AAC is considered stale, all descendant authorities (hierarchy sub tree) will also be removed from the mediator’s cache. Resembling this scheme the IIA would hold a list of service mediators that have executed the identity authentication protocol. This way when a digital identity has been revoked the interested service mediators will be notified about the change. Both lists of interested parties affect the amount of information held by the authorities in the DIM hierarchy, evidence that may be exploited when attacking our infrastructure.

The most demanding task performed by the DIM framework is the policy on how authorities enter and remain in the hierarchy. The continuous burden of an AAC is how to ensure the “good behaviour” of its children. To facilitate this we already mentioned the development of a custom software component the traverses the hierarchy in order to verify the integrity of the structure. Another solution is to include the notion of “trust” among
authorities and among services and identity providers. This belief layer would facilitate authorities as the structure would self-regulate, dropping out malicious or faulty authorities. The inclusion of trust would also benefit services in spotting misbehaving IIAs and denying access to credentials originating from them.

Until now we have examined situations dealing with how our identity management infrastructure behaves on various attack patterns. We also gave guarantees about the absence of a single point, where mass identity exposure can be achieved. What we have not considered is how to resume proper execution after the theft of a digital identity or even after compromising one of the components of our system. When handling an identity theft situation two steps need to be performed: first the invalidation of the digital identity, and second the notification of the service mediators to deny access to the in question digital identity. Furthermore investigation is required on how to notify all transacting parties about a security breach. The main reason is to invalidate any illegal activities that where performed by the adversary, but also facilitate the gathering of information to estimate the magnitude of security exposure.

Another important direction for future work is to incorporate standards proposed by organizations such as OASIS, W3C and make use of well established Internet technologies. For example we will mention the XMLEncryption and XMLSignature, SAML and SSL socket connections. An open issue remains a standard for user digital identities, the information they include and how to fully achieve mobility.
Bibliography


   http://www.microsoft.com/net/services/passport/review_guide.asp


   http://www.oasis-open.org/committees/security/


[40] “XML Signature WG”. http://www.w3.org/Signature/