Identity Management for Hyper-Linked Entities in reTHINK

Gil Dias
Instituto Superior Técnico, Universidade de Lisboa
Lisbon, Portugal
Email: gil.dias@tecnico.ulisboa.pt

***Abstract*—The typical telecommunication operators are losing ground to new communication services that emerge from the rapid growth of the Internet. In order to take advantage of the wide network offered by the Internet, these operators must reinvent themselves. This is the main idea of the reTHINK project, a European project sponsored by the European Commission: to develop a new open and standardised platform for digital communication between user devices using modern web technologies.**

Identity management plays an important role in the reTHINK project and is the main focus of this Thesis. Having an identity associated to the user device is crucial to provide the means for users to communicate with each other. The goal of this work is to design and implement two key components to support identity management in reTHINK: the Identity Module and the Runtime Registry. The Identity Module is responsible for supporting the lifecycle of end users identities, devices, and other resources of the reTHINK ecosystem. It is also responsible for the establishment of secure communication channels for users to communicate with both confidentiality and integrity guarantees. To assist in the association of an identity to the user device, another module is developed to manage this information, the Runtime Registry. This module is mainly responsible to create that association and publish it in a public discovery service. The evaluations of these two modules suggest adequate results, managing to withstand heavy workload as demonstrated by the experimental tests and the ongoing usage by the reTHINK partners.

I. Introduction

In today's world, mobile communications are present worldwide and are brought to us as a centralised telecommunication service held by telephone service providers. These service providers are characterised for having absolute control over the communication environment, ranging from the subscriber identification module (SIM) card inside the user device, to the telephone infrastructure.

The introduction of Internet communication systems allows for novel approaches that can operate Worldwide without obstacles, contrary to what happens with the Telephone communication services, where physical barriers exist, such as country borders. Although some interoperability exists between these Telephone companies, this type of communication starts to become not only expensive but also outdated, comparatively to the growth of the Internet. The telecommunication providers are struggling to compete against the agility and innovation of web based services. Instant messaging services and Voice over IP (VoIP) via Internet has revolutionised the way people can communicate dethroning the business models of telecommunication services. With a need to adapt communications to modern technologies, the reTHINK project, a new communication framework, arises to introduce a new concept of a Web-centric P2P Service Architecture.

The reTHINK aims to provide a framework, to run in the user devices, that introduces interoperability among Communication Service Providers (CSPs) and enables the communications between clients using services from different CSPs. It allows CSPs to create services compatible with reTHINK that can communicate with other services from other CSPs. For example, the reTHINK framework is foreseen to allow a communication service created by Skype, reTHINK compatible, to communicate with a communication service created by Google, also compatible with reTHINK. The framework does not restrict the type of information that can be conveyed, which can be voice, text or file sharing, for example. If two services from different CSPs have the same capabilities on the type it supports, then is possible to create a communication between those two services.

To allow devices to be identifiable, a user identity is required. The standard telecommunication providers make use of the SIM cards to enable the user device to be identifiable and discoverable in the mobile phone network. With the use of the Internet as the platform for the operations, it is required to use other methods in order to allow for the users devices to be identifiable. Given this, identity management features an important component in the project, and the larger the population using the service, the more impact it will have. Given an individual identity, it is crucial to provide a trustful management of it, to allow a proper functioning of the architecture.

Considering the current state of traditional communications, it is easy to observe that the association of a person to an identity, a phone number in this case, is rather basic and restricted, whereas the number associated is limited to a single SIM card. Although a communication between two users using the telecommunications' infrastructure can be classified as private, given the use of encryption, this

communication is not completely confidential, mainly because telephony services have access to both identities and infrastructures used for the communication, gathering all the mechanisms required to restore a conversation that occurred in the past. So with this problem in mind, reTHINK aims to give a clear distinction and separation of operational services on the entities providing the Identity, and communication services.

II. Background and Goals

A. reTHINK background

The reTHINK project introduces a new web service - Hyperlinked Entities (Hyperties) [1]. This Hyperty is a module of software provided by a CSP which enable which enables the creation of trusted relationships among these Hyperties and can be deployed in a wide variety of devices ranging from smartphones, laptops, cloud infrastructures, just to name a few. The Hyperty communication can be carried out in two ways, through a message communication, providing an asynchronous communication among the Hyperties instances, or by a stream communication, where media streams are established between Hyperties, granting an audio and video communication. The goal of reTHINK is to provide a trusted worldwide service communication, where different services providers can be dynamically created to better meet consumers' needs, and thus overcome the geographical implications that affect the telephone companies.

Starting with the user device, this component is responsible for running the reTHINK client side application, which is composed of one or more Hyperties instances from the CSPs and the Runtime Core. The Runtime Core is the component responsible for ensuring the correct functioning of the reTHINK application, as well as managing the Hyperties running on it. Following with the CSP, this system is comprised by several internal components, namely the: Domain Registry, Discovery, Message Bus and Catalogue. The Domain Registry has the responsibility to register publicly the Hyperties upon their installation in the user device, as well as the identity associated to it. The Message Bus is the component responsible for intercepting and forwarding messages to be exchanged between users. These messages are usually signalling messages used to initiate a call. The Discovery is the public interface that enables the search for users registered in their own Domain Registry, as well as users registered in others CSPs Domain Registries. The catalogue is a repository provided by the CSP that hosts the web oriented software to be used by user devices, for example, the Hyperty code to be installed in the user device.

To give an example of a complete operation of the reTHINK framework, it is demonstrated a use case where Alice initialises the reTHINK application from the start and then try to contact Bob, also running the reTHINK application. Figure 1 demonstrates the flow with the messages that need to be exchanged to achieve this objective.

When Alice runs the reTHINK application in her device, she is given the choice to install a Hyperty from several CSPs Hyperties list. Upon the choice of the desired Hyperty, on step 1 of Figure 1, she is prompted to obtain an identity from a preferred Identity Provider (IdP). The acquisition of this identity requires a user authentication towards the IdP. In step 2 and 3, after the Runtime Core receives the identity Assertion (idAssertion), it starts downloading and installing the Hyperty from the Communication Service Provider (CSP) A. Once the installation is done a Hyperty instance URL is generated, and in the step 4 the Runtime Core associates the initially obtained identity with the Hyperty instance URL and registers it in the Domain Registry of the Alice CSP A.

In steps 5 to 8, Alice tries to search for Bob by inserting the email address of Bob she knows, for example, in a search bar in the reTHINK application. The Runtime core then uses the Discovery service from the CSP A, but since Bob is not registered in that CSP, the Discovery service queries the Global Registry using the Bob's GUID to resolve the current domain to which the identity is registered. After being contacted, the domain registry from Bob's CSP, replies to Alice's device with Bob's Hyperty instance URL, for which he can be contacted.

Steps 9 to 12, occurs after Alice receives the address of Bob, representing the call initiation between Alice and Bob. The call initiated by Alice contains her idAssertion. The Runtime Core on Bob's device consequently requests to the IdP of Alice to verify the legitimacy of Alice identity. After receiving the response from the IdP chosen by Alice, Bob's Runtime Core initiates a mutual Authentication between Alice and Bob, to authenticate themselves mutually and to exchange the keys used to establish a secure communication. Finally, after the successful completion of the mutual authentication, Bob's device replies to the call offer with his idAssertion already verified by Alice and the call starts.

Most of these steps are transparent to the user, being the user only responsible to login into an IdP to obtain an identity, choose the Hyperty to install and search for another user with a known identity, in order to start a call.

B. Goals

To instantiate and execute the obtained CSPs communication services in an end user device, in order to provide a communication channel between two users, it is required to do the proper storage and identity management in the reTHINK framework. The goal of this thesis consists in the development of a front-end Identity Module component in the reTHINK framework which runs on an end-user device, providing the capabilities of identity acquisition, authentication, and access to resources using an identity selected by the user. It is required an appropriate management of identities and access tokens, to trustfully manage communication between user devices including the association between those identities and the user devices.

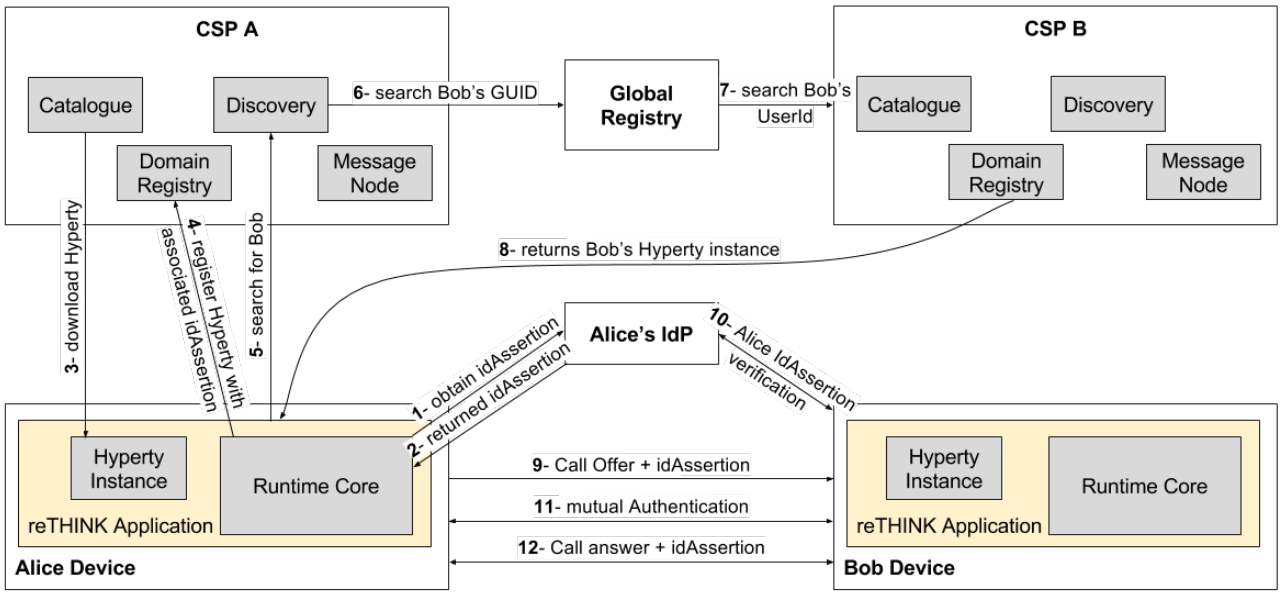


Figure 1. reTHINK communication between two users

The acquisition of an identity can be performed using different IdPs, for example Google or Microsoft. The Identity Module must allow users to register to a CSP service using the identity obtained through authentication mechanism. This module is intended to prevent CSPs from monitoring the communications performed by the user devices, providing privacy communications.

This Identity Module is to be developed in JavaScript, a high-level, dynamic programming Language supported by the majority of web browsers, in order to be supported the maximum number of devices. Since the information about the identities and other components are required to be stored, until the user ends the application, a Registry module also needs to be developed in order to support the needed storage mechanism. This Registry module will handle not only the registration of Identity Module, but all the components required to run in the end-user application referring to the reTHINK framework. The Registry will handle the allocation of URL addresses for all these components, and ensure synchronisation with the public Back-end Service Provider Registry.

So that a Hyperty instance running in the user's device can be associated with an identity, a module is required to store the registration of Hyperties, being that module the Runtime Registry. The Runtime Registry component has the responsibility to register and manage several runtime components, including Hyperties along with the respective sandbox provided by a Service Provider.

III. Architecture

The architecture of the system herein proposed consists of two major components in the reTHINK framework, namely the Identity Module and its Proxy sub-component and the Runtime Registry. These are two of the core components of the Runtime Core. Along with the remaining components developed by others partners in the reTHINK project, as depicted in Figure 2. The Message Bus and the Policy Engine components are worth mentioning and describing because they play an active role in the interaction with the components developed in this Thesis.

The Policy Engine has the role of enforcing user policies in the reTHINK framework. For example, if a user wants to block the communications with another user, he can do it by adding a rule to block that identity in the Policy Engine configuration page. For the Policy Engine to be effective, it needs to listen to all messages that go through the Message Bus. As such, both Policy Engine and Message Bus work together to intercept all messages and analyse each one, to check whether a given policy needs to to be applied.

The identity management comprises not only the action of managing the user's identity but also the acquisition of these identities. To do so, the standard OpenID Connect (OIDC) Protocol [2] is used to allow for users to get an identity from the identity provider, which will be managed by the Identity Module. This identity token, provided by the IdP, is managed by the Identity Module, and used to associate an identity with the Hyperties instantiated in the user's device.

OIDC is a popular authentication protocol used in the authentication of a large number of users in several services, for instance, Google and Microsoft. This protocol has some features of great relevance for the reTHINK project. On the one hand, it offers a field that can be used to send the user's public key, which is necessary for the generation of

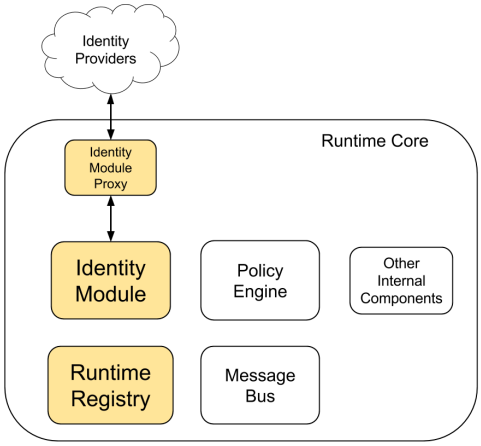


Figure 2. Runtime Core architecture

an identity token with the capabilities of a digital certificate, and consequently to have the IdP as a certification authority.

A. Mutual Authentication Protocol

The reTHINK architecture is designed to operate in a peer-to-peer architecture, and as a result there is no centralised service that proceeds to authenticate the users in reTHINK. Because of that, the identity assertions play a very important role in reTHINK, by enabling mutual authentication between users. Using IdP's that supports OpenID Connect, it is possible to request the IdP to assert a particular content on the request to authenticate the user. The identity token received after the user's successful authentication contains the user identity assertion and the content provided during the authentication request, also asserted. In reTHINK, the content to be asserted by the IdP is a public key specified by the Identity Module, later used to prove the user's identity to a third party. This way, the identity assertion provided by the IdP acts as a digital certificate, where the IdP plays the role of a Certification Authority (CA).

As such, whenever a user intends to initiate a communication with another user, the mutual authentication protocol is triggered so that users can authenticate each other mutually. In order for the mutual authentication to be successful, all the messages are required to have an identity assertion, which will work as a digital certificate. Therefore, to authenticate to a message, the sender identity assertion obtained through the IdPs is attached to the message, containing the user public key. To confirm that the public key actually corresponds to the claimed identity, the receiving user contacts the IdP to validate the content of the Identity assertion. With this public key, the receiver can validate the sender message digital signature and encrypt the response to the sender challenge to conclude and successfully authenticate the sender identity. For mutual authentication, the roles invert and the receiving users becomes the one who must prove his identity, using the same procedure. For user privacy assurance, the Identity Module may frequently request the generation of a new Identity assertions, each with different public keys.

Taking the mutual authentication process described above, the IdM performs this authentication process and along with it, generates the symmetric session keys to be used for the protection of the messages exchanged after the mutual authentication process. This is done in an identical manner as in the TLS protocol. By doing this, we provide the same procedures and the same security properties of the TLS, including the security assurance for the message integrity and confidentiality. The proposed solution used in rethink for authentication is also separated in the handshake and record phases. Some simplifications are introduced in the authentication protocol compared to TLS, mainly because there is no need to support some of the features. The negotiation steps of cryptographic methods in TLS are not taken in consideration since it is the Identity Module that defines the cryptographic methods to be used, and all devices running the reTHINK application will use the same version of the Identity Module. Additionally, compression will not be considered, since this protocol is already running on top another TLS communication (i.e. the communication of each client with the Message Node).

B. Identity Module

The Identity Module is the component responsible for handling all the user identities as well as ensuring confidentiality and integrity on the messages exchanged either in a direct communication between Hyperties or in chat groups where multiple Hyperties are connected.

The identity in the reTHINK framework is not fixed to a unique Identity Service Provider (IdP). It can be obtained from several IdP's, being the IdModule responsible for providing the user with the option to choose which IdP he prefers. To do so, a graphic user interface (GUI) component that communicates with the Identity Module is required in order to provide the best experience to the user. With this GUI component, the user is presented with the possibility to add new identities, select the preferred identity and remove identities previously registered, all this, in a simple and accessible interface.

For an identity token to be obtained by Identity Module from an IdP, a module needs to be acquired from the IdP, the IdP proxy. This proxy is the component responsible for the communication with the external IdP server. So, when an Identity Module asks the IdP for a user assertion, this request is made to the IdP proxy via the internal Message Bus. Then, the proxy communicates directly with the IdP requesting the user assertion. The decision to support proxies came up with the need to support multiple IdP's without the need to modify the identity module itself, being this IdP proxy downloaded and compiled on the fly. With a proxy, the Identity Module can be agnostic to the available IdP's. In order for the Identity Module supports the load of the

proxy dynamically, the proxy should be standardised, and for that, the WebRTC IdP proxy standard [3] was chosen. With this API well defined, any Identity Provider following the WebRTC standard for proxies can provide their proxy, compatible with reTHINK.

The communication between users is one of the major characteristic in reTHINK. As such, the authentication of each user takes a big role to ensure that no personification can occur. Every time a user starts a communication with another user, the process of mutual authentication, described above, is initiated by the Identity Module. This mutual authentication is not only useful for the authentication of the users, but is also essential for the exchange of the symmetric keys used in the established secure communications. The reTHINK framework provides native support for two types of communication, a direct communication between two users through their respective Hyperties and a group chat communication where the messages are exchanged between all participants in the group. The secure channel, for these communications, consists on the creation of a HMAC of the message followed by the encryption of the message, to ensure confidentiality and integrity, being these tasks, the responsibility of the Identity Module.

C. Runtime Registry

The Runtime Registry has the responsibility to register and unregister several internal components. Among these components, we can find Hyperties supplied by Service Providers, Protocol stubs (Protostubs), which implements the protocol to be used to communicate with the Service Providers and Data Objects, a component that allows to establish group communication among Hyperties. Besides the storage of those components, the Runtime Registry allows for internal components running in Runtime Core to request information about the registered components, such as, return all the Hyperties registered in the device.

The registration of Hyperties requires a bit more than just storing internally the component. As depicted in Figure 3, when a request is made to register an Hyperty, the Runtime Registry receives information regards the Hyperty and the respective sandbox with the executable code. Then a request for a user identity is made to the Identity Module, which promptly asks the user, to select or add a new identity using the GUI. When the identity is received, the Runtime Registry requests the public Domain Registry to generate a unique Hyperty URL and return it to the Runtime Registry. After receiving the unique URL, it is then associated with the Hyperty sandbox received in the Runtime Registry, allowing this Hyperty to be identifiable and addressable. This URL is important to allow the Hyperty sandbox to have a listener for that address. So, incoming messages to that specific URL are rerouted to the Hyperty sandbox. After the Hyperty has been associated to the Hyperty URL and an identity, the Runtime Registry sends a request to register an association of the Hyperty URL and the identity within the Domain Registry, to allow this association to be public. With this information, users can query the Domain Registry for the Hyperties registered with a given identity, receiving as result the Hyperty URL, which can be used to start a communication. After successfully registering the information in the Domain Registry, it returns a confirmation message, which is propagated until reaches the Runtime Registry. In the end, the Hyperty URL is returned, to whom made the request to register the Hyperty.

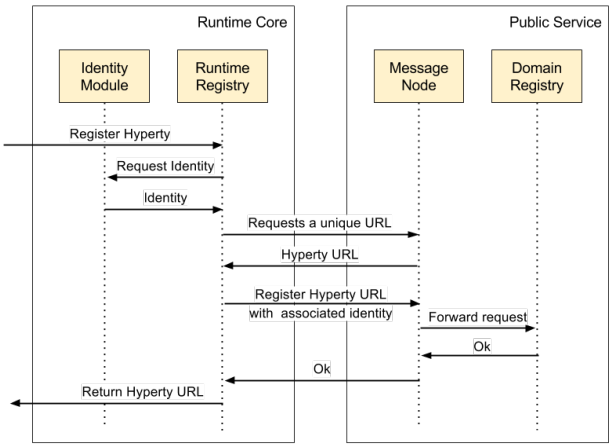


Figure 3. Hyperty registration

IV. Implementation

The implementation of all the modules was done in JavaScript, a high-level programming language, using the ECMAScript 6 which is the latest standardised language specification. The code was developed to run in a browser, with Chrome web browser chosen as the main platform for the development and test of the reTHINK framework.

A. Identity Module

The reTHINK project aims to bring interoperability among different service providers and for it to be successful, some components are required to be modular in order to support different implementations of the same component, from different service providers, to do so, it is essential that the IdP Proxy complies with the requirements, explained in the previous chapter. So, the Identity Module must provide the support for all IdP Proxies develop using the WebRTC IdP proxy[3] implementation.

Trustful communications between users can be assured by trigger the mutual authentication protocol. This mutual authentication protocol follows the message flow similar to the TLS protocol[4], in order to provide the same security properties. During this protocol, symmetric cryptographic keys are generated. These keys are useful for the secure

communication that occurs after authentication of users. Because several cryptographic methods are used by the Identity Module during the establishment of secure communications, a cryptographic library was created exposing an API, to ease the tasks.

All these features are provided in a well-defined API by the Identity Module, so it can be used by other components belonging in the Runtime Core. Since latency in the network is a constant presence, the main application should not stall, waiting for the response that may never come. Given this, most of the Identity Module methods use Promises, which are a useful tool for applications that make requests over the Internet. Promises work like callbacks and are intended to be used for asynchronous computations in order to allow the code to resume its execution, even if the method is still waiting for the response.

1) Identity acquisition: In order for the IdM to be able to manage identities, these need to be acquired in the first place. This is achieved by supporting and using the OpenID connect (OIDC) authentication protocol, as described in the previous chapter.

The OIDC protocol has the advantage of using REST calls to make requests, providing in return identity assertions in a JSON format, which is an asset for the reTHINK framework. This allows the IdM running in JavaScript to make HTTP requests and easily handle the received JSON token. Additionally, because OIDC offers a standardised API, all Identity Service Providers implementing the OIDC, will follow the same standard REST API for the user's requests.

2) Identity GUI: To allow users to easily select the identity they want to associate with a Hyperty, the identity GUI component was developed. This identity GUI allows the user to: add new identities, select from existing ones and remove previously added identities. This GUI was developed using the materialize framework[1], a CSS framework that combines both: CSS, HTML and JavaScript language into a easy to use framework

The identity GUI is available in two occasions: when the user clicks the configuration button and the identities button, and when the users register a Hyperty. The first one stays open until the user hits the exit button. In the second one, when the user selects a Hyperty to be registered, the identity GUI opens immediately waiting for the user to select a previously registered identity or to add a new one. When the user chooses an identity or registers a new one, the identity GUI closes and the Runtime Core proceeds with the Hyperty installation. The decision to show and close the identity GUI when registering a new Hyperty, came from the need to provide a better experience for the user. With this solution the user is not required to open and close the identity GUI whenever an identity is required when installing a Hyperty.

[1]http://materializecss.com/

3) IdP proxy: The requirement for the development of the IdP Proxy is to follow the WebRTC standard[3] for IdP Proxies. The IdP proxies created by the IdPs that follow the WebRTC standard are automatically compatible with reTHINK. This is possible because the WebRTC defines two methods for the IdP proxy implementation, namely the 'generateAssertion()' and 'validateAssertion()'.

The IdP proxy communication with the IdM is performed using the Message Bus, but for the IdP proxy to use the Message Bus, a small component need to be installed, namely the Protocol Stub. This IdP proxy Protocol Stub component is installed along with the IdP Proxy in the isolated sandbox and enables the use of the Message Bus.

For the reTHINK project, two IdP proxies were developed. One for the Google IdP and another for the Microsoft IdP, both using OpenID connect protocol.

4) Mutual Authentication: The mutual authentication protocol used in reTHINK is based on the TLS protocol. Because the TLS is typically a protocol for the transport layer, the mutual authentication protocol had to be develop from scratch, in order to be used in the application layer. During the protocol development, the TLS protocol was taken into consideration.

The mutual authentication can be triggered in two ways, either by sending a initial message to the other user, or by calling specifically the method to start the mutual authentication between two Hyperties. Usually this method is called when there is a request to subscribe a Data Object that enables group chat communications. So, when a the Data Object subscription triggered this method, the necessary keys for a secure communication are exchanged in the end of the mutual authentication protocol.

For asymmetric encryption and decryption, was used the RSA cryptosystem, more specifically the RSA-OAEP with SHA-256, was chosen with a key size of 2048 bits. The signature and validation methods uses the RSASSA-PKCS1-v1_5 with SHA-256 cryptosystem, also with a key size of 2048 bits.

5) Secure communication: The secure communications in reTHINK are established after a successful mutual authentication between two users IdMs. It supports two types of communication, a direct communication between users in this case a Hyperty to Hyperty communication, or a group chat communication where one Hyperty can communicate with many Hyperties subscribed in a data object.

To enforce the secure communications, two methods were created within the Identity Module, the 'secureSend(message)' and the 'secureReceive(message)'. The 'secureSend' method receives as input a message, creates a HMAC of the message, encrypts the message sensitive content, swaps the message value with the encrypted one and returns the message encrypted with the HMAC. The 'secureReceive' method does the exact opposite of 'secureSend' does. This method receives a protected message, decrypts it,

verifies the integrity and returns the original message, with the data in plain text.

For the symmetric encryption, was used the AES cryptosystem with Cipher Block Chaining (AES-CBC) using a key size of 256 bits. To create keyed-hashes and validate them, the HMAC SHA-256 cryptosystem was chosen, using a 256 bits key size.

B. Runtime Registry

The Runtime Registry is the component responsible for registering the multiple data from the several internal components of the reTHINK framework. To do so, it provides a specific API so other components from the Runtime Core can use it. All the API methods are implemented using the Promise API to turn them asynchronous. This is required for some methods since they have dependencies on others asynchronous requests.

Information within the Runtime Registry is not required to be persistently stored, being the responsibility on the persistence of another component of reTHINK. Given this, the information in the Runtime Registry can be stored in volatile data structures being the hash tables the chosen data structure. The hash table is a key value data storage, where the key can map to the respective value, using hash functions to compute a key that will be assigned to an array of buckets to where the information will be stored.

1) Discovery Library: One of the tasks of the Runtime Registry is to register the information regarding the association of identities to components registered locally, on a public service. To facilitate the search for these public associations by other components, the Discovery Library at the Runtime Core was created. The Discovery library is implemented using the JavaScript Class object, allowing it to be instantiated in any reTHINK component. It exposes the API for the Hyperties to use, to allow querying the Domain Registry for the required information.

V. Evaluation

This works presents the solution proposed to manage the users authentication towards the IdP and their mutual authentication, in order to ensure the establishment of the secure communication between users, within the reTHINK framework. In order, to evaluate the performance of the proposed and developed solution, several tests were performed to evaluate the performance cost to the reTHINK framework in several aspects.

This evaluation covers: the Identity Module and the Runtime Registry components. For this evaluation, 1000 samples were taken for each test and averages of these samples were considered. The tests were performed on a computer with the following characteristics: CPU with an intel core i5-3210M running at 2.5 Ghz, 8GB of memory RAM and the Ubuntu 14.04 LTS operating system.

A. Mutual Authentication

To evaluate possible extreme usage scenarios for the mutual authentication protocol, the time required to perform multiple the mutual authentication processes simultaneously was evaluated, as depicted in Figure 4. From the values obtained it is possible to observe that the values grow linearly and do not introduce a bottleneck to the reTHINK framework.

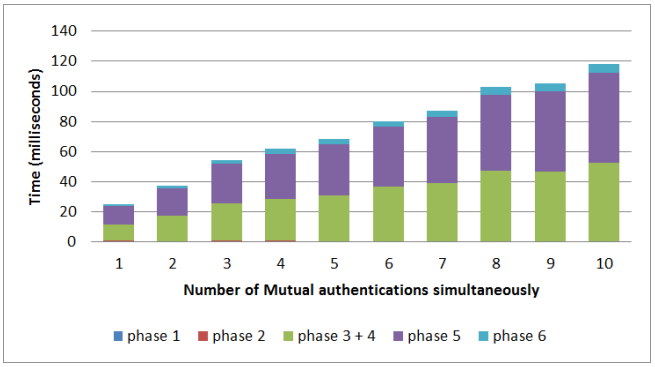


Figure 4. Chart representing the average times of several mutual authentications running simultaneously

The above evaluation does not contemplate the round trip time. Since a public Message Node is required, its time must also be taken into account. To evaluate the impact of this node in the reTHINK framework a Message Node was defined to run in a localhost environment. The Figure I illustrates the total time of the mutual authentication execution including the round trip time imposed by the messages exchanged itself.

Number of mutual authentications running simultaneously	Total mutual authentication execution time (ms)	Round trip time (ms)	Total time (ms)
1	24.94	23.20	48.15
2	37.54	28.40	65.93
3	54.51	39.09	93.60
4	61.71	39.32	101.03
5	68.14	40.23	108.37
6	80.30	44.74	125.04
7	87.33	46.50	133.83
8	102.76	55.13	157.89
9	105.47	55.45	160.92
10	118.26	59.32	177.57

Table I
TABLE REPRESENTING TIME OF THE MUTUAL AUTHENTICATION SEVERAL TIMES AN ITS ROUND TRIP TIME

From this analysis, it is possible to conclude that the mutual authentication, in an ideal communication network, presents values almost unnoticeable to the common user

(<200 milliseconds). Even in rare scenarios when 10 mutual authentication processes are executed. Since the execution time of the protocol is independent from the round trip time, the biggest bottleneck of this mutual authentication is in the communication time itself, where the communication latency can introduce a significant time to the conclusion of the mutual authentication process.

B. Secure communications

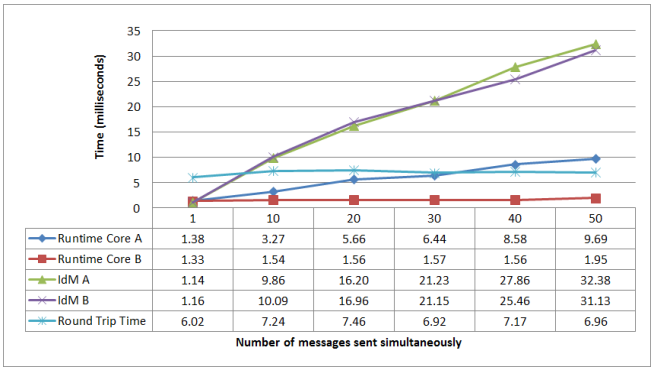
To perform the evaluations of the messages exchanged over the secure communication it was tested a scenario where multiple messages are send simultaneously from one Web Browser running an instance of the reTHINK application to another Web Browser also running instances of reTHINK, replicating the communication between two users. Both of reTHINK instances have a Hyperty running, representing a real case of two Hyperties communicating between them.

For each sent message the following operations were measured:

- **Runtime Core A**: Time it takes for the Alice's Runtime Core to process the message created by a Hyperty and send it, excluding the time taken by the IdM.
- **Runtime Core B**: Time it takes for the Bob's Runtime Core to receive the message, process it and delivery it to the Hyperty destination, excluding the time taken by the IdM.
- **IdM A**: Time Alice's IdM takes to encrypt and authenticate the message.
- **IdM B**: Time Bob's IdM takes to decrypt and to validate the received message.

To test the responsiveness of the IdM to several message transmission requests at the same time, a test was performed that send multiples messages simultaneously ranging from 1 up to 50 messages. Each message contains a 32 Bytes value to be handled by the IdM, simulating a conversation with the exchange of short messages between users. The results obtained for both tests are depicted in Figure II. These Figures depicts the average time in milliseconds that each component takes to send a single message when multiple messages are sent, including the round trip time. From these results it is visible that the A and B IdM delay increases with a linear proportion, for up to 50 small messages sent simultaneously. This linear increase is due to the fact that browsers do not possess concurrency, meaning they only use a single thread to run JavaScript code. Because the IdM methods are asynchronous, it is possible for the IdM method to loses its thread processing time to another method, only regarding the processor later on to conclude the task. The time the method is interrupted is accounted in the total time the method takes.

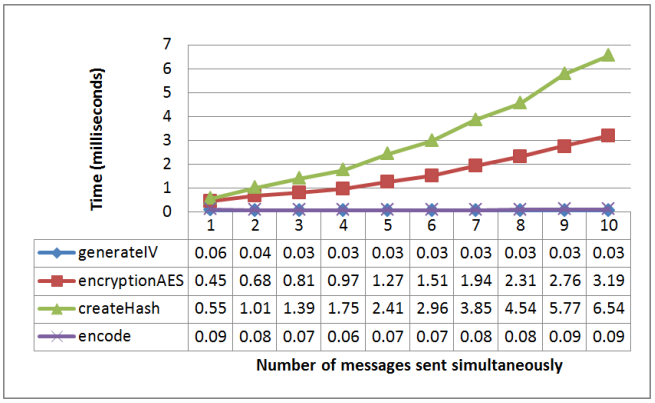
From this result it is possible to conclude that the performance of the IdM is only slightly affected when sending multiple messages at the same time.



	1	10	20	30	40	50
Runtime Core A	1.38	3.27	5.66	6.44	8.58	9.69
Runtime Core B	1.33	1.54	1.56	1.57	1.56	1.95
IdM A	1.14	9.86	16.20	21.23	27.86	32.38
IdM B	1.16	10.09	16.96	21.15	25.46	31.13
Round Trip Time	6.02	7.24	7.46	6.92	7.17	6.96

Table II
CHART SHOWING THE AVERAGE TIME EACH COMPONENT TAKES TO SEND EACH MESSAGE, BY SENDING MULTIPLE MESSAGES SIMULTANEOUSLY (1 TO 50 MESSAGES)

Making a deeper analysis of the IdM times, the IdM times were decomposed on the following aspects: initial vector (IV) generation, encryption using AES, message authentication and encoding, for the IdM A and the opposite, for the IdM B. The values depicted in Figure III and FigureIV suggest that when sending multiple messages simultaneously the values that tend to increase are related to the encryption and hash computation. The IV generation and the encode/decode steps are barely noticeable. It is also possible to observe that both A and B IdM take approximately the same to process a message, for the same number of messages sent simultaneously.



	1	2	3	4	5	6	7	8	9	10
generateIV	0.06	0.04	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03
encryptionAES	0.45	0.68	0.81	0.97	1.27	1.51	1.94	2.31	2.76	3.19
createHash	0.55	1.01	1.39	1.75	2.41	2.96	3.85	4.54	5.77	6.54
encode	0.09	0.08	0.07	0.06	0.07	0.07	0.08	0.08	0.09	0.09

Table III
CHART REPRESENTING THE TIME OF EACH STEP USED BY IDM A TO PROCESS A MESSAGE

To provide a overall conclusion on secure communication using messages, it is possible to conclude that the IdM methods do not increase exponentially, not even in the most extreme cases. This allow to conclude that the system is able to perform with adequate metrics, independently of the number of messages exchanged.

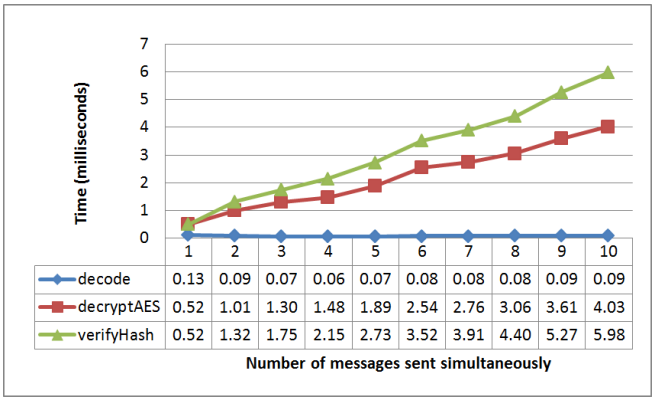


Table IV
CHART REPRESENTING THE TIME OF EACH STEP USED BY IDM B TO PROCESS A MESSAGE

C. Runtime Registry

The components that can be searched are and registered in the Runtime Registry are stored in hash tables. To evaluate its performance, the load times for the Protostub component in Runtime Registry was tested.

To test the search of the Protostub component stored in hash tables, the time it takes to search for one Protostub in a Runtime Registry with 1, 10, 100, 1000, 10 000 Protostubs registered was measured. The obtained results, depicted in Figure V, shows that the search times is practically the same independently of the number of Protostubs registered in the Runtime Registry. This results from the efficiency of the hash tables, which scales very well. The obtained tests

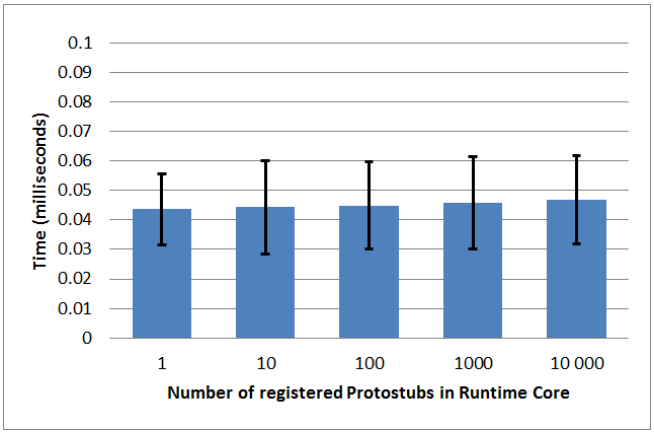


Table V
CHART REPRESENTING THE AVERAGE TIME FOR THE PROTOSTUB SEARCH

proved the capacity of the Identity Module to endure the tests with the heaviest load. For the Runtime Registry, tests were made to evaluate the search capacities of the registered components. The obtained results validate the decision to use hash tables, showing good performances for the search of components.

VI. Related Work

In our modern society, technology is ubiquitous, and transactions are evermore accomplished using digital technologies without the need to involve physical contact. An example of this situation can be observed in money transactions, where a few years ago if someone needed to make a bank transfer, it would require that person to move personally into a bank agency to order it, and in current days these money transfers can be performed using a smartphone. Since identification binds a person identity with the respective individual attributes [5], an authentication of identity is required. Given this, and since the majority of the current transactions are performed digitally, we need, a sometimes called, digital identity to prove who we are in remote communication. This concept of Identity comprises two important information security mechanisms, the authentication and authorisation [5]. In a short description, the authentication is an identification followed by verification.

It was studied the state-of-the-art on identities, authentication mechanisms and its management. The concept of identity tokens and several mechanisms to obtain this token were analysed. Were made a survey and a deep study on several authentication mechanisms, namely the BrowserID [6], the WebID-TLS [7], the OpenID connect [2], the Kerberos [8] and the smart card [9]. The transport layer protocol (TLS) [4] was also studied, with more focus on the mutual authentication phase.

VII. Conclusion

This paper presents and details the proposed solution for two important modules of the reTHINK framework: the Identity Module and the Runtime Registry. The Identity Module is the component in reTHINK responsible for user authentication, identity management and deployment of secure communications. The Runtime Registry is responsible for the management and storage of the Runtime Core components, and, more importantly, for associating an identity to the user's device.

The proposed and implemented solution introduces a new mutual authentication protocol, similar to the well-known TLS, using the received identity token as a digital certificate. The mutual authentication protocol is achieved using Identity Providers with the OpenID Connect protocol, which can be used to assert a given public key, sent in the identity request.

References

[1] reTHINK Consortium, "Global Identity and Reachability Framework for Interoperable P2P Communication Services," *19th International ICIN Conference*, 2016.

[2] N. Sakimura, "OpenID Connect Core 1.0, http://openid.net/specs/openid-connect-core-1_0.html, December 2015." [Online]. Available: http://openid.net/specs/openid-connect-core-1_0.html

[3] E. Rescorla, "Webrtc security architecture," *draft-ietf-rtcweb-security-arch-12*, June 2016.

[4] T. Dierks, "The transport layer security (tls) protocol version 1.2," *RFC 5246*, 2008.

[5] J. L. Camp, "Digital identity," *Ieee Technology And Society Magazine*, vol. 23, pp. 34–41, 2004.

[6] Mozilla, "Persona - Mozilla — MDN, https://developer.mozilla.org/en-US/Persona, December 2015." [Online]. Available: https://developer.mozilla.org/en-US/Persona

[7] H. Story, "WebID Specifications, http://www.w3.org/2005/Incubator/webid/spec/, December 2015." [Online]. Available: http://www.w3.org/2005/Incubator/webid/spec/

[8] B. C. Neuman and T. Ts'o, "Kerberos. An authentication service for computer networks," *IEEE Communications Magazine*, vol. 32, no. 9, pp. 33–38, 1994.

[9] U. Hansmann, M. S. Nicklous, T. Schäck, A. Schneider, and F. Seliger, *Smart card application development using Java*. Springer Science & Business Media, 2012.