# Local data storage : security and availability

Mayssa Jemel

ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

TELECOM ParisTech

2016-ENST-0053

EDITE - ED 130

# Doctorat ParisTech

# T H È S E

**pour obtenir le grade de docteur délivré par**

# TELECOM ParisTech

## Spécialité « Informatique et réseaux »

*présentée et soutenue publiquement par*

**Mayssa JEMEL**

le 29 septembre 2016

# Stockage des données locales : sécurité et disponibilité

Directeur de thèse : **Ahmed SERHROUCHNI**

**Jury**

| | |
|---|---|
| **M. Pascal LORENZ**, Professeur, Université de Haute Alsace Colmar | Rapporteur |
| **M. Yacine CHALLAL**, Maitre de conférences, Université de Technologie de Compiègne | Rapporteur |
| **M. Ken CHEN**, Professeur, Université Paris 13 | Examinateur |
| **M. Bernard COUSIN**, Professeur, Université de Rennes 1 | Examinateur |
| **M. Rida KHATOUN**, Maitre de conférences, Télécom ParisTech | Examinateur |
| **M. Guy PUJOLLE**, Professeur, Université Pierre et Marie Curie | Examinateur |
| **M. Hassan NOURA**, Professeur associé, Université Libanaise, faculté d'ingénieur | Examinateur |

**TELECOM ParisTech**
école de l'Institut Mines-Télécom - membre de ParisTech
46 rue Barrault 75013 Paris - (+33) 1 45 81 77 77 - www.telecom-paristech.fr

T H È S E

# Acknowledgements

It is with a great pleasure that I would like first of all to express my gratitude to all those who participated in the development of this work in best conditions.

I sincerely thank Mr Pascal LORENZ professor at the University of Haute-Alsace and Mr Yacine CHALLAL Associate Professor at the Higher National School of Computer Engineering of Algeria for reviewing the manuscripts and making helpful observations. I would also like to thank Mr Bernard COUSIN professor at the University of Rennes, Mr Guy PUJOLLE Professor at the University of Pierre et Marie Curie of Paris, Mr Hassan NOURA HDR at the University of Pierre et Marie Curie of Paris, Mr Ken CHEN Professor at the University of Paris 13 and Mr Rida KHATOUN associate professor at Télécom ParisTech for accepting to be part of my thesis juries.

I would like to express my deepest gratitude to my supervisor Ahmed Serhrouchni. I sincerely thank him for the continuous guidance and support. Without his encouragement, I would not be able to accomplish this work.

I couldn't complete those acknowledgments without expressing my sincere recognition and respect to all my friends and colleagues.

# Abstract

Due to technological advancements, people are constantly manipulating multiple connected and smart devices in their daily lives. Cross-device data management, therefore, remains the concern of several academic and industrial studies. The proposed frameworks are mainly based on proprietary solutions called private or closed solutions. This strategy has shown its deficiency on security issues, cost, developer support and customization. In recent years, however, the Web has faced a revolution in developing standardized solutions triggered by the significant improvements of HTML5. With this new version, innovative features and APIs are introduced to follow business and user requirements. The main purpose is to provide the web developer with a vendor-neutral language that enables the implementation of competing application with lower cost. These applications are related neither to the used devices nor to the installed software.

The main motivation of this PhD thesis is to migrate towards the adoption of standardized solutions to ensure secure and reliable cross-device data management in both the client and server side. There is already a proposed standardized Cloud Digital Safe on the server side storage that follows the AFNOR specification while there is no standardized solution yet on the client-side. This thesis is focused on two main areas: 1) the proposal of a standardized Client Digital Safe where user data are stored locally; and 2) the synchronization of these data between the Client and the Cloud Digital Safe and between the different user devices.

We contribute in this research area in three ways. First, we propose a Client Digital Safe based on HTML5 Local Storage APIs. We start by strengthening the security of these APIs to be used by our Client Digital Safe. Second, we propose an efficient synchronization protocol called SyncDS with minimum resource consumption that ensures the synchronization of user data between the Client and the Cloud Digital Safe. And finally, we address security concerns; in particular, the access control on data sharing following the Digital Safe requirements.

**Keywords**

Digital Safe, HTML APIs, Local Storage APIs, syncrhonization protocol, Hierarchical Hash Tree, WebSocket, CP-ABE, time based access control.

# Résumé

Le progrès technologique offre désormais de plus en plus aux utilisateurs divers équipements connectés et intelligents. En conséquence, la gestion des données entre ces équipements a fait l'objet d' un nombre croissant d'études. Les applications déjà proposées sont principalement basées sur des solutions propriétaires dites solutions privées ou fermées. Toutefois, cette stratégie a toujours montré ses insuffisances en termes de problèmes de sécurité, de coût, de simplicité pour les développeurs et de transparence des solutions. Migrant vers des solutions standardisées, HTML5 propose de nouvelles fonctionnalités pour répondre aux exigences des entreprises et des utilisateurs. L'objectif principal est de mettre à la disposition des développeurs web un langage simple pour la mise en œuvre des applications concurrentes à moindre coût. Ces applications ne sont pas liées ni aux dispositifs utilisés ni aux logiciels installés.

Nous nous intéressons dans notre thèse à proposer des solutions standardisées adoptées par l'utilisateur afin d'assurer la gestion des données entre ses différentes machines. Deux entités sont concernées par la gestion des données: stockage du côté client et du côté serveur. Pour le stockage côté serveur, un Coffre Fort Cloud standardisé est déjà présent. Il suit les spécifications d'AFNOR. En ce qui concerne le stockage côté client, aucune solution standardisée n'est encore proposée. A cet égard, la thèse porte sur deux problématiques principales. La première concerne la proposition d'un Coffre Fort Client standardisé où les données de l'utilisateur sont stockées localement. La deuxième problématique traite la synchronisation entre les Coffres Forts Numériques Client et Cloud.

Trois contributions font l'objet de nos travaux. Dans la première partie, nous proposons un Coffre Fort Client basé sur les APIs HTML5 de stockage. Tout d'abord, nous commençons par le renforcement de la sécurité de ces API pour fournir une base sécurisée à notre Coffre Fort Client. Dans la deuxième contribution, nous proposons un protocole de synchronisation appelé SyncDS qui est caractérisé par son efficacité avec une consommation minimale des ressources. Nous traitons enfin les problèmes de sécurité, et nous nous concentrons principalement sur le contrôle d'accès dans le cas de partage des données tout en respectant les exigences des Coffres Forts.

**Keywords**

Digital Safe, HTML APIs, Local Storage APIs, syncrhonization protocol, Hierarchical Hash Tree, WebSocket, CP-ABE, time based access control.

# Contents

# Glossary

| | |
|---|---|
| ABAC | Attribute-Based Access Control. |
| ACL | Access Control List. |
| AES | Advanced Encryption Standard. |
| AFNOR | Association Française de Normalisation. |
| API | Application Programming Interface. |
| BEP | Block Exchange Protocol. |
| COP | Configurable origin policy. |
| CORS | Cross Origin Resource Sharing. |
| CP-ABE | Ciphertext Policy Attribute Based Encryption. |
| CSRF | Cross-Site Request Forgery. |
| CSS | Cascading Style Sheets. |
| CVS | Concurrent Versions System. |
| DB | DataBase. |
| DDP | Device Discovery Protocol. |
| DES | Data Encryption Standard. |
| DOM | Document Object Model. |
| DoS | Denial of Service. |
| DTLS | Datagram Transport Layer security. |
| EFSS | Enterprise File Synchronization and Sharing. |
| GUID | Global Unique Identifier. |
| HDFS | Hadoop Distribute File System. |
| HHT | Hierarchical Hash Tree. |
| HLPSL | High Level Protocol Specification Language. |

| | |
|---|---|
| HMAC | Hash Message Authentication Code. |
| HTML | Hypertext Markup Language. |
| HTML5 | HyperText Markup Language 5. |
| HTTP | Hypertext Transfer Protocol. |
| IBAC | Identity-Based Access Control. |
| ICE | Internet Communication Engine. |
| ICN | Information-Centric networking. |
| IETF | Internet Engineering Task Force. |
| IPC | Inter-Process Communication. |
| JSON | JavaScript Object Notation. |
| LCS | Longest Common Subsequence. |
| MAC | Mandatory Access Control. |
| MD5 | Message Digest Version 5. |
| MSK | Master Secret Key. |
| NAT | Network Address Translation. |
| OWSAP | Open Web Application Security Project. |
| P2P | Peer-To-Peer. |
| PBKDF2 | Password Based Key Derivation Function 2. |
| PK | Public Key. |
| PK-ABE | Policy Key Attribute Based Encryption. |
| PKCS | Password-based Cryptography Standard. |
| RBAC | Role-based Access Control. |
| RTC | Reat Time Protocol. |
| SCTP | Stream control Transmission Protocol. |
| SDP | Session Description Protocol. |
| SFJ | Server File Journal. |
| SHA | Secure Hash Algorithm. |
| SMP | Strong Master Password. |

| | |
|---|---|
| SOP | Same Origin Policy. |
| SRTP | Secured Real Time Transport Protocol. |
| STUN | Servers Session traversal Utilities. |
| SyncDS | Digital Safe Based File Synchronization Approach. |
| TCB | Trusted Computing Base. |
| TCP | Transport Control Protocol. |
| TLS | Transport Layer security. |
| TPM | Trusted Platform Module. |
| TURN | Traversal Using Relay NAT. |
| W3C | Wide Web Consortium. |
| WHATWG | Web Hypertext Application Technology Working Group community. |
| WS | WebSocket. |
| XAuth | X Window Authentication. |
| XHR | XMLHttpRequest. |
| XML | Extensible Markup Language. |
| XMPP | Extensible Messaging and Presence Protocol. |
| XSS | Cross-site Scriting. |

# List of Figures

# List of Tables

# Chapter 1

# General Introduction

## 1.1 Problem statement and motivations

With the technological progress, the users are using multiple connected and smart devices. Therefore, the cross-device data management remains the concern of multiple research and industrial works. In fact, various devices, owned by the same user or different ones, rely on synchronization protocols in order to maintain data consistency. In this context, several Cloud storage solutions are proposed to handle this issue. They offer to the clients the possibility to externalize their data. So, they can upload, download, share and access them whenever and wherever they want.

Dealing with the cross-device management leads to address two entities. First, the server side framework guarantees the storage of users data in centralized servers. Second, the client side application ensures the storage locally in the user device and their synchronization with the server side content.

The existent frameworks in both sides are mainly based on proprietary solutions called private or closed solutions. This strategy has always shown its deficiency compared to the standard solutions in terms of security issues, cost, developer support and customization.

The standardization of the storage solution on the server side raises with the Gsafe project[10]. Adding the probative value in the Cloud storage, the project proposes a Cloud Digital Safe. It is standardized architecture that provides a secure environment for storing sensitive documents while guaranteeing integrity over time. The conception of this safe follows the AFNOR specifications [2].

The motivation of our work is to deal with standardized solutions. In particular, we incite, first, the proposal of a standardized client side storage solution which guarantees the local storage of the user data based on HTML5. Second, we emphasizes the proposal of a standardized protocol which ensures the synchronization between the client and the Cloud storage servers. In addition to the standardization, the efficiency and the security are the most important pillars of a synchronization architecture and protocol.

### 1.1.1 The potential of HTML5

How many times a browser asks the user to update his plugins? Do they have any idea of the changes brought by this plugin? Is it transparent to them which data this plugin can handle and which limits are set up to this access? Too many ambiguities are present in the use of plugin in a browser. The best of proof is that many operating systems such as iOS and Android refuse the support of browser's plugins such as the Adobe flash player because of security and management issues. As an alternative, they migrate towards the standardized solution which is HTML5.

In terms of security issues, the standardized and open solutions are usually adjusted to address the security thread. This dynamicity comes first, because of the availability of specification details, second because of the wide range of public people which are looking for potential threads. Unlike standardized solutions, proprietary ones rely on its development teams or try to hide its security problems to prevent outsider exploiting them.

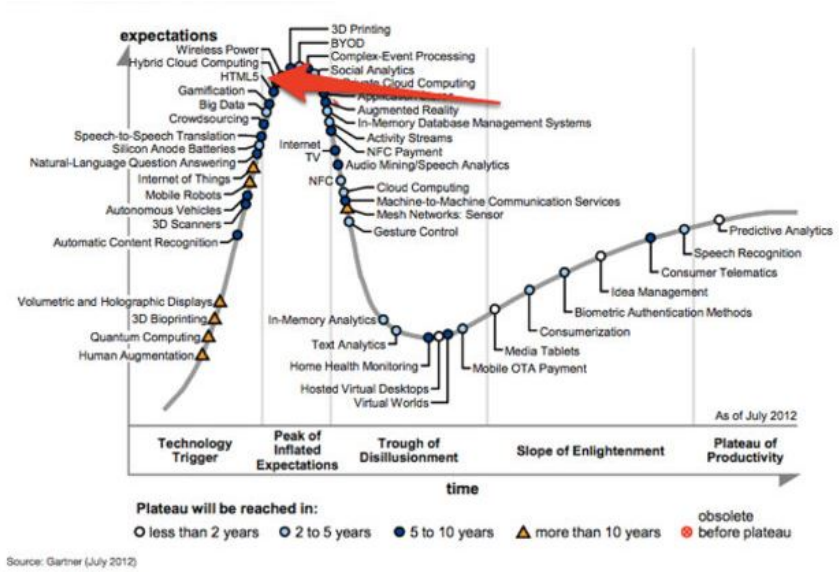



**Figure 1.1:** *Gartner Hyper Cycle for Emerging Technologies 2012 [98]*

HTML5 comes with new features to place the functionalities of proprietary solutions into a standardized format. Multiple analysis reports the prosperous future of HTML5. Gartner has been publishing its Hype Cycle schemes for Emerging Technologies since 1995. The interest of this scheme is to represent the evolution of the technologies according to their maturity, their visibility and their adoption. Published every year, this diagram illustrates the evolution of different technologies in a progression curve. The report shows that in 2012, HTML5 is located within the Peak of inflated expectations (figure 1.1). This phase is characterized by the exaggerated and unrealistic expectations of the technologies. After this phase, HTML5 will drop into the "trough of disillusionment" (the phase where the technology fails to meet the expectations of users). It will climb the " slope of enlightenment" in five to ten years where the technology becomes more reliable and mature.

Gartner analysis [25] consider also HTML5 as one of the TOP 10 technologies and capacities that will be crucial for organizations that want to exploit the full potential of mobility as part of their digital business strategy. Even with the lack of HTML5 maturity , it stills an essential technology for organizations which need multiple platforms applications.

### 1.1.2 The interest of data synchronization

Remote synchronization has been the subject of significant works that depend on the purpose of the software and the nature of objects to synchronize. In fact, these objects can be stored in structured or unstructured databases, in files and folders. The existent solutions are mainly proprietary. In this context, the issue of file synchronization and sharing had been the subject of the "Gartner magic Quadrant" [35] which details the different actors and their positioning (figure 1.2). Gartner has published a report on the current strategy of companies regarding the interest of sharing and synchronizing documents produced and used by their clients [33]. This report is entitled "How to build EFSS plans to address current and future business requirements" (EFSS: Enterprise File Synchronization and Sharing).


**Figure 1.2:** *Magic Quadrant for Enterprise file synchronization and sharing [33]*

Gartner's, report published at the end of 2014 highlights the interest of data synchronization solutions. It discusses the strategic issues such as the user expectations regarding the services associated to their documents. First, it emphasizes their benefits in terms of fluidity of the information exchange process. Second, it highlights the interest of adopting EFSS in the companies to protect corporate data.

## 1.2 Contributions of the thesis

Introducing the synchronization between the different Digital Safes needs (I) to introduce a Client Digital Safe, (II) define a file synchronization protocol with high quality and minimal resource consumption and (III) to ensure a secure data synchronization and data sharing.

As depicted in figure 1.3, the manuscript is divided into four main parts. The first part presents the security features of the Local Storage and communication APIs which are adopted in our Client Digital Safe and synchronization protocol. The three last parts deals with the three main contributions that follow the file synchronization requirements within the context of Digital Safe.

- (I) The conception of a Digital Safe based on the local Storage APIs;
- (II) Dealing with the efficiency issues of the standardized synchronization protocol;
- (III) Dealing with the security features of the standardized synchronization protocol.

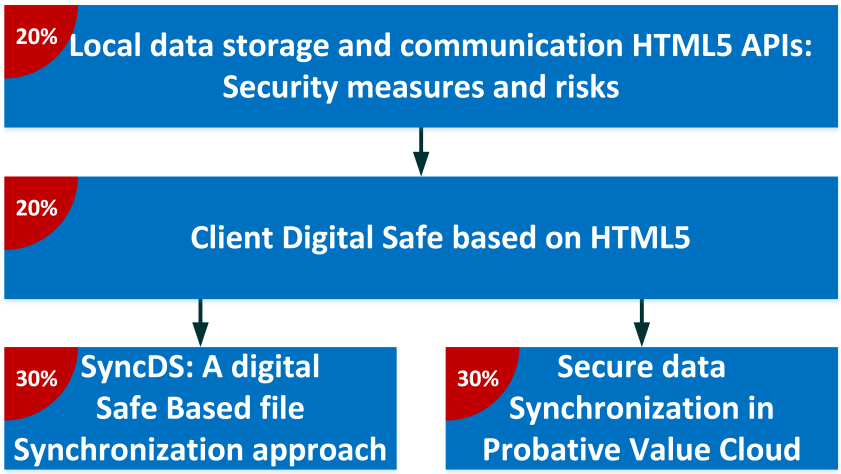



**Figure 1.3:** *Contributions of the thesis*

### 1.2.1 Client Digital Safe based on HTML5

Proprietary solutions are usually adopted for the client side storage and the file synchronization, and require the installation of software or plugins. To overpass these closed tools and to work with standardized solutions, we propose a HTML5 based Client Digital Safe. It enables the user to manage his data securely when he is offline or online. Our client side storage is based mainly on the HTML5 Local Storage API with additional security considerations to follow the Digital Safe standard requirements. In fact, we add the confidentiality, the integrity and the metadata integrity into the stored data. We define also a new Digital Safe API where the Digital Safe specifications are introduced. This API guarantees the interoperability between the Client and the Cloud Digital Safe. The data stored in this Digital Safe are subject of synchronization in the second contribution.

### 1.2.2 SyncDS: A Digital Safe Based File Synchronization Approach

The second contribution of the thesis focuses on a synchronization architecture and protocol SyncDS (Synchronization protocol in the context of Digital Safe). The synchronization is ensured between the proposed Client Digital Safe based on HTML5 and the Cloud Digital Safe. The architecture is divided into four main layers, which are: (1) *Client Storage Layer*: the Client Digital Safe, (2) *Application Layer*: handles the synchronization at the level of the application, (3) *Synchronization Layer*: manages the messages exchanged between the Local and Cloud Digital Safe based on the WebSocket protocol and finally, (4) the *Secure Storage Layer*: a standardized architecture that provides a secure environment for storing sensitive document in the Cloud. The particularity of our architecture is the adoption of the WebSocket which ensures a bidirectional communication between the client and the server. This protocol is used to notify the user in case of modifications which need synchronization. It is used also for the exchanges of synchronization messages and for the data transfer. Within the different synchronization approaches, the SyncDS protocol holds three phases: (1) *Offline phase*, (2) *On Connection phase*: with the *Post changes* and the *Synchronize* steps and finally the (3) *Online phase*.

During the on connection phase and more particularly, in the Synchronize step, the user sends to the server an abstract of his file system to be compared with the one of the server. This procedure detects the changes performed on the server when the user was offline.

We focus, in this part, on the format of the abstract. We chose to adopt the Hierarchical Hash Tree into the abstract format. Indeed, the Hierarchical Hash Tree (HHT) has a tree structure following the structure of the file system. Each node is identified by a hash besides the other metadata. For files, this hash matches to the file's content hash. For directories, the hash is based on the hashes of directory's content.

### 1.2.3 Secure data synchronization in probative value Cloud

In this part, we address the security requirements of the synchronization protocol that gathers the file integrity, non-repudiation, authentication, secure synchronization and the access control. We focus mainly on the access control, especially when the data are shared between different users. In fact, it is clear that synchronizing data that are already encrypted by users to multiple destinations may introduce key management challenges. We propose to use the CP-ABE Ciphertext Policy Attribute based encryption for the access control management. We add to CP-ABE the notion of timely based access control where data can be accessed only for a period of time. We address also the security of the key generation to be retrieved only by the concerned user.

## 1.3 Organisation of the manuscript

Following this introduction, the thesis manuscript is structured into five main chapters as follows:

In **Chapter 2** we present the web revolution and we introduce the innovative features and APIs of HTML5. Following the requirements of the cross-device data management, we detail the specification of the HTML5 APIs which ensure the communication and the local data storage. The security measures taken by each API and their security gaps are detailed.

In **Chapter 3**, we start by introducing the security measures fundamentals of the local storage. In the second part, we present the strength that we brought to the HTML5 Local Storage APIs. Based on the enhanced HTML5 APIs, we present then the conception of our standardized client Digital Safe by identifying and describing the different specification of this safe. The implementation and the integration of this safe into the chromium browser are specified at the end of the chapter.

**Chapter 4** deals with the synchronization protocol between the Client Digital Safe and the Cloud Digital Safe. We start by presenting, comparing and analyzing the different approaches adopted for data synchronization. We deal also with the different strategies used to detect changes between two versions of file systems. Dealing with our contributions, in the second part, we start by identifying and detailing the different entities of our SyncDS synchronization architecture as well as the messages exchanged between them. In the third part, we propose the introduction of the Hierarchical Hash Tree into the abstract structure, and we detail its associated algorithm. To prove the efficiency of our protocol with the adoption of HHT and the WebSocket protocol, we present at the end of the chapter, analytical explanations and empirical evidences.

**Chapter 5** starts with a survey of the security services and mechanisms adopted by data synchronization architectures in general. We address then security concerns; in particular, the access control on data sharing following the Digital Safe requirements. Subsequently, a timely-based access control is proposed based on the CP-ABE. We end our proposal with the validation of our protocol in terms of security.

**Chapter 6** We end up with the conclusion of our thesis, and we give a discussion of the achieved work and the future perspectives.

# Chapter 2

# Local data storage and communication HTML5 APIs: Security measures and risks

## Sommaire

## 2.1 Introduction

The Web has faced a revolution with HTML5. This new standard introduces innovative features and APIs to follow the business opportunities and users requirements. As a vendor-neutral language, these APIs enable the implementation of competing application with lower cost. Thus, numerous Web actors adopt it even if some security and privacy issues still exist.

Among the most known HTML5 APIs, we find those that ensure the storage of web application data locally in the user machine and those that ensure the communication over the Internet. They have proven their worth in terms of quality of experience enhancement and web application efficiency. Throughout our research work, we focus on these two kinds of APIs as they will be adopted in the Digital Safe conception and the synchronization protocol implementation. As depicted in figure 2.1, 20% of the thesis works focus on the analysis of security issues and risks of these APIs.

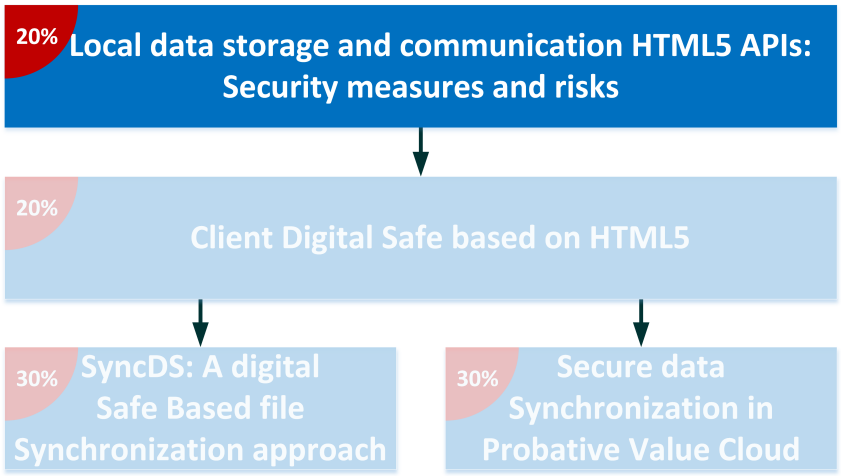



**Figure 2.1:** *Structure of the thesis*

This chapter is structured into four sections. We start with an overview on HTML5 standards as well as its global security features. The next two sections deal with the security measures and gaps of communication and local storage HTML5 APIs. Finally, we deal with the access control strategies adopted by browsers to secure the web application and users data.

## 2.2 Overview on HTML5 security

### 2.2.1 The Web Revolution

Since 2007, improvement has been made on the HTML standard to promote the web revolution. Such improvement are held through the experiences feedback of the existing browsers (Opera, IE, Firefox, Chrome, etc.) as well as the user requirements. As shown in figure 2.2 since 2004, the Web Hypertext Application Technology Working Group community (WHATWG) [46] has started working on improving the HTML standard.

Three years later, the HTML5 specification was adopted by the Working group World Wide Web Consortium (W3C) [22] which is headed by Google. In 2012, W3C interacts with the IETF and proposes Working Draft specifications of HTML5 APIs . So far, a part of these APIs becomes stable and is implemented in the majority of the browsers.

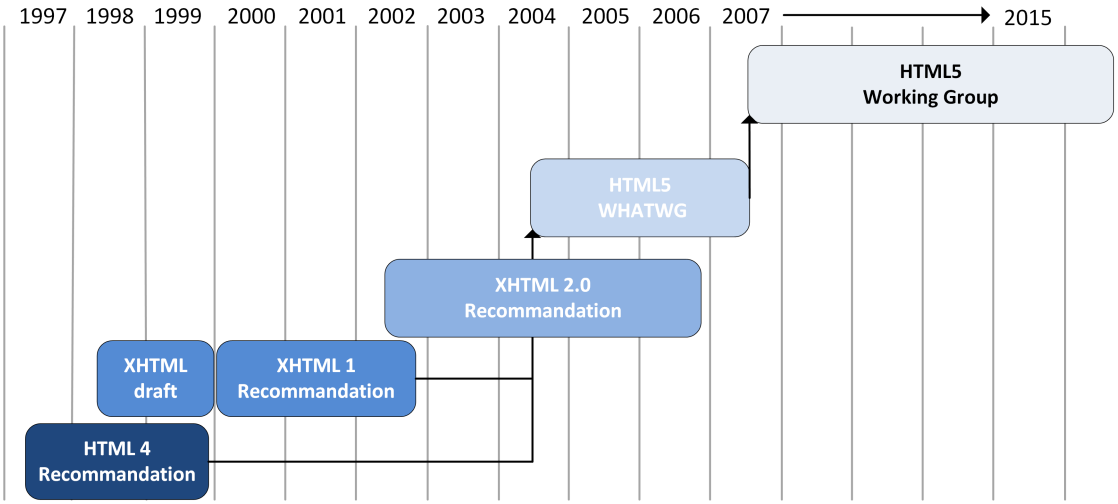



**Figure 2.2:** *HTML5 evolution*

The main goal of the specification is to define an open and an infrastructure independent language. Thus, it can be deployed by a wide range of products and devices and can reduce the development cost. First, the web revolution with HTML5 comes, as shown in figure 2.3, by the elimination of the proprietary solutions which needs additional installations and adaptation to extra languages. Second, HTML5 aims to offer new features already treated by the Web2.0 services. It aligns with the requirements of the Internet of things, augmented reality, real-time and peer-to-peer communications, web video, geolocalisation, offline application etc.

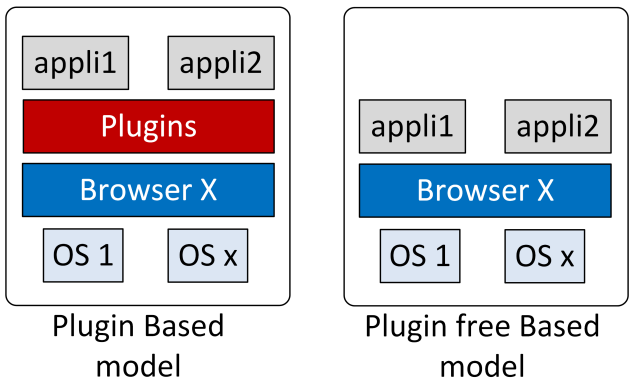



**Figure 2.3:** *HTML5 vs Non HTML5*

### 2.2.2 Security issues of HTML5

An HTML5 API is an interface which gathers a set of objects, methods and properties brought together for a particular purpose. With different degrees of maturity, the HTML5 APIs are classified from the less mature to the most mature as following: Working Draft, Candidate Recommendation, Proposed Recommendation and W3C Recommendation. These APIs are implemented in the browser, and everyone chooses its own strategy of deployment. Different criteria can be adopted to classify the whole HTML5 APIs such as the nature of manipulated elements or the nature of the brought functionality. Throughout our work, we choose to classify the APIs according to their functionalities. As shown in

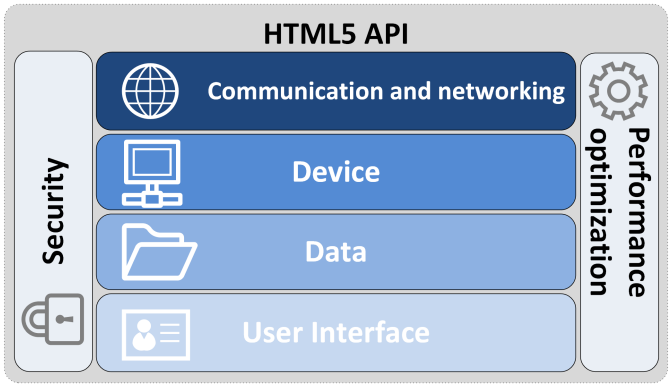



**Figure 2.4:** *HTML5 API classification*

figure 2.4, they can be classified into six main categories:
- **Communication**: APIs that ensure the communication between the client, the server (synchronous or asynchronous), the browsers and the threads;
- **Device**: APIs that exploit device's component during the navigation. For example, we find displaying videos and animations, extracting the device position and orientation, etc;
- **Data**: APIs that enable the local storage of data in the user's machine and then their management;
- **User interface**: APIs that deal with the ergonomics of web pages to obtain an easier environment used by the final client;
- **Performance optimization**: APIs used to compute or ameliorate the navigation performances;
- **Security**: APIs that add secure functionalities to different web applications.

Each technology has a set of vulnerabilities, and its evolution pushes the attacker to look for new opportunities of attacks to exploit. Before focusing in details on the security risk of HTML5 communication and storage APIs, we need to pass through different browser vulnerabilities. Figure 2.5 shows the browser structure with the HTML5 technology stack and its adjacent technologies. With the integration of HTML5, new technologies were emerged and new interactions between them were built. Among the main threats, we find:

- The Javascript is subject to code injection attacks [116]. It is known that web

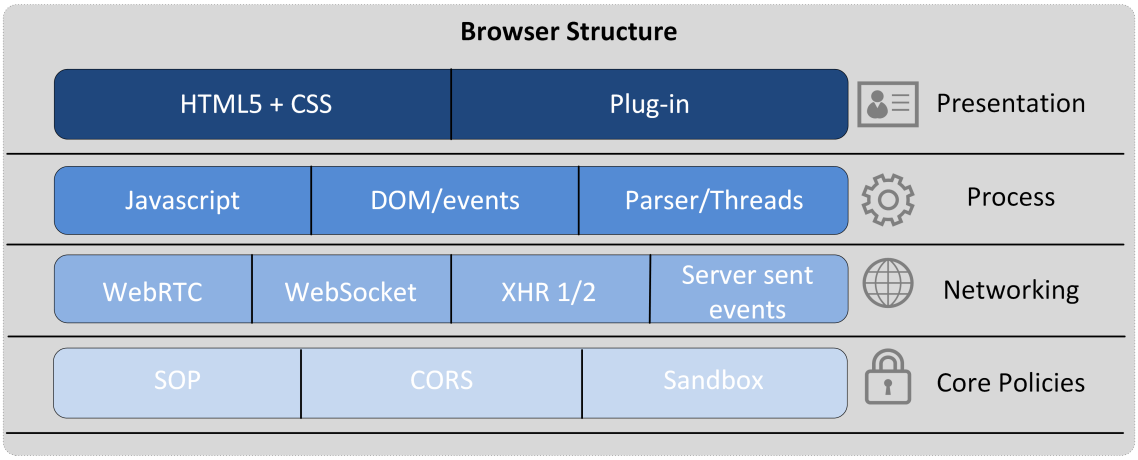



**Figure 2.5:** *Browser structure with HTML5*

technologies allow the data and code to be mixed together. Therefore, when a malicious code is embedded inside an HTML page as a text, it can be sent for Javascript engine for execution. Cross-site Scriting (XSS) is a special type of this attacks and is considered by the OWSAP [11] as one of the top ten security risks in web application. Besides the data disclosure, XSS concerns the HTML5 tags such as media, canvas, form and buttons, HTML5 attributes such as form autofocus, sandbox and HTML5 events and objects such as Drag and Drop API, history API etc. In addition, XSS can be used for HTML5 Denial of Service (DoS) when the injected code sends many requests to the victim website.

- Dom based XSS: This kind of attack is on the rise as large applications are built on Document Object Model, XmlHTTPRequest and Web Messaging [102]. It is an XSS attack that occurs when an untrusted code is inserted into dynamic code evaluation constructs without any verification process. In fact, the malicious modifications of the DOM environment lead to a different execution of the client side code. This kind of attack has important effects on the HTML5 application that runs widgets as the entire DOM can be accessed by the attacker. This attack can also affect the plug-ins installed into the browser since the DOM loads different objects such as Silverlight and Flash extension.

- Misuse of the Cross Origin Resource Sharing (CORS): The Same Origin Policy (SOP) is a core policy adopted by some HTML5 API to disable any inter-domain data exchange. Even if this technique serves as a security function, it stills insufficient for certain applications. This leads to the emergence of Cross Origin Resource Sharing. The CORS allows the communication between documents from different domains. It is built on top of the XmlHttpRequest with new parameters in the header. However, the CORS header can be crafted or misconfigured to get access to sensitive informations. This misuse can lead essentially to the Cross-Site Request Forgery (CSRF). With this attack, the attacker forces the victim to execute actions

within the authenticated context of a web application without victim's consent. When the victim visits the attacker's web site, it receives a response with the CSRF header. This response initiates, therefore, an HTTP request to the web application.

- Access to local resources: Even if the browser has the full access to the local resources of the user's device such as files, webcam, microphone or geolocalisation, each HTML5 API should ask the user permission to access these resources. For example, the user selects explicitly a file when using an HTML forms which allow file upload. In addition, if a web application needs to locate the user's device, a pop-up appears in the browser to ask the user's permission. The user consent is not restricted to the HTML5 API as it concerns also the plug-ins and extensions. It is the case of the Shockwave flash. Some HTML5 APIs need to pass through the procedure of user's consent and are not taken into consideration by the browser. It is the case of the LocalStorage API. In fact, a website can store or retrieve freely data on the user's machine without his notice.

- Divers HTML5 APIs: The different HTML5 APIs that carries new functionalities, unfortunately, brings vulnerabilities exploited by attackers. As an example, we find that the HTML5 fullscreen API can lead to phishing attacks [20], Web Workers API can be exploited for DoS attacks by using excessive CPU for computation, and a misuse of the Web Messaging can lead to messaging hijacking [11].

## 2.3 HTML5 Communication APIs

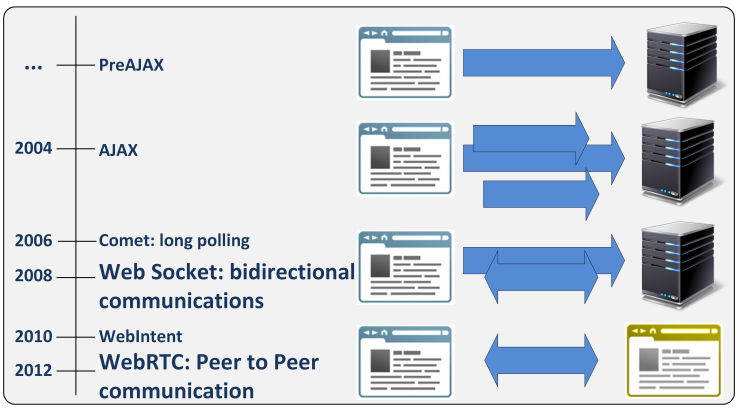



**Figure 2.6:** *Evolution of HTML communication protocols*

Many researchers are working on deploying web environment to be used in devices with limited capacity [61]. To this end, communication APIs should be adopted. As shown in figure 2.6, the evolution of the communication technologies over the web life can be presented as follows:

- PreAjax: Before Ajax, the response to each request sent to the server is a static web page;
- Ajax: The dynamicity was added into the web pages through the adoption of Ajax. In fact, it is used to build dynamic and interactive web site which can be updated automatically without refreshing. Ajax combines JavaScript, CSS, JSON, XML and DOM to improve the management of Rich Internet Application;
- Comet [100] is a model that uses the HTTP protocol. It allows a server to push data to the browser without any request. Therefore, the client polls data by sending a request that stills open for a long time waiting the availability of a data to be sent back to the client. With an additional HTTP connection, Comet facilitate bi-directional communications over two HTTP connections;
- The WebSocket [31] that ensures a bidirectional communication based on full duplex sockets;
- Web-intent [41] is a framework that enables rich interaction between web applications. It enables their interconnection without recognizing each other with a limited code line number. It is modeled after the integration of the Intent system into Android where the intents are used to ensure the data exchange between the application components;
- The WebRTC [21] enables a P2P communication between browsers.

With the progress of the communication protocol, new features are considered to add the efficiency into the Web application. Dealing with the efficient synchronization protocol in the context of Digital Safe, we will focus mainly on the two new HTML5 APIs which are the WebRTC and WebSocket. A detailed specification of these APIs and their related protocol will be presented, and their security analysis will be discussed.

### 2.3.1 WebRTC and security requirements

In this section, we introduce the WebRTC API and protocol which ensure a Peer-to-Peer communication between browsers. We study the security and privacy issues of this protocol.

#### Specification of WebRTC API and protocol

WebRTC [21] changes the way of interaction between users and how a user interacts with the Internet. It is a free and open project that ensures a bidirectional communication between browsers. No software or plugins needs to be installed as browser-browser communication is ensured through simple JavaScript and HTML5 APIs. The basic use cases of this API are video and audio in real time, web conferences, chatting and direct data transfer. Standardized by both W3C and IETF, it is based on three steps to ensure a P2P communication for data exchange(figure 2.7):

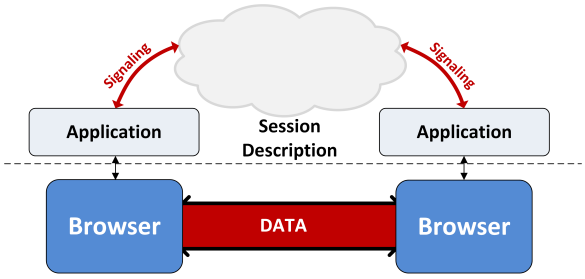



**Figure 2.7:** *WebRTC protocol*

- **PeerConection**: It creates a connection with the peer through predefined servers (ICE elements: STUN and TURN servers);

- **Session description**: It exchanges Session Description Messages (SDP) between the different peers. These messages cover the channel informations and ICE elements;

- **DataChannel**: It creates a direct channel between browsers to be used for data exchange.

The Network Address Translation (NAT) causes difficulties in Peer to Peer communication as the peer acts as a client and server at the same time. Through these problems, we find, first, that the communication initiated by a peer will be blocked by the NAT as there is no mapping between the exterior and interior address and port pairs. Second, a peer is unable to know its public address to provide it to the other peer. To overcome the complexity of real-world networking, WebRTC uses the Internet Communication Engine (ICE) with the Servers Session traversal Utilities (STUN) and Traversal Using Relay NAT (TURN). STUN is a protocol used to help NAT traversal. It helps to obtain the peer's IP address-port and to discover if the peer is behind a NAT. TURN is a STUN's extension that provides a relay for interchanging data between peers.

WebRTC stills a working draft in the W3C standard specification. Therefore, it is the concern of multiple research and industrial works. In [87], Kurento software overpasses the basic exploitation of WebRTC in P2P communication. It proposes a media server based on WebRTC and a set of APIs which handle transcoding, mixing and routing of audiovisual flows. This solution enables the development of advanced video applications and advanced media processing capabilities such as the video indexing and the augmented reality, to integrate it into the IMS systems for professional application. Based on WebRTC, BOPlish [56] introduces an architecture for decentralized content publishing between the same web application launched on different devices. The content distribution is ensured in the Information-centric networking (ICN) context and used for online games, chat group and file sharing.

### Security measures taken by the WebRTC

A series of security measures is adopted by the WebRTC API and protocol to prevent possible attacks. Bellow, we list these principal measures.

- TLS is the most widely adopted protocol for securing the network traffic. It is based on the TCP as transport protocol. As many applications are adopting the datagram mode for data transfer, the Datagram Transport Layer security (DTLS) emerged as a datagram-compatible variant of TLS. DTLS is adopted by the WebRTC below the Stream control Transmission Protocol SCTP. This protocol guarantees a secured data exchanges between the far end peers during the datachannel step;

- WebRTC adopts the Secured Real Time Transport Protocol (SRTP) to encrypt the media streams. This protocol is used rather than DTLS for the exchange of this kind of stream as it is characterized by its lighter-weight;

- The fingerprint identifies the certificate presented during the DTLS handshake. It is transported in the SDP where the key exchange occurs in the data channel. The fingerprint is used to verify that the devices communicating with the DTLS are the same exchanging the signaling messages, to be sure that there is not man in the middle attacks;

- The WebRTC has the access to the user's device resources such as camera and microphone. However, the user's consent is required to give permission to the API to exploit these resources. His consent avoids malicious applications to record his video or voice without his accord. To meet these requirements, the current browsers are implementing the user's permission request for one-time or permanent access. To go further, they display explicitly in the user interface that these resources are currently opened and used. For example, Google Chrome shows a red spot for the camera and a loudspeaker icon for the microphone. Nonetheless, nothing is taken into consideration in case of data exchange between the peers which can lead to DoS attacks on his device;

- A solution for far-end authentication is proposed in [21]. It considers that the web application is launched as untrusted and likely hostile while the browser is considered as a Trusted Computing Base (TCB). The security architecture is based on the Identity Provider that supports a protocol such as OpenID or BrowserID. In fact, each user should have an account with an IdP that he uses to authenticate to other web sites. The authentication elements are first exchanged between the browser and the IdP to prove the identities. Second, they are exchanged between the browsers during the signaling phase over the SDP messages.

#### Security gaps of WebRTC

Even if a series of security measures are considered by the WebRTC protocol and API specification, there are still areas for attackers:

- One of the ICE server disadvantages is the disclosure of the peer IP address. Since the IP Addresses are stored publicly, the position of the node can be revealed even if the user refuses to disclose it;
- DOS attacks can be performed by a malicious web application on connected devices and even on the NAT infrastructure servers [38]. For example, the credentials of the TURN server are introduced into the web application to configure the connection to the ICE servers. These credentials can be sent to unauthorized parties and which can exploit them to attack the TURN server and consume its bandwidth. Therefore, it is interesting to revoke frequently these credentials and use new ones;
- Once the WebRTC connection is established, a malicious web application can send and retrieve freely traffic without the consent of the user;
- The signaling protocol is not predefined and imposed by the WebRTC specification. Therefore, the security of the WebRTC depends heavily on the choice of the adopted signaling protocol. There are mainly five protocols which can be adopted: WebSocket, SIP over WebsSocket, XMPP/Jingle, XHR/Comet or even the WebRTC's Data channel. It is essential to adopt a secured signaling protocol. Otherwise interception a signaling messages by an attacker leads to a man in the middle attack [37].

### 2.3.2 WebSocket and security requirements

In this section, we present the WebSochet API and the bidirectional real time protocol. Afterward, we list the security measures already considered by the protocol specification and, we present its security risks.

#### Specification of WebSocket API and protocol

WebSocket is a new protocol standardized by the IETF [31]. It provides a permanent two-way communication between the client and the Web server using a single socket. The first phase of the communication between the client and the server, as shown in figure 2.8, is the handshake. Once the handshake is established, the client and the server exchange in both directions a series of messages, each one is composed of one or more frames. There are several types of frames, and all frames of the same message have the same type. For example, we find the binary and text types, where frames contain Unicode characters encoded in UTF-8. There are also control frames (ping and pong) designed for the use of the protocol itself.
The main characteristics of the Websocket are:

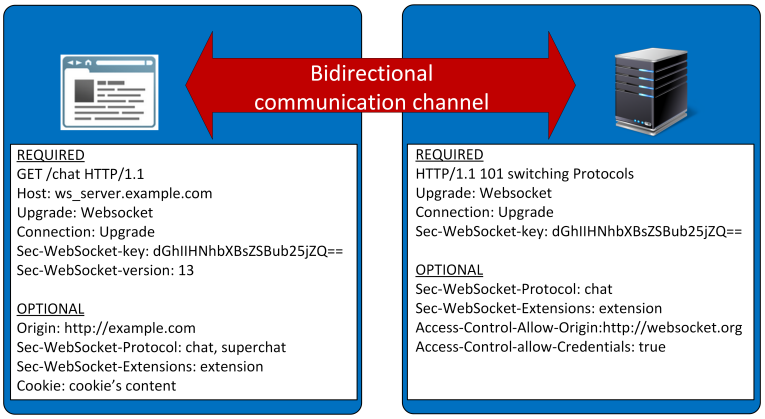



**Figure 2.8:** *Websocket handshake*

- the specification of the API is developed by the W3C and the communication protocol is standardized by the IETF;
- it is a bidirectional and asynchronous communication that ensures both the push and pull of informations between the client and the severer;
- it basically uses the port 80 and in case of secured connection the port 443;
- it is based on the TCP transport layer;
- it optimizes the network traffic with only 2 bytes of overhead;
- it reduces the latency.

### Security measures taken by WebSocket

Security measures are introduced into the specification of the WebSocket protocol to fight against possible attacks. Among these measures, we find:

**- The encryption of the WebSocket traffic:**

A TLS handshake can be achieved just after the WebSocket Handshake. In fact, using the encryption of the message payload guarantees both the confidentiality and the integrity.

**- The specification of the origin into the WebSocket header:**

The field origin in the message header of the handshake mentions the origin of the Web page that loaded the client script. The server can verify this field and choose to accept or not the request.

**- The exchange of a key between the client and the server:**

The connection is established only when the browser receives the appropriate value of the Sec-WebSocket-Accept field. The client will be sure that the response comes from the server that received his request. The value of Sec-WebSocket-Accept matches to the SHA-1 of the concatenation of Sec-WebSocket-key and a fixed GUID. This technique prevents an attacker from deceiving the WebSocket server as the Sec-WebSocket field cannot be added by the XMLHttpRequest code. Thus, the use of the key guarantees that the WebSocket

protocol is adopted by the two nodes .

**- The use of the mask for an unpredictable traffic**:

Adopting the WebSocket protocol without the message encryption can be used to disturb the proper functioning of the intermediate network elements such as proxies. Considering the following scenario:

GET /ga.js HTTP/1.1
Host: www.google-analytics.com
Protocol: Upgrade

An attacker establishes a WebSocket connection to a target server via a proxy. It sends then a WebSocket request that mimics a simple HTTP request. The attacker programs its server to response with a malicious file to such a request. With this approach, the attacker realizes a cache poisoning of the proxy which is not aware that the communication is based on the WebSocket protocol (the Upgrade field is not understandable). As consequences, an HTTP proxy cache poisoning is performed [80]. Each time the client accesses a web page with the URL http://www.google-analytics.com/ga.js, he receives systematically the malicious file.

To overcome this type of attack, WebSocket masks the data with a key controlled by the runtime environment (example: the JavaScript interpreter) and not by the application. The basic idea is that the payload content cannot be predictable. To overpass the HTTP proxy cache poisoning , [80] proposes to improve the Websocket protocol by encrypting the bytes using the stream cipher. Later, the Websocket working group adopted a variant of this proposal by masking the attacker-controlled bytes using the XOR with a predefined mask.

### Security gaps of WebSocket

The WebSoket API and protocol have adopted several security measures to prevent attacks on web applications. However, WebSocket stills present security gaps:

- Web applications which use the WebSocket are exposed to the attacks of basic web applications. In fact, Cross Site Scripting XSS and man in the middle still presents a risk to the application security. Each attacker who can sniff the HTTP traffic can use the same methods to intercept or inject code into the WebSocket traffic;
- The denial of service attack (DoS), is also possible with WebSockets. Indeed, it is possible to use all the resources of a server as no check on the number of WebSocket connection is imposed. This attack can be done by opening simultaneously multiple WebSocket connections. This problem is solved at the browser level by imposing a maximum number of connections. An attacker can also set off a DoS attack. It exhausts the memory of another node by sending a large frame or by sending a large number of small frames. Therefore, it is interesting to set a maximum size for frames and a message size after collecting frames that constitute it.

## 2.4 HTML5 Local data storage APIs

In this section, we start by presenting the different HTML5 Local Storage APIs to deal, thereafter, with the analysis of their security.

### Specification of HTML5 Local Storage APIs

To enable the storage of data in the browser, several proprietary solutions have been implemented before the emergence of HTML5 standards [96]. These solutions can be extensions and plugins which should be installed into the browser or libraries and should be integrated into the Web application code. Among the possible storage solutions, we find:

- **Local Shared Objects [48]**: Based on Adobe flash and called Flash cookies, it stores data on user's device. These data can be found in folders located on paths that differ according to the operating system. However, this solution is not supported by the majority of mobile devices;
- **Google Gears Storage Provider [9]**: Implemented by Google to offer offline functionalities for the Web application, it uses SQLite database to store information. Nevertheless, it has been abandoned as it has the same functionalities of HTML5 Local Storage and Offline APIs;
- **UserData behavior [14]**: It is introduced by the Internet explorer browser. It stores information and page states in a hierarchical data structure using XML. This solution is proprietary to IE browser and it is used in several applications to increase the storage volume of the HTML5 Local Storage APIs.
- **Dojox.storage [7]**: It is a part of the JavaScript toolkit Dojo which includes languages utilities, user interface components and other libraries used for building a web application. Dojox Storage provides a persistent client side storage based on existent HTML5 APIs. It proposes an encrypted data storage for confidential informations.

To overpass the proprietary solutions and the introduction of additional plugins, HTML5 proposes three principal APIs that ensure the local storage based on standardized specifications. These APIs differ by the nature of locally stored data. Throughout the manuscript, we call the set of these APIs as HTML5 Local Storage APIs.

- **WebStorage [39]** stores key/value pair data in the browser. According to the usage requirement, two possibilities of storage are available. First, there is the persistent storage. The data are shared between different tabs and can be deleted only by the application or by the user. Second, in the case of temporary storage with the SessionStorage, the data are deleted when the tab is closed. The data of these API are stored into a SQLite database.

- **Indexed DB [24]** is an API that enables web application to store data in an indexed database. It replaces the traditional relational databases architecture. In fact, an indexed system allows an optimized and fast access to data.

- **FileSystem API [32]** simulates a virtual file system in the browser. The implementation of this API allows web applications to manage files. Hence, applications can read, write and create files and folders in a sandbox. FileSystem API offers also two possibilities of storage, persistent and temporary storage.

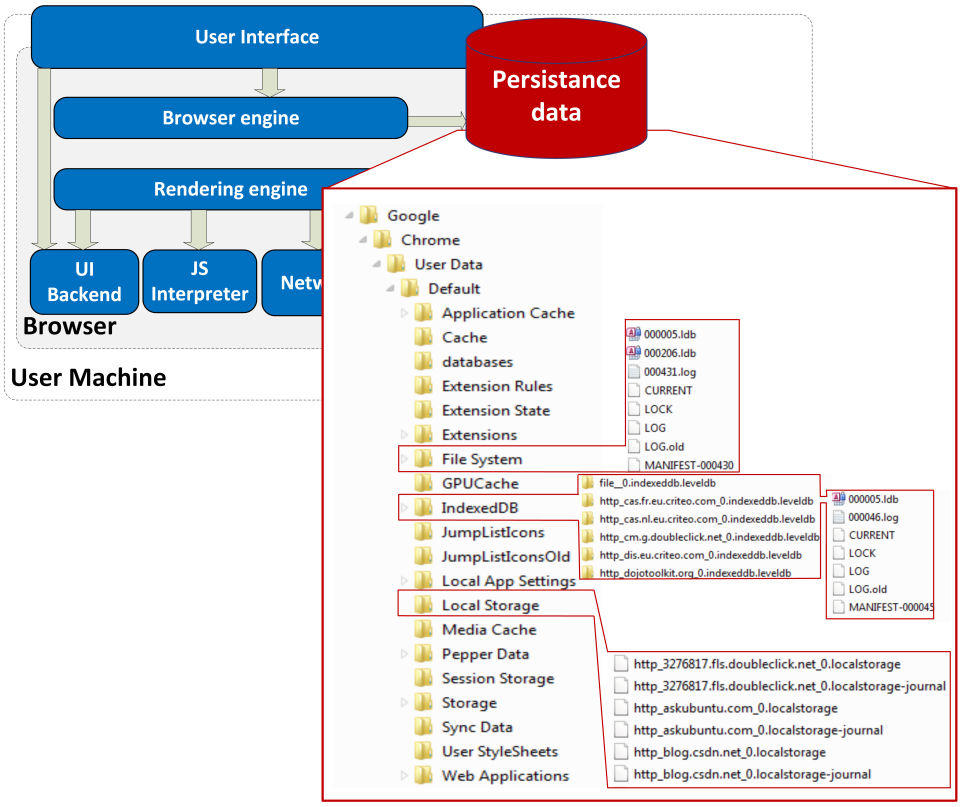



**Figure 2.9:** *Browser's architecture with the local data storage*

These APIs are implemented in the browser. A browser consists on a group of structured codes that interpret the web development languages (HTML5, CSS and javascript) and transform them into a set of instructions (displaying webpage, sending message, storing data, etc.). As shown in figure 2.9, in the data storage context, the browser engine is responsible for storing application data in the user machine in appropriate files. These files, with different extensions, match to the databases (SQLite DB, Level DB) where the web application data are inserted. In fact, the domain name is used to separate different domain's data.

Although browsers follow the same specification, each one implements differently HTML5 APIs. Several experiment evaluations of the HTML5 Local Storage API [78] [121] underline that the performances differ greatly from one browser to another. The results, in [78], show that the indexed DB performs faster in Firefox (using the SQLite database)

than in Google Chrome (using the LevelDB, which is a library proposed by Google to provide an ordered mapping from string keys to string values).

### Security measures taken by HTML5 Local Storage APIs

As it is detailed in the W3C specification, each browser should store the data of each domain in a separate location. This strategy is called the SOP (Same Origin Policy). It represents the only form that enables the content protection. This practice prevents any inter-domain access by linking the stored data to a particular domain.

### Security gaps of HTML5 Local Storage APIs

If we analyze in depth the Local Storage APIs, we uncover that attacker can exploit the APIs conception in the browser to retrieve and to modify user informations [79][39]. The following attacks affect the confidentiality and availability of data:

**- Extracting private information on users**

An advertising website can use the LocalStorage API to track the user by extracting his information across multiple sessions. This strategy allows these web sites to build a complete profile on user preferences for a marketing and business interests. To reduce the risk of user tracking, it is possible to adopt several techniques such as:

- refusing access to local data of a domain from other domains which run in iframes;
- removing persistent data after a period of time;
- asking the user's permission to access the local storage;
- defining a blacklist of domains that should not save data locally.

**- DNS Spoofing**

It is possible that the data associated with a domain are exposed to DNS spoofing attack. The attacker can simply use his domain and the DNS server to redirect all the traffic of the victim to another domain that appears to be those of the Web application. For the WebStorage API, the hacker can retrieve the keys of data by opening a copy of the same application on his side and scanning the keys using the browser. Retrieving the key leads automatically to the recovering of value.

**- Shared environment**

A user who accesses a given browser may also access data stored in the machine. On the one hand, the domains focus on saving their data based on the used browser and not on the user. On the other hand, the Local Storage data is stored unencrypted in SQLite and indexed file on the client machine. Using the Local Storage represents a significant risk that mainly concerns confidential data such as passwords and information about the credit card.

To reduce the risk, the data can be encrypted before storing them. However, if the attacker can access the stored data, he can also launch the associated application and use them. To avoid this scenario, it is essential to link the data to the user. Therefore, a user authentication needs to be performed before Local Storage data decryption.

**- Cross Site scripting**

The Cross-Site Scripting allows the injection of malicious code in the contents of a page. With XSS, the attacker can, for example, retrieve or change data stored in the browser by LocalStorage API. In the case of the Webstorage API, the injected JavaScript code contains necessarily the attributes 'getItem' or 'setItem' and the name of the relevant key and value.

**- Implementation of the browser**

HTML5 introduces the specification of the different APIs used by a Web application to provide a set of features. Each browser chooses its own strategy to implement these APIs. We focus on the implementation of the WebStorage API. The W3C standard specifies that the sub-domains (prefixes or Origin) of a domain should not have extra space for the storage. However, Chrome, Safari and IE do not respects this rule. Thus, it is possible to multiply indefinitely sub-domains to fill the space of the disc with content of the LocalStorage (if the client uses the Google Chrome, the browser will crash when the data size reaches 970MB). Firefox is the unique browser that implements adequate measures against this type of attack. In fact, it imposes that all sub-domains of the same domain share the same space.

**- Exploiting metadata**

The attacks based on metadata tampering which concern the storage of password in the browser are well discussed in [53]. In the context of the HTML5 local storage, data encryption is not enough against the metadata tampering and data recovery. In fact, an attacker can alter the name of the database files. Therefore, the browser will consider that the stored data match to the attacker domain. If the domain of the attacker gains accidentally the permission of the user to manage local data, the attacker will have the full access. To avoid this kind of attacks, encrypting the metadata of database file remains essential.

## 2.5 Access control in browsers

The Same origin Policy (SOP) is a core policy adopted by some HTML5 API to disable any inter-domain data exchange. Even if this technique serves as a security function, it stills insufficient for certain applications. This leads to the emergence of other security measures. The security measure taken by the browser, defined as core policies in figure 2.5, can be classified into two main categories [57]: strategies which are based on the principle of SOP by defining additional criteria for policies and other strategies different from the SOP for more flexibility.

The table below summarizes the different strategies of access control found in the literature. They are adopted by the browser to manage the web application resources.

**Table 2.1** Access strategies for browser

| Access strategy | Example |
|---|---|
| **Additions to the SOP** | - **Splitting one Single Principal**: creating an isolated environment. Exp: iframe in HTML5<br>- **Combining multiple Principal at the server side**: Specifying access control policies at the server side through a HTTP headers. Exp: CORS, SOMA<br>- **Combining multiple Principal at the client side**: Specifying access control policies at the client side. Exp: PostMessaging, Gazelle<br>- **Cooperation between the client and the server**: Specifying access control policies after a cooperation between the client and the server. Principals focus mainly on resources exchanged between the web application and the server. Exp: COP |
| **Changes to the SOP origin** | - **Fainer grained Label**: The access control scheme is <Protocol, Host, Port, optionally extra data>. Exp: optionally extra data can be : Path, server public key, ring access control, token |

### 2.5.1 Additions to Same-Origin Policy

The policy can be used in addition to the Same Origin Policy:

- Cross Origin Resource Sharing (CORS) is a W3C specification that allows the communication between documents from different domains. It is built on top of the XmlHttpRequest with new parameters in the header. For example, a resource loaded from domain A (http://a.example) as an HTML page, makes a request for a resource of the domain B (http://b.example) such as image using img tag (http://b.example/image.png);

- iframe [12] is a tag improved by HTML5 to specify an inline frame. It is mainly used to embed a document within the actual HTML document. Therefore, the resources of document managed by this tag supports a sandbox properties. These resources are separated from those of the host document with the same origin;

- SOMA [97] enhances the SOP by adopting a mutual approval to send cross-domain HTTP requests. In fact, the inclusion of resources into a web page is performed only by the approval of both the target site and the resource provider. When an approve is needed from the site operators or from external domains, the content is prevented to be retrieved from attackers or malicious servers;

- The first draft the Webstorage specification by WHATWG [28] introduced the globalStorage object used to specify the domains that could access the current domain's data. Due to security concerns, this object was removed from the spec and

replaced with the same origin policy. Cross domain LocalStorage [120] implements an extension of the web storage specification to enable cross origin communication. The cross domain storage framework is inspired by the X Window Authentication (XAuth) protocol implementation, which shares third party authentication information in the browser. It combines the use of the cross document messaging and the LocalStorage API to get access to data that was stored by a different domain. Therefore, the cross origin is performed at the JavaScript level;

- Configurable Origin Principals [57] focuses mainly on the principal that depends on defining different principals for a unique domain and regrouping many domains into the same principal. The configuration of the principals is performed between the different web sites by exchanging the principals between the web sites. With this strategy, changes concern both sides: the web server and the browser. First, the web server makes modifications to the web application and the responses to the principal communications. Second, the browser makes modifications to the WebKit level to override the Same Origin policy. This strategy focuses on sharing the resources, mainly on the iframe;

- Multi-principal browser such as Gazelle [112] is proposed. The main goal of the browser is to separate each element of the same web page. Therefore, each component does not affect the other elements either the browser or the machine. The separation of each principal and the security measures are introduced with a novel browser's architecture.

### 2.5.2 Finer grained Label

Finer gained label uses additional element to those defined by the HTML5 specification. The basic scheme used by the SOP is <Protocol, Host, Port> with the fundamental problem of coarse granularity. Several researchers worked on adding extra information for the policy to avoid specific attacks. Therefore, the origin of a resource depends on the scheme <Protocol, Host, Port, optional extra data> to provide finer granularity. These enhancements includes:

- Specifying the URL that a resource has the right to access. This strategy is adopted by the Tahoma Web browsing system [62].

- Enforcing the access using the servers x509 certificates and public key [77] to deal against the Dynamic Pharming attacks.

- Defining a Mandatory Access Control (MAC) policy based on rings and ACL to control the access to the objects such is the case of ESCUDO [70].

- Adopting a capability-based access control model for the web browsers. In this context, the strategy adopted in Contego [86] requires that a principal need to have

a token to perform certain actions. For example, a JavaScript needs to have an AJAX request token if it wants to send an AJAX request.

## 2.6 Conclusion

In this chapter, we introduce, in a first part, the HTML5 as well as an overview of the security issues of the browsers which adopt this standard. In the second and third parts, we point out in detail the security issues of the HTML5 APIs of communication and local storage. Finally, we deal with the security measures taken by the browser to control the access to the web application resources.

This chapter described the general security of HTML5 with some of its APIs. In the next chapter, we will focus mainly on the Local Storage APIs to build a secure Digital Safe based on.

# Chapter 3

# Client Digital Safe based on HTML5

## Sommaire

## 3.1 Introduction

Several research and industrial works focus on guaranteeing the security and confidentiality of the user data when externalized and managed by Cloud Service Provider. While the majority of the proposition is dedicated to the Cloud infrastructure, few efforts deal with the security of the data before their externalization and more specifically when they are stored locally in the user machine.

The existent frameworks are mainly based on proprietary solutions called private or closed solutions. This strategy has always shown its deficiency compared to the standard solutions in terms of security issues, cost, developer support and customization.

We propose to overpass the proprietary characteristics to focus on standardized solutions. As illustrated in figure 3.1, 20% of the thesis works focus on the introduction of a standardized Client Digital Safe. Two main contributions are subject of this chapter to propose a secure and standardized local storage. Based on the two standard HTML5 and AFNOR, we enhance first the security of the HTML5 Local Storage APIS. Second, we introduce the Client Digital Safe which is based on the enhanced HTML5 Local Storage and which follows the AFNOR specification.

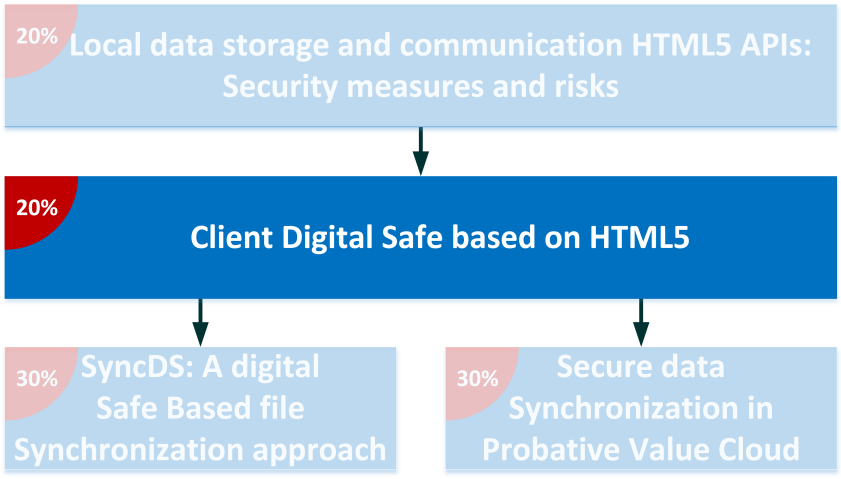



**Figure 3.1:** *Structure of the thesis*

This chapter is structured as follows. We highlight, first, the security measures considered by the local storage solutions. In the second part, we detail the first contribution which deals with the enhancement of the security of HTML5 Local Storage APIs. In the third part, we introduce the Client Digital Safe. We end up with the implementation and the discussion of performances.

## 3.2 Security measures for Local storage

The majority of existent solutions focuses their efforts on securing the Cloud infrastructure to guarantee the user data security. In our work, we give as much importance to the security of the local storage as the security of the Cloud storage. In this section, we present

the security measures which should be taken for the local storage.

It is essential to highlight that in this chapter, we focus mainly on applications which enable the synchronization of user data across his different devices. We do not consider the case of data sharing between different users as this context will have the full interest of the chapter 5.

### 3.2.1 Data encryption

Data confidentiality is one of the main concerns of users who store their sensitive data locally or externalized to the Cloud. The problem is how they can be sure that their data cannot be accessed by attackers who have the full access privilege to their machine or storage server. In this context, we focus on the different security measures taken by web applications and browsers.

#### Web application data encryption

In applications that are Cloud oriented data storage, encryption can be integrated into the different level of the storage process [94].

- First, we find the server-side encryption where the data are stored encrypted in the storage servers. In this case, the keys are usually user independent and are managed by the server owner.
- The second level is the data transmission. It is based on the encryption of the traffic between the client and the server. Some storage applications use merely the encrypted transmission such as Google Drive [18]. Others combine it with the server-side encryption, such as Dropbox [23]. In the case of Dropbox, the connection between the client and the server is secured with SSL/TLS and the uploaded data are stored in Amazon S3 storage service encrypted with AES-256 [94].
- The last level is the client side. Cloud-based storage services are fighting again the attacks originating from the servers that host their data. To this end, they are migrating toward the encryption of their data on the client side before externalizing them rather than adopting the server-side encryption. The servers store therefore, the data without any knowledge of their content. In this case, all the elements related to the security, such as the keys and the encryption functions, are managed by the client. Among the Cloud storage solutions that follow this strategy, we find CloudFogger [5] , Amazon S3 with SafeNet integration [16] and Wuala [17].

#### Browser data encryption

The measures of security taken by the browser are applied on browser data. It covers user password (stored for autofill in case of authentication to a web application), bookmarks, history, open tabs, etc. The browser ensures their security when they are stored locally in the machine (client side encryption) and when synchronized to the Cloud server. Besides the browser implementation, additional applications (1Password [1], LastPass [13]) can

be installed to manage and secure the web application password. As shown in the table 3.2, two techniques are adopted by the browser to encrypt their data locally. The first technique relay on encrypting data with a user independent key, while the second one uses a master key strongly linked to the user.

**Table 3.1** Security features of storage applications

| Web App | Key management | Encryption | Data Integrity | Metadata protection |
|---|---|---|---|---|
| **Web Storage application** | | | | |
| **Wuala** | PBKDF2 | Client side encryption. AES-256 for blob encryption, RSA2048 for signature and for key exchange | HMAC | YES |
| **Amazon with safeNet** | KMIP | Client side encryption. AES-256 for file encryption | NO | NO |
| **CloudFogger** | PBKDF2 | Client side encryption. AES-256 for file encryption | NO | NO |
| **Google Drive** | No key management | No data encryption. | NO | NO |
| **Dropbox** | key management with a user independent key | Server side encryption. | SHA-256 | NO |
| **Password Manager in Browsers** | | | | |
| **1Password** | PBKDF2-SHA-1 of a chosen master Password. It stores password in the Cloud | AES-256 | SHA-256 | NO |
| **LastPass** | PBKDF2-SHA256 of the Mail address and the password. | AES-256 | NO | NO |

### 3.2.2 Key management

Dealing with the data encryption, leads necessary to raise the issue of key management. As we focus, in this work, on the client side data security, we list the different strategies that can be adopted by Web storage application and by browsers.

#### Web application key management

Ensuring the key management and storage by hardware such as TPM [34] stills among the most secure solutions. However, this technique is related to the used machine as the key does not leave it. The commonly used techniques are as follows:

- Encryption key can be fixed by the application and stored locally in files on the user equipment;
- Encryption key can outsourced to be managed by third parties such as via the Key Management Interoperability Protocol (KMIP) [67];

**Table 3.2** Key management by different browsers

| Browser | Algorithm | Encrypted data | Keys storage |
|---|---|---|---|
| **Without master key** | | | |
| **Firefox** | Three Key Triple-DES | Stores encrypted username-password and plaintext webpage URL in the SQlite file signons.sqlite | Stores the Triple-DES keys in the key3.db file |
| **Opera** | Three key triple-DES | Stores encrypted username, password and URL in the Sqlite database Login data | the Triple-DES keys are stored in the SQlite database Login data |
| **chrome** | Operating System API | Stores encrypted password, username and webpase URL in the SQLite database file Login data | The password is encrypted using the Windows API functions. No additional entropy is provided to the Windows API. |
| **IE** | Operating System API | Stores encrypted password and encrypted username as a value data under Windows registery | The password is encrypted using the Windows API functions. |
| **With master key** | | | |
| **Firefox** | SHA-1 | Hach of the master key and the global 160 bit salt to obtain the master key | Master key used to encrypt the 3-DES keys before storing it to the file key3.db |
| **Opera** | SHA-1 | Hach of the master key and the global 128-bit salt to obtain the master key | Master key and 3DES keys are used to double encrypt passwords |

- Encryption key can be neither stored locally nor outsourced. They are generated instantly using some user information and passwords.

#### Browser key management

Regarding the user password, the five most popular browsers, Firefox, Google, Opera, Internet Explorer and Safari use different strategies (table 3.2). They either store the encryption keys in files in the user machine (such as key3.db for Firefox, wandet.dat for Opera) or use operating system functions like CryptoProtectData and CrypUnprotectData to perform encryption and decryption operations with predefined keys (Google Chrome, Internet explorer and Safari).

In other cases, the encryption key is not stored neither locally nor externally. It is generated from a chosen passphrase that must be remembered by the user. Specific algorithms are defined for this purpose such as the Password-Based key Derivation Function (PBKDF2) [75] and cryptographic hash functions. The two browsers Firefox and Chrome use the Master Password as second mechanism to strengthen the password security. Fixed by the user, it is used to generate the master key which encrypts the encryption key or double-encrypts the web application password.

### 3.2.3 Data Integrity

Clearly "Encryption without integrity-checking is all but useless"[50]. In fact, Guaranteeing the data integrity is being sure that data are not tampered by a third party without the

user consent. The data modification may be ensured through a malicious application or directly by accessing the files and the database where data are stored. To avoid the tampering attacks, Hash Message Authentication Code (HMAC) algorithm can be adopted such is the case of Wuala [17] or the Secure Hash Algorithm as in 1Password [1] and Dropbox [63]. Nevertheless, this safety measure is not being taken seriously by all the applications.

### 3.2.4 MetaData Integrity

It is not only the data which are concerned by the integrity as it should be even the case for the metadata. In fact, metadata such as the directory and the filename may be encrypted to protect the retrieval of their content. According the application data structure, this strategy avoids metadata tampering [53]. Some application adopts the metadata protection by encrypting the filename [17].

## 3.3 Enhancing the security of the HTML5 local storage

The Local Storage APIs in their current status can not be used as a fundamental basis of a secure storage of sensitive data. In this section, we start by highlighting the imminent interest of adding the security level into these APIs. Our enhancements are essential in general for web applications in different context, and in particular for web applications that ensure the storage of user data. We detail, in a second part, these security enhancements.

### 3.3.1 Imperative interest of securing the Local Storage

The Web has faced a revolution triggered by the new improvement of the HTML standard. With HTML5, innovative features and APIs are introduced to follow the business and user requirements. The main purpose is to provide the web developer with a vendor-neutral language that enables the implementation of competing application with lower cost. These applications are not related neither to the used devices nor to the installed plugins or software.

Recognizing the importance of the local storage in web application, HTML5 comes with three APIs that ensure the local data storage. These APIs differ according to the nature of stored data. Before the HTML5, the local storage is performed using the cookies or proprietary solutions.

In fact, client side storage [72] relies on storing web application data on the client equipment rather than the server. Thus, a part of the application code is transferred from the server to the browser. The HTML5 Local Storage APIs have several advantages:

- With the local storage, web application on offline mode becomes possible. Since all required data are stored locally, the user can continue interacting with the application

even when the network connection is off. The HTML5 offline application API [40] detects the interruption of the connection to switch to the offline mode. The different URLs of resources which constitute the web application are listed in a manifest file, and the resources are stored by the browser in the user machine. The HTML5 Local Storage APIs can be used in this context to store the web application data;

- Local storage provides richer and more interactive user interfaces [121]. It achieves also better performance and increases the user experience. In fact, storing the application resources locally adds the velocity into the access and management of these resources instead of bringing them from a long away;

- The transfer of the data to the client side removes unnecessary message exchanges between the client and the server. It is possible to use intelligent asynchronous requests to exchange only small parts of the data [72] [90];

- The server load is improved, and the service availability is increased [114]. In fact, a part of the code is transferred to the client side and a part of data stored by web application is distributed between the different users' devices.

Dealing with the W3C standard and browser APIs implementation, we notice that stored data are not protected enough and many attacks can risk their availability and confidentiality. Thus, we focus on the imperative need to assure the security of these data. However, the security is not the unique trouble. When a user moves from a machine to another, the data stored in the first used cannot be retrieved. As a consequence, the data should be synchronized following the user on his different equipments to ensure their availability. From these three points, security, synchronization and availability, begins our contribution. As illustrated in figure 3.2:

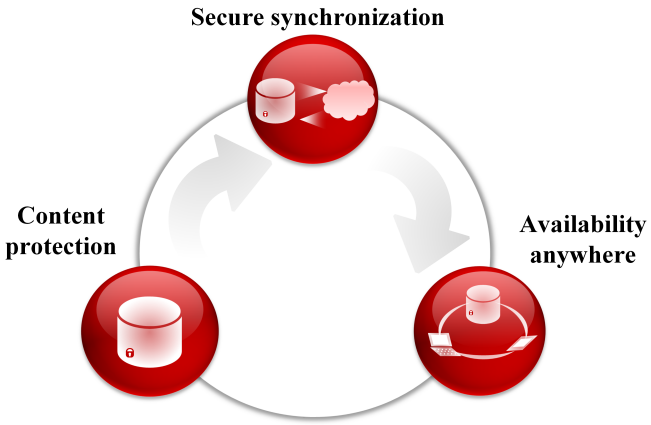



**Figure 3.2:** *Content protection and data synchronization*

- **Content protection** In a first step, securing the data locally is required. Therefore, we propose that the browser gives each user his own secure space where data and

some metadata are stored encrypted and ready for the synchronization. With this practice, only the user can have a look on his own data and only web application with the user permission can retrieve it;

- **Secure synchronization** Second, the data should be synchronized automatically, instantly and continuously when the user moves from equipment to another. Ensuring the synchronization means that data will be stored somewhere in the cloud. Thus, we notice the supreme interest of the first step and ensure the client-side data protection before the Cloud storage. The Cloud is considered as an encrypted blob store. Therefore, the storage provider cannot access to these data;
- **Availability anywhere** After the content protection in client side and data synchronization, the web application can access the data anytime and from any device. Of course, clients will be reassured that their data are securely stored and securely synchronized.

To highlight the necessity of data security and availability, we pursue with use cases which range from the simplest to the most complicated.

- Some websites are based on the HTML5 Local Storage APIs to autofill a user informations that can be public or confidential. The current browser API implementation stores this identifier in clear and only in local machine. With the security enhancement, the user does not worry about the confidentiality of these information and with the synchronization, information will be filed automatically even when the user changes the equipment.
- Externalizing files is one of the main services that came with the Cloud Computing. However, files are not the unique form that represents the user data. Data can be saved in databases in key/value form. For more complex one, they can be stored in an indexed database.
- We propose to overpass the typical use of HTML5 Local Storage APIs and use them for Cloud storage services. For further details, we can take the example of Dropbox. This application needs to be installed. It dedicates a specific part in the machine file system, where data are stored locally. With HTML5 Local Storage APIs, no application installation is required, and the web applications take advantage of the API already implemented by the browser. Data will be stored encrypted in every user machine and in the storage server.

### 3.3.2 Enhancement of the HTML5 Local data storage APIs

The first contribution of this chapter is the enhancement of the security of the HTML5 Local Storage API. The goal is to make these APIs in line with the Web application expectations in terms of secure storage.

### Global system structure of the proposed framework

Through the figure 3.3 are presented the different entities of the proposed system, as well as the interactions between them. The different entities that interact are:

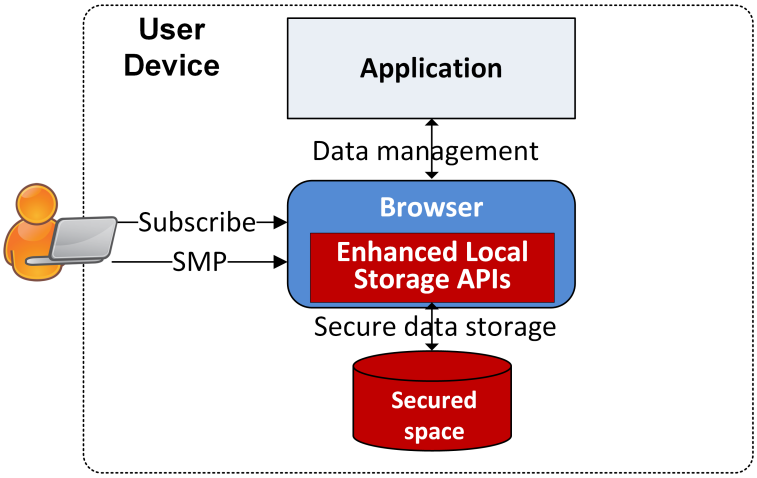



**Figure 3.3:** *System structure for data protection*

- **The user**: In our proposal, the user should, in the first step, introduce his ID and password to be authenticated by the browser. In the case of the chrome browser and Firefox, the user is identified by his email address and its corresponding password. The user is asked also to introduce a Strong Master Password. As we will see later in details, this password will be used to generate the key of encryption;

- **The browser**: With security consideration, we propose to alter the browser with its Local Storage. In fact, the browser is the main program which enables the user to visit the Web application using its address. The whole HTML5 APIs are implemented at this level following the W3C specification. In the context of data storage, the browser is the intermediate engine between the web application and the storage space in the device. Therefore, It enables the application to manage the data stored locally;

- **The Web application**: Implemented using the JavaScript language, the Web application follows the predefined functions to store or to recover the data stored by the browser. These functions depend on the nature of managed data and the used Local Storage API. For security consideration, this management can be achieved only with user permission;

- **The enhanced Local Storage APIs**: These APIs are the main concern of our contribution. The goal is to add a security layer to the stored data. The Local Storage APIs of HTML5 are mainly the WebStorage, the indexedDB and the FileSystem API. After the analysis of the security gaps of these APIs, we emphasize the need of

adding the data confidentiality, the data integrity and the metadata integrity. These security features are detailed later.

- **The secured space**: The data stored by the Local Storage APIs can be found in the file system of the user. They are stored in clear without any integrity verification. With our security enhancement, we affirm that data are stored secure.

Every user will have his own Strong Master Password (SMP)[122]. Different techniques can be used to derive the encryption key from a simple passphrase. They differ according to the strength of the generated key against brute force attacks.

In our system, we rely on the PBKDF2 (formula (3.1)) as a key deviation function specified in the password-based Cryptography Specification PKCS#5 [75]. The salt guarantees that different keys generated from the same SMP are strongly independent. The iteration with a high count is used to secure the data against brute force and dictionary attacks [122].

Using the DerivedKey, the browser encrypts or decrypts the web application data regardless the used device. In our case, the user is the unique entity that has the required elements to generate the encryption key.

$$DirevedKey = PBKDF2(SMP, Salt, Iteration, kLen) \tag{3.1}$$

*where:*

***PBKDF2**: Password-Based Key Derivation Function 2*
***SMP**: Strong Master Password*
***Salt**: The salt is generated randomly for the derivation.*
***Iteration**: Number of intern iterations for the derivation*
***kLen**: Derivation Key length*

### The added security features to the HTML5 Local Storage APIs

As defined in the W3C standard, the unique security strategy for the HTML5 Local Storage APIs is the single Origin Policy (SOP). With this strategy each domain saves its data in a separated database as it is illustrated in figure 3.4.

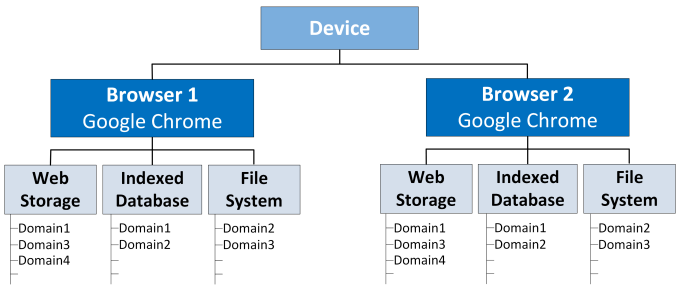



**Figure 3.4:** *Management of local storage*

We propose to strengthen the HTML5 Local Data Storage security by enforcing a user dependent security. As it is depicted in figure 3.5, the user is added to the storage

strategy. In this case, a specific storage space is defined for each subscribed user. This space guarantees the data confidentiality, data integrity and metadata integrity.

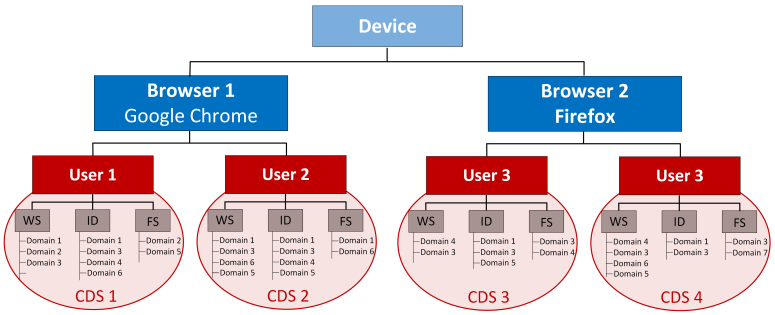



**Figure 3.5:** *User integration in the management of local storage*

- **Data encryption**

  Figuratively speaking, the single key to encrypt and decrypt data is the Strong Master Password (SMP). For this reason, the derived key or the passphrase should not be saved in plaintext in the user machine.

  As we expect to synchronize the HTML5 Local Storage data, this Client Digital Safe represents a double-edged sword. First, it ensures the security of data when stored in the user machine. Thus, there is no risk to recover data without the SMP. Second, it protects data when synchronized and stored in the Cloud as they are externalized encrypted.

  To deal more in detail with the encryption, only the web application data will be stored encrypted in the file database and not the whole file. This limitation saves the database global structure and its basic interaction with the browser. In fact, in case of data storage by the HTML5 APIs, data are encrypted by the browser in the first step then saved in the appropriate file of the database. The reverse process is considered in the case of data retrieval. The browser recover the data from the database file then decrypt it.

- **Metadata integrity**

  In the context of the HTML5 local storage, data encryption is not enough again the metadata tampering [53] and data recovery. To avoid this kind of attacks, we propose to encrypt the metadata of database files using the same SMP derived key previously adopted for data encryption.

  In fact, the database, where the web application data are stored, uses the domain as a part of its files name. The metadata tampering can be achieved following these steps:

  – An attacker can alter the name of the database files related to another domain. For example in the case of the WebStorage API, it changes the SQlite File name from domain-D1.sqlite to domain-attacker.sqlite;

  - The browser will consider that the data stored there match to the attacker domain;
  - The domain of the attacker gains accidentally the permission of the user to manage local data;
  - The user introduces his SMP to allow the web application of the attacker to manage his data. As a consequence, the attacker will have the full access to the encrypted data of the domain D1.

  These attacks are no longer possible with the metadata encryption as the attacker cannot predict the name of the database files.

- **Data Integrity**

  In the basic implementation of HTML5 Local Storage APIs, any attacker, who has the full access to the machine, can modify data from the database files. Thus, no integrity verification is considered. To strengthen this characteristic, a new event log file is added into the database folder. This guarantees the data integrity by computing the hash of the database after every data management. Concisely, we insert, for every event of data management, the timestamp, the domain name that initiates the event and the cryptographic hash of the database. Data integrity can be verified by comparing the last stored hash with the newly computed one. The event log file can be in turn tampered, that is why we chose to encrypt the information stored there using the user derived key.

## 3.4 Client Digital Safe based on HTML5 Local Storage APIs

After providing a secure basis to store data by Web application in the previous section, we present the conception of the Client Digital Safe in this section, which is the subject of our second contribution. We start by presenting the major interest of our standardized Client Digital Safe compared to the existent solutions. We detail in the second part, the entities and the specifications of our safe.

### 3.4.1 Motivation of a Client Digital Safe based on HTML5

On the one hand, the HTML5 standard comes with the Local Storage APIs to enable the storage of data locally in the user device. In the other side, the standard AFNOR defines the specifications of a Digital Safe. In our contribution, we mix the both standards in the context of local data storage and we propose a standardized Client Digital Safe based on the HTML5 API. By choosing our storage solution, the user decides to have the full control on his data and digital documents. Simple to implement for service providers

and to use for clients, this digital safe meets the requirements of the digitization and the communication of all kinds of digital content. In fact, the characteristics of our **Client Digital Safe** is as follows:

- It is a standardized Digital Safe solution which meets the AFNOR specification [2] first, in terms of structure and format, second in terms of provided security services;
- It integrates the interoperability between the standardized Cloud Digital Safe and the HTML5 Local Storage APIs. In fact, the Cloud Digital Safe takes into account the requirements of the Digital Safe, the technical characteristics of Cloud Computing and the specification of the AFNOR standards. Regarding the HTML5 Local Storag APIs, they provide a client side storage of user data with all its forms;
- It is an alternative service which replaces the commercialized storage solutions and presents remedies against frequently encountered problems in terms of standardization, transparency, mobility efficiency and security.

### Standardized structure

The conception of our Digital safe is based on the HTML5 and AFNOR. They are the two main standards used and merged within our framework:

- **HTML5**: HTML5 is the first major update to the HTML specification. New features for multimedia, interactivity, smart forms are introduced. HTML5 still supports all HTML features describing not only a new version of this specification, but a series or combination of technologies designed to make the Web much richer and more attractive to interact with (through the introduction of API).
- **AFNOR**: The AFNOR standard is used for archiving sensitive documents while guaranteeing integrity over time. The Digital Safe specifications are published under the standard NF Z42-020. The main reason behind the adoption of the AFNOR specification is that it is better to integrate standardized solution than building customized one.

### Transparency

Transparency is one of the important characteristics of an application and a framework. In this context, transparency means the complete predictability of the application toward the users. The output of each operation performed on the Client Digital Safe is completely expected for a particular input. Frameworks adopting standardized solution in general and HTML5 standard, in particular, have the advantage of preserving the transparency with the end user and the service provider.

### Mobility and portability :

Users are owning multiple devices used in different contexts and goals. Thus, it is tedious to follow each device requirements when installing any software.

This is the case of the majority of existent storage solutions. They propose different versions according to the used device (mobile and desktop), and according to the OS installed on the device (Linux, MAC, Windows, Android, IOS, etc.). In the context of mobile devices, the native applications are used. They can be downloaded from application stores such as App Store and Google Play.

To overpass these difficulties, we propose a client side storage solution which is based primary on the HTML5 standard. The use of web applications guarantees that all the service is controlled by the service provider. Therefore, the compatibility issues between the user terminal and the service provider network are avoided. The benefits of using web applications include:

- No software installation is required;
- Automatic update without complex procurement process;
- A wide variety of operating systems (OS) is supported;
- Availability on all devices with a compatible Web HTML5 engine.

**Efficient traffic exchange** :

Dropbox application, when installed locally in the desktop, exchanges permanently data with the centralized distant storage servers. This exchange is performed in the background often without the user knowledge. In our case, the user has the full control on the application and on the traffic. The access to the distant Digital Safe and the synchronization between the local Digital Safe and the Cloud Digital Safe starts when the user opens the Web application and authenticates to the service.

The efficiency of ou framework raises with the choice of the communication protocol between the local and Cloud Digital Safe as well as the structure of the exchanged data. This is the main focus of the chapter 4.

**Security** :

To win the trust of the user and boost their confidence, it is essential to give a high priority to the security of his stored data. Our Client Digital Safe is the image of the Cloud Digital Safe on the user device. Therefore, this projection conserves the safety rules and specifications defined on the standardized Cloud Digital Safe. Data are stored encrypted and externalized encrypted, integrity of data is preserved over the time and metadata integrity is guaranteed.

### 3.4.2 Conception of the Client Digital Safe

The concept of a standardized Digital Safe, was introduced in [93]. It follows the AFNOR specifications[2] which published the specification under the standard NF Z42-020 [30]. Considered as a subset of NF Z42-013, this standard offers the best features of security, integrity and quality to preserve the user data in a Digital Safe Component. The requirement of the Digital Safe is to guarantee the integrity of the stored data over time. It provides a secure environment for storing sensitive document. This environment full fit both the user requirements and Cloud security challenges. It is characterized also by its

probative value as a proof of data storage is stored in a third trusted party.

Our proposed Client Digital Safe accomplishes the standardize characteristic by following the AFNOR standard. Therefore, we detail the networking and deployment architecture of our framework.

### Client Digital Safe: networking architecture

Our Client Digital Safe is the image of the Cloud Digital Safe on the user device. Therefore, it should have the same characteristics and specification of the Safe hosted in the Cloud. Different from the existent solution, our Client Digital Safe is distinguished by its non-proprietary character. In fact, it is based on HTML5 Local Storage APIs.

In the figure 3.6, we present the networking architecture of our framework. We start by presenting the Cloud Digital Safe component, then those of the Client Digital Safe.

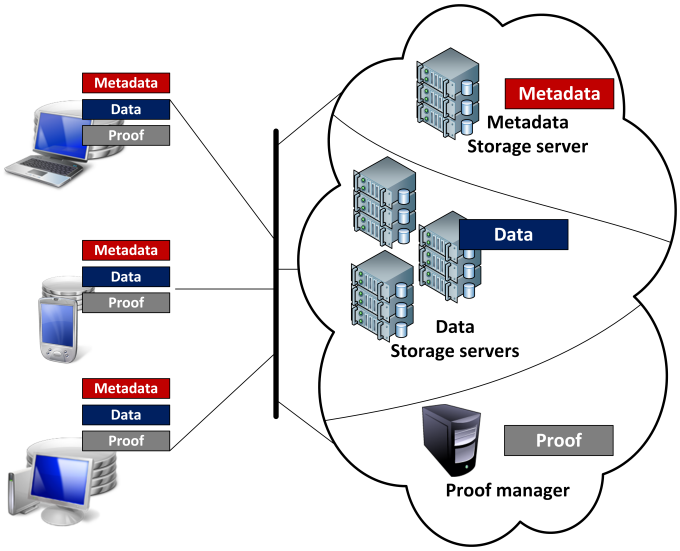



**Figure 3.6:** *Networking architecture*

The Cloud Digital Safe is composed of three main components, which are:

- **Data Storage server**: The user data are divided into blocks and stored in servers. These storage servers should have large scales and many Cloud storage services can be used such as Amazon S3 [3], Hadoop Distribute File System (HDFS) [54], Google File System (GFS) [101], etc. Usually, a Cloud storage service should satisfy the data availability, high reliability, security, fact access to the data, etc. These system should support, therefore, data distribution, SLA matching, Quality of service, authority assignment, audit, certification and access control;

- **Metadata Storage server**: Each file has his metadata which can be classified into three kinds:
  - Technical metadata: information related to the object representation, integrity information, and identification informations;
  - Management metadata: describing the rule that constitutes the descriptive metadata,

the service level that involves the availability and security, the rules of accessibility to the objects and the duration beyond which the data are eliminated or conserved for eternity;
- Descriptive metadata: includes the descriptive informations of each object. It implies sustainability informations like objects origin, object description, date, key words, etc. These data follows a model defined in the management metadata.

A metadata in the case of a Digital Safe is an XML file that lists the different file information. These XML files are kept stored separated from the file blocs in a server managed by the Digital Safe service provider;

- **Proof Manager**: It is a trusted third party Proof Manager that preserves the proof of data storage. The proof consists on metadata signed by the owner using his private key. This proof guarantees the non-repudiation and adds the probative values into the storage. Therefore, it can be used in case of litigation.

Three kind of information are managed by the Cloud Digital Safe which are: the metadata, the data and the proof of storage. These information are stored respectively in the Metatada storage server, Data storage servers and Proof manager. Nevertheless, the HTML5 Storage APIs as defined by the W3C standard can not be used in the context of the Digital Safe. That is why, additional features should be considered by these APIs. These APIs should be used to store :

- the metadata following the structure of those stored in the Cloud Digital Safe;
- the data following the AFNOR specification. In the context of the Digital Safe, the data of the user are stored in a file system with a set of directories and files. Thus, we focus of the HTML5 FileSystem API among the whole Local Storage APIs ;
- the proof of the storage to be send then to the Proof manager. In our case, a part of the proof will be considered as the record of the different operations performed on the client Digital Safe in a log file.

### Client Digital Safe: deployment architecture

The deployment architecture is created by mapping the logical functional blocks of the Digital Safe to a physical environment in order to respect the requirement of the networking architecture and the Digital Safe specification. As depicted in figure 3.7, in the deployment architecture, we add the following entities at the application level:

- **The synchronization API**: It ensures the communication between the Client and Digital Safe to synchronize their data content. This API has the full access to the stored data and their metadata and to the log file where stored the storage proofs. The synchronization protocol is detailed in the chapter 4;

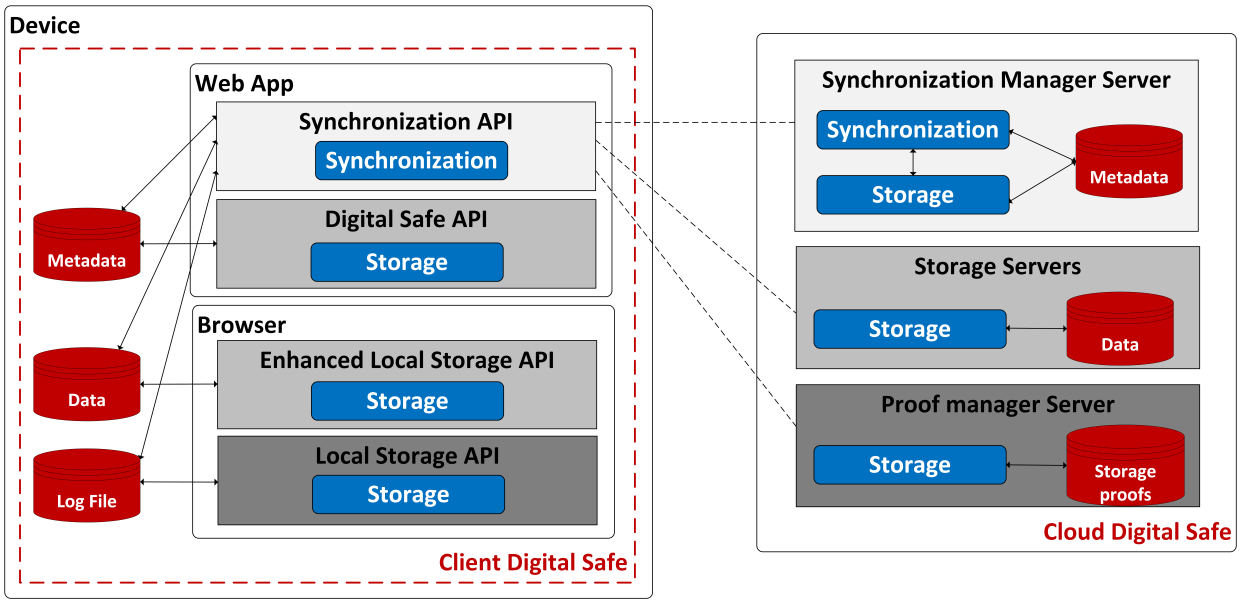



**Figure 3.7:** *Deployment architecture*

**Table 3.3** functions of the Digital Safe

| Function | Description |
|---|---|
| **Function for one Digital Object** | |
| **Deposit** | This function aims to insert a Digital Object into the Digital Safe Component after checking the rules of the user and his legitimacy to perform this action. |
| **Read** | This function aims to obtain a complete copy of one Digital Object that are already stored in the Digital Safe Component. |
| **Drop** | This function aims to make inaccessible of one Digital Object already stored in the Digital Safe Component. All records in the log related to this Digital Object are not affected by the destruction. |
| **Read Metadata** | This function is designed to obtain the technical metadata associated to one Digital Object already stored in the Digital Safe Component. |
| **Control** | This function is used to verify the existence and integrity of one Digital Object already stored in the Digital Safe Component. The verification concerns the existence of the Digital Object and its non-alteration since the insertion into the Digital Safe Component. |
| **Function for one or multiple Digital Objects** | |
| **Read log** | This function is used to get all the information stored on the log file associated to a Digital Object preserved or has been preserved in the Digital Safe Component. |
| **List** | This function aims to obtain a list of unique identifiers associated of the different Digital Objects stored in the Digital Safe Component. |
| **Count** | This function is designed to get the number of the Digital Objects stored in the Digital Safe Component. |

- **The Digital Safe API**: To manage the Digital Object, the client can use a set of functions. These functions are described by the standard AFNOR NF Z42-020. To be conform to the standard, the Digital Safe Component must implement these functions. An exhaustive list of these actions with there detailed description is shown

in the table 3.3. Of course, before any execution by the system, it is crucial to verify that the user has the right to execute the action according to his profile, and the access control policy.

At the level of this API starts the conception of the Client Digital Safe. It defines the different specification of the safe following the AFNOR standard. This API ensures the interoperability between the HTML5 Local Storage APIs and the Cloud Digital Safe. It is added at the Web application level and it is managed by the service provider. The specifications of the Local Storage APIs focus on the three HTML5 API: fileSystem API, IndexedDB API and WebStorage API.

In the table 3.4, we compare the specifications of the AFNOR standard and the specifications of the HTML5 Local Storage API as defined by W3C. We can say that the Local Storage APIs as they are defined cannot be used in the context of the Digital Safe. In fact, a part of the Digital Safe methods are managed by the HTML5 APIs and another part is not taken into consideration such as Read Metadata, Control and Read log functions (pointed by the X symbol in the table 3.4).

- **Deposit**: The deposit function is provided by the different HTML5 Local Storage APIs. The user can therefore, store files locally in appropriate directories using a virtual file system. He can also store data as either in key/values format or in an indexed database;
- **Read**: The data retrieval is ensured by the different HTML5 Local Storage APIs;
- **Drop**: The removal of the data stored locally under all its forms is guaranteed by the different HTML5 Local Storage APIs ;
- **Read Metadata**: The metadata retrieval is related only to the FileSystem API. Neither the WebStorage API nor the Indexeddb API are concerned. However, the format of files and directories metadata is completely different from the one defined in the Cloud Digital Safe. With getMetadata method, the web application can retrieve few information on the file such as its size and its last modification time. Regarding the standardized Cloud Digital Safe, it stores the metadata of one file in a separated XML file following the standardized Metadata Format of Dublin Core [27]. Bellow, in figure 3.8, an example of a file metadata as should be stored in the Digital Safe. In this example, the metadata are sorted into four main categories: information related to the file, information related to the storage, information about the access control policies and finally information about the synchronization which will be the focus of chapter 5;

  **Remedies**: As the data management is ensured at the Digital Safe API, we decide to add the construction of the metadata following the standard at this

**Table 3.4** comparison between NF Z42-020 and HTML5 Filesystem API specifications

| Specification in the standard NF Z42-020 | Specification in the W3C of Local Storage APIs |
|---|---|
| **Deposit** | - dirEntry.getDirectory (path, create:true, opt_success Callback, opt_errorCallback);<br>- fileEntry.createWriter(successCallback, opt_ errorCallback); |
| | - put(Index, value); |
| | - localStorage.setitem(key, value) |
| **Read** | - fileEntry.file(successCallback, opt_ errorCallback); |
| | - get(Index); |
| | - localStorage.getItem(key);<br>- localStorage.key(n) |
| **Drop** | - fileEntry.remove(successCallback, opt_errorCallback);<br>- dirEntry.removeRecursively(successCallback, opt_ errorCallback); |
| | - delete();<br>- clear(); |
| | - localStorage.removeItem(key);<br>- localStorage.clear() |
| **Read Metadata** | fileEntry.getMetadata; |
| | X |
| | X |
| **Control** | X |
| **Read Log** | X |
| **List** | - dirEntry.createReader.readEntries(successCallback, opt_errorCallback); |
| | - get(Index); |
| | - localStorage.getItem(key);<br>- localStorage.key(n) |
| **Count** | - dirEntry.createReader.readEntries(successCallback, opt_errorCallback); |
| | - get(Index); |
| | - localStorage.getItem(key);<br>- localStorage.key(n) |

level. To add this metadata file, we chose to add for each created file an adjacent one where the metadata information are stored. The title of this file is the concatenation of the created file name with the string "_metadata.xml". The management of the metadata will, therefore, be done automatically according to the performed operation.

- **Control**: The verification of the integrity of the files is not considered by the Filesystem API. This function gives the user the possibility to control if the objects stored in the Digital Safe still in their original version without being altered by an intruder. This intruder can be either a user who has the access to the machine and to change the database file of the APIs or a malicious web application.

  **Remedies**: We have already enhanced the HTML5 Local Storage in the previous section where the data integrity remains essential, The integrity verification, in this case, is integrated into the Local Storage APIs;

- **Read log**: The log file is not integrated into the conception of the FileSystem

API. In fact this file allows to keep traces of the different operation performed on the Client Digital Safe.

**Remedies**: We choose to use the WebStorage API to store the historic of operations performed on the client side ;

- **List and Count**: No direct instruction is provided by HTML5 APIs to retrieve the list and the number of the stored data.

  **Remedies**: The Local Storage APIs instruction which ensure the retrieval of data, should be used in a loop while used in the Digital Safe API to detect both the list of existent files and their number;

```
<?xml version="1.0"?>
- <Metadata>
  - <File>
      <Title> The title of the file </Title>
      <Author> The user Id that own the file </Author>
      <FileID>The file uinque identifier </FileID>
      <Path> The path of the file in the conatiner </Path>
      <ContainerID> The container that contains the file </ContainerID>
    </File>
  - <Storage>
      <Format> The format of the file </Format>
      <Size> The total size of the file </Size>
      <Sensitivity> Sensitivity of the file</Sensitivity>
      <Number_of_Duplicates> number of duplicates </Number_of_Duplicates>
      <Description>Desription introduced on files </Description>
      <Start_Date_Storage>The date of file archiving </Start_Date_Storage>
      <End_Date_Storage> The date of file erasure </End_Date_Storage>
    </Storage>
  - <Rights>
      <Role>The role of the user </Role>
      <Right> </Right>
      <Encrypted_key>The encryption of the key used for file encryption</Encrypted_key>
    </Rights>
  - <Synchronizer>
      <hash> hash of the file </hash>
      <Sync_version> The version of the file synchronized </Sync_version>
      <Last_modification> The date of last modification made on the file </Last_modification>
    </Synchronizer>
</Metadata>
```

**Figure 3.8:** *File metadata in the Digtal Safe*

- **The Enhanced Local Storage APIs**: These APIs match to the enhancements made to the basic Local Storage APIs with the following security considerations: data confidentiality, data integrity and metadata integrity. These enhancements are subject of the first contribution of this chapter.

  The Digital Object managed by the Client Digital Safe should be limited to files and directories. This fits exactly to the kind of objects managed by the HTML5 FileSystem API.

- **The WebStorage APIs**: It is important, first, to add a log file where the slightest modification to the Client Digital Safe is kept there. Second, the Safe owner should be able to access the log information. For the creation of the log file, we will use the WebStorage API. The historic will be therefor stored in this file. For each modification made on the Digital Safe, we capture the timestamp, the nature of the action and the object concerned by the action. The historic of the operations will be used later for data synchronization. It will be maintained for a period of time imposed by the server provider.

## 3.5 Implementation and evaluation

The objective of our work is to propose a standardized Client Digital Safe based on HTML5. To achieve this goal, both contributions are the subject of this chapter. In the first one, we enhance the security of HTML5 Locla Storage APIs. Our enhanced APIs are the basis of the second contribution where we propose to follow the standard specifications for the conception of our Client Digital Safe.

After the conception comes the implementation step. As a proof of concept, we implemented our enhancement of the Local Data Storage APIs and our Local Digital Safe in the Chromium browser. Chromium is chosen, among the existent browsers, because first it is the open source behind the most used browser Google Chrome. Second, it supports the local storage HTML5 APIs. Our contribution can be performed on any browser. However, the purpose of this implementation is to prove the feasibility of our approach.

This browser is installed on a virtual machine with dual 3.5 GHz processors and 5 GB of memory running in a 64 bit Windows 7 OS.

### 3.5.1 Data protection into the Chromium browser

Among the Local Storage APIs, we focus in this part on the WebStorage API. In our case, no modifications are involved into the conception of the Local Storage APIs. The idea is to add new functionalities to enhance the security management of stored data by being fully compliant with the standard. The changes brought to the browser code have affected three files of Chromium which are:

- *"/content/renderer/dom_storage/webstoragearea_impl.cc"*
- *"/content/browser/dom_storage/dom_storage_database.cc"*
- *"/content/browser/dom_storage/dom_storage_area.cc"*

We present in figure 3.9 an architectural overview of the Chromium browser part which ensures the WebStorage API functionalities. In addition, we highlight the enhancement introduced into this browser by adding our main security elements (red blocs) which are: the data encryption, the metadata and the data integrity.

The architecture of the WebStorage API enhancement (figure 3.9) is composed of four main elements:

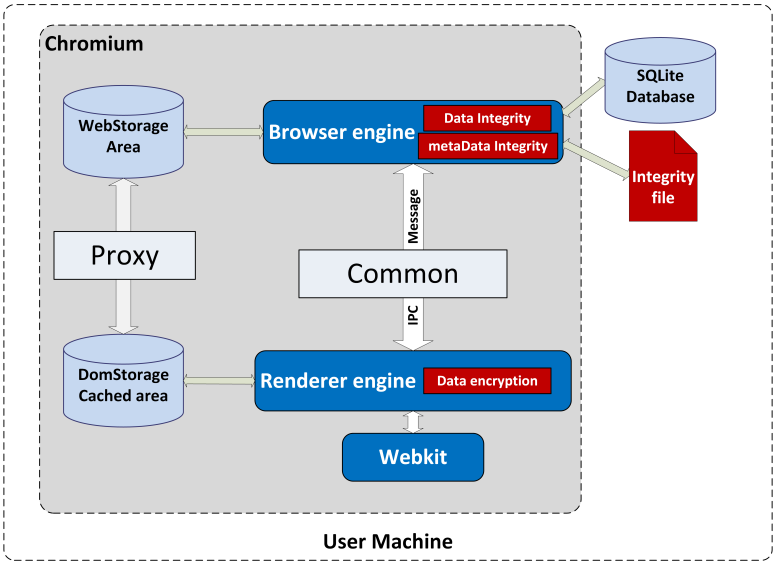



**Figure 3.9:** *Chromium DomStorage Architecture with the HTML5 data protection*

- Browser engine: It is the main process and it gathers all the user interface and I/O. It is the engine that stores and recovers the API data in the SQLite database. As explained in the chapter 2, the SQLite databases of the different domains are separated using the domain name in the database name. In the *"dom_storage_database.cc"* file to guarantee the metadata integrity, we choose to create the database with a name that matches to the encryption of the domain using the key derived from the SMP. In addition, after each modification, we compute the hash of the database content and we store it in the Integrity file. In case of access to the database, the current hash is compared to the last stored. In fact, this verification reassures that no modification was performed directly from the user filesystem and that the data managed was achieved from the web application. These enhancement are ensured at the "dom_storage/dom_storage_database.cc" file.

- Renderer engine: It stores a copy of WebStorage API data collected from the Webkit directly in the DomStorage cached Area. This cached area is intended to be used in the renderer processes. It contains a complete cache of the stored data for fast access. To secure the data, we add the data encryption in the renderer engine. Thereby, encryption occurs before data storage in the DomStorage Cached area with the SetItem function. Regarding the decryption of data, it happens when the GetItem function is called in the renderer.

- Webkit: It consists primary of the webcore that represents the core layout functionality and the JavascriptCore that runs JavaScript. No modifications are attributed at this level.

- Common: Shared between the browser and the renderer. It ensures their communication through Inter-process Communication (IPC). No one modification has been

made in this entity.

### 3.5.2 Integration of the Client Digital Safe into the Chromium browser

As we have detailed previously in the deployment architecture, the Client Digital Safe concerns mainly the FileSystem API. The implementation of our Client Digital Safe concerns two parts, the chromium browser to enhance the security of the FileSystem API and the Web application where the Digital Safe specifications are implemented. Therefore, as depicted in figure 3.10, we focus on the Chromium components that ensure the management of the virtual file system. We focus also on the web application to implement the Digital Safe API. Our contribution and implementation are highlighted in figure 3.10 by the red blocks.

#### Implementation at the browser level

This architecture of the browser is structured as follow:

- Webkit: The main code of the FileSystem API is hosted in the Webkit. In fact, the virtual file system with its data base files are created at this level. This component receives the Javascript code to interpret it. It sends then the messages of the FileSystem API into the browser engine and more specifically the FileSystem bloc;
- Browser engine (FileSystem bloc): This engine receives the messages related to the FileSystem API. It filters them and executes the matched actions on the virtual file system. The main file that handles the FileSystem operations is *"content/browser-/fileapi/fileapi_message_filter.cc"*. It is at this level that we integrate the main Digital Safe requirements. Thus, we add:

  - the encryption of the files before their storage in the Client Digital Safe;

  - the verification of the file integrity. As the hash of the file are computed in integrity file, this function verifies that the hash of the current file remains the same as stored previously. In case of modification, the hash is updated;

  - the encryption of the file name to guarantee the metadata integrity.

#### Implementation at the Web application level

Following the deployment architecture, we add a new Digital Safe API based on Javascript. At this level, we find the different functions that match to each specification of the standard.

These function are called from the main web application code to manage the different files and directories stored in the Client Digital Safe.

As we detailed in the Client Digital Safe conception, some of these functions are already supported by the FileSystem API. Therefore, a simple call with the appropriate attributes is performed. When the function is not ensured by the HTML5 API, we define new one. It is the case of the following functions:

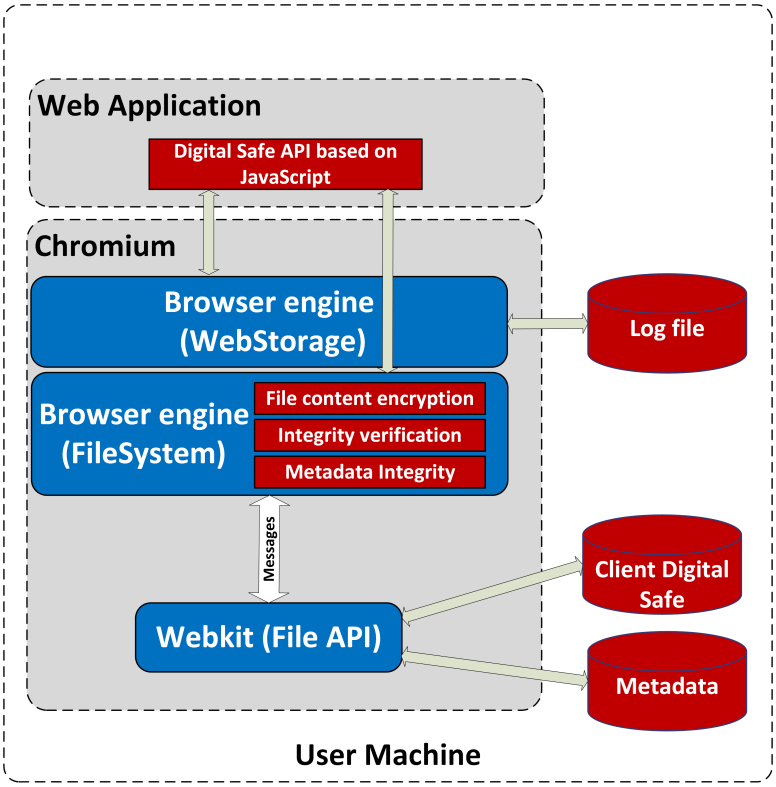



**Figure 3.10:** *The Client Digital Safe implementation*

- DigitalSafe.Deposit(file_name, Path): In addition to the creation of the file using the FileSystem API, we add the creation of the metadata following the structure defined in the Digital Safe. We add also the action in the log file;
- DigitalSafe.ReadMetadata(file_name): This function reads the metadata of each stored file;
- DigitalSafe.ReadLog(): This function displays the log file content to the user.

### 3.5.3 Results and performances discussion

For performance evaluations, we compare the time needed to achieve the data management of our solution with the Webstorage API and Dojox toolkit [7]. In fact, Dojox proposes a secure client side storage by setting a passphrase used for encryption. However, this technology does not ensure neither data integrity nor metadata integrity. Therefore, we implement three HTML5 applications that store the same data amount using respectively the basic WebStorage API, the secured WebStorage API and the secured Dojox storage toolkit.

- The first application stores the data using the basic WebStorage API as defined by the W3C standard;
- The second application stores the data using our enhanced WebStorage API with the security considerations (data encryption, data and metadata integrity);

- The third application stores the data using the Dojox Storage. Dojo toolkit [7] is an open source modular JavaScript library created to make the development of cross-platform, Ajax-based applications and websites easier and quicker. It proposes a new syntax, to store data encrypted in the browser.

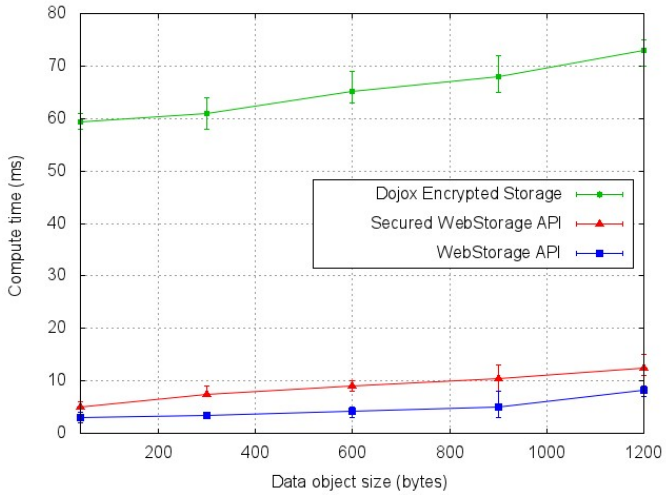



**Figure 3.11:** *Performance evaluation of the SetItem operation*

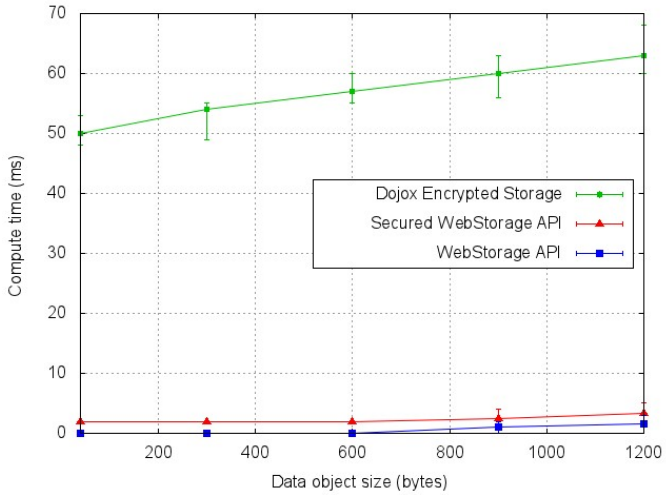



**Figure 3.12:** *Performance evaluation of the GetItem operation*

To analyze the performances, we capture the time needed to achieve two of the main CRUD (Create Read Update Delete) operations: reading and writing. In fact, the Update operation gives the same results of the Create operation, and the Delete operation gives the same results for the three different applications.

We repeat the operations around forty times to have distinct final results. The figures 3.11 and 3.12 expose respectively the computation time when setting and getting data. The axis of X and Y match respectively to the data size in bytes and the time of operation execution in Milliseconds.

Compared to the Dojox's solution, our secured WebStorage API exhibits better performances with a very low operation achievement time. Ensuring the confidentiality and integrity of data stored by the HTML5 WebStorage API comes at a weak cost. As we can notice, operation execution time when data are secured is higher than the operation execution time of the basic WebStorage API.

The performance degradation is explained by the fact that addition preprocessing is applied on data. However, in our case, it stills slightly higher if we compare it with Dojox Storage Toolkit. In front of the full data protection interest, the minor performance degradation can be tolerable.

## 3.6 Conclusion

The HTML5 Local Storage APIs have proven their ability to improve the web application performances and the quality of experience while giving up the imperative need to focus on the data security and availability.

In this chapter, we mix the HTML5 standard and the Digital Safe standard to propose a standardized Client Digital Safe. In the first part, we present the security measures which should be guaranteed by the local storage. In the second, we give the security gaps of the Local Storage APIs and we enhance their security. After the security enhancement, these APIs are used, in the third part, to define the Client Digital Safe.

We implement the enhanced Local Storage APIs and the Client Digital Safe. Implementing our proposition in the Chromium browser and evaluating performance prove that adding the data protection into the Local Storage APIs is crucial and efficient. In next chapter, we focus on the protocol that ensures the synchronization between the Client Digital Safe and the Cloud Digital Safe.

# Chapter 4

# SyncDS: A Digital Safe Based File Synchronization Approach

## Sommaire

## 4.1 Introduction

One of the main concerns of Cloud storage solutions is to offer the availability to the end user. Thus, addressing the mobility needs and devices variety have emerged as major challenges. At first, data should be synchronized automatically and continuously when the user moves from one equipment to another. Secondly, the Cloud service should offer to owners the possibility to share data with specific users.

While the client Digital Safe is the main concern of the previous chapter, in this chapter, we propose a secure framework and protocol called SyncDS (Synchronization of file in the context of Digital Safe) that ensure file synchronization between the Client and Cloud Digital Safes with high quality and minimal resource consumption. As shown in figure 4.1, this contribution represents 30% of the thesis works.

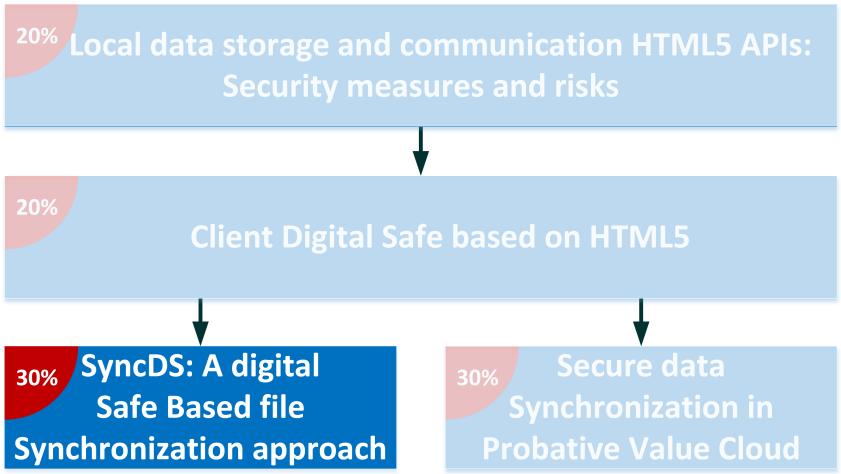



**Figure 4.1:** *Structure of the thesis*

The chapter is organized as follows. The second section surveys synchronization protocols. The third section deals with our SyncDS protocol and its architecture. The section four focuses on efficiency of the protocol and more precisely on the strategy of changes detection between two versions of file systems located in different devices. We highlight in section five the adoption of the SyncDS protocol in the context of Peer-to-Peer. The last two sections address the implementation and the analysis of our protocol SyncDS.

## 4.2 Synchronization protocol: principle and requirements

### 4.2.1 Overview on synchronization protocols

Different strategies can be adopted to ensure data availability facing the user mobility and devices variety. As shown in figure 4.2, these strategies can be classified into two principal categories: centralized and decentralized strategies and each one can be, in turn, sorted into replication and distribution.

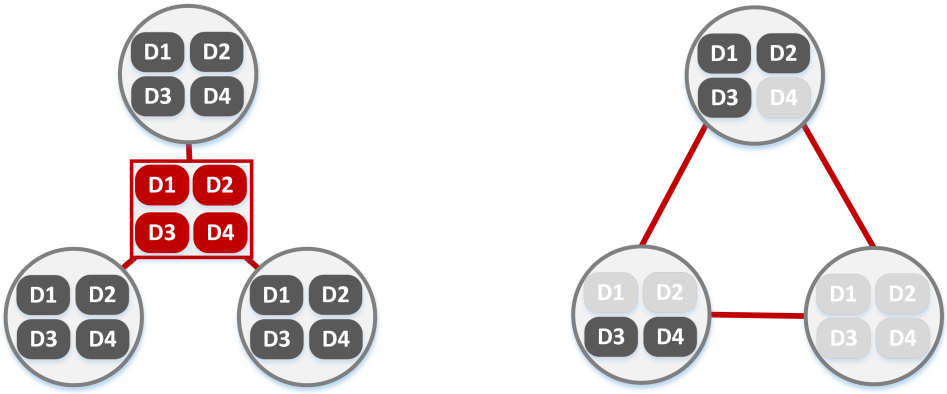



**Figure 4.2:** *Centralized-replication and Decentralized-distribution strategies*

- **Centralized strategy:** This strategy relies on centralizing the entire data. Therefore, they are completely externalized and stored in appropriate servers. Two cases exist. In the first one, data are stored only in the servers. When a peer needs any information, it retrieves the data directly from the storage server [93]. In this case, a continuous connection should be available to manage the data. In the second case, a copy of user data is stored in his machine too. Thus, the user can manage his data when he is offline and a continuous synchronization is performed when he is online.

- **Decentralized strategy:** The privacy offered by popular Cloud file synchronization and replication services is becoming the concern of the press. In fact, some services have recently been reported as sharing information with governmental security agencies without warrants [69]. To overcome these deficiencies, synchronization solutions are migrating to completely decentralized services using P2P protocols. Data are, therefore, stored in personal devices without the need of third parties. Besides the security, these measures rely on the user devices without a need to pay for external storage solutions.

- **Replication:** In case of replication, data are completely synchronized. Any modification that occurs in one device should be applied to the rest of user devices. As a result, the full data are present in every device. This replication can either be completely decentralized based on the P2P network such as bittorrent sync [69], SuperNova [103], PeerSoN [55] or can rely on third parties that manage the synchronization [73] such as Google drive and Dropbox.

- **Distribution:** Regarding the distribution, each device holds a portion of data. The responsibility is load balanced between the different devices [99]. The distribution respects the low storage memory of devices and adds the efficiency to the data access.

### 4.2.2 Efficient Synchronization protocol requirements

Remote synchronization has been the subject of significant works that depend on the purpose of the software and the nature of objects to synchronize. In fact, object can be

personal data, and information stored either in structured databases or in files and folders. Among the most known synchronization protocols for personal information, we find Palm Hotsync , Intellisync and CPISync [107]. These solutions are basically used for PDA and mobile devices. However, they do not consider the problems of distributed services based on Cloud and Grid environment. To follow the technological evolutions and the consumer demands, new protocols are proposed.
In the context of file synchronization, the protocol has to provide four main key properties that should be respected in order to guarantee an efficient bandwidth, the fastest data exchange and an effective computation [95] [84] [44] [115] [117]:

- **Property 1**: Low computation of the data and metadata at the client side. This property adds the scalability and enables simultaneous synchronization.
- **Property 2**: Efficient change detection between the file system versions of the client and the server. The goal is to reduce the time of the changes detection and therefore, the time of whole synchronization.
- **property 3**: Reducing the number of synchronized files by finding the maximum number of matched content.
- **Property 4**: Reducing the messages exchanged between the client and the server.

### 4.2.3 Sub-protocol of the synchronization protocol

To define a new synchronization file protocol, it is essential to specify the stack of sub-protocols where each one ensures a specific functionality [51] [84]. These sub-protocols are: identities, objects, network, routing, conflict resolution, change and exchange.

#### Identity and object:

This sub-protocol defines how the devices, users, and objects are identified in the architecture of synchronization.

- **The user identification**: Each user subscribed to the synchronization solution is identified by a unique identifier which can be his mail address, a pseudonym, a random number, etc.
- **The data identification**: Data are stored in a container or a namespace which has a unique identifier too. Thus, these data are identified by their path in the user file system as well as the identifier of the space where they are stored. In addition, when the file is partitioned into blocks, each block is identified by its hash [108] [95]. Additional identifiers can be added for data synchronization purpose such as in the case of BittorentSync where the root folder of the user is identified by the SharedID. This key is requested by one device to be concerned by the data synchronization.

- **The device identification**: The device of the user is identified by a unique identifier to allow the synchronization between the different user devices. It can be based on the hash of the user certificate (the case of IPFS [51] and SyncThing [36]) or the name of the used device (case of DropBox and BittorentSync).

#### NETWORK:

It identifies the different networking protocols used to ensure the communication between the client and the server or between nodes in the case of peer to peer file synchronization.

Choosing the best protocol for the data exchange is very important for the synchronization protocol efficiency. In fact, it is crucial to reduce the amount and size of messages exchanged between both end points. There is two kinds of communications in the case of file synchronization:

- a channel to ensure the exchange of the data between the nodes.
- a channel to notify the devices concerned by the synchronization. The objective is to alert these devices that changes occurred and that they need to update their content.

Various networking protocols and architectures are used including both the notification and data transfer. As example, Dropbox [63] uses the HTTPS for the data transfer and the long HTTP polling for the notification. Regarding Google Chrome synchronization, the data transfer is also based on HTTPS and the notification exploits an existing XMPP-based Google Talk server [4]. Rest API is also used to ensure data replication in multiple solutions such as CouchDB [47]. Indeed, Representational State Transfer (REST) is an architecture that generally runs over HTTP.

In the Peer-to-Peer synchronization solutions, we find for example, BittorentSync [69] which uses the libutp for data transport. The uTP protocol was developed by Bittorent to replace TCP. We find also the solution SyncThing [36] which defines the Block Exchange protocol (BEP) deployed in the highest level on the protocols stack and the Device Discovery Protocol (DDP) used before the synchronization for nodes discovery.

#### ROUTING:

It includes information about the connected nodes, their content and the log of the synchronization. Several solutions are adopted, and they depend mainly on the synchronization strategy: centralized or decentralized.

These information can be stored in the server side. In the case of Dropbox a Server File Journal (SFJ) is adopted. It is a big metadata database which contains the blocklist (hash of file) without the file content. The journal is an append-only record. Each row in this file matches to the version of the namespace updates, and the keys of the rows are: Namespace Id, Namespace relative Path, blocklist, journalID. To have an idea about the different user device, a centralized list of devices linked to the user account is performed.

It contains the IP address, the country where the device is connected and the approximate time of most recent activity.

In the decentralized synchronization, information can be exchanged directly between the nodes and others require to be saved on servers. After the establishment of the connection in Syncthing [36], both peers send an Index message. This message contains the name of the folder the number of files, their names and the hashes of file blocs. These messages help both peers to recompute their knowledge of the data that exist in the cluster of user devices. A global Discovery server is also used. The purpose is to provide the IP address of other user peers in the same cluster.

Bittorent sync uses a Tracker server. In this server, several information about the peers are centralized. To synchronize the folder identified by the ShraedID, a peer sends a discovery message to the Tracker server. Therefore, the server sends back the list of other peers that share this folder with their deviceID, IP address and port number. The particularity of Bittorent Sync is that the synchronization is performed across multiple devices using distributed technology.

#### Exchange:

At this level, different steps of the synchronization protocol are defined. The exchange depends chiefly on the synchronization architecture (centralized or decentralized) as well as the entities adopted in the architecture.

In fact, in a centralized architecture, the synchronization between two nodes passes through a third server. Changes performed on the client side are transferred to the centralized server which transfers them later into the legitimate devices.

In the context of decentralized architecture, the signalization passes through a third server, and the data transfer is ensured directly between nodes. It is essential to find the following steps [36] [69] :
- The node retrieves information about the other nodes. These information are stored in a centralized server or exchanged between the peers.
- Once the nodes are discovered, the push and retrieve of files is similar to the centralized solutions.

#### Conflict mechanisms

As the user is using many devices, new versions of the same local file upload can occur from the different devices at the same time. This can probably lead to a conflict. The conflict is considered when we need to update a specific file with a new operation and we find that other operations have already been considered. In our context, we focus mainly on synchronization of files which accept simultaneous access and a single write at the same time. Two main approaches are adopted to solve the problem of conflict in our case:

- Saving the original file and creating a second version with the same name followed by the device and the date of conflict. These both versions are synchronized across

the devices and the owner decides, when he wants, the version to keep and the one to remove.

- Identifying the nature of conflict and raising a alert to the user who should decide immediately which version he wants to keep and synchronize. The protocol SyncView [115] uses only the timestamp of the file modification in different devices to detect a conflict. However, this approach may miss update information in case of offline updates. To overcome this deficiency, the synchronization protocol Synccs [84] specifies that a file conflict can be caused by four categories of operation: create, modify, delete and rename. Thus, six kinds of conflict are highlighted: file name (creating a file with a name that already exists), modify-delete (the local operation tries to delete an already modified file), modify-modify, delete-modify, rename-delete, rename-rename.

### Changes detection:

This sub-protocol detects the changes that occur in one device and which part of the file system should be sent to have the same objects in different nodes. Different algorithms of changes detection was proposed depending on the purpose of the software and the nature of objects to synchronize.

In the context of file synchronization over the network, the proposed algorithms, as depicted in figures 4.3, 4.4 and 4.5, can be classified into three main approaches. We consider that the node A intends to send the changes performed on its file system to the node B.

Otherwise, the change detection can use operating system modules such as Inotify on Linux

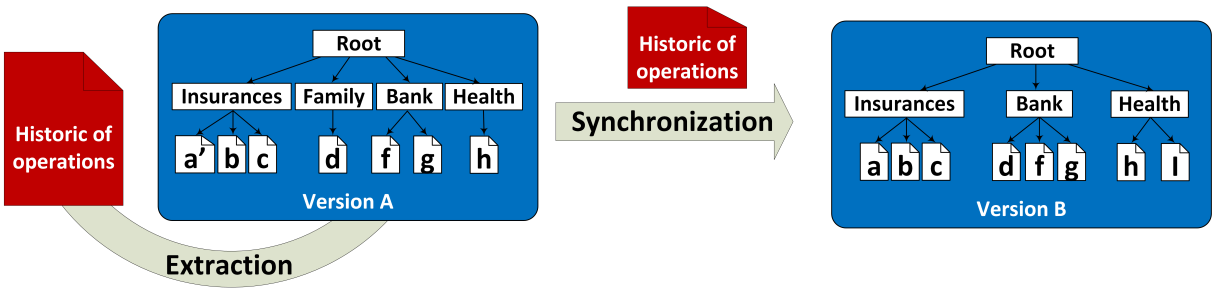



**Figure 4.3:** *Operation approach*

or FSEvents API on Mac OS X. It is obvious then that the synchronization application will depend heavily on the operating system.

**- Operation Approach** (figure 4.3):

This approach is based on recording the different operations performed on the node A and sending them to the remote node B [115]. These operations are sent to the node B to update its replica. However, storing the operation is a consumption of the storage memory, unless a log merging is adopted such as the case in [84].

**- Changes Approach** (figure 4.4):

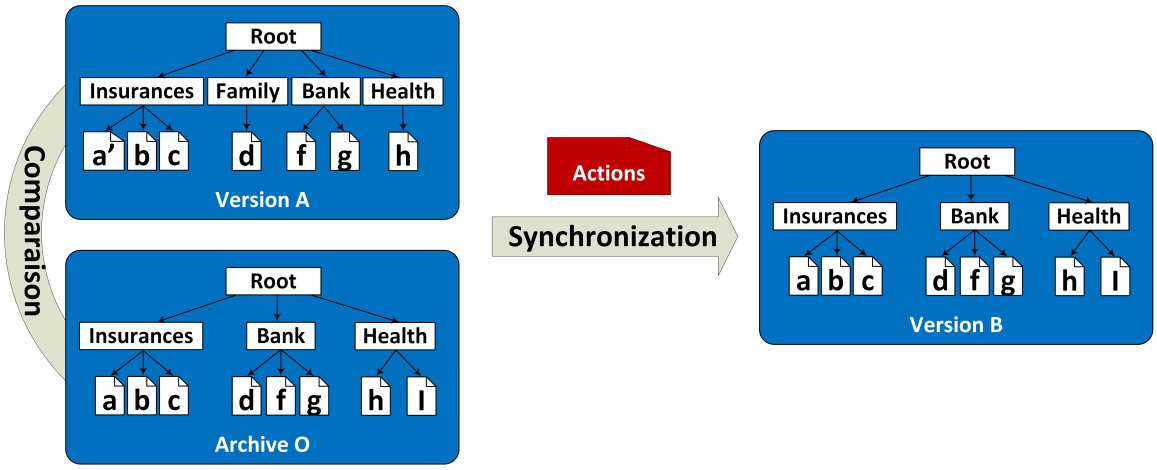



**Figure 4.4:** *Changes approach*

The main idea behind this approach is to save a copy of the last synchronized file system version (i.e. just before being offline). After a series of modifications and when the user becomes online, the old copy and the new version are compared. The different actions performed offline are therefore detected and sent to the remote node B. Unison [52] is one of the algorithms that adopts this approach. In fact, it is a bidirectional synchronization algorithm which detects updates on both sides based on the comparison of the archive and the file system. It applies then the updates based on the recursive multi-round synchronization protocol.
This approach proved its worth. However, it needs additional storage space for archiving the file system. In addition, there is a lack of automatic detection since it must be triggered periodically.

**- Differencing Approach** (figure 4.5):

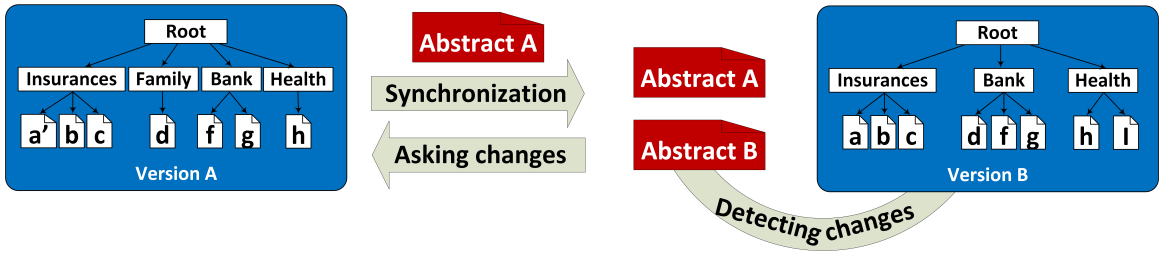



**Figure 4.5:** *Differencing approach*

The majority of proposed algorithms focuses mainly on detecting the changed bloc within one file. The Rsync [108] is the most used algorithm in the existing solutions. It is a single round protocol that splits the file into chunks with the same size. It computes for each chunk two checksums (MD5 hash and rolling hash). It sends these checksums to the server which compares them with its version and asks the missing blocks. Research

works are making effort to enhance it by integrating additional paradigms such as the divide and coquer [105] and the content defined chunks [43]. In addition, we find the set reconciliation algorithm [92]. It is based on finding data that are in the intersection and union of two files with the minimum amount of communication. This protocol is based on characteristic polynomials. Taper [95] follows also the differencing approach. It is a tiered approach based on four phases where each phase moves from a larger to a finer matching granularity. The master sends a tree structure Hierarchical Hash Tree that contains the hash of the different directories and files of the file system. The main goal is to find the changed part of files. The phases of Taper are ordered as follows: directory matching, chunks matching, blocks matching, bytes matching.

In this approach, the node A sends an abstract of its files and directories to the remote node. This abstract usually includes some metadata of files. In the case of dropbox [63], the metadata contains the object path, object type (files or directories) and the hashs of 4Mb file blocks. In the case of Rsync [108], it sends the hash of file blocks, and Taper [95] sends the Hierarchical Hash Tree, the hash of file chunks and the hash of file blocks. The node B, therefore, compares its abstract with the one received from A to detect the changes. It applies the changes and it asks A to send back missing blocks of files.

This approach cannot guaranty an automatic synchronization and sending periodically the abstract leads to an inefficient bandwidth usage. In this case, even the structure of the abstract should be well chosen to guaranty a fast matching at the target.

The existent solutions focus on detection the changed blocks of files. However, in a whole synchronization protocol, we need to detect the changes which occur within the whole file system with its directories and files. Therefore, we focus on different strategies used to detect changes between two structured data.

### 4.2.4 Detecting changes in file systems

Adopting the differencing approach leads to focus on detecting the changes between the abstract sent by node A and the abstract of the node B. In the case of data synchronization, the abstract consists on a part of the file system metadata. Speaking in terms of graph theory, a file system can be modeled as a tree structure. Therefore, the metadata of a file system can be organized to be modeled as a tree structure. The tree is a rooted and ordered graph built on nodes interconnected with a parent-child relationship. In the file system context, a node that having no parent is called a root. It is the main directory of our file system. However, the nodes having no child are the leafs which matches to the files. Nodes in between are called inner nodes. They represent the different directories of the file system.

Many works are focusing on general problems of detecting changes between documents, mostly flat files. For example, Unix diff is one of the most popular change detection utilities that use the Longest Common Subsequence (LCS) algorithm to compare two files. We find also the Concurrent Versions System (CVS) which uses the diff algorithm to

show the differences between different revisions of a given program. However, these two solutions cannot be generalized as they do not understand the all structure of the data. Later, structured document differencing algorithms were proposed to fit the requirement of structured data for Latex and nested-object documents such as LaDiff [60] and MH-Diff [59] algorithm. Algorithms are proposed also for XML documents such as diffX [45]. It is based on the bottom-up mapping and the DOM-hash to reduce the size of the trees to compare. Our change detection algorithm, which will be detailed in section **??** is inspired from the diffX [45] algorithm, first because it proved its worth in change detection compared to the other solutions related to structured documents. Second, it is based on the tree structure considering a structure similar to the Hierarchical Hash Tree.

Using the Hierarchical Hash Tree (HHT), each node of the tree is identified by a hash. For files, this hash matches to the file content hash. For directories, the hash is based on the hashes of directory content. In Taper [68], the HHT was basically used to detect the changed blocks of files. However, in our proposal, the HHT is introduced to detect the changes that occurred in the whole file system. These changes are basically modify, move, create, rename and delete operations applied on files or directories. The HHT will be explained in details in section 4.4.

## 4.3 SyncDS synchronization protocols

### 4.3.1 Overall SyncDS architecture

The goal of our framework is to ensure a secure data synchronization between a Client Digital Safe and a Cloud Digital Safe while considering the probative value. As a first step, we need to define the architecture as well as the messages exchanged between its different entities. The SyncDS architecture is divided into four main layers, which are

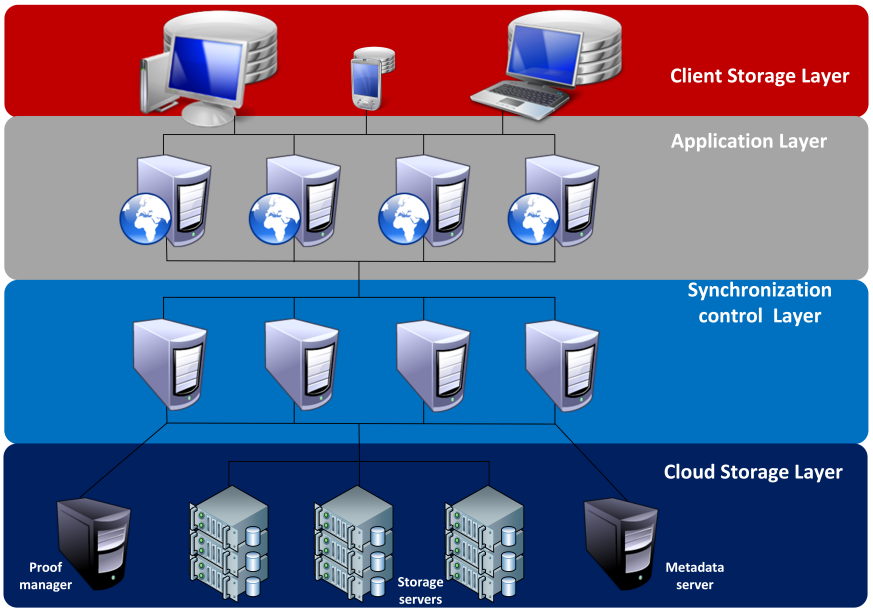



**Figure 4.6:** *Digital Safe Synchronization Architecture*

from top to bottom of figure 4.6:

- **Client storage layer**: The data are stored securely in a Client Digital Safe. The client side storage is based on the HTML5 Local Storage API with additional security considerations to follow the Digital Safe requirements. The conception and the implementation of a Client Digital Safe was detailed in the chapter 3.
- **Application layer**: This part includes the web application with the different used APIs such as the Synchronization API. This API is introduced into the web application to detect the user operations on the Client Digital Safe and record them. It manages then the data synchronization between the Client Digital Safe and the Cloud Digital Safe in the application side.
- **Synchronization control layer**: This server is responsible of managing the synchronization. Therefore, it handles the different synchronization requests and responses as well as the conflict resolution. It also notifies the devices concerned by the modification to propagate the changes. The concerned devices can be those owned by the main user or those that share the modified file. In our architecture and for great performances, we choose the Websocket protocol to ensure the bidirectional communication between the local and remote Digital Safe.

  One of the main challenges that face the synchronization service provider is to set up a synchronization architecture without a single point of failure. Their first concern is to provide a continuously service. To address this problem it is essential to build the redundancy into the synchronization servers and to adopt the load balancing.
- **Cloud storage layer**: As introduced in [93], it is a standardized architecture that provides a secure environment for storing sensitive document. This environment full fit both the user requirements and Cloud security challenges. Going into further details, this Cloud Digital Safe is composed of three main components.
  - First, the metadata server stores the metadata generated for each stored file. A metadata is an XML file that stores information about its file. We find information related to the file, information related to the storage, information about the access control policies and finally information about the synchronization.
  - Second, the Storage servers with large scales store the blocs of files. Data replication is guaranteed by these servers to ensure the data availability.
  - Finally, the third party Proof Manager preserves the proof of data storage and guarantees the non-repudiation. The proof consists on the metadata signed by the owner using his private key.

### 4.3.2 SyncDS sub-protocols

Dealing with the SyncDS sub-prototols leads to give first an overview on the identity and network synchronization sub-protocols. Second, it is essential to detail the exchange,

change detection and conflict resolution.

### Identity

An object, file or directory, is identified by its name preceded by its path. This identifier is unique as two objects cannot have the same name if located in the same directory. The user is identified by its mail address as it is unique and his certificate. The device is identified by the device name followed by the used operating system.

### Network

As we defined previously, two kinds of communication should be ensured. The first one for the notification and the second one to exchange synchronization messages between the client and the server. In our case, we adopt the WebSocket protocol for both types of communication.

- Data exchange: The WebSocket is used to ensure the exchange of synchronization messages between the user device on one side and the synchronization Manager, the storage servers and the Proof Manager on the other side. Therefore, it is proven that WebSocket reduces unnecessary network traffic and latency when compared with HTTP based protocols. It has also additional advantages:
  - Bi-directional: In the case of HTTP, a request is usually initiated by the client. The servers then process this request and answer. Using WebSocket, a bi-directional protocol is ensured. Both the client and the server can initiate the message sent without predefined message patterns like the request and the response.
  - Full-duplex: With HTTP, we find the client talks to the server or the server talks to the client in a particular time. In fact, the client sends a request and waits for the server response. This is not the case for WebSocket, as the message exchange of the client is independent of those of the server.
  - Single TCP connection: In the basic use case of HTTP, a new TCP connection should be initiated for each HTTP request/response. Otherwise, an HTTP persistent connection with the keep-alive header to guarantee multiple request/response over a single TCP connection. Regarding WebSocket, basically this protocol provides a long-held TCP socket connection.
  - Efficiency: Compared to HTTP, WebSocket eliminates the extra headers sizes and especially reduces the number opening and closing of the socket connections for a request.
- Notification: It is interesting to use the same protocol for the data exchange and for the notification. We adopt WebSocket as the bi-directional characteristic fits to the notification requirement. In fact, it is interesting for a server to initiate the message

sent. First, the need of synchronization is detected at this level. Secondly, the server should notify the synchronization request.

### 4.3.3 Overview on exchange steps

We have defined previously the architecture of our synchronization solution, and this section focuses on the messages exchanged between the different entities of the architecture. The figure 4.7 presents an overview on the SyncDS protocol messages. The considered entities are:

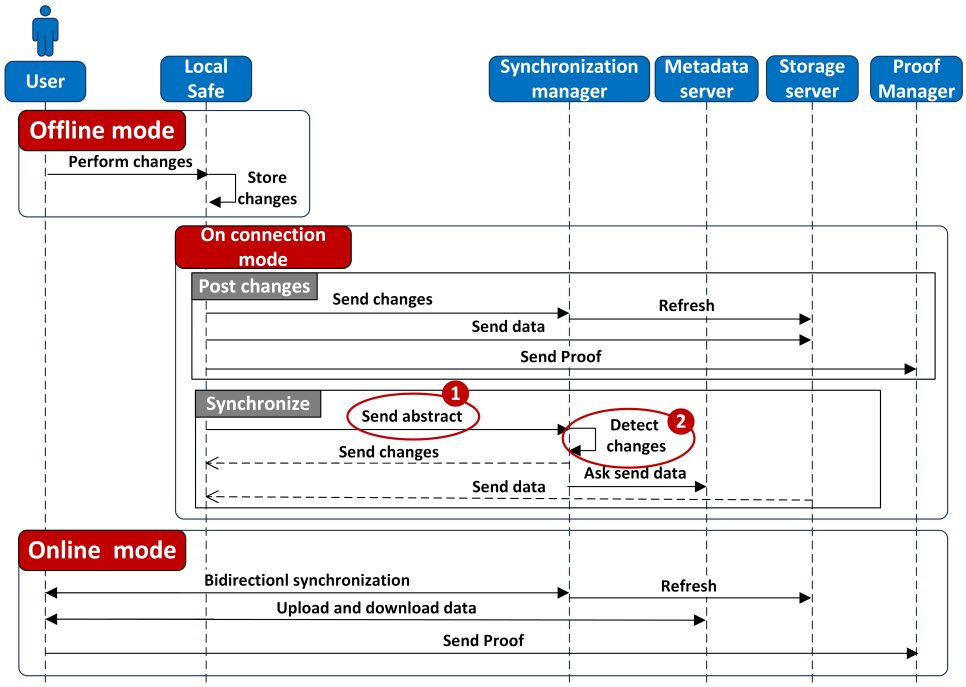



**Figure 4.7:** *Overview on the synchronization protocol SyncDS*

- **User**: The owner of the Digital Safe;
- **Local Storage**: The Client Digital Safe;
- **Synchronization manager**: The server which manages the synchronization;
- **Metadata server**: The server where metadata are stored;
- **Storage server**: The server where the file blocs are stored;
- **Proof manager**: The third party that stores the proofs of storage.

Our synchronization protocol SyncDS holds three phases:

- **Offline phase**: The user makes changes when he is in the offline mode.

- **On connection phase**: It represents the action performed when the device comes back online. This mode has two steps:
  - **Post changes**: The changes performed offline are sent to the server.
  - **Synchronize**: The changes performed on the server side are sent to the device.
- **The online mode**: The user makes changes when he is online.

These phases are detailed in the next section as we will zoom in on the different blocs. The novelty in our synchronization protocol is the introduction of the Hierarchical Hash Tree (HHT) into the abstract structure in the synchronize step (circle 1 in figure 4.7). The goal is to detect changes between two versions of the same file system. The HHT was adopted in previous work to detect changes between two versions of one file. To the best of our knowledge, our work is the first one that adopts the HHT in the context of secure file synchronization to detect changes between whole file systems. We propose therefore, the appropriate algorithm to ensure this efficient detection (circle 2 in figure 4.7).

### Offline phase

**Table 4.1** SyncDS operations

| Operation | Operation signature |
|---|---|
| **Adding** | Add(object, nature_object) |
| **Deleting** | Delete(object) |
| **Modifying** | Modify(object) |
| **Copying** | Copy(object, destination) |
| **Moving** | Move(object, destination) |
| **Renaming** | Rename(object, new_name) |

The particularity of our standardized solution is the possibility to make changes on the Client Digital even when the web application is offline. All made changes will be updated in the Cloud Digital Safe just when the user comes back online.
If the connection is not available or possible, the Client Digital Safe records in a log file the different operations performed locally by the user.

These operations are detected through the enhanced HTML5 fileSystem APIs specification that follows the Digital Safe requirements. Table 4.1 lists these operations with their signatures as they will be stored. The object represents the full path of the digital object and not limited to its name. In addition, nature_object matches to a file or directory.

### On connection phase

This phase starts when the user device comes back online while the Digital Safe application is opened or the user opens the appropriate web application to resume the communication with his Cloud Digital Safe. The "on connection" phase is a two way synchronization that includes two steps Post changes and Synchronize.

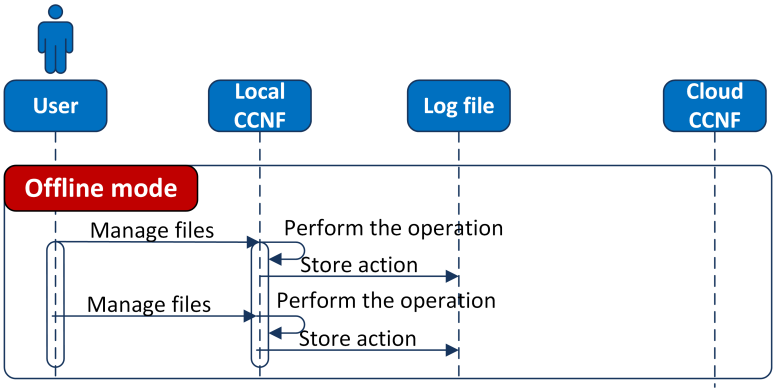



**Figure 4.8:** *Offline phase of SyncDS*

- **Post changes:**
  In the first step, the client posts changes performed when it was offline. This step is based on the operation approach. Thus, the log file is sent to the server. The server then applies on his version the changes as listed in the log file. The different steps are as follows:

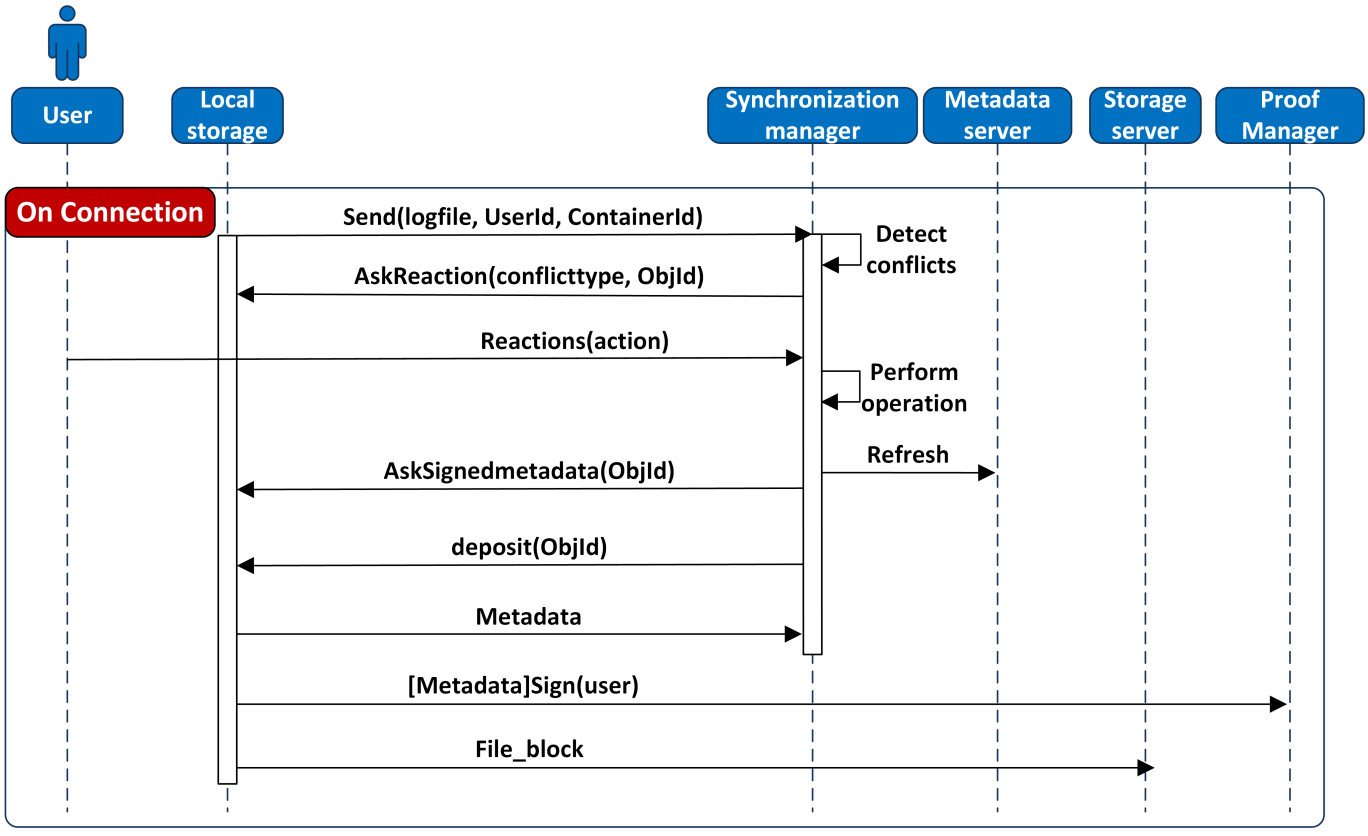



**Figure 4.9:** *On connection phase: post changes*

  - The client sends to the server the log file where the operations performed offline are stored.
  - The server receiving this file, detects the possible conflicts. In fact, it verifies if

the same file were managed by the same user in another device or by another user who shares it when the current device was offline. In our protocol, in case of conflict detection, the server sends an alert to the client to ask him his reaction to the conflict. This message explains the conflict type and the concerned object as shown in the table 4.2.

- When the server receives the reaction of the user, it performs the appropriate action. It modifies the server version by refreshing the metadata.
- The server asks the client to send the missing blocks of files to the storage server and the new version of signed metadata to the proof manager.

**Table 4.2** Conflict resolution

| Conflict type | Definition | Possible reactions |
|---|---|---|
| **File name** | Creating an object with a name that already exists | - Rename the object added by the user.<br>- Deleting the object added by the user.<br>- Deleting the object added when the user was offline. |
| **Modify-Delete** | Deleting a modified object | - Deleting the object modified by the user.<br>- Adding the object modified by the user. |
| **Modify-Modify** | Modifying a modified object | - Keeping the version of the Cloud.<br>- Keeping the version of the user. |
| **Delete-Modify** | Deleting a modified object | - Deleting the object.<br>- Keeping the version of the Cloud. |
| **Rename-Delete** | Renaming a deleted object | - Sending the object with the new name.<br>- Deleting the object. |
| **Rename-Rename** | Renaming a renamed object | - Keeping the old name of the object.<br>- Keeping the new name of the object. |

- **Synchronize changes:**
  In the second step, the server reveals changes performed on his side when the user was offline. These changes can be performed by the same user in a different device or by another user who shares a part of the file system.

  In this step, the differencing approach is adopted. It is based on comparing two abstracts to detect the changes between two versions of the same file system. In fact, as depicted in figure 4.10:

  - The Client Safe sends an abstract of its file system to the server following the Hierarchical Hash Tree (HHT) structure.
  - Receiving the client abstract version, the synchronization manager generates the abstract from the metadata server.
  - The server compares the two versions of abstracts to detect the changes between both versions of file systems.
  - The changes are sent to the client to apply them at his side. If a new file was found, the Client Safe retrieves the object from the Cloud Safe.

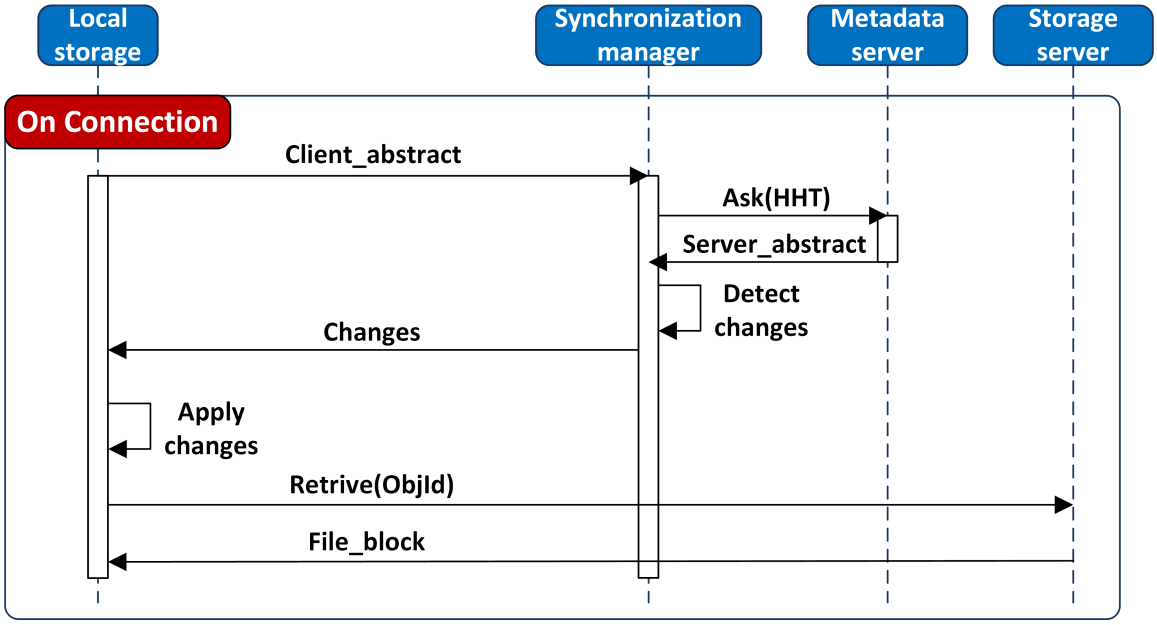



**Figure 4.10:** *"On connection" phase: synchronize changes*

### Online phase

This phase starts when the two file systems versions of far-end are identical. It is a two way synchronization while the data can have "multiple access and unique modification" at the same time. Changes made in the client side are sent to the server, while and changes, made by different devices and synchronized to the server, are sent to the user.

This step is based on the operation approach. Even if Synccs [84] proposes a similar protocol, its online phase considers a one way synchronization. However, this protocol does not deal with multi-locations changes. It synchronizes only changes that occur at the client side which remains insufficient. This point is taken full account in our proposal and choosing the WebSocket is the best proof. In our synchronization protocol, WebSocket is used to send data from the client to the server. It is also adopted by the server to send notifications and data to the client.

The messages exchanged between the Client Digital Safe and the Cloud Digital Safe are as follows:

- Each operation performed on the client side is sent to the Synchronization manager;
- The Synchronization manager applies these operations and refreshes the Metadata server content;
- In the case of adding a file, first, the metadata are sent to the Synchronization manager. Then, the signed metadata are sent to the Proof manager and the file blocks are sent to the storage server;
- In the case of a modification in the server, the user is notified and receives the changes.

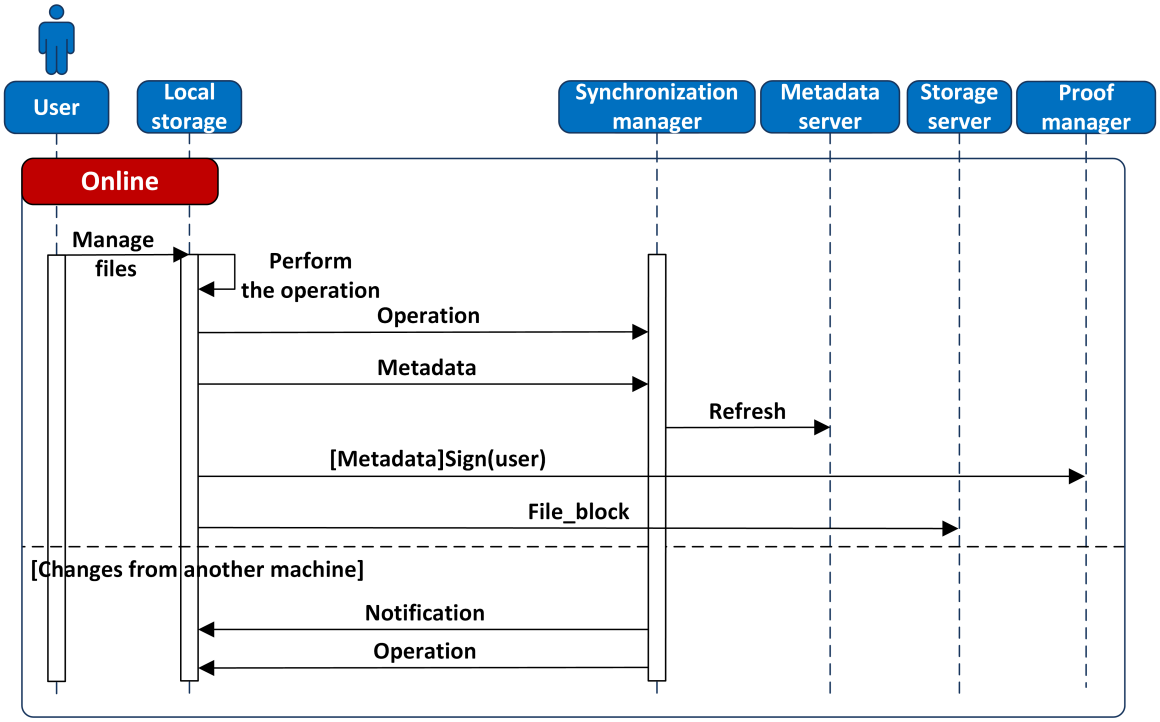



**Figure 4.11:** *Online phase*

## 4.4 SyncDS efficiency with the Hierarchical Hash Tree

In this part, we focus on the "on connection" mode and more specifically on the synchronize phase. The novelty is the introduction of the Hierarchical Hash Tree (HHT) into the file system abstract structure to detect changes between two versions of the same file system. The HHT was adopted in previous work [95] but only to detect changes between two versions of one file. To the best of our knowledge, our work is the first one that adopts the HHT in the context of secure file synchronization to detect changes between whole file systems. We propose the appropriate algorithm to ensure an efficient detection of changes.

In the following sections, we describe, first, the structure of the abstract as it should be sent by the client and generated from the Metadata server. Secondly, we present the algorithm of abstracts comparison to generate the script of changes.

### 4.4.1 Abstract structure

Among the metadata information, we extract the ID, the path, the type and the hash of the object (file or directory) to build the abstract. The abstract, in this context, can be modeled into a rooted and ordered tree: rooted as it has a root directory and ordered as there is an hierarchical relation between files and directories. The special feature of the abstract, in our proposal, is the introduction of the Hierarchical Hash Tree (HHT). In fact, two main functions can be considered when talking about HHT:

- The hash of the file: It is computed using the hash function (Message Digest 5) of

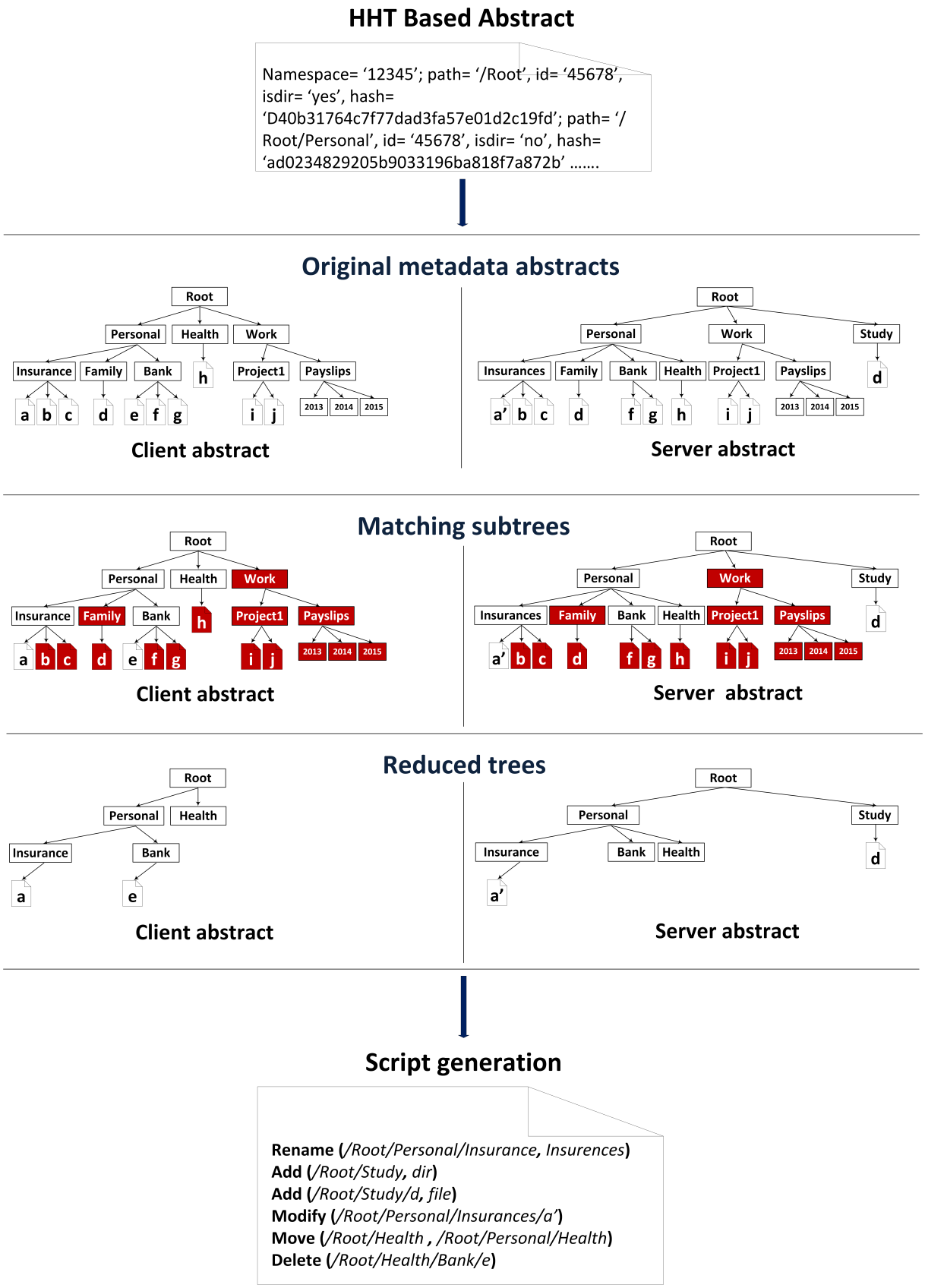



**Figure 4.12:** *Changes detection algorithm*

the file content. This hash is already computed and introduced into the metadata structure:

$$Hash_F(file) = Hash_{MD5}(file) \tag{4.1}$$

- The hash of the directory: To compute this hash, we need first to concatenate the different hashes of the directory objects, files or directories. Then, we compute the hash of the concatenation result. The concatenation should follow a specific order of objects to have a unique hash. In our case, we order the objects in the ascending number of their IDs.

$$Hash_D(dir) = Hash_{MD5}(\sum_{f,d \in dir} Hash_F(f) + Hash_D(d)) \tag{4.2}$$

### 4.4.2 Changes detection algorithms

As depicted in figure 4.12, the whole algorithm is composed of two algorithms: isolated subtree matching and edit script generation. The output of the first algorithm is the input of the second one. The isolated subtree matching allows the detection of objects that matches while comparing both abstracts to generate the reduced trees. The second algorithm uses the both reduced trees to generate the script.

#### Isolated subtree matching (Algorithm 1)

While comparing both versions of abstracts, the function match(x,y) where x ∈ T1 and y ∈ T2 detects the subtrees that has not been changed in the trees T1 and T2. The match function should have the following conditions:

$$match(x,y) = \begin{cases} true & \text{if } Hash_{F/D}(x) = Hash_{F/D}(y) \\ & Name(Parent(x)) = Name(Parent(y)) \\ false & \text{otherwise.} \end{cases} \tag{4.3}$$

*where:*

$x \in T1$;

$y \in T2$;

$Hash_{F/D}(n)$: *computes the hash of a node which can be a file using the equation 4.1 or a directory using the equation 4.2;*

*Parent(n): matches to the parent node of the node n;*

*Name(n): matches to the name of the node n.*

**Algorithm 1** Isolated subtree matching

```
1: procedure DETECT MATCHING(T1, T2)
2: Input T1, T2: tree
3: Output T1′, T2′: tree
4: Traverse T1 in a level order and top-down
5: let x be the current node
6:     if ∃ y ∈ T2/ match(x,y)=true then
7:         Delete (subtree(x), T1);
8:         Delete (subtree(y), T2);
9:          if Name(x) ≠ Name(y) then
10:           Rename(x,Name(y))
11:         end if
12:         if ∃ y ∈ T2/ Hash(x)=hash(y) and Parent(x)=Parent(y) then
13:        Mark (y, x);
14:
15:         end if
16:    end if
17: End-traverse
18:    if ∃ y ∈ T2/ Mark(y)=True then
19:        Copy (x, Name(y) );
20:        Delete (subtree(y), T2);
21:    end if
22:
23: end procedure
```

Using a top-down tree analyses, the first algorithm looks for the set of subtree that matches between both trees, i.e., matching the trees T1 and T2 leads to find the pair(x,y) where the subtree of T1 rooted at x and the subtree of T2 rooted at y match (using equation 4.3).
Starting from the root node, the algorithm finds all the nodes of T2 that matches to the node of T1. While a match is found both subtrees are erased from the trees (Lines 7 and 8 of the algorithm 1). In case of unmatched, the algorithm recursively passes to the level bellow of the tree T1 and continues the detection of matched subtree.

When the matched subtrees are found and erased from the trees, the outputs of the algorithm are two reduced trees each one refers to a version of a file system.

During this phase, if we have two nodes from two trees that match but with different names (Line 9 of the algorithm 1), this leads to detect the rename operation. We consider particularly this case for efficiency reasons. Otherwise, the whole directory will be considered different and will be synchronized while its content has not changed.

### Edit script generation (Algorithm 2)

This algorithm has as input the reduced trees T1 and T2 which are the outputs of the first algorithm. Its output is a script. This edit script contains the different operations when we applied them of the different nodes of the tree T1, we obtain the tree T2.

In the context of file system synchronization, the tree T1 refers to the reduced tree of the client file system and T2 refers to the reduced tree of the server file system. During the synchronize step of the "on connection" phase, the server should therefore send this script to the client. The client applies the operation of the script on his file system to provide a version identical to the version of the server.

The algorithm in this phase follows a down-top traversal of the new version of the trees T1 and T2.
The order of operation detection in this algorithm is important and is considered as follows: add ,modify, move, delete. Every time that changes have been detected for an object of the tree, this object will be deleted from the reduced tree.

In this algorithm, the different used functions are detailed in the table 4.3 bellow.

The script generation algorithm is composed of two parts:

- **Traversing the reduced tree T2:** The tree T2 is traversed to detect the four operations which are adding, modifying and moving, copying and deleting an object. In this part of the algorithm, the refresh of hashes' reduced trees should be done after the operation Modify, Move and copy. In fact, the new values of hashes will be used to detect the operations Move, Copy and Delete.

  - **Add operation** (line 6 to 12): It is introduced into a while loop to get the folder root in case of adding a folder with objects inside;
  - **Modify operation** (line 13 and 14): It needs to find an object in the tree T1

**Algorithm 2** Script generation

```
1: procedure GENERATE SCRIPT(T1, T2, S)
2: Input T1, T2: tree
3: Output S:List
4: Traverse T2 in a level-order sequence and down-top
5: let y be the current node
6:     while ∄ x ∈ T1 / Match(x,y)=True  do
7:         if ∄ x' ∈ T1 / Match(x',Parent(y))=True then
8:             y==Parent(y) ;
9:         else break;
10:        end if
11:    end while
12: Add(Path(Parent(y))/Path(y), Type(y))
13:    if ∃ x ∈ T1 /( Name(y) = Name(x) and Type(y) = Type(x)) then
14:        Modify (Path(x));
15:    else if ∃ x ∈ T1 / Parent(y) ≠ Parent(x) and Name(y)= Name(x) and Type(y) =
   Type(x) and Hash(x) = Hash(y) and parent(y) ∈ T1 then
16:        Move (Path(x), Path(Parent(y)));
17:    else if ∃ x ∈ T1 / Parent(y) = Parent(x) and Name(y) ≠ Name(x) and Type(y)
   = Type(x) and Hash(x) = Hash(y) then
18:        Rename(x,Name(y));
19:    end if
20: End-traverse
21: Traverse T1 in a level-order sequence and down-top
22: let x be the current node
23:    for  x ∈ T1 do
24:        if ∄ y ∈ T2 / Hash(x)= Hash(y) or Name(x)=Name(y) then
25:            Delete(x)
26:        end if
27:    end for
28: end procedure
```

**Table 4.3** Detailed functions of the script generation algorithm

| Function | Description |
|---|---|
| **Type(n)** | returns the type of the node n. It can be a file or directory. |
| **Path(n)** | returns the path of the node n. |
| **Name(n)** | returns the name of the node n. |
| **Subtree(n)** | returns the subtree rooted at the node a and extracted from the Tree T. This subtree matches to the repository n with its content. |
| **Match(n1,n2)** | verifies if two nodes n1 and n2 from two different trees are identical. |
| **Parent(n)** | returns the node parent of the node n. |
| $Hash_{F/D}(n)$ | returns the hash of the file or directory. |

which has the same name and type. Having different hash is not checked at this level as it has been already verified in the first algorithm;

- **Move operation** (line 15 and 16): It needs to find an object of the tree T1 which has the same name, type and hash and has a different parent node;

- **Copy operation**: This operation is detected in the algorithm 1 (line 18 to 20). In fact, if we find an object which has the same hash and a different parent, we mark it and we do not delete it. In fact, it can match to another object in T1 detected it throughout the algorithm. At the end of the algorithm, if this marked object still exists, therefore, we detect that it has been copied. Unfortunately, we cannot detect a copied object which has been modified. First, its hash has changed, second, in the reduced tree we do not have an idea about the existent of the origin of this copy.

- **Rename operation**: The rename operation is detected in the algorithm 1 (line 9 and 10) when the directory is subject of only name modification. In the algorithm 2 (line 17 and 18), the rename operation is detected in case of renaming a directory with a modified content. It is sufficient therefore, to verify the hash of directories;

- **Traversing the reduced tree T1:**
  In this part, the delete operation is detected. For each node in the tree T1, the delete operation is detected if there is not a node of the tree T2 that has the same hash or the same name.

## 4.5 SyncDS in the Peer-to-Peer context

Remaining in the objective of synchronizing Client Digital Safe files, in this section, we will focus mainly on the decentralized strategy. The goal is to emphasize the adoption of the SyncDS in the Peer-to-Peer context. We notice that besides the widespread adoption of the Cloud for the storage, a substantial proportion of storage solutions opts for decentralized services using P2P protocols. Data are, therefore, stored in personal devices without the need of third parties. They justify their choice by the fact that some Cloud based services

have recently been reported as sharing information with governmental security agencies without warrants[69]. In addition, as the user chooses exactly the devices where the data are saved, he has the full control.

Several solutions are proposed to synchronize data using Peeer-to-Peer protocol. However, they still adopt proprietary solutions which need additional installation. The most known ones are BittorentSync [69] and SyncThing [36]. With Bittorent sync, a key should be entered to synchronize the folder. Neither authentication to approve the add of a user nor password are required. Even if the risk is limited, anyone can guess or steal this key and synchronize the relevant data. Syncthing is closely similar to BittorrentSync. Once downloaded, the application need to be launched and the user will be automatically redirected to a local interface in the browser.

In this section, we focus on the decentralized strategy, and we highlight the data synchronization using Peer-to-Peer protocol. The standardized characteristic is guaranteed as first data are stored in standardized Client Digital Safes. Second, we adopt the standardized WebRTC protocol. Therefore, we are providing a standardized, efficient, flexible and cross platform data synchronization framework.

### 4.5.1 P2P-SyncDS: System requirements:

We focus on the synchronization of data across the user devices. Addressing the Peer-to-Peer protocol in a standardized synchronization architecture brings to deal with the WebRTC protocol. It is a free and open project that ensures a bidirectional communication between browsers of different machines. No software or plugins need to be installed as browser-browser communication is ensured through simple JavaScript and HTML5 APIs. Standardized by both W3C and IETF, it is based on three steps to ensure a P2P communication for data exchange: PeerConnection, Session description and DataChannel. WebRTC remains a work draft and a concern of multiple research works [56] [88] [21].

The WebRTC is based on two main protocols SRTP and SCTP. The Secured Real Time Protocol SRTP is specifically designed for the transfer of audio and video data. The Stream Control Transmission Protocol SCTP which runs on top of the DTLS protocol is used by the Data channel for application data exchange.

With SCTP, WebRTC mimics the WebSocket protocol. Therefore, it has the following characteristics which emphasize its adoption for the file transfer and synchronization:

- The support of string transfer as well as the blob Javascript binary type;
- The data channel can be configured to support the reliable mode. This mode guarantees the transmission of the messages as well as the order of their delivery. The Flow and congestion control mechanisms are also ensured;
- In the context of data synchronization between the user devices, three aspects should be considered (figure 4.13):

**(1)** Securing the data when stored locally in the user device;
**(2)** Authenticating the far-end devices;
**(3)** Securing the data transfer between two devices.

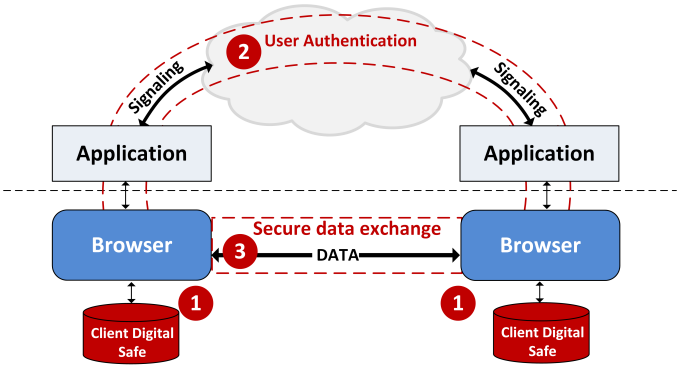



**Figure 4.13:** *Secure data distribution*

The secure data transfer **(3)** is already integrated into the WebRTC protocol. In fact, this protocol uses the Datagram Transport Layer Security (DTLS) to ensure an end-to-end data encryption. A solution for far-end authentication is proposed in [21]. It considers that the web application is launched as untrusted and likely hostile while the browser is considered as a Trusted Computing Base (TCB). The security architecture is based on the Identity Provider. In fact, each user should have an account with an IdP that he uses to authenticate to other web sites. The authentication elements are exchanged between the browser and the IdP to prove identities. They are also exchanged between the browsers during the signaling phase over the SDP messages. Regarding the local data protection, adopting the standardized Client Digital Safe based on the HTML5 Local Storage APIs meets this security requirement.

### 4.5.2 P2P-SyncDS: Standardized architecture

In figure 4.14, we detail the interactions between the elements of the architecture:

- Signaling servers: The authentication of the different nodes is performed at this level. The centralized server detects the devices which are online and those that have just log in and need to synchronize their Digital Safe content.

- Synchronization API: This API is implemented at the Web Application layer based on Javascript. It is used to manage the different messages of synchronization between the two far-end Client Digital Safes. It stores the operations performed when the user is offline. It starts the proceeding of synchronization whenever the device becomes online.

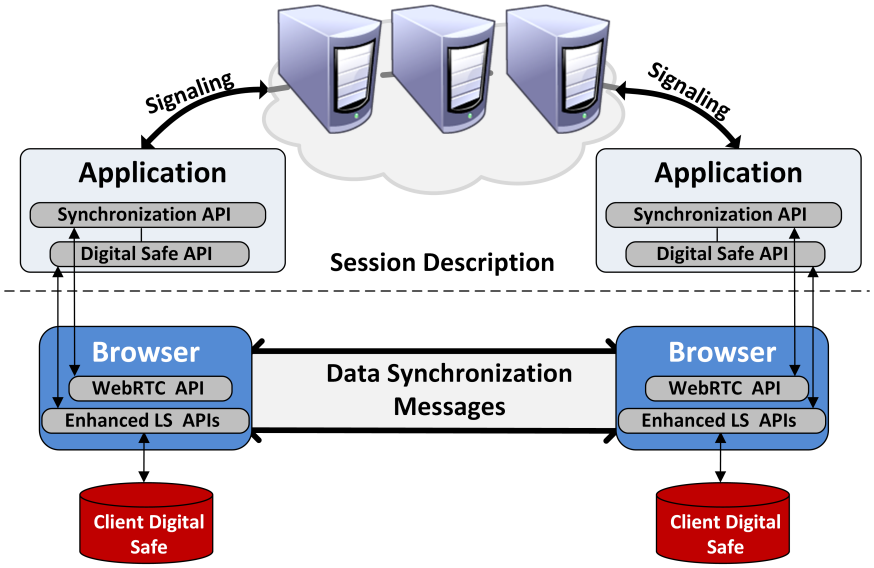



**Figure 4.14:** *Remote data management architecture*

- WebRTC API: It is used to ensure the transport of the synchronization messages between the two browsers and thereby, between the two Digital Safes.

- Enhanced Local Storage APIs: These APIs are the extensions of the basic Local Storage APIs with security considerations. Data confidentiality, data integrity and metadata integrity are added.

The presence of the synchronized directory abstract is essential to guarantee the synchronization of the Digital Safes contents. What ever the chosen strategy, stored in the centralized center or exchanged directly between the Digital Safes, the abstract should adopt the HHT in its structure. As shown in the previous section, adopting the Hierarchical Hash Tree remains crucial to guarantee an efficient changes detection between two versions of a file system as well as an efficient integrity verification.

## 4.6 Implementation and proof of concept

Even if the Hierarchical Hash Tree has proven theoretically its efficiency, we need to verify it through the synchronization framework. Therefore, we focus in this section on the implementation and the performance evaluation of our SyncDS architecture and protocol.

As a proof of concept and proof of the protocol efficiency, we address mainly three parts of the synchronization architecture. First, the implementation covers the client storage layer by granting modification on the chromium browser and particularly in the file system HTML5 API. Second, it includes the web application. Finally, it holds the synchronization control layer with its WebSocket server. As shown in figure 4.15, the red blocks are those added.

### The Chromium browser:

The implementation at this level concerns mainly the HTML5 fileSystem API. AS detailed

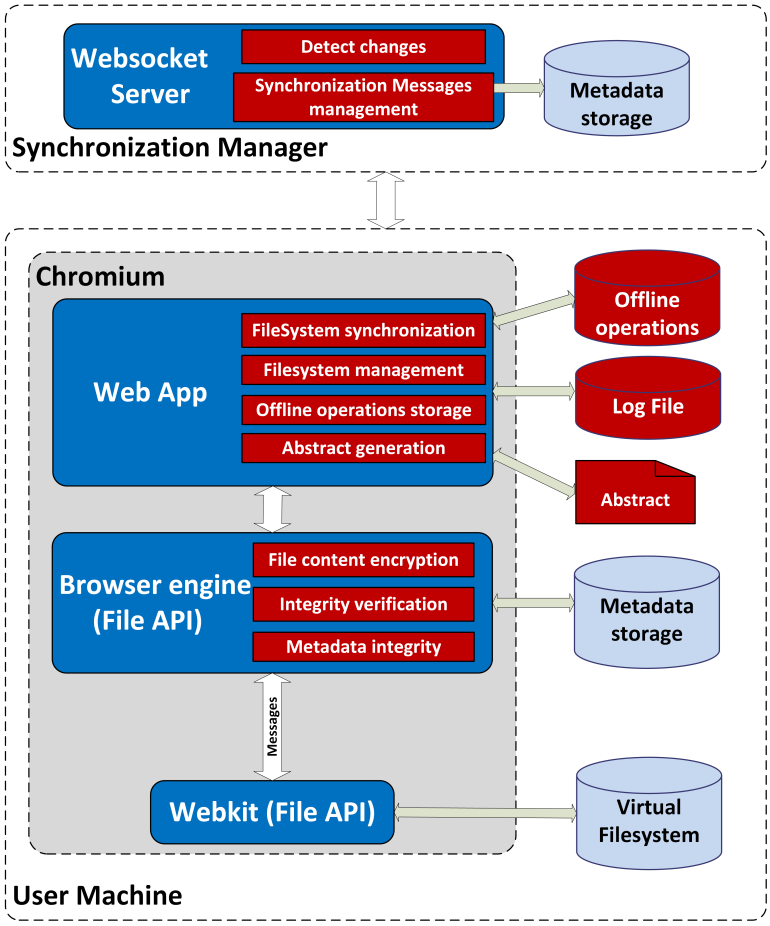



**Figure 4.15:** *Implementation of the synchronization architecture and protocol*

in the previous chapter, modifications are added into the HTML5 APIs to enhance their security. These enhancements deal with the data confidentiality, data integrity and metadata integrity.

### Synchronization in the Web application:

A HTML5 web application is developed based on the enhanced File System API and the basic Webstorage API. This application allows the user to manage his Digital Safe, store operations when he is offline and ensure synchronization of his Digital Safe content following the SyncDS protocol.

The added functionalities are introduced based on the JavaScript language using mainly HTML5 APIs:

- **File system synchronization:** This block manages the different messages exchanged between the client and the synchronization manager server. It is based on the WebSocket API calling mainly the WebSocket object *send()* method for each sent message and the *onmessage()* function for each received message. These exchanges should necessarily start by a connection to the WebSocket server using the method *new WebSocket("wss://IP@")*.

- **File system management:** It is at this level that user can manage his Client

Digital Safe. The Javascript code uses therefore the different methods of the HTML5 FileSystem API to define the Digital Safe specifications.

- **Offline operations storage:** To store the different operations which were performed on offline mode, the application uses the Offline application API and The WebStorage API. The first one is essential to detect that the application is disconnected. The second one follows the user actions when it is offline to store them in a key-value format.

- **Abstract generation:** It is at this level that the abstract of the file client file system is generated following the Hierarchical Hash Tree structure.

#### Synchronization in the WebSocket Server:

To introduce the WebSocket server, we use the web server Apache with its extension mod_pywebsocket that supports WebSocket. We add in the server side the detection of changes by comparing two abstracts (the one sent by the Client Digital Safe and the other extracted from the Cloud Digital Safe). The Hierarchical Hash Tree is introduced into the abstract structure. Indeed, the comparison of both abstracts is implemented using the Java language and more specifically using the TreeModel interface.

In this algorithm, the content of the abstract received from the client is converted into a tree structure using the *Java TreeModel interface*. The same step is applied to the abstract of the server. The algorithm of subtree matching isolation is applied first on these two trees to obtain the reduced trees. Then, the algorithm of script generation is applied on the reduced trees to generate the operation which should be applied on the client version to obtain the server version.

## 4.7 Analysis of the SyncDS protocol

The proposed protocol SyncDS is a file synchronization protocol that focuses on guaranteeing a high quality and minimal resource consumption within a secure environment. It presents two main novelties. The first one is the non-proprietary characteristic of the synchronization architecture which has a great importance to integrate a broad range of competing products and devices. In fact, we enhance HTML5 Local Storage APIs to store locally user data in a client Digital Safe. We use also the HTML5 Websocket API for the data transfer between the Cloud and the Client Digital Safe.

The second novelty is the introduction of the HHT to detect changes between two versions of the same file system. The HHT was adopted in previous work to detect changes between two versions of the same file. To the best of our knowledge, our work is the first one that adopts the HHT in the context of secure file synchronization to detect changes between whole file systems. We proposed therefore, the appropriate algorithm to ensure this efficient detection. Our protocols brings efficiency in terms of data synchronization and security verification.

### 4.7.1 Efficiency perspective

The proposed protocol adheres to the properties of an efficient synchronization protocol. To prove this adherence, analytical explanations and empirical evidences are presented.

#### Property 1:

The low computation of the data and metadata at the client side is respected within our algorithm. If we compare SyncDS with the already existent synchronization protocols, the unique additional computation at the source machine is the extraction of the Hierarchical Hash Tree. More specifically, we add only the computation of the directories hashes as the files hashes should be computed and introduced into the abstract even with the basic synchronization protocol.

The overhead on the client side brought on by the adoption of the Hierarchical Hash Tree is highlighted in figure 4.16. This overhead consists exactly on the computation of directories hashes. We present first, the time needed to generate the abstract. Second, we compute the time required to generate the hashes of directories for the HHT structure while varying the total number of files.

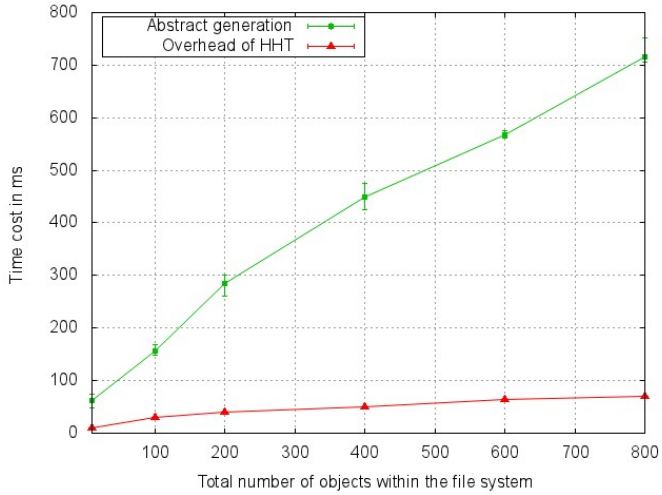



**Figure 4.16:** *Overhead on the client side caused by HHT*

As we can notice from figure 4.16, the overload of the Hierarchical Hash tree is low compared to the whole abstract generation time. Even if the adoption of the HHT comes with an additional cost, it ensures efficiency of the synchronization protocol which is highlighted with the following proprieties.

#### Property 2:

Introducing the Hierarchical Hash Tree improves the efficiency of the change detection between the file systems of the client and the server. In fact, processing reduced versions of trees is more efficient than processing the whole trees. This minimizes systematically the time of script generation.

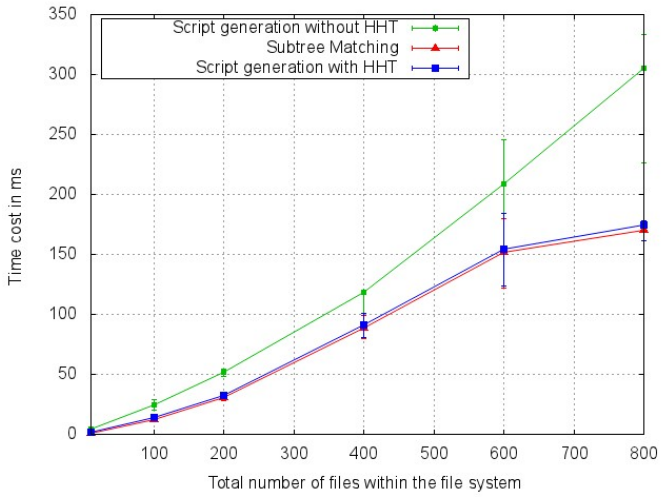



**Figure 4.17:** *Performances of the Hierarchical Hash Tree in change detection*

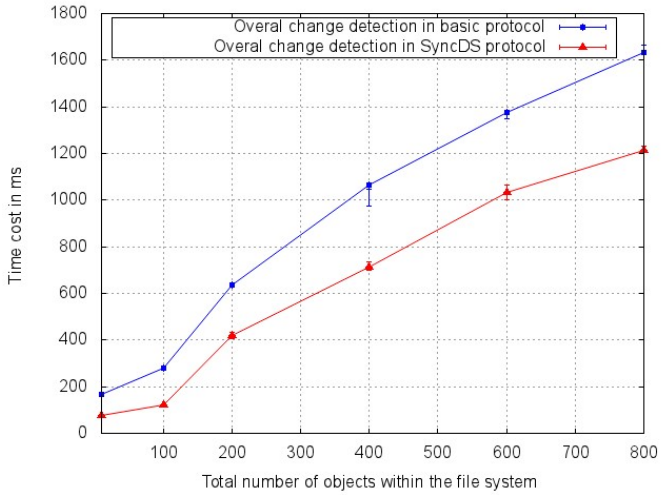



**Figure 4.18:** *Overall change detection*

To emphasize the congruency of our protocol with this property, we find that the time parameter of script generation is the most important as it is strongly linked to the other performance parameters such as the time of whole file synchronization and resources consumption. In addition, comparing the script generation with and without the HHT is equivalent to comparing the points of differences between our proposed architecture and the already commercialized one under the same experimental conditions.

In our framework, we start by capturing the time needed for detecting the changes performed on the server side to achieve the synchronization process. To highlight the performances of HHT integration, we compare, in figure 4.17, the time needed for the script generation with and without the integration of the HHT into the abstract structure. The experiment is repeated several times for a same number of total files within the same file system and a random number of changed files and directories (illustrated by the error-bars in the figure). The axis of X and Y match respectively to the total number of files in the file system and the time cost of script generation (change detection) in milliseconds.

In fact, the time of the script generation includes here the time of the subtree matching phase. As we can notice, the script generation needs more time in the basic method of change detection than when the HHT is integrated. We notice also that the most time of script generation when introducing the HHT is consumed by the subtree matching phase.

In figure 4.18, we compare the overall time of change detection of our SyncDS protocol with the basic protocol. This time includes the abstract sent, the script generation and the script sent to the client. We remind that the basic protocol uses the HTTP protocol for the communication between the Client and Cloud Digital Safe without a particular structure of the abstract. However, in the SyncDS protocol, the WebSocket protocol is adopted and the HHT is used for the abstract structure. The results show that the change detection in the basic protocol needs more time compared to our protocol. These results lead to confirm that the proposed framework and SyncDS protocol, reduce the time of file synchronization across devices.

**Property 3:**

The number of synchronized files is reduced while using the Hierarchical Hash Tree. For classic changes detection [108] [92], they rely on objects name to detect the changes. Therefore, when an object is renamed, copied or moved, the whole object with its content is synchronized since it is considered as a new element in the new version of the file system. With HHT, and as the hash is introduced into the conditions of detection, these repositories and files are no more considered as new elements and will not be synchronized as their content already exists in the storage servers.

To demonstrate the strength of the HHT adoption, we explain analytically in details the example of the copy operation. Let's assume that when the client was offline, the content of a directory D1 (its files and directories) has been copied (by another user or by the same user in a different device) to a new directory D2 with a new name. In the typical case, when the client connects and becomes online, the directory D2 will be considered as recently added with a new content. As a consequence, the whole content of D2 will be synchronized and sent to the client device.

However, using the hash of the directory in the change detection algorithm detects that the D2 content already exists in the client's file system. Consequently, this content does not need to be synchronized, and only the metadata should be updated.

**Property 4:**

Adopting the WebSocket protocol reduces the messages exchanged between the client and the server. This reduction is justified by the bidirectional communication ensured by WebSocket as the server can send any messages to the client without needing a request from the client. In addition, the size of messages is reduced using the WebSocket protocol.

As depicted in figure 4.19, we compute the time needed to ensure the synchronization of the files using our architecture and protocol. To emphasize the imminent interest of the WebSocket protocol integration, we build our synchronization architecture, first based

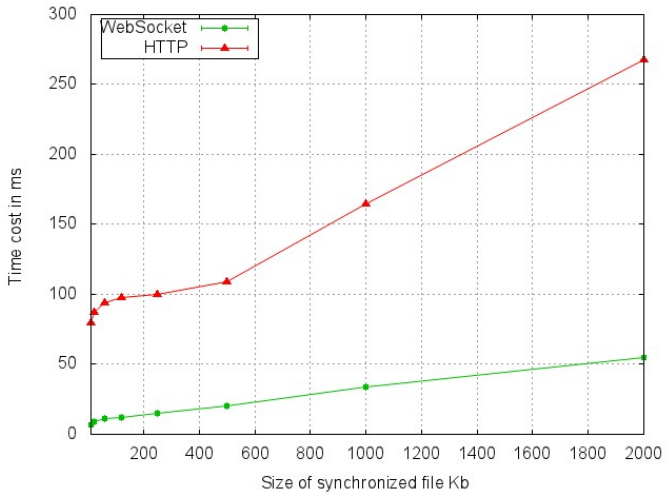



**Figure 4.19:** *Performances of the WebSocket protocol in file synchronization architecture*

on WebSocket servers, second based on basic Web servers which use HTTP protocol. We compare then the performances of both architectures. We choose to compare with the HTTP as it is the basic protocol used by the most synchronization tools. The axis of X and Y match respectively to the size of synchronized files and the time cost of achieving the synchronization in milliseconds. As shown by the results, the WebSocket protocol is by far the most efficient, especially with large files.

### 4.7.2 Security perspective

The architecture of synchronization is based on the Digital Safe context. This context focuses mainly on adding the probative value and preserving the integrity of a digital object over the time.

Introducing the Hierarchical Hash Tree into the synchronization participates directly to ensure the security of data in addition to the performances enhancement. In fact, verifying the hash value of the root directory of one file system detects systematically if one object of the file system has been altered by a third party, or the file system keeps its original version. In this case, it is no longer necessary to check the hash of objects one by one to verify the integrity.

## 4.8 Conclusion

In front of the various owned devices and the need of sharing files between different users, ensuring an efficient file synchronization protocol is crucial. In this chapter, we propose an architecture and a protocol that ensure file synchronization in a probative value Cloud. Two keynote novelties of the SyncDS protocol are highlighted: first, the integration of the Hierarchical Hash Tree into the metadata abstract to detect the changes during the second step of the "on connection" mode, second, the integration of the WebSocket protocol and server into the synchronization architecture.

Our experimental results show that adopting our framework, reduces the time of file

synchronization across devices and reduces the time of change detection.

The security stills the main concern of the user besides the efficiency. In the next chapter, we will address the security features of the SyncDS architecture and protocol. We will focus mainly on the access control in case on data sharing.

# Chapter 5

# Secure data synchronization in probative value Cloud

## Sommaire

## 5.1 Introduction

The main challenges of Cloud storage solutions are to ensure a continuous data synchronization and to guaranty data sharing between different users. Dealing with synchronizing shared data, two issues are raised. First, synchronizing encrypted data which are shared by multiple destinations may introduce key and access control management challenges. Second, it is essential to deal with the states constrains of owners and consumers. In fact, neither the owner who shared data nor the consumer are usually connected at the same time. In the context of probative value storage of sensitive data, we address the security requirement of shared data synchronization which represents 30% of the thesis works ( figure 5.1 ). We focus mainly on the access control, and on the concept of timely file sharing where shared data are synchronized to legitimate users only for a specified period of time.

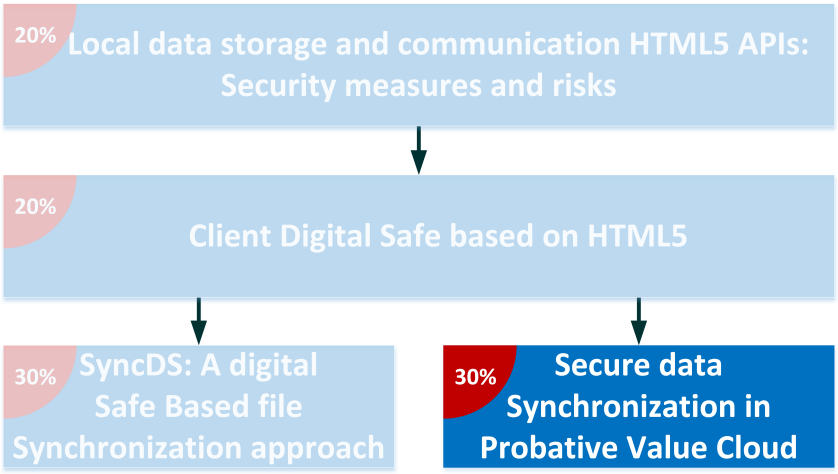



**Figure 5.1:** *Structure of the thesis*

This chapter is structured into three parts. We introduce, in the first one, security measures and mechanisms in the context of file synchronization and sharing. In the second part, we present the security of SyncDS protocol, and we highlight the timely file sharing. In the last part, we present the implementation and validation of our SyncDS protocol security.

## 5.2 Security concept for file sharing and synchronization

In this section, we introduce the concept of synchronizing shared files with its security services and mechanisms.

### 5.2.1 Synchronization of shared data

The Cloud storage services offer a variety of services for data management and especially for data sharing. In fact, the data owner can share his data with a specified user or a group of persons letting them access his data. The users within the context of file sharing are divided into data owner and data consumer. The owner, besides data creation, storage and sharing, is responsible of defining the access policy. The data consumer is the user who downloads and decrypts files shared by the owner. Often, neither the user who shared data nor the consumer are usually connected. Against this background, the data access can follow one of these methods:

- Public Sharing: The data is designed to the public without any access control consideration. In fact, a link is published to access the data.
- Secret-URL Sharing: The owner sends a sharing URL generated by the Cloud Service Provider to the consumer. Retrieving this URL leads to access to the data without any authentication or access verification. The owner is therefore, the only responsible of his data security.
- Secret-Sharing: The owner specifies the list of users to access his data (usually identified by his email or a series of attributes). The Cloud service provider is then in command of authenticating the identity of the consumer. It asks them usually to sign into their accounts to access the data.

Choosing the sharing method depends on the privacy of the shared data. The public sharing is the simplest method but cannot be applied for sensitive data. The secret sharing is the most secured as each user is authenticated to the Cloud service provider. A compromise between simplicity and security is illustrated by the Secret-URL sharing as the owner gives the URLs to each consumer.

In our thesis, we focus on the share of sensitive documents which are externalized encrypted. Therefore, it is clear that sharing data, already encrypted by users with multiple destinations, may introduce key management and access control challenges.

To deal with encrypted data, many issues are raised. How the owner will define the access policies. How the legitimacies of the consumers are detected? The most challenging issue is how decryption keys are sent to consumer to decrypt the data.

### 5.2.2 Security services

In terms of security, the baseline synchronization architecture can be the subject of multiple attacks. The main attacker goals are the access to user data content and the disturbance of data exchange. In reality, user data are hosted in large-scale systems, thus, the storage services are not necessarily trusted. In addition, the access to stored data can be impossible when the user goes offline or in case of network or service failure.

To avoid these threats, the synchronization architecture should rely on the following security services [26]:

- **Confidentiality**: Data confidentiality is one of the main concerns of users who store their sensitive data locally or externalized to the Cloud. The problem is how they can be sure that their data cannot be accessed by attackers who have the full access privilege to their machine or to storage servers.

  First, we find the server-side encryption [18] [23] where data are stored encrypted in storage servers. In this case, the keys are usually user independent and are managed by the server owner. Second, when the user does not trust remote servers, it is comforting for him to send his data encrypted [5] [16] [17]. In this case, the data encryption is performed at the client side before externalizing them to the Cloud.

- **File integrity**: This service consists on ensuring that synchronized data still in their original version as stored by the user [26]. The integrity of file is violated when the content undergoes changes either during their storage in the servers or during their transmission.

- **Authentication**: It consists on authenticating the user before any access to the service. It verifies the identities of users based on different attributes chosen by the application.

- **Non-repudiation**: It is the act of guaranteeing that no one of the two parties denies the contract of data externalization and storage by the service provider.

- **Secure synchronization**: It consists on securing the messages of synchronization between different architecture entities.

- **Access control**: This service ensures that only users who have the privilege can access data. These data can be owned by a unique user and can be shared by multiple users having the privilege to access them.

### 5.2.3 Security mechanisms

Among the security mechanisms in the context of file synchronization, in this part, we will focus on the authentication, the secure data exchange and the access control mechanisms.

In storage and synchronization architecture, the authentication and the access control are two fundamental mechanisms to guaranty the security of a system. Authentication is used to verify that the user is the one that he claims to be, and the access control verifies which data he can access in the storage system.

**Authentication:**

The commonly used technique to authenticate a user is the password-based authentication. For a high level of security, the passwords chosen by users are replaced by certificates which are supplied by trusted third parties. Tokens may be also used as an authentication mechanism. According to the use case, the token guarantees both user authentication and authorization.

Our SyncDS synchronization architecture is based on the WebSocket protocol. This protocol uses the Transport Layer Security (TLS) to ensure the authentication of the entities and the security of the exchanged traffic [31]. The secured WebSocket starts with the basic handshake followed by the TCP protocol. The data then is exchanged secured between clients and servers.

**Access control:**

The access control verification is performed just after the authentication of the user account credential and identity. Thus, it comes after the gain of the access to the system. In this step, the control focuses on system resources. It is mainly used to decide who can do what to whom in a system. Therefore, three components need to be outlined:

- the subject (who) that matches to the entity (user, process thread or program) that wishes to manipulate the resources within a system;
- the object (whom) that matches to the resources that need to be controlled through the access control policy;
- the access rights (what) that present the access policy with the description of rules used to make the decision.

### Basic Access Control Methods

Basic access control mechanisms are based on trusted third parties to protect the access to data. As depicted in figure 5.3, these servers save the access policy imposed by the owner. In case on access request by the consumer, they just verify the policy and take the decision. Various types of mechanisms can be used according to system requirements. In this context, NIST [91], lists three distinguishable access control methods which are:

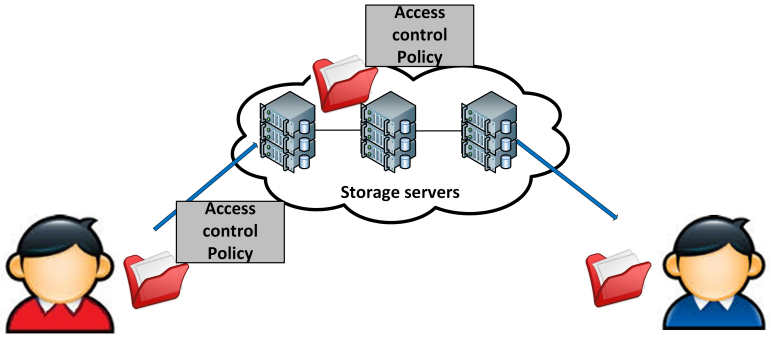



**Figure 5.2:** *Basic Access control methods*

- **Identity-Based Access Control (IBAC)**: This kind of access control policy is based on the identity of the subject to grant or deny access requests. An example of IBAC commonly used in network security services is the Access Control List (ACL). It associates to each system resource the set of authorized subjects with their access rights. However, this method is not suitable for enterprise level use. In fact, it adopts only one dimension and bypasses the context business functions and characteristics of the user [119]. Furthermore, IBAC is not scalable as the number of identifiers and user request access increases in large enterprise [91].

- **Role-Based Access Control (RBAC)**: The access to resources is not based on the user identity but rather on the user role within the organization [104]. In this case, the permissions are no more assigned to individual users which leads systematically to the reduction of the access administrations overheads [119]. However, the RBAC does not support time based access control while defining an access policy [85].

- **Attribute-Based Access control (ABAC)**: It is a logic access control model that introduces the subject, the resources and the environment attributes. The first kind of attributes defines the identity and characteristics of the subject. The second one is related to the characteristics of the resources acted upon by a subject. Finally, the environment resources matches to the context or the technical and operational environment where the access verification occurs.

**Advanced encryption for access control**

In the context of data synchronization, we need to focus on the data sharing while highlighting the privacy. The data privacy is susceptible to be compromised as the storage and the protection are usually delegated to remote storage servers.
The basic access control methods consider that the entities that manage the access control as fully trusted which is not the case in the Cloud storage services. For this purpose, advanced encryption methods are adopted to ensure the access control. It preserves the confidentiality in addition to the key management.
Several cryptographic access control mechanisms were proposed to ensure both file sharing and privacy when stored in untrusted servers. Their basic approach is based on externalizing the file encrypted by the owner. Thus, the decryption keys are retrieved only by authorized users. The retrieval of the decryption key can follow one of these three strategies:

- **Generated from user profile:**
  The encryption and decryption keys are designed for a specific or a group of users. Identity-based encryption (IBE) [42] is one of the first solutions which extends the public-key paradigm. The basic idea is to generate the public key based on the consumer identity to encrypt the data. A private key is then generated by a third trusted party and sent to the consumer to decrypt these data.

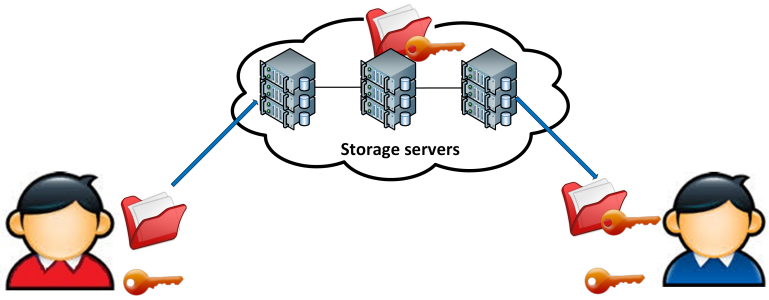



**Figure 5.3:** *Cryptographic Access control methods*

Different from the basic public key cryptography used by IBE, the Attribute-Based Encryption (ABE) is used when the encrypted messages are not intended for a specific destination. In fact, one to many encryptions is the target. Two kinds of ABE exist: Ciphertext Policy Attribute Based Encryption (CP-ABE) [71] and Policy Key Attribute Based Encryption [110] (PK-ABE). In CP-ABE, the access policy is integrated into the ciphertext, while it is linked to the decryption key in PK-ABE. In our architecture, we adapt mainly the CP-ABE into the context of shared file synchronization. This choice is based on the fact that the user is usually defined by a set of attributes, and these attributes will be linked to the decryption key. In addition, the time of the synchronization request will be added to the user attributes to verify his legitimacy.

- **Transfered directly to the user:**
  Under this case, the owner acts as a key distribution entity. It transfers directly the decryption key to the consumer who is authorized to share the data. Among the realized works, we find, Plutus [76] which divides the files into filegroups (group of files having similar sharing attributes). Each file is encrypted with a unique symmetric key then encrypted with the symmetric key of the filegroup to which it refers. The filegroup key is further delivered by the owner to authorized consumers. In case of revocation, all the keys should be updated, and whole files should be re-encrypted with the new keys. This strategy is not appropriate for fine-grained access control with a huge number of filegroups. In addition, the owner should be usually connected to send the decryption key.

- **Transfered from third parties:**

  - Some solutions store the file as a couple of file metadata and file content. With Sirius [65], besides information related to the file, the metadata includes the access control policy and a series of data decryption keys encrypted with each authorized consumer public key. Thus, the size of the metadata is proportional to the number of authorized users.

  - A proxy can be used such as the case in [49]. It is based on the proxy re-encryption

to secure the file storage. Without retrieving the plaintext, a semi-trusted proxy converts a ciphertext encrypted under the owner public key into another ciphertext. The last ciphertext can be decrypted only by the private key of the consumer. The private and public keys are generated using a set of proxy re-encryption operations.

- The fine-grained access control is also adopted in [118] by mixing the CP-ABE with the proxy re-encryption. In this solution, the decryption key generation and the access verification are ensured by the owner. However, in the context of a timely file sharing, the verification of access privilege cannot be performed when the owner is offline.

**Table 5.1** Comparison of access control schemes used for file sharing

| Scheme | Retrieving the decryption key | User revocation | Timely file sharing context |
|---|---|---|---|
| **PLUTUS** [76] | The decryption key is directly transfered to the authorized consumer | - Re-encrypting the data with new key;<br>- The consumer should be online in case of user revocation. | The key decrypts the data without any time constraint |
| **SIRIUS** [65] | The decryption key are saved encrypted with the consumer public key in the file metadata | - The size of the metadata is proportional to the number of authorized consumers;<br>- Updating the metadata in case of user revocation. | The key decrypts the data without any time constraint |
| **CP-ABE** [71] | Using CP-ABE for key generation | Re-encrypting the whole files | The key can have time constraint with the problem of confidentiality against the Cloud servers |
| **Proxy re-encryption** [118] | Using the proxy re-encryption and CP-ABE for key generation | - The consumer can be offline in case of user revocation;<br>- Several keys are managed by the proxy those related to CP-ABE and those related to the proxy re-encryption. | The verification of access privilege in the context of timely file sharing cannot be performed when the owner is offline |

### 5.2.4 Background on CP-ABE in file synchronization

The Ciphertext Policy Attribute Based Encryption (CP-ABE) can be used both for access control and key management. In fact, it allows the consumer to access the data if it has the appropriate attributes. Meanwhile, it generates the key to the consumer to decrypt the data already encrypted by the owner. The problem of the adoption of CP-ABE, as it is basically defined, presents security threats. In fact, integrating the access verification into the owner side requires a permanent presence to verify access especially when considering the time constraint. In addition, when integrating the CP-ABE access verification into the Synchronization manager, this server has full access to all user decryption keys. This

constitutes an architecture vulnerability to attacks that might be initiated by the server when acting maliciously.

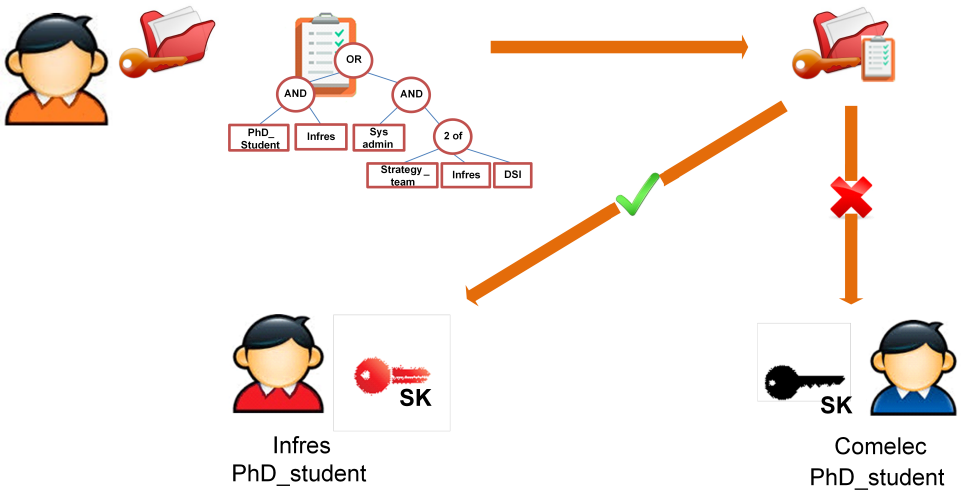



**Figure 5.4:** *Ciphertext Policy Attribute Based Encryption*

For the access control decision, CP-ABE is based on a set of attributes A, and an access policy defined by a series of attribute included in A.

- **Universal Attributes set (A):** It matches to the set of attributes, which can be related to the externalized data, to the user whether it is owner or consumer and can be related also to the environment.
- **Access structure (P):** It is the access policy defined by the consumer. This policy is defined in the guise of a tree structure where the leafs are the different attributes, and the interior nodes are the threshold gates. The "and" and the "or" operators specify the relation between the different attributes.

Four main functions should be used for the generation of the different keys and for the decryption and decryption of the message:

- **Setup:** This function generates the Public Key (PK) and the Master Secret Key (MSK). PK is used by the encryption and decryption functions while the MSK is used by the Key generation algorithm for secret key generation;
- **Key generation (MK, S $\subset$ A):** It generates the Consumer_public_key. This key is customized to the consumer according to the set of his attributes;
- **Encryption(PK, M, P):** This function encrypts the message M using the Public Key (PK) and according to the Policy chosen by the user. It generates therefore, the message M.cpabe;
- **Decryption(PK, M.cpabe, SK):** This function verifies if the consumer attributes meet the owner access control policy. It decrypts the message M.cpabe using the

Consumer_Private_Key. If the message M is retrieved, the user with its attributes is legitimate to access the message content. Otherwise, he is not legitimate.

Several enhancements are introduced into the ABE to meet the secure Cloud storage requirements. They can be classified as follow:

- **Hidding attributes:**
  With ABE, the ciphertexts reveals some information about the attributes used in the defined access policy. Therefore, encryption systems were added to hide the access policies and attributes from the Storage service providers. Several research works focus on this extension to the ABE. It includes proposals that support restricted access structure [81] [74] [106] and other proposals more flexible and expressive with any boolean structure [82].

- **Performance enhancement:**
  Improving the efficiency concerns mainly the applications which need the lowest energy consumption. It is the case of [109] which aims lightweight devices. It adopts constant size secret keys using elliptic curve cryptography. B.Waters who participates in the proposal of the basic CP-ABE, introduces later more enhancement [113]. It proposes a efficient CP-ABE by choosing the size of the ciphertext which depends linearly on the number of attributes used in the access policy.

  Other proposals focus on raising the efficiency by outsourcing the decryption operations into the Cloud [66] as the decryption complexity strongly depends on the access structure. Cryptographic techniques such as the proxy re-encryption were also adopted in the Cloud [49] [118] to reduce the ABE operations.

- **Decentralized trusted authority:**
  In the basic Ciphertext Policy Attribute Based Encryption scheme, data are encrypted under a unique key given by a central trusted authority. Several proposals have migrated towards the adoption of multi-authorities. In fact, each party can become an authority to generate the secret keys. In this context, B.Waters [83] proposes a new collusion resistant solution. There is no need of a perpetual communication between the authorities as only one is required to create a set of common global parameters. Each party creates the public key and private keys of the users according to their attributes. A user then encrypts his data based on an access matrix, the set of public keys gathered from concerned parties and the global parameters. In this proposal, unlike the one adopted in [58], any boolean formula can be introduced, and no central authority is required.

Unlike the previously proposed solutions that deal with a simple access to shared data, the novelty of our proposal is to adjust the CP-ABE to synchronizing shared data while considering the time constraint. In fact, the owner chooses to share his data with specific

consumers only for a period of time. Beyond this period, the consumer has no more privilege to access the data. Considering the context of timely shared data synchronization takes into consideration the states constrains of owners and consumers (Online or offline).

## 5.3 SyncDS: system model and security requirements

The goal of our framework is to ensure a secure data synchronization and sharing between a Local Digital Safe and a Cloud Digital Safe. As a step towards this goal, we need to remind the architecture for data synchronization and present the different security roles of each architecture component.

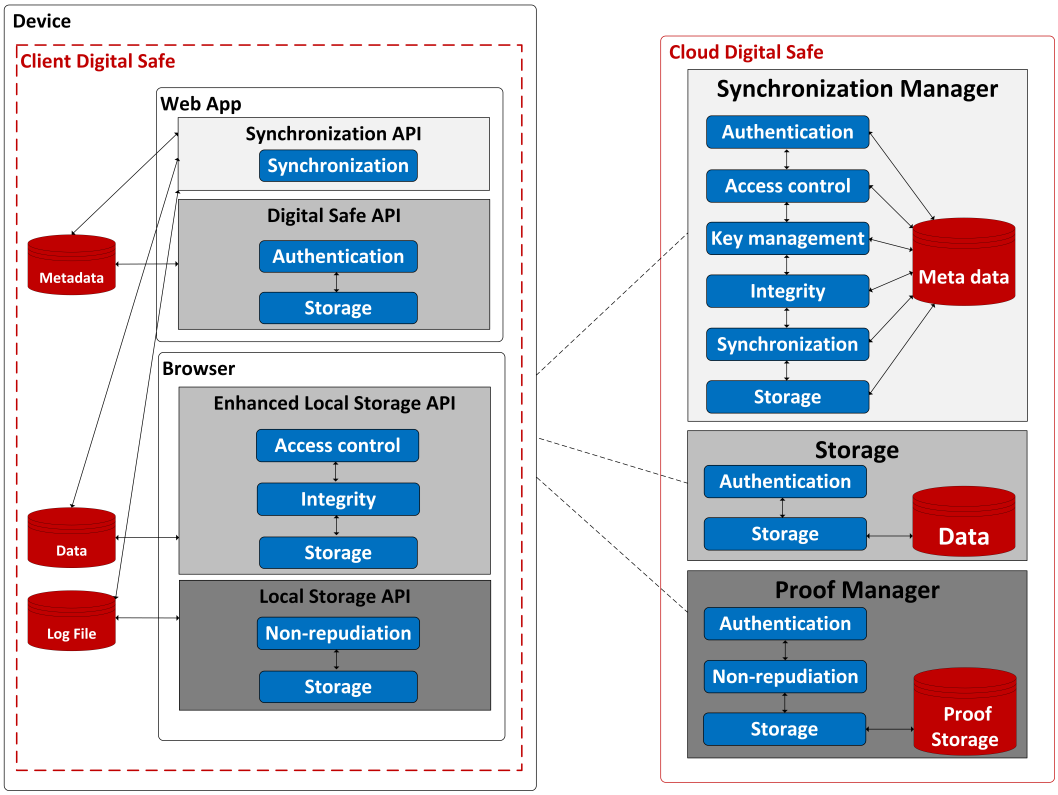



**Figure 5.5:** *Security aspect in SyncDS architecture*

In terms of architecture, shown in figure 5.5, the SyncDS synchronization framework consists on two main components:

- **Client Digital Safe**: The data are stored in a Local Digital Safe. The storage is based on the HTML5 Local Storage API with additional security considerations to follow the Digital Safe requirements.
- **Cloud Digital Safe**: As introduced in [93], it is a standardized architecture that provides a secure environment for storing sensitive document. This environment fully fits both the user and Cloud security requirements. Three main servers form the Cloud Digital Safe:

**- Synchronization manager server:** It is the front component of the synchronization. It notifies the devices of the synchronization and handles the synchronization requests and responses;
**- Storage servers:** They are used to store the blocks of data;
**- Proof manager server:** It preserves the proof of data storage and guarantees the non-repudiation.

The security services are distributed between the architecture components according to their roles in the synchronization. The detailed functionalities are described as follows:

- **Secure storage**:
  To win the trust of users and boost their confidence, it is essential to give a high priority to the security of data both when stored locally and when stored in the Cloud. As the user does not trust remote servers, it is comforting for him to send his data encrypted. The Cloud is considered therefore, as an encrypted blob storage server. Thus, the storage provider cannot access to these data. After the analysis of the different access control strategies used within the context of data sharing, we propose the timely CP-ABE which will be detailed in the next section. The encryption key is therefore generated based on access policy and period of legitimacy.

- **File integrity**:
  In our architecture, the files and directories hashes is introduced into the file metadata. In fact, this mechanism preserves the file integrity. With the introduction of the Hierarchical Hash Tree (HHT) as detailed in the chapter 4, checking the hash value of the file system root directory verifies the integrity of the whole file system. It detects systematically if an object has been altered by a third party, or the file system keeps its original version. In this case, it is no longer necessary to check the hash of files one by one to verify their integrity.

- **Non-repudiation**:
  In the Cloud Digital Safe, a Proof Manager is the entity that stores the metadata signed by the user private key. Thereby, storing a proof guarantees the non-repudiation. Therefore, neither the client nor the service provider can deny having participated in the storage process.

- **Authentication**:
  It consists on verifying the user identity by the Synchronization Manager before any access to the storage and synchronization services. For the data and proof storage, the authentication of the user by the Storage servers and the Proof Manager are also required.

- **Secure synchronization**:
  Securing the messages of synchronization between different architecture entities is

crucial in our architecture. Encrypting the exchanged messages using a mutually agreed session key avoids the retrieval of confidential information in case of session interception. In the conception of our synchronization protocol, the user files are sent encrypted. To avoid multiple encryptions of the same sent message, the encryption will occur at the application layer instead of the transport layer. In this case, only messages with high level of confidentiality are encrypted.

- **Access control**:
  In the case of data sharing between different users, only authorized consumer can access the data. Even the Cloud provider is concerned by the access control not to give him any right to access the data. The synchronization Manager, as a front component of the synchronization architecture, is the first responsible of the access control. In our proposal, the data owner can share data to consumers over a specific period of time. Beyond this period the access is denied. This leads to propose a timely CP-ABE which is subject of our contribution and add the timestamp into the access control verification.

  **- Confidentiality against the Synchronization Manager**:
  The synchronization Manager is a semi-trusted server. Therefore, it is characterized by its honesty and curiosity. It perfectly follows the architecture and protocol requirements, but it tries to collect as much information and data as possible. In our architecture, this server is in charge of the access control and the key management. It is essential, thus, to prevent it to retrieve the key of data decryption.

  **- Confidentiality against access of unauthorized consumer**:
  In our architecture and protocol, access and synchronization of data are restricted to authorized users with appropriate attributes, and within a specific period defined by the data owner.

- **Key management**:
  In case of file sharing, retrieving the decryption key by the consumer is essential. The proposed Timely CP-ABE is used to ensure a secure key distribution at the Synchronization Manager level.

## 5.4 SyncDS: Secured file synchronization

In this section, among the security requirements, we will give an overview on the authentication that should be carried between the different entities of the architecture. We will give a big attention to the access control and key management for the data sharing.

### 5.4.1 Authentication and secure data exchange

The user of the Client Digital Safe should be authenticated by three entities, which are: the Synchronization Manager, the Storage server and the Proof Manager. To avoid multiple authentications based on passwords and certificate, we propose to adopt a hybrid

authentication paradigm. The certificate is used in the authentication between the client and the Synchronization Manager to meet the WebSocket security requirements [31]. A token is used in the authentication between the client and other entities to respect the security design of Cloud Digital Safe storage servers [93]. This token authorizes also the user to post or to retrieve the file from the Storage server and to send the proof to the Proof Manager. The authentication and authorization are managed mainly by the Synchronization Manager. The tokens are generated by this entity since this server is in the foreground of the synchronization, and as it controls the access to the Cloud Digital Safe.

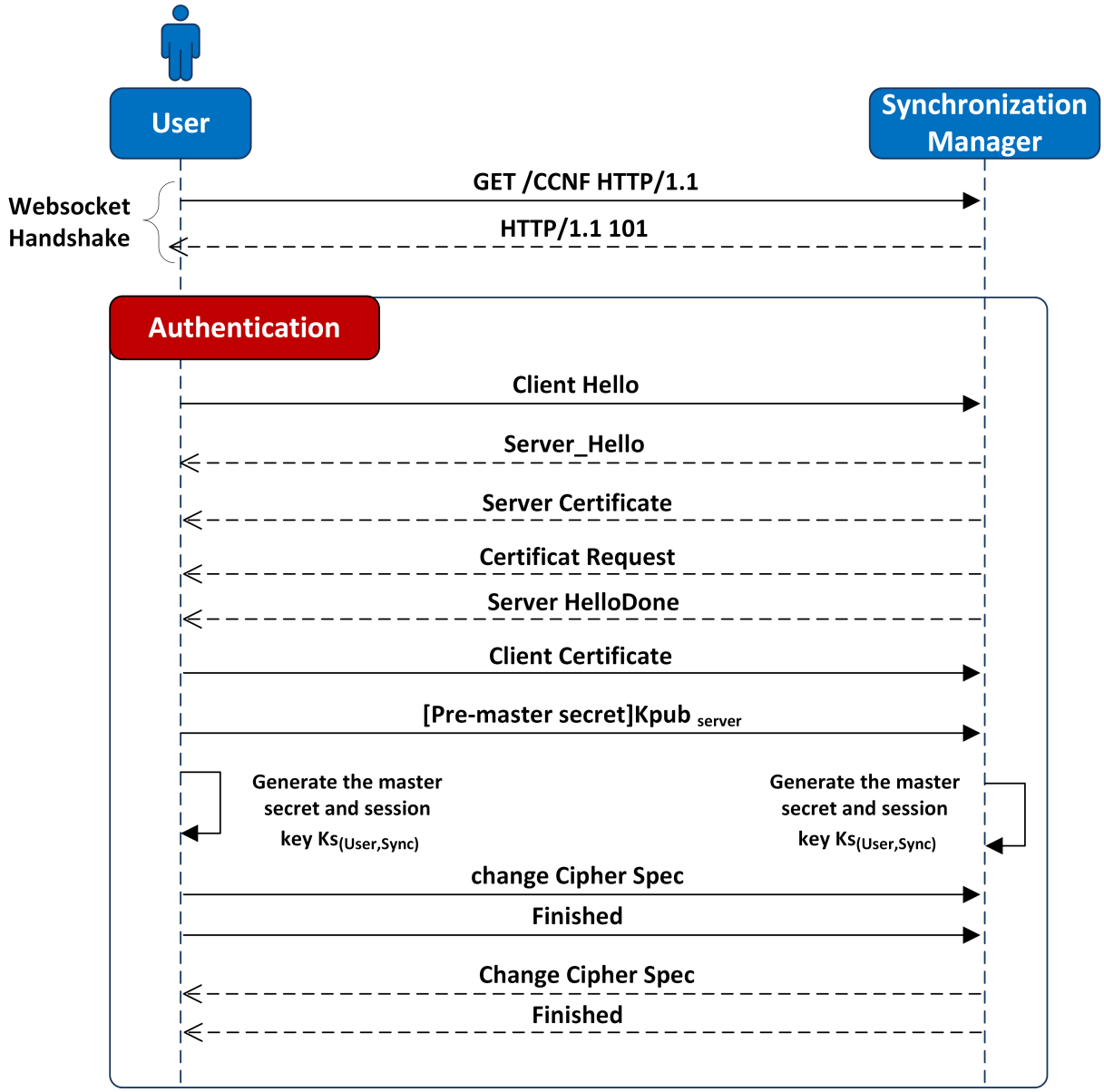



**Figure 5.6:** *Certificate based authentication*

- Certificate based authentication
  As detailed in figure 5.6, the authentication based on the TLS protocol is as follows:

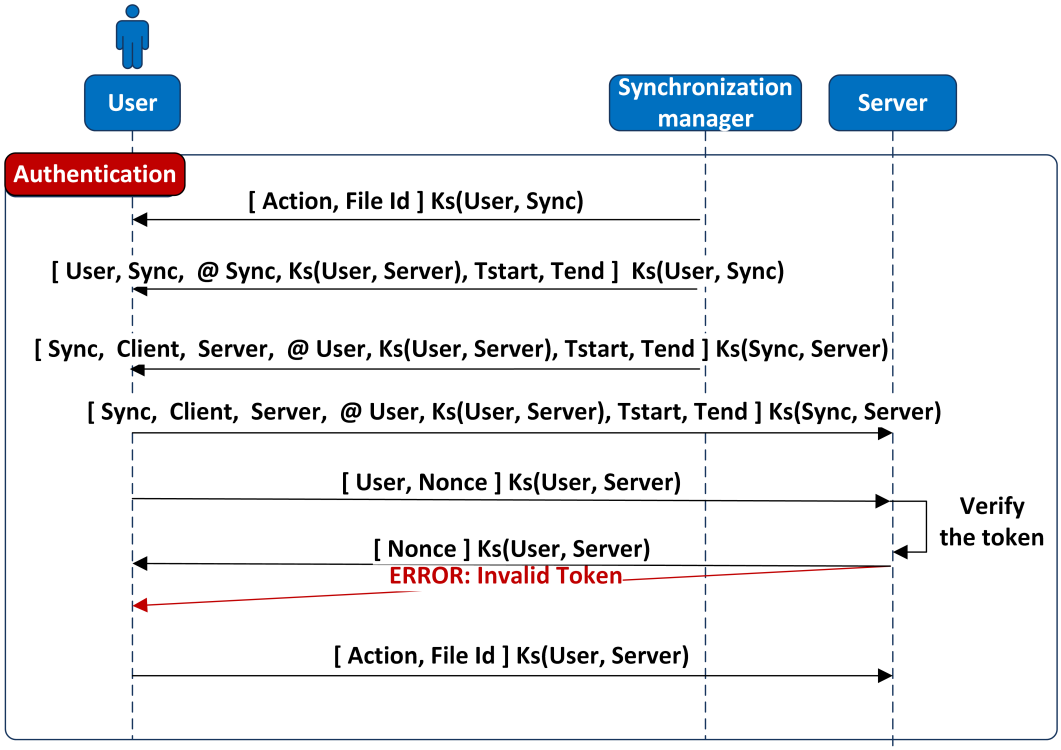



**Figure 5.7:** *Token based authentication*

- First, the client and the Synchronization Manager exchange their certificates and their nonces to ensure a mutual authentication.
- The client sends the Pre-master-secret encrypted with the server public key.
- The both entities generate the master secret based on the sent pre-master-key. Thereafter, the session key Ks(User,Sync) is generated based on the two exchanged nonces and the master key. This key is used to encrypt a part of the synchronization messages exchanged between the user and the synchronization Manager.
- Change cipher spec messages are exchanged preceded by the Finished message in case of changing the specification of the Cipher.

- Token based authentication
  After the authentication between the user and Synchronization manager, the client sends actions which should be carried at the Cloud Digital Safe side. According to the action, related to the Storage servers or the Proof manager, an authentication to the concerned server should be performed following these steps:

  - The Synchronization Manager sends to the user two tokens. The first one T1 is intended to the user and it is encrypted using the session key generated during the certificate based authentication. The second token T2 is designed to the server which can be the Storage server or the Proof Manager. This token is encrypted using a key shared previously between the Synchronization Manager and the server;
  - The user retrieves the encryption key Ks(User,Server) generated by the Synchronization Manager from T1. It transmits to the server the token T2 and a nonce

encrypted with the retrieved key;
- At the server side, the session key Ks(User,Server) is retrieved from the received token T2. The nonce and the validity of the token are verified;
- After the verification, if the user is authorized, a confirmation is sent to the user containing the nonce encrypted with their session key. Otherwise, a refuse message is sent;
- Finally, the user sends the action to the server (deposit or retrieve a file in case of communication to the Storage server and deposit the metadata in the case of the Proof Manager).

**Table 5.2** details on the token fields

| Token field | Field role |
|---|---|
| **Client** | The entity that needs to be authenticated which is the client in our architecture |
| **Server** | the entity that authenticates the client. In our case, we have two servers which should authenticate the client: the Storage servers and the Proof Manager |
| **Address of the client** | The IP address of the client |
| **IP address of the server** | It matches to the IP address of the Storage server or the IP address of the Proof Manager |
| **Symmetric key** | The session key used to encrypt the traffic between the client and the server. This key was a part of the token sent by the Synchronization Manager |
| **Start time:** | the start time of token validity |
| **Expiration time:** | the expiration time of token validity |

The information that should be introduced within the token are listed in the table 5.2 bellow.

### 5.4.2 Timely Ciphertext Policy Attribute Based Encryption

An owner when sharing data has the full right to choose the consumer who can access them. Our contribution, in this context, is to consider a timely file sharing. In fact, in addition to the basic access policy, the owner imposes the period of access (figure 5.8). The privilege to access data begins at the start time of the period and finish at its end time.

The majority of works based on ABE that we found in the literature places the main features under the owner control. However, this strategy does not fit to the timely file sharing expectations. In fact,

- neither the owner nor the consumer is online at the same time during the data access and synchronization;
- with the timely access constraint, the access verification is not performed once. The access or synchronization request should be timestamped to verify periodically if the consumer still has the access privilege or not.

To highlight our timely CP-ABE, the scheme file sharing will be presented at two levels: system level and algorithm level. At the system level, we present the interaction between the different entities with high level operations. The second level describes the low-level algorithms used at the system level.

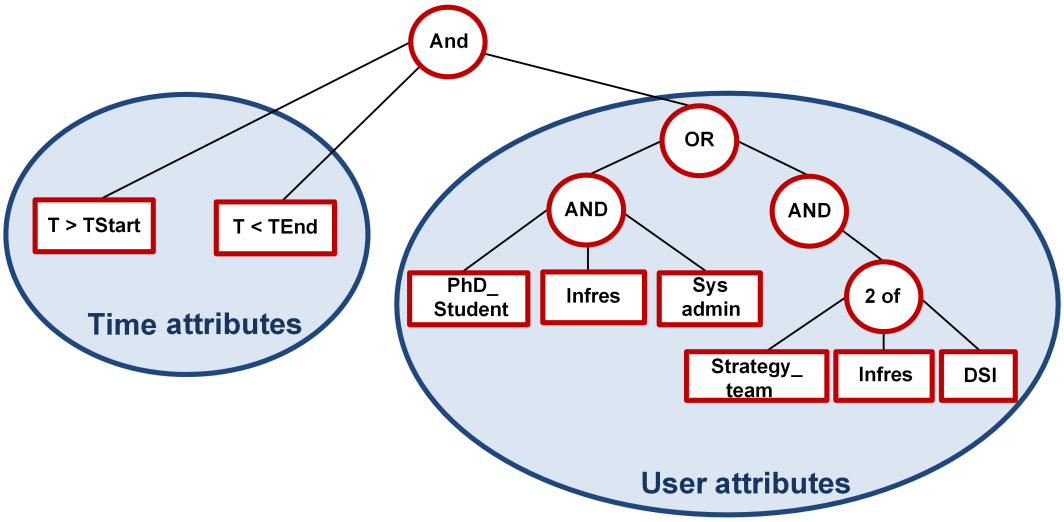



**Figure 5.8:** *Timely access control policy*

### An overview on the access control

We consider that the Synchronization Manager is a semi-trusted server. This means, first, that the server is honest to follow the architecture and protocol requirements. Second the server is curious, so it tries to collect as much information and data as possible. As the data are saved encrypted at the server side, the key of file encryption should not be retrieved by the server. Otherwise, the server can have the full access to the user data.

For this reason, we divide the encryption key into two keys $K_1$ and $K_2$.

- The first key $K_1$ is sent encrypted to the consumer using his public key. Thus, the consumer is the unique entity that can retrieve it. This key is sent only once.
- The second key $K_2$ is sent encrypted using the timely CP-ABE. It is then decrypted by the server to verify the access right of the consumer.

With this division of encryption keys, to access the data, the consumer should obtain from:

- **the owner** the key $K_1$ during the first request of synchronization;

- **the Synchronization Manager** the key $K_2$ and the authorization of downloading the encrypted file as long as his attributes meet the owner access control policy and the access is requested during the legitimate period.

With $K_1$ and $K_2$, the consumer computes the encryption key K and can decrypt the file.

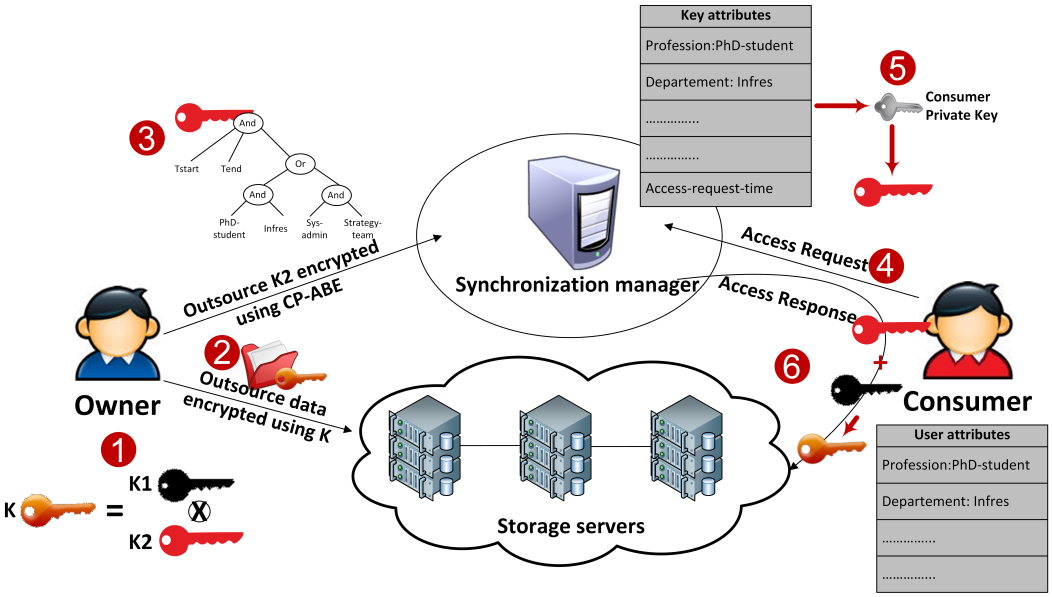



**Figure 5.9:** *Timely file sharing*

### Timely CP-ABE: System Level operation

As depicted in figure 5.9, the high level operation in our architecture can be designed as follows:

- **System setup**: This step focuses on the system initialization:
  -The Synchronization Manager executes the CPABE_setup to generate the Public key and the Master Secret Key.
  -The Public Key is sent then securely to the owner.

- **New File generation**:

  This procedure is ensured by the data owner as depicted in figure 5.10. This entity:

  - generates two keys values $K_1$ and $K_2$;
  - computes K = $K_1 \otimes K_2$ to encrypt the file;
  - encrypts $K_1$ using the public key of the consumer certificate and sends it securely to the consumer;
  - encrypts $K_2$ using CP-ABE with the appropriate policy to be sent to the Synchronization Manager;
  - integrates the $k_2$ into the file metadata;
  - sends metadata to the Synchronization Manager;
  - encrypts the file using the Key K and sends it to the Storage Server.

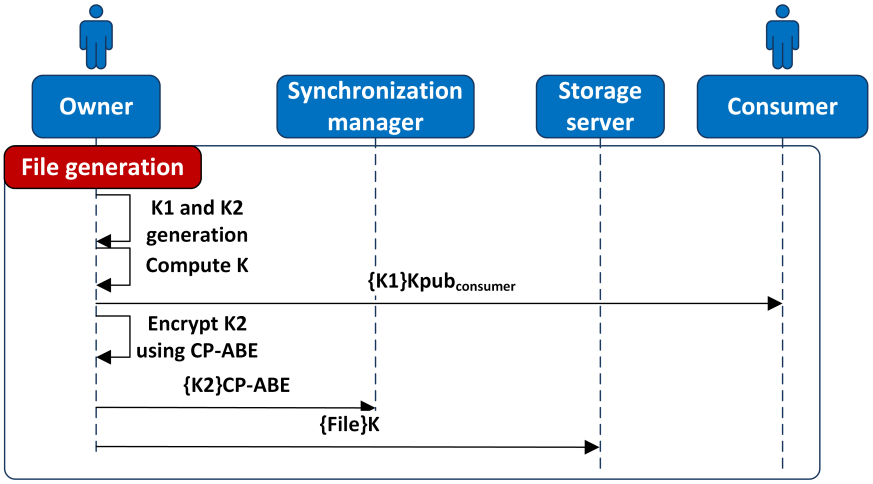



**Figure 5.10:** *New file generation*

- **Consumer access verification**:

  In our proposal, we consider the distinction between the first and continuous synchronization request. In fact, the consumer can be offline when the owner makes changes to his data or to the access policies. In addition, the owner will not necessarily be online when the consumer receives notification for synchronizing. However, we require that both parties are online for the first request of data sharing.

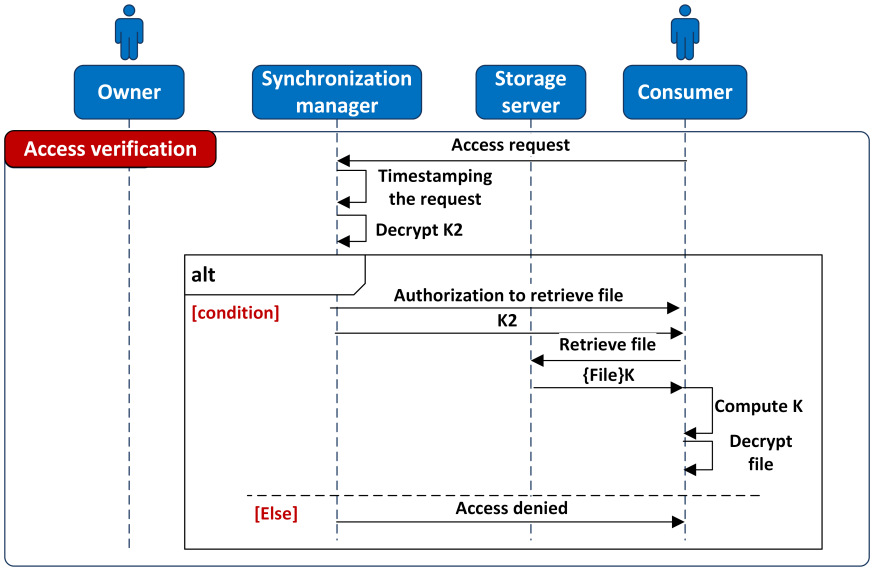



**Figure 5.11:** *Consumer access verification*

In figure 5.11, we present different steps of the verification of consumer legitimacy.

- When a legitimate consumer asks for a file sharing for the first time, it receives, from the data owner, the key $K_1$ encrypted with his public key $\{K_1\}Kpub_{consumer}$. It receives then the key $K_2$ from the Synchronization Manager. In fact, the Synchronization Manager decrypts $\{K_2\}CP-ABE$ based on consumer attributes, current time and timely CP-ABE decrypt operation.

- For a continuous synchronization request, only $K_2$ is sent to the consumer. In fact, the Synchronization Manager rechecks the access privilege of the consumer, and regenerates the new key $K_2$.

The legitimate consumer is the unique entity that can retrieve the data. Even if the Synchronization Manager has the Key $K_2$, it lacks the key $K_1$ to compute K. In the other side, even if the consumer has the key $K_1$, without the access privilege, he cannot retrieve the key $K_2$ from the Synchronization Manager to compute K and to get the authorization to access the data.

- **Owner revocation**:
  When the owner needs to change the access policy, the $K_2$ is encrypted according to the new policy. Even if the owner goes offline after changing the access policy, the Synchronization Manager still handles the access verification. When the consumer has no longer access to the owner data, he is not authorized to retrieve the $K_2$ and to retrieve the latest version of the file.

- **Data deletion**:
  When an owner wants to delete the data sharing, it updates the concerned metadata. It is sent then to the Synchronization manager to update the version of the Cloud.

### Timely CP-ABE: Algorithm Level operation

Among the known ABE methods, we choose to work with the CP-ABE. In fact, in our architecture each user has a set of roles and properties. Depending on these properties, a user gets permission to access the files. These properties can be translated to a set of attributes. In addition, as we focus on a timely file sharing, the time of an access request can be merged with the user attributes to generate the encryption key. The different operations which need to be defined by our Timely CP-ABE are:

- **Setup**:
  It generates the Public Key (PK) and the master secret key (MSK). This function is performed by the Synchronization Manager.

- **Encryption**(PK, $K_2$, Policy):
  Used by the owner, the function encrypts $K_2$ using the Public Key (PK) and according to the chosen access Policy. It generates $\{K_2\}CP-ABE$. This policy is defined by the access in the guise of a tree structure where the leafs are the different attributes, and the interior nodes are the threshold gates. To guarantee that file can be accessed only for a specific period of time (equation 5.1), the user introduces, besides the attributes that form his access policy (Policy'), the time of start and end of the access period interval (figure 5.9).

$$Policy = Policy' \wedge (TStart \wedge TEnd) \tag{5.1}$$

In this case, when the access date is overtaken, the consumer is no more concerned by the synchronization.

- **Key_generation**(PK, Consumer_public_key, MSK, A):
  Used by the Synchronization Manager, this function generates the Consumer_public_key. This key is customized to the consumer according to the set of his attributes. In order to ensure a timely file sharing, the access depends heavily on the period predefined by the owner in his access policy.

  Consequently, the timestamping of the request should be performed at this level, and this time should be added to the set of user attributes when generation the decryption key (equation 5.2).

$$A = UserAttributes \cup Time\_Access\_Request \tag{5.2}$$

- **Decryption** (PK, Consumer_Private_Key, $\{K_2\}CP-ABE$):
  This function verifies the consumer legitimacy. In fact, it verifies first that the consumer has the attributes that meet the owner access policy. Second, it verifies that this request is performed during the period chosen by the owner. When these conditions are satisfied, this function can decrypt $\{K_2\}CP-ABE$ using the Consumer_Private_Key. Otherwise, an access denied alert is raised.

### 5.4.3 Security analysis of data sharing with SyncDS

In our architecture, the secure storage is solved by the externalization the data after their encryption. The file integrity is guaranteed by the verification of the files and directories hash stored in the metadata following the Hierarchical Hash Tree structure and finally the Non-repudiation is ensured by the Proof Manager. In this part, we analyse the satisfaction of the rest of the security requirement.

- **Confidentiality against the Synchronization Manager**:
  While this server is supposed to harm the confidentiality of data, it must be honest to follow the data and key management instructions and to process the request of data access. Even if the Synchronization Manager handles the key managements and verifies the access legitimacy, it holds only a part of the secret. This secret is used only to verify that the consumer and the time of access request fit the access policy imposed by the owner. Thus, it cannot retrieve the user data with this key.

- **Confidentiality against the access of unauthorized consumer**:
  In our proposed protocol, the legitimate consumer who has the access privilege is the unique entity that can retrieve and decrypt the data. Even if the Synchronization Manager has the Key $K_2$, the key $K_1$ misses to compute the file encryption key K.

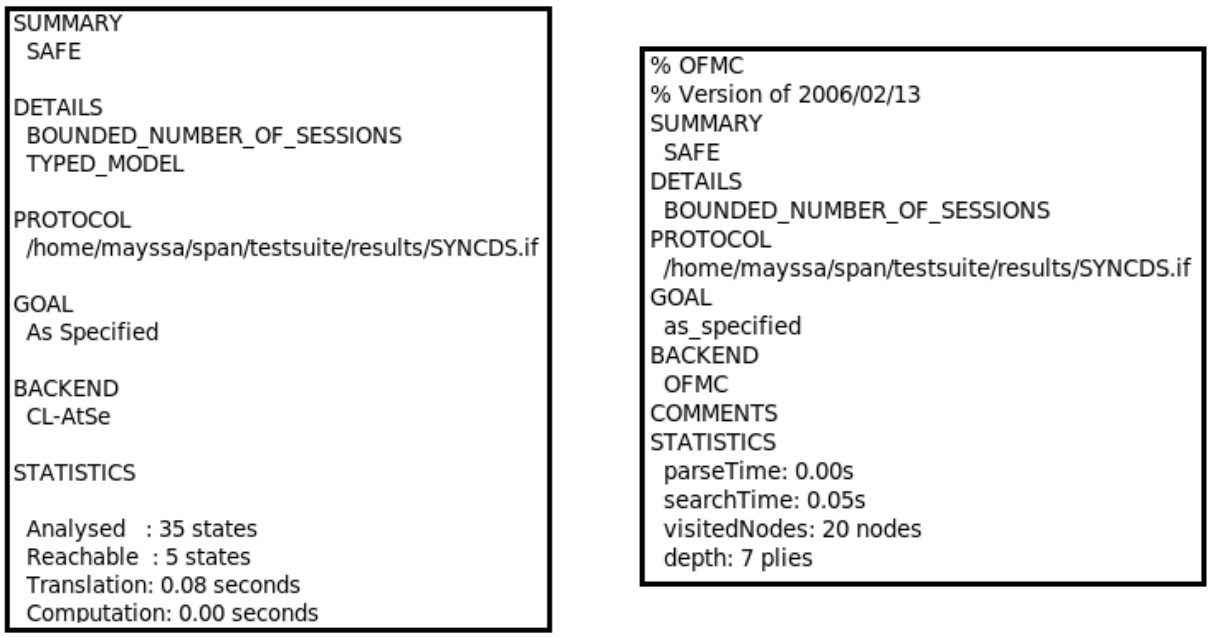



**Figure 5.12:** *Formal security validation using ATSE and OFMC*

In addition, even if the consumer has the key $K_1$ and he has not the privilege of accessing the owner data, he cannot retrieve the key $K_2$ from the Synchronization Manager.

Another case is possible. The consumer retrieved previously the both keys $K_1$ and $K_2$ and actually he has no longer the privilege of accessing the data due to time expiration or policy changes. In this case, the Synchronization Manager, when computing the new key $K_2$ does not give him the authorization to retrieve and synchronize the new version of the data.

## 5.5 Validation and proof of concept

We need to validate the different security requirement of our SyncDS protocol. Therefore, we validate the authentication, authorization and confidentiality of shared data against the service provider using Avispa. We implement also our Timely CP-ABE on the CPABE toolkit.

### 5.5.1 Formal security validation of SyncDS

To validate the security properties of the SyncDS protocol, we need to formalize it. Avispa is one of the most used tools that automatically validate the security of Internet protocols and applications. This tool is based on the specification language called High Level Protocol specification Language (HLPSL). HLPSL is an expressive, modular and role-based formal language. The goals expected from this tool are the authentication of the entities and the secure data exchange.

The protocol model, in algorithm3, presents the messages exchanged between the different entities following the Avispa notations. We choose the back-ends Cl-AtSe and

**Algorithm 3** Formal security validation of SyncDS

*Notation and Initialization*

**A**: Client
**B**: Synchronization Manager
**C**: Storage server
**D**: Proof Manager
**E**: Consumer
**CA**: Certificate Authority

**_Ki**: private key of the entity i
**Ki**: public key of the entity i
**Kij**: key shared only between the entity i and the entity j
**Sid**: id of the session; **Ni**: nonce generated by the entity i
**Ti**: token number i generated by the Synchronization Manager

**Keygen**: function known by A and B to generate a symmetric key based on (Ni, Nj)
**T-CP-ABE**: function known by A and B to encrypt the key K1
**Compute**: function that computes the key K based on K1 and K2
**Initially shared**: $K_{BC}$, $K_{BD}$
**Established during protocol**: $K_{AC}$, $K_{AD}$ retrieved from tokens

*Authentication based on certificate between the client and the Synchronization Manager*
1. A→ B: {A, $N_A$, Sid}$K_B$, {A, $K_A$}_$K_{CA}$
2. B→ A: {B, $N_B$, Sid}$K_A$, {B, $K_B$}_$K_{CA}$
3. A→ B: {A, $N_B$} Keygen($N_A$, $N_B$)
4. B→ A: {B, $N_A$} Keygen($N_A$, $N_B$)

*Authentication based on token between the client and the Storage Server*
5. B→ A: {Action, FileID, T1, {T2}$K_{BC}$}Keygen($N_A$, $N_B$)
6. A→ C: {T2}$K_{BC}$; {userID, $N'_A$}$K_{AC}$
7. C→ $A$ : {$N'_A$} $K_{AC}$
8. A→ C: {Action, FileID}$K_{AC}$

*Authentication based on token between the client and the Proof Manager*
9. B→ $A$: {Action, FileID, T1', {T2'}$K_{BD}$}Keygen($N_A$, $N_B$)
10. A→ $D$ : {$T2'$}$K_{BD}$; {$userID$,$N"_A$}$K_{AD}$
11. D→ $A$: {$N"_A$} $K_{AD}$
12. A→ $D$ : {{$metadata$}_$K_A$} $K_{AD}$

*Access control verification and key management*
13. A → $E$: {$K_1$}$K_E$
14. A → $B$: {$K_2$}T-CP-ABE
15. A → $C$: {File}Compute($K_1$,$K_2$)
16. E → $B$: {Get, FileID}$K_{EB}$
17. B → $E$: {$K_2$}$K_{EB}$
18. C → $E$: {File}Compute($K_1$,$K_2$)

the OFMC for the execution tests. Cl-AtSe is a Constraint Logic (CL-Atse) based Model-Checking of Security Protocols and OFMC is a symbolic model checker for security protocols.

According to the summary results of the tool output, as shown in figure 5.12, the proposed protocol is safe, the goals are as specified and there are no major attacks to SyncDS protocol.

### 5.5.2 Timely CP-ABE implementation

**Table 5.3** Implementation of the timely access control operations

| | Entity | Executed operation | Output of the operation |
|---|---|---|---|
| **0** | **Consumer** | Attributes (Student, Infres_dep) | |
| **1** | **Owner** | Generate (K1) | file named K1 containing the key $K_1$ |
| **2** | **Owner** | Generate (K2) | file named K2 containing the key $K_2$ |
| **3** | **Owner** | Compute (K) | file named K containing the key K |
| **4** | **Synchronization Manager** | cpabe-setup | two files named master_key and pub_key |
| **5** | **Owner** | cpabe-enc pub_key $K_2$ current_date $<$ 1427752800 and *current_date* $>$ 1425164400 and ((Student and Infres_dep) or ( sys_admin and 2 of (strategy_team, comelec_department, DSI) )) | a file named K2.cpabe containing the key $K_2$ encrypted |
| **6** | **Synchronization Manager** | cpabe-keygen -o consumer_public_key pub_key master_key Student Infres_dep 'current_date = '‘*date*+%*s*‘ | file named consumer_public_key |
| **7** | **Synchronization Manager** | cpabe-dec pub_key consumer_priv_key K2.cpabe | the file named K2 |
| **8** | **Consumer** | Compute(K1, K2) | file named K containing the shared data decryption key K |

The validation of our timely file sharing is based on the CPABE toolkit [29]. This toolkit provides a set of cryptographic operations used to implement the Ciphertext-Policy Attribute-Based Encryption scheme. In our architecture, we adjust the toolkit to support our Timely CP-ABE. As shown in table 5.3, the operations to ensure the timely access control are applied to the owner, Synchronization Manager and consumer. We add also to this toolkit the operations handled by the owner to generate $K_1$ and $K_2$ and to compute the Key K. In addition, we implemented the operation handled by the consumer to retrieve

the decryption key.

The owner starts by generating the keys $K_1$, $K_2$ and then K (steps 1,2,3). After the generation of Public Key and Master Secret Key by the Synchronization Manager (4), the owner encrypts $K_2$ following a specific policy for an access during March 2015 (converted into 1425164400-1427752800 as the number of seconds since midnight on January 1, 1970) (5). It sends then the encrypted key to the Synchronization Manager. The Synchronization Manager generates the consumer key (6) by adding the current time into the consumer's attributes (0) and decrypts K2.cpabe (7) to verify the access privilege of the consumer. We consider that the owner has already sent $K_1$ to the consumer during the first synchronization request. The consumer retrieves the $K_2$ from the Synchronization Manager and computes the key K (8) to decrypt the shared data.

### 5.5.3 Performance analysis

For the performance evaluation, we compare the introduction of the timely-file sharing when it is based on our Timely CP-ABE and when it uses a simple access control list. We considered, in our analysis, several random access policy schemes and user attributes that can meet the real file sharing in the context of Digital Safe.

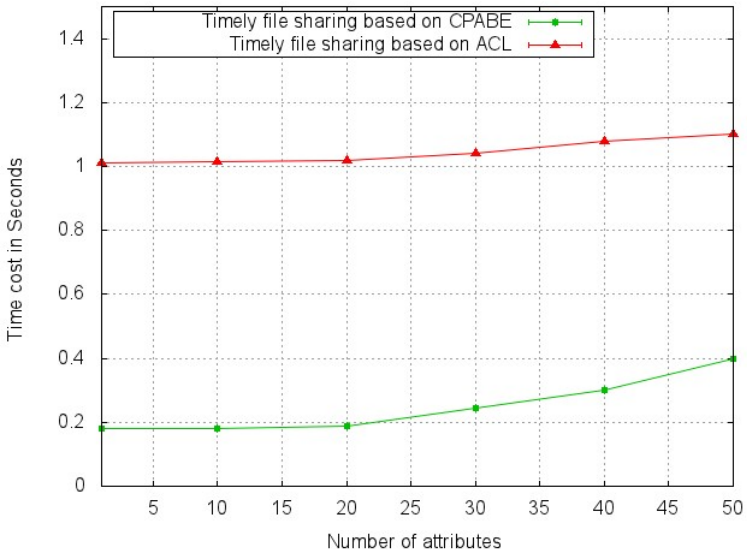



**Figure 5.13:** *Performances of the timely file sharing based on CP-ABE*

In the figure 5.13, the axis of X and Y match respectively to the total number of attributes used in the access policy and the time cost of access verification in seconds. The results show that it is more efficient to adopt the enhanced CP-ABE in the architecture of synchronization. More precisely, the enhanced CP-ABE when adopted for the timely-file-sharing synchronization reduces the verification cost by the fifth compared to ACL. It is even more efficient as it handles the key management besides the access verification which is not done by the ACL.

## 5.6 Conclusion

The main goal of our work is to develop a framework that ensures the data synchronization between the Client Digital Safe based on the HTML5 Local Storage APIs and the Cloud Digital Safe. In addition to the efficiency, our protocol should be characterized by its high quality and security. In this chapter, we have defined the different security requirements of the SyncDS architecture and protocol. We focus on the concept of data sharing by proposing the Timely CP-ABE. It guarantees first that only the legitimate consumers can decrypt the shared files. It highlights also the notion of the timely access control into the synchronization of shared files.
We strongly prove that our architecture and protocol are in line with all the safety requirements in the context of file externalization and synchronization for Digital Safes.

# Chapter 6

# General Conclusion

The majority of existing Cloud storage products adopt proprietary solutions. Their first concern is to offer the data availability to the end user across his different devices. Second, they compete on guaranteeing the best security and confidentiality of user data when externalized and managed by Cloud Service Provider. Proposing proprietary solution has shown its deficiency on security, cost, developer support and customization issues. In this respect, our thesis focuses on standardized solutions and Digital Safe context.

Proposing standardized storage solutions leads to deal with three features: storage at the client side, storage at the server side and data synchronization between both sides. There is already a proposed standardized Cloud Digital Safe on the server side storage that follows the AFNOR specification while there is no standardized solution yet on the client side. This thesis proposes three main contributions that follow the secure and standardizes storage requirements.

The first contribution deals with the proposal of a standardized Client Digital Safe based on HTML5 Local Storage APIs and AFNOR specifications.

In the second contribution, we define a file synchronization protocol with high quality and minimal resource consumption. This efficiency is raised with the WebSocket protocol and with the integration of the Hierarchical Hash Tree into abstract structure to detect changes between two versions of the same file system.

Security challenges of our synchronization protocol is subject of the third contribution. We mainly focus on the access control, and on the concept of timely file sharing were shared data are synchronized to legitimate users only for a specified period of time. We propose therefore, a Timely Ciphertext Policy Attribute Based Encryption.

## 6.1 Summary

Our thesis contributions revolve around the HTML5 APIs and the Digital Safe. Therefore, the first chapter presents an overview of the HTML5 standard and the security issues of the browsers which adopt this standard. Among the HTML5 APIs, we focus on the communication and local storage API. We present their specifications, the considered security measures of these APIs and their gaps.

**Table 6.1** Commercialized storage vs SyncDS solutions

| Criteria | Existent solutions | SyncDS | Contributions |
|---|---|---|---|
| Cloud data storage | Proprietary solutions | Standardized solution based on AFNOR. | - The requirement of the Digital Safe is to guarantee the integrity of the stored data over time.<br>- It is characterized by its probative value as a proof of data storage is stored in a third trusted party. |
| Local data storage | Proprietary solutions | It is a standardized Digital Safe solution which meets the AFNOR specification first, in terms of structure and format, second in terms of provided security services. | - Adopting standardized solutions guarantees transparency to users and to service providers.<br>- No software installation is required.<br>- A wide variety of operating systems (OS) is supported.<br>- Availability on all devices with a compatible Web HTML5 engine is guaranteed. |
| Adopted protocols | The synchronization protocol is based on HTTP and Rest API for messages exchange. It uses long HTTP polling (Dropbox) and XMPP (Google drive) to push notifications to users. | The WebSocket protocol is adopted to ensure a bidirectional communication between the Client and the server. It is used both for data exchange and notification sent from the server to the client. | - Reducing the number of exchanged messages with the bidirectional communication.<br>- Reducing the time of synchronization with a reduced overhead size of exchange messages. |
| Data security | Data are encrypted by the server with a user independent key. | Data are encrypted at the Client side with a user dependent key. | - The data are stored secured with the enhanced HTML5.<br>- The Cloud is considered as an encrypted blob store.<br>- The storage provider cannot access to user data. |
| Data sharing | Access Control List (ACL) is used to define who can share the owner data. | The owner defines his security policy. He uses the Timely CP-ABE to encrypt a part of the key which will be sent to servers | - Confidentiality against the Service provider<br>- An access control just for a period of time. |
| Change detection | Change detection is based on file names and folders and hash of files | Change detection is based on the Hierarchical Hash Tree in addition to files and folders names. | - Efficient file synchronization with a reduced change detection of two file system versions.<br>- Efficient verification of files integrity. |

In the second chapter, we focus on the HTML5 Local Storage APIs. We mix HTML5 and Digital Safe standard to propose a standardized Client Digital Safe. In the first contribution, we discuss the security measures which should be guaranteed by local storage solutions and we propose to enhance the security of data stored using HTML5 APIS. In fact, we add data confidentiality by encrypting data locally, data integrity and metadata integrity. In the second contribution, we use the enhanced APIs in the conception of Client Digital Safe based on AFNOR specifications. As a proof of concept, we implement the enhanced Local Storage APIs and the Client Digital Safe. Implementing our proposition in the Chromium browser and evaluating performance prove that adding the data protection into the Local Storage APIs is crucial and efficient.

In the third chapter, we propose an architecture and a protocol called SyncDS, that ensure file synchronization in a probative value Cloud. To present an efficient protocol with minimum resource consumption, two keynote novelties are highlighted: first, the integration of the Hierarchical Hash Tree into the metadata abstract to detect the changes

between two versions of the file system, second, the integration of the WebSocket protocol and server into the synchronization architecture.
Several analytical explanations and empirical evidences are presented to emphasize that using our framework, reduces the time of file synchronization across devices and reduces the time of change detection.

In addition to the efficiency, SyncDS protocol and architecture should be characterized by their high quality in terms of security. In the fourth chapter, we have defined the different security requirements of the SyncDS architecture and protocol. We mainly focus on the concept of timely data sharing based on the CP-ABE to guarantee that only the legitimate consumers can decrypt the shared files. By highlighting the notion of the timely-based access control into the synchronization of shared files, the consumer legitimacy is limited in time. He can receive the last version of owner version only during a period predefined by the owner. This period is added into the attributes, and the event of data synchronization is timestamped.
We strongly prove that our architecture and protocol are in line with all the safety requirements in the context of file externalization and synchronization for Digital Safes.

The table 6.1 highlights the contributions of our synchronization architecture and protocol SyncDS compared to the commercialized solutions.

## 6.2 Future work and open issues

There are several interesting open issues and possible extensions that deserve to be the concern of future works:

- **HTML5 Local Storage APIs in the IoT context**
  Web applications are adopting intensively the client side storage. Besides the web application and user data, these APIs can be used in the context of Web of Things (WoT). First, many projects [6] [15] [19] are focusing on the skills of client side web developing languages (JavaScript, HTML5) to start managing elements in the physical world. Second, the fog computing [64] and its integration into the IoT make the data storage and computing dense geographically distributed across different smart things. Therefore, web applications used in WoT context emphasize the need to store data locally in smart things.

- **Collaborative data sharing**
  In the context of Digital Safe, we focus mainly on sharing document with a single write and multiple reads constraint. With our protocol, modifications into the same file cannot be occurred by different users at the same time. Otherwise, a conflict resolution is raised. The owners then chooses the version he wants to keep.

  Several proposed solutions [111] [89] [8] provide the possibility of sharing and collaborating on files. Multiple users can therefore access the same file and edit it at the

same time. These solutions adopt the server side encryption rather than encrypting data before their externalization. Thus, data and history of edits are revealed to the service provider.

As future work, it is interesting to deal with file sharing and collaborative editing when the owner chooses to externalize his data encrypted. Clearly, several issues can be raised such as the data conflict resolution, time constraint of the data management. Of course, data security remains the most challenging issue and especially the choice of the appropriate access control strategy that should meet all the collaborative edits requirements.

- **Dealing with data distribution in the context of Digital Safe**:
  Different strategies can be adopted to ensure the data availability facing user mobility and devices variety. These strategies can be classified into replication and distribution.

  In case of replication which is adopted by our protocol SyncDS, data are completely synchronized. Any modification that occurs in one device should happen in the rest of user devices. As a result, the full data are present in every device. Regarding the distribution, each device holds a portion of data. The responsibility is load balanced between the different devices.

  Adopting the data replication is the appropriate strategy in the case of smart user devices with a low storage memory and that are unable to store a big amount of data. Future works can deal with the secure distribution of local stored data across many devices according to the Digital Safe requirements, the user behavior and the device capacities.

## 6.3 Publications

- M.Jemel, M.Msahli, A.Serhrouchni: SyncDS: A Digital Safe Based File Synchronization Approach. The 13th Annual IEEE Consumer Communications and Networking conference CCNC 2016. Janvier, 2014, Las Vegas, USA.

- M.Jemel, M.Msahli, A.Serhrouchni: Digital Safe: Secure synchronization of shared files. The 11th International Conference on Information Assurance and Security IAS 2015. December, 2015, Marrakech, Morroco.

- M.Jemel, A.Serhrouchni: Toward user's devices collaboration to distribute securely the client side storage. CFIP & NOTERE 2015 . July, 2015, Paris, France.

- M.Jemel, A.Serhrouchni: Security assurance of local data stored by HTML5 web application. The 10th International Conference on Information Assurance and Security IAS 2014 . Novembre, 2014, Okinawa, Japon.

- M.Jemel, A.Serhrouchni: Content protection and secure synchronization of HTML5 local storage data. The 11th Annual IEEE Consumer Communications and Networking conference CCNC 2014 . Janvier, 2014, Las Vegas, USA.
- M.Jemel, M.Msahli, A.Serhrouchni: Towards an efficient file synchronization in probative value Cloud. 12th International Conference on Collaborative Computing: Networking, Applications and Worksharing. November 2016, BEIJING, China.

# Résumé de thèse

## 1 Introduction

L'un des principaux objectifs des solutions de stockage Cloud est de garantir la flexibilité et la haute disponibilité des données des utilisateurs. Ainsi, faire face aux besoins de mobilité et la diversité des appareils utilisés émerge comme un défi majeur. D'un côté, les données doivent être synchronisées automatiquement et en contenu lorsque l'utilisateur passe d'un équipement à un autre. D'un autre côté, les services doivent offrir à chaque utilisateur la possibilité de partager ses propres données avec différents utilisateurs. La gestion des données entre de multiples appareils nous mène à mettre le point sur deux entités principales. Premièrement, nous trouvons le framework du côté du serveur qui sauvegarde les données des utilisateurs dans des serveurs centralisés. En deuxième lieu, il y a l'application du côté client qui assure le stockage des données localement dans l'appareil de l'utilisateur et leur synchronisation avec le contenu du côté serveur.

Cependant, les frameworks développés dans les deux côtés sont principalement basés sur des solutions propriétaires dites solutions privées ou fermées. Toutefois, cette stratégie a toujours montré ses lacunes en termes de problèmes de sécurité, de coût, de simplicité pour les développeurs et de transparence des solutions.

La motivation de nos travaux est liée à la proposition d'un Coffre Fort Cloud standardisé. Pour assurer la disponibilité des données à travers les différentes machines, retrouver un Coffre Fort coté serveur nous mène à proposer un Coffre Fort coté client. Il permettra ainsi aux utilisateurs d'une part de gérer ses données même lorsqu'ils sont déconnectés. D'autre part, il assurera la synchronisation de son contenu avec celui du Cloud et des autres machines.

Le Coffre Fort numérique Cloud est défini dans le cadre du projet de recherche Gsafe (Governmenet Safe). Tout en attachant une importance à la valeur probante dans le stockage Cloud, le projet propose une architecture standardisée pour le stockage des documents sensibles. Il fournit un environnement sécurisé pour garantir l'intégrité au fil du temps. La conception de ce Coffre Fort suit les spécifications définies dans le standard d'AFNOR.

Pour suivre les exigences du stockage standardisé et la disponibilité des données, trois contributions font l'objet de cette thèse. Nous proposons dans une première partie (I) un Coffre Fort numérique Client standardisé basé sur les APIs de HTML5 et sur les spécifications

d'AFNOR. Dans les deux autres contributions, nous proposons un protocole de synchronisation d'une part, (II) efficace avec une consommation minimale des ressources et d'autre part, (III) sécurisé pour assurer le partage des données en suivant les exigences du Coffre Fort Numérique.

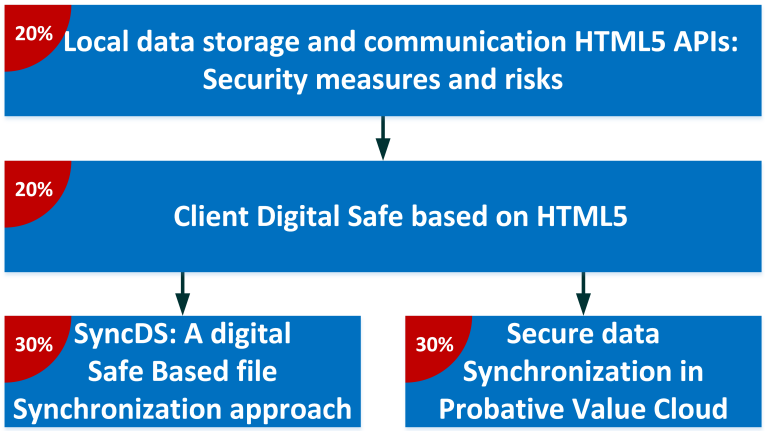



**Figure 1:** *Le plan de contributions*

Nos travaux de thèse manuscrit de thèse est structuré en cinq chapitres principaux comme illustré dans la figure 1

Dans le deuxième chapitre, nous présentons la révolution web ainsi que les fonctionnalités innovantes et les API de HTML5. Conformément aux exigences de la gestion des données multi-appareils, nous détaillons les spécifications des API HTML5 qui assurent la communication et le stockage des données locales. Les mesures de sécurité prises par chaque API et leurs lacunes en matière de sécurité sont détaillées.

Dans le troisième chapitre, nous commençons par introduire les mesures de sécurité fondamentales du stockage local. Ensuite, nous présentons les améliorations que nous avons apportées aux APIs de stockage local de HTML5. Sur la base des APIs de HTML5 améliorées, nous présentons ensuite la conception de notre Coffre Fort Client standardisé en identifiant et décrivant les différentes spécifications de ce Coffre Fort.

Le quatrième chapitre traite le protocole de synchronisation entre le Coffre Fort Client et le Coffre Fort Cloud. Nous commençons par présenter, la comparaison et l'analyse des différentes approches adoptées pour les protocoles de synchronisation existants. Nous traitons aussi les différentes stratégies utilisées pour détecter les changements entre deux versions d'un même systèmes de fichiers. Pour faire face à nos contributions, dans la deuxième partie, nous commençons par identifier et détailler les différentes entités de notre architecture de synchronisation ainsi que les messages échangés entre elles. Dans la troisième partie, nous proposons l'introduction de l'arbre hiérarchique de Hashage (HHT) dans la structure de l'abstrait, et nous détaillons l'algorithme associé pour la détection des changements entre deux version du système de fichier. Pour prouver l'efficacité de notre protocole avec l'adoption

de HHT et le protocole WebSocket, nous présentons à la fin du chapitre, des explications analytiques et empiriques évidences.

Le chapitre cinq commence par une introduction des services de sécurité et des mécanismes adoptés par les architectures de synchronisation des données en général. Nous abordons ensuite les problèmes de sécurité; en particulier, le contrôle d'accès pour le partage des données tout en suivant les exigences du Coffre Fort numérique. Par la suite, un contrôle d'accès qui adopte une dimension temporelle est proposé en se basant sur CP-ABE. Nous terminons notre proposition avec la validation de notre protocole en matière de sécurité.

## 1.1 Coffre Fort Numérique Client basé sur HTML5:

Des solutions propriétaires sont toujours adoptées pour assurer le stockage côté client et la synchronisation des données. Ces solutions ont prouvé leurs insuffisances en terme de mobilité, portabilité, transparence, efficacité et sécurité.

Pour remédier à toutes ces lacunes et adopter des solutions standardisées, nous proposons un Coffre Fort Numérique basé sur HTML5. Il permet à l'utilisateur de gérer ses propres données en toute sécurité lorsqu' il est connecté ou déconnecté. Standardisé, notre stockage côté client repose principalement sur deux standards: HTML5 et AFNOR.

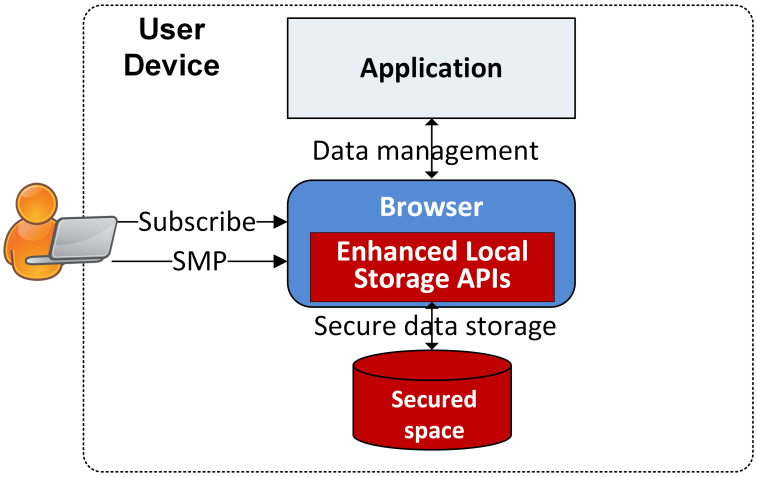



**Figure 2:** *Structure du système pour la protection des données*

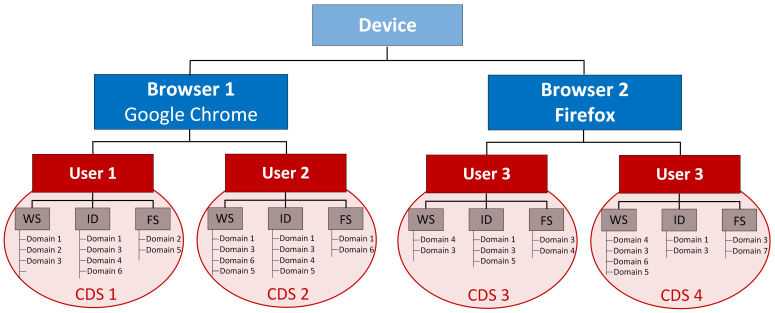



**Figure 3:** *Niveau utilisateur dans la hiérarchie du stockage*

La première partie de nos travaux apporte deux contributions principales. Dans un premier temps, nous nous focalisons sur la sécurité des APIs de HTML5 qui assurent le stockage des données en local (figure 2). Cette étape demeure indispensable pour la conception de notre Coffre Fort qui repose principalement sur ces APIs. Il est alors impératif de s'appuyer sur une base sécurisée. Après une analyse de la sécurité de ces APIs, nous ajoutons la confidentialité des données en les chiffrant localement, la vérification de l'intégrité des données et l'intégrité des métadonnées en chiffrant les noms attribués aux bases de données utilisées par les APIs. Nous avons ajouté aussi un niveau utilisateur à la hiérarchie du stockage des données comme illustré dans la figure 3.

Dans la deuxième contribution, nous nous basons sur les APIs de stockage de HTML5 améliorées pour définir un Coffre Fort numérique Client. Il s'agit de la projection du Coffre Fort Numérique Cloud sur la machine de l'utilisateur. Pour ce faire, nous définissons une nouvelle API là où les spécifications du standard AFNOR sont introduites. Cette API garantit l'interopérabilité entre les APIs de HTML5 et le Coffre Fort situé dans le Cloud. Les données stockées dans ce Coffre Fort font l'objet de synchronisation dans la deuxième partie de la thèse. Notre Coffre Fort Client apporte les caractéristiques suivantes par rapport aux solutions existantes:
- La standardisation puisqu'il s'appuie sur deux standards qui sont HTML5 et AFNOR;
- La portabilité et la mobilité avec des solutions qui ne dépendent pas de la machine utilisée;
- La sécurité puisqu'il suit les exigences définies dans le Coffre Fort Cloud en termes d'accès et de fonctionnalité ;
- La transparence qui est apportée par l'utilisation des standards.

### 1.1.1 Architecture du réseau

Notre Coffre Fort Client est l'image du Coffre Fort Cloud sur la machine de l'utilisateur. Par conséquent, il devrait avoir les mêmes caractéristiques et les spécifications du Coffre Fort hébergé dans le Cloud. Différent de la solution existante, notre Coffre Fort Client se distingue par son caractère non-propriétaire. En fait, il est basé sur les API de stockage local de HTML5.

Dans la figure 4, nous présentons l'architecture réseau de notre framework. Nous commençons par présenter les composants du Coffre Fort Cloud, puis ceux du Coffre Fort Client.

Le Coffre Fort Cloud est composé de trois éléments principaux, qui sont:

- **Serveur de stockage des données**: Les données utilisateur sont divisées en blocs et stockés dans des serveurs. De nombreux services de stockage Cloud peuvent être utilisés tels qu'Amazon S3, Hadoop Distribuer Système de Fichiers (HDFS), Google File System (GFS) etc. En général, un service de stockage dans le Cloud devrait satisfaire la disponibilité des données, la fiabilité, la sécurité, l'accès aux données, etc. Ce système

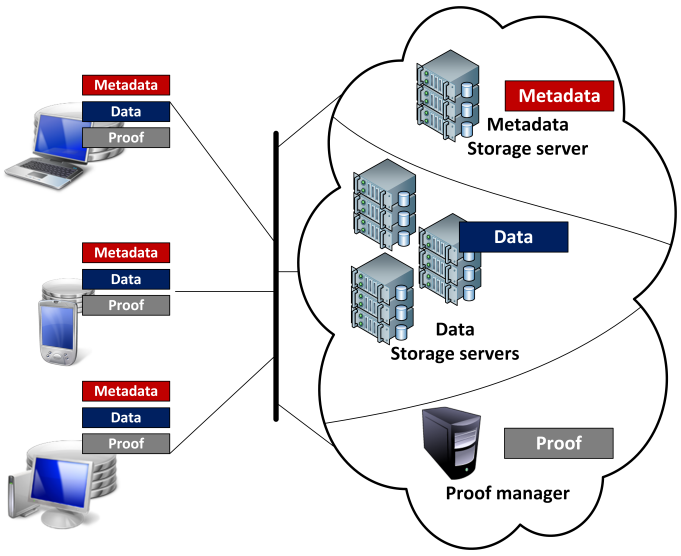



**Figure 4:** *Architecture réseau*

devrait donc prendre en charge la distribution des données, la concordance des SLA, la qualité de service, l'attribution des pouvoirs, l'audit, la certification et le contrôle d'accès;

- **Serveur de stockage des Metadonnées**: Une métadonnée dans le cas d'un Coffre Fort est un fichier XML qui répertorie les différentes informations sur un fichier. Ces fichiers XML sont stockés séparément des blocs de fichiers dans un serveur géré par le fournisseur de services du Coffre Fort Numérique;

- **Gestionnaire de preuves**: Il s'agit d'un tiers de confiance qui préserve la preuve de stockage de données. La preuve consiste en des métadonnées signées par le propriétaire à l'aide de sa clé privée. Cette preuve garantit la non-répudiation et ajoute la valeur probantes dans le stockage. Par conséquent, il peut être utilisé en cas de litige.

Trois types d'informations sont gérés par le Coffre Fort Cloud: les métadonnées, les données et la preuve de stockage. Ces informations sont stockées respectivement dans le serveur de stockage de métadonnées, les serveurs de stockage de données et le gestionnaire de preuves. Néanmoins, les APIs de stockage HTML5 définies par la norme W3C ne peuvent pas être utilisées dans le contexte du Coffre Fort. C'est pour cette raison que des fonctionnalités supplémentaires doivent être prises en compte par ces API. Ces API doivent être utilisées pour stocker:

- les métadonnées qui suivent la structure de celles stockées dans le Coffre Fort Cloud;

- les données qui suivent la spécification AFNOR. Dans le contexte du Coffre Fort numérique, les données de l'utilisateur sont stockées dans un système de fichiers

avec un ensemble de répertoires et fichiers. Ainsi, nous nous concentrons, parmi toutes les API de stockage local, sur l'API du HTML5 FileSytem;

- la preuve du stockage est envoyée au gestionnaire de preuves. Dans notre cas, une partie de la preuve sera considérée comme l'enregistrement des différentes opérations effectuées sur le Coffre Fort Client dans un fichier de journal.

### 1.1.2 Architecture du déploiement

L'architecture de déploiement est créée en projetant des blocs fonctionnels logiques du Coffre Fort vers un environnement physique afin de respecter les exigences de l'architecture réseau et de la spécification du Coffre Fort. Comme l'illustre la figure 5, dans l'architecture de déploiement, nous ajoutons les entités suivantes au niveau de l'application:

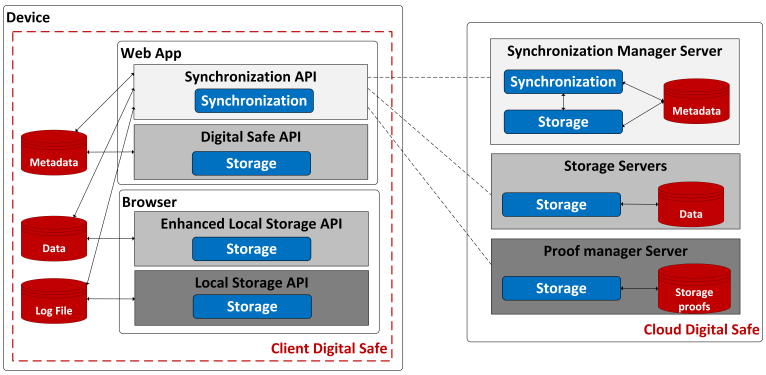



**Figure 5:** *Architecture de déploiement*

- **API de synchronisation**: Elle assure la communication entre le Coffre Fort Client et Cloud pour synchroniser leur contenu de données. Cette API a un accès complet aux données et métadonnées stockées et au fichier journal où sont stockées les épreuves de stockage.

- **API Digital Safe**: Pour gérer l'objet numérique, le client peut utiliser un ensemble de fonctions qui sont décrites par la norme AFNOR NF Z42-020. Pour être conforme à la norme, la composante de sécurité numérique doit mettre en oeuvre ces fonctions. Bien entendu, avant toute exécution par le système, il est essentiel de vérifier que l'utilisateur a le droit d'exécuter l'action en fonction de son profil et de la politique de contrôle d'accès.

### 1.1.3 Implementation et evaluation des performances

Pour la validation de notre Coffre Fort, nous implémentons, d'une part, les modifications apportées aux APIs de HTML5 de stockage. D'autre part, nous implémentons notre API Digital Safe basé sur Javascript. Comme indiqué dans la figure 6 avec les rectangles en rouge,

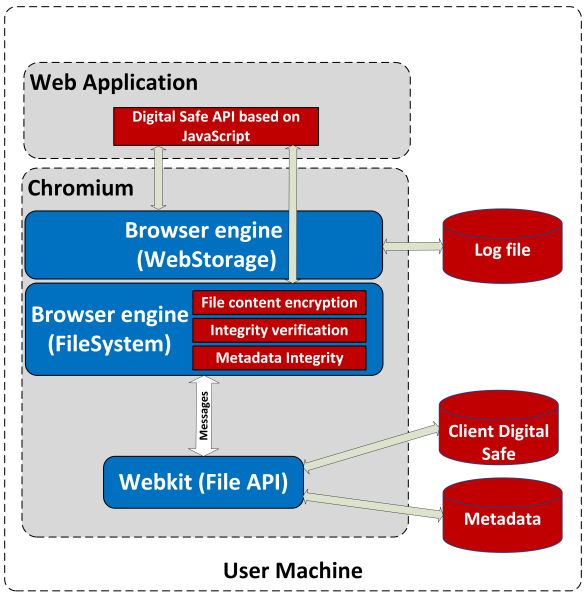



**Figure 6:** *Implementation du Coffre Fort Client*

nous avons ajouté les fonctions de sécurité au niveau du Browser Engine de Chromium et nous avons développé l'API Digital Safe au niveau de l'application Web.

Nous comparons, dans les figures 7 et 8, les performances des APIs améliorées avec une librairie qui n'apporte que la confidentialité des données. Les résultats montrent la présence d'une légère dégradation des performances par rapport aux APIs basiques de HTML5, ce qui nous semble attendu vu l'ajout de nouvelles fonctionnalités à chaque API. Toutefois, ces dégradations restent minimes comparées à celles apportées par la librairie Dojox.

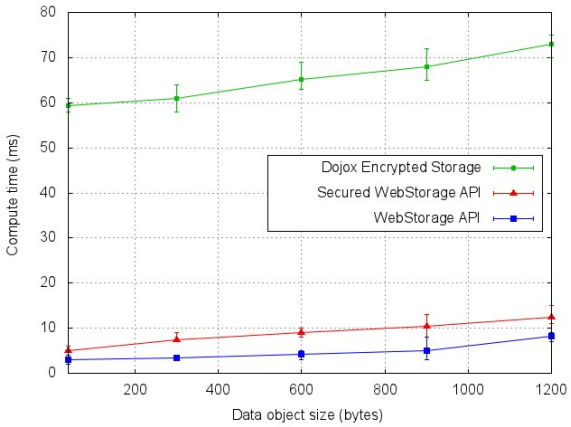



**Figure 7:** *Performance du stockage des données avec setitem*

## 2 SyncDS: Synchronization efficace des données:

La deuxième partie de notre thèse porte sur une architecture et un protocole de synchronisation dans le contexte du Coffre Fort Numérique appelé SyncDS (Synchronization protocol in the context of Digital Safe). La synchronisation est assurée entre le Coffre Fort Numérique

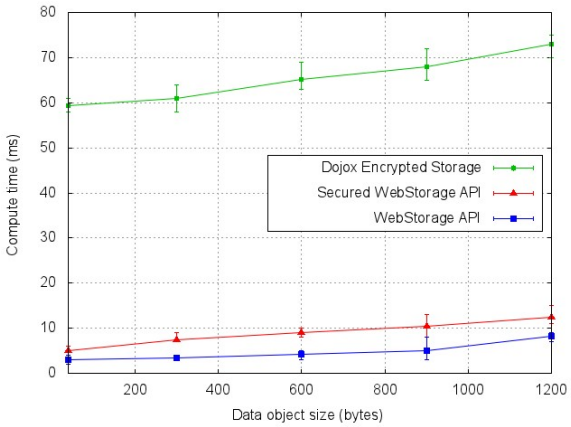



**Figure 8:** *Performance de la récupération des données avec getitem*

Client basé sur HTML5 et le Coffre Fort Numérique Cloud.

## 2.1 SYNCDS: ARCHITECTURE DE SYNCHRONISATION

Dans le cadre du Coffre Fort, nous proposons une architecture, figure 9, composée de quatre couches principales, qui sont:

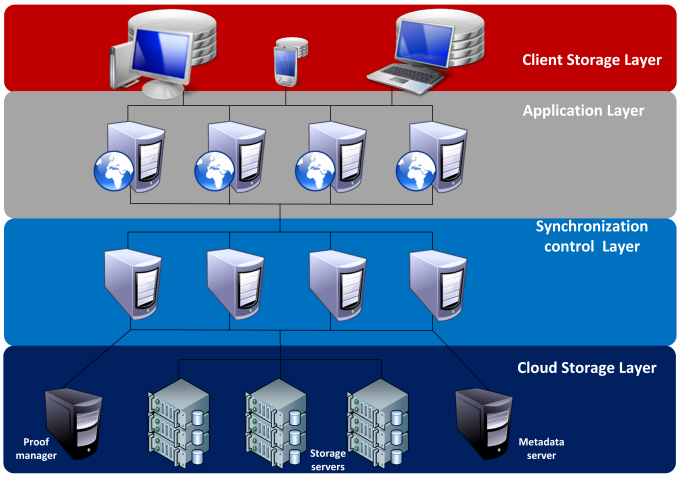



**Figure 9:** *Architecture de synchronisation entre Coffres Forts Client et Cloud*

- *la couche de stockage Client*: le Coffre Fort Numérique Client qui sauvegarde les données localement en toute sécurité
- *la couche Application*: gère la synchronisation au niveau de l'application
- *la couche de synchronisation*: gère les messages échangés entre le Coffre Fort Client et Cloud en se basant sur le protocole WebSocket. A ce niveau, nous trouvons la gestion des conflits ainsi que la notification des utilisateurs concernés par les modifications
- *la couche de Stockage Cloud*: une architecture standardisée qui fournit un environnement sécurisé pour le stockage des documents sensibles dans le Cloud.

## 2.2 SyncDS: protocole de synchronisation

Notre protocole de synchronisation SyncDS met l'accent sur la haute qualité et la consommation minimale des ressources dans un environnement sécurisé. Il présente deux nouveautés principales.

La première est le caractère non propriétaire de l'architecture de synchronisation. Ce caractère demeure d'une très grande importance pour intégrer une large gamme de produits et d'appareils concurrents. Nous nous basons sur un Coffre Fort Client standardisé qui fait le sujet de la première partie de la thèse. Nous nous basons également sur l'API HTML5 Websocket. Ce dernier assure une communication bidirectionnelle entre le client et le serveur. Nous proposons de l'utiliser pour notifier l'utilisateur des éventuelles modifications qui ont besoin d'être synchronisées. Il est utilisé aussi pour les échanges de messages de synchronisation et pour le transfert de fichiers

La deuxième nouveauté apportée par notre solution est l'intégration de l'Arbre Hierarchique de Hashage (HHT) pour détecter les changements entre deux versions du même système de fichiers. Le HHT a été adopté dans des travaux antérieurs pour détecter les changements entre deux versions d'un même fichier. Notre travail est le premier dans son genre qui adopte le HHT dans le contexte de synchronisation de fichiers sécurisé pour détecter les changements entre les systèmes de fichiers en entiers. Le protocole SyncDS détient trois phases (figure 10):

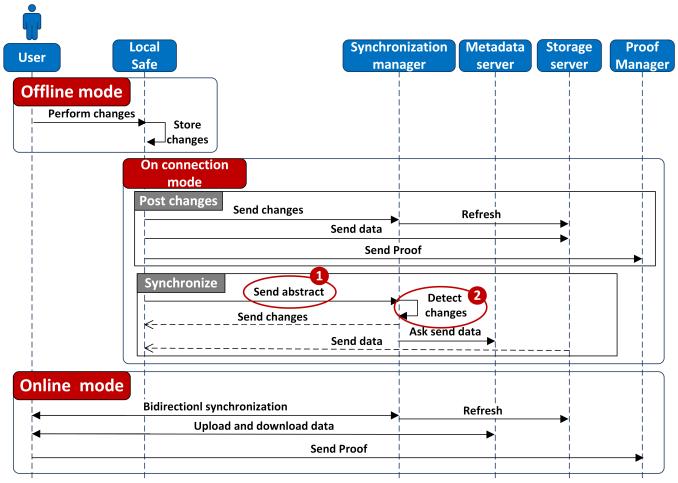



**Figure 10:** *Différentes étapes du protocole de synchronisation SyncDS*

- *phase offline*: Lorsque l'utilisateur est hors ligne, il peut effectuer des modifications au niveau de son Coffre Fort Client. Toutes les modifications sont sauvegardées localement
- *phase de connexion*: Cette étape commence lorsque l'utilisateur revient en ligne. Les modifications effectuées du côté client et du côté Cloud doivent être synchronisés.

Deux étapes font l'objet de cette partie, l'*envoie des changements* et la *synchronisation*. Durant la première phase, le Coffre Fort

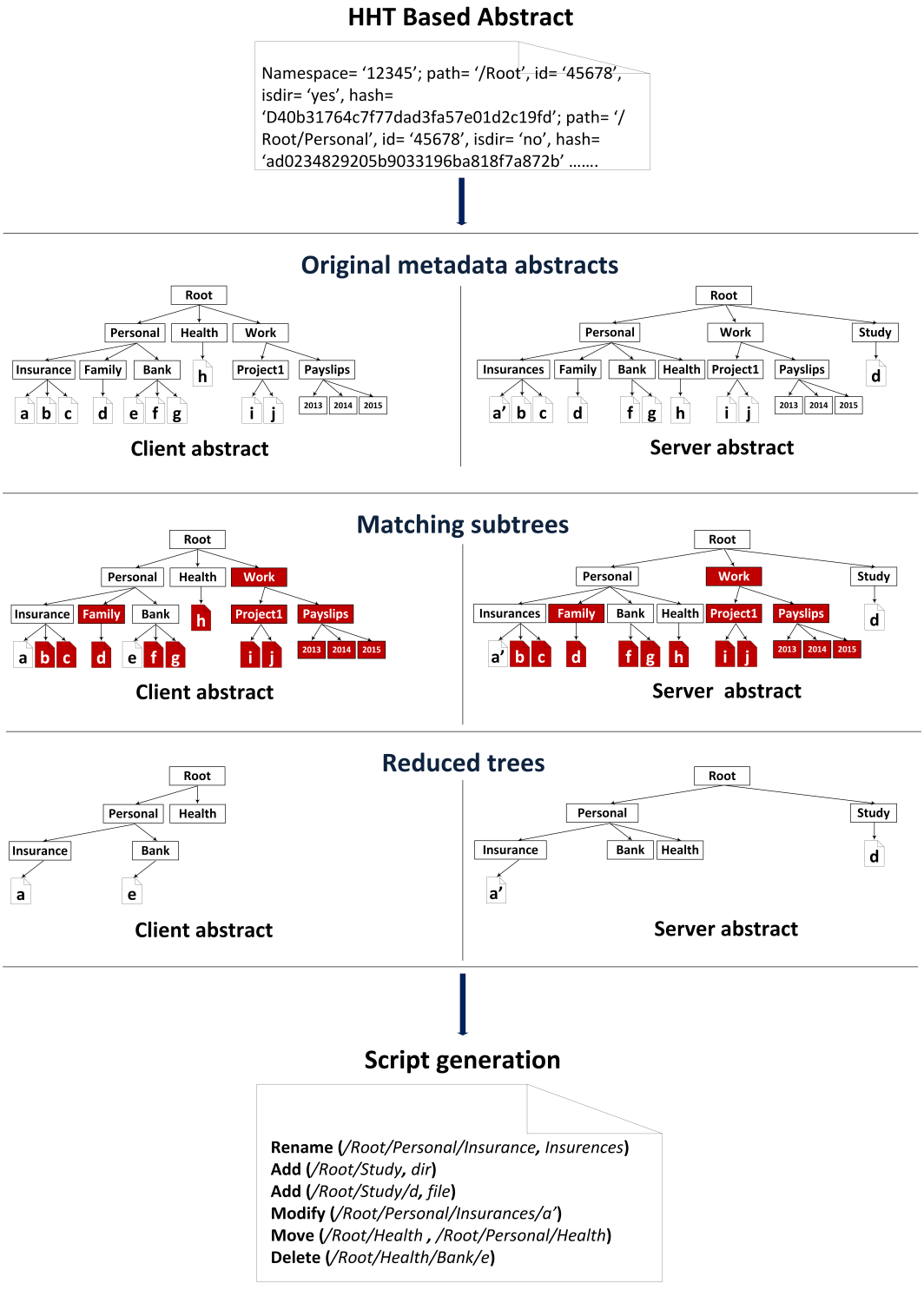



**Figure 11:** *Détection des changements dans SyncDS*

Client envoie les opérations qui ont été effectués hors ligne. Au cours de la phase de la *connexion*, et plus particulièrement, au niveau de l'étape *Synchronisation*, l'utilisateur envoie au serveur un extrait de son système de fichiers qui donne une image du

contenu de son système de fichiers. Cet extrait doit être comparé avec celui du serveur pour détecter les modifications effectuées sur le serveur lorsque l'utilisateur était hors ligne. Le HHT est introduit à ce niveau pour structurer l'extrait envoyé. En effet, le Hierarchique Hash Tree a une structure d'arbre qui suit la structure du système de fichiers. LEs répertoires et les fichiers sont les noeuds des arbres. Outre les métadonnées basiques, chaque nœud est identifié par un hash. Pour les fichiers, ce hash correspond à la valeur de hachage du contenu du fichier. Pour les répertoires, le hash est basé sur les hachage du contenu du répertoire avec ses fichiers et ses répertoires.

- *phase en ligne*: A ce niveau, on retrouve une synchronisation bidirectionnelle entre le Coffre Fort Client et le Coffre Fort Cloud. En effet, toute modification qui s'effectue d'un côté est directement envoyée vers l'autre côté.

Comme représenté dans la figure 11, l'algorithme entier est composé de deux algorithmes: *Matching subtrees* et *Script generation*. La sortie du premier algorithme est l'entrée du second. A partir des deux extraits des deux versions du système de fichiers, notre algorithme génère deux arbres tout en suivant la structure de HHT. L'algorithme *Matching subtrees* récupère les deux arbres et détecte les nœuds identiques. Ces nœuds sont alors éliminés de l'arbre pour retrouver des arbres réduits *Matching subtrees*. Les deux arbres réduits sont l'entrée de l'algorithme *Script generation* pour générer le script qui contient différentes opérations. Dans le cas ou ces opérations sont appliquées sur le premier arbre réduit, nous retrouvons le deuxième arbre. Le script en question est envoyé par le Coffre Fort Client au cours de la phase *Synchronize* de la deuxième étape de notre protocole de synchronisation.

## 2.3 Étude des performances du protocole SyncDS

Le protocole proposé SyncDS est un protocole de synchronisation de fichiers qui garantie une meilleure qualité et une consommation minimale des ressources dans un environnement sécurisé. Il présente deux nouveautés principales. La première est le caractère non propriétaire de l'architecture de synchronisation qui a une grande importance pour intégrer une large gamme de produits et d'appareils concurrents. En fait, nous améliorons les API de stockage local HTML5 pour stocker localement les données utilisateur dans un Coffre Fort Client. Nous utilisons également l'API HTML5 Websocket pour le transfert de données entre le Cloud et le Coffre Fort Cloud et le Coffre Fort Client.

La deuxième nouveauté de notre architecture est l'introduction du HHT pour détecter les changements entre deux versions du même système de fichiers. Le HHT a été adopté dans les travaux précédents pour détecter les changements entre deux versions du même fichier. A notre connaissance, notre travail est le premier qui adopte le HHT dans le contexte de la synchronisation sécurisée des fichiers pour détecter les changements entre les systèmes de

fichiers entiers. En effet, cet arbre représente une empreinte des objets stockés localement ainsi que leur organisation dans le système de stockage local Nous avons alors proposé des algorithmes de parcours et de comparaison basés sur d'arbres de hachage hiérarchiques ce qui permet de localiser efficacement les objets à synchroniser.

Pour prouver le concept et valider l'efficacité de notre architecture et protocole, nous nous concentrons principalement sur trois parties de l'architecture de synchronisation. Tout d'abord, l'implémentation couvre la couche de stockage côté client par l'amélioration des APIs de HTML5 et en particulier l'API FileSystem. En second lieu, elle comprend l'application web avec le développement de l'API de synchronisation. Enfin, elle détient la couche de synchronisation du côté du serveur de synchronisation basée sur WebSocket et l'intégration de l'algorithme de détection des changements du système de fichier basé sur le Hierarchical Hash Tree (HHT).

Nos résultats empiriques et preuves analytiques montrent que l'utilisation de notre framework, respecte les propriétés d'un protocole de synchronisation efficace. Ces propriétés sont : Le faible calcul des données et de métadonnées du côté client (figure 12), l'efficacité de la détection des changements entre les systèmes de fichiers du client et du serveur (figures 13 et 14 ), la réduction du nombre de fichiers synchronisés et enfin la réduction des messages échangés entre le client et le serveur (figure 15). Les figures représentent des comparaisons entre les performances de notre architecture et celles des autres protocoles adoptés par les solutions existantes.

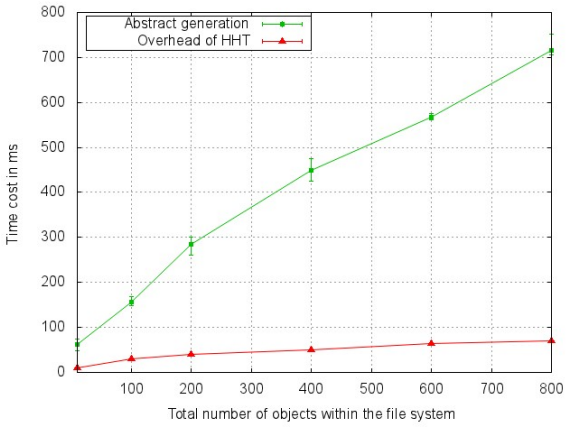



**Figure 12:** *Délai supplémentaire introduit par HHT au niveau du Coffre Fort Client*

Notre architecture de synchronisation est proposées dans le contexte Coffre Fort Numérique. Ce contexte se concentre principalement sur l'introduction de la valeur probante et la préservation de l'intégrité d'un objet numérique au fil du temps.

L'introduction de l'Hierarchical Hash Tree (HHT) dans la synchronisation participe directement à la sécurisation des données en plus de l'amélioration des performances. En effet, la vérification de la valeur du hash du répertoire racine d'un système de fichiers détecte

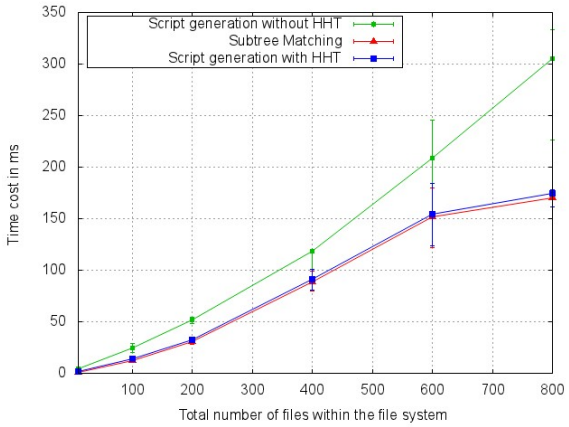



**Figure 13:** *Performances de HHT lors de la détection des changements*

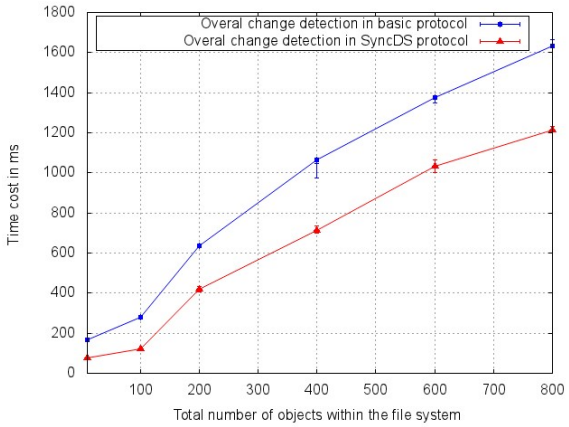



**Figure 14:** *Temps total de la détection des changements*

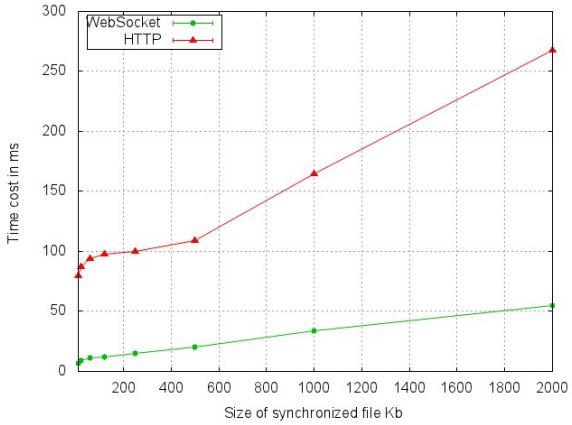



**Figure 15:** *Performances du protocoles WebSocket dans l'architecture de synchronisation*

systématiquement si un objet du système de fichiers a été altéré par un tiers ou si le système de fichiers conserve sa version d'origine. Dans ce cas, il n'est plus nécessaire de vérifier le hash des objets un par un pour vérifier l'intégrité du système du fichiers.

# 3 SyncDS: Partage des données sécurisé

Dans cette dernière partie de la thèse, nous abordons les exigences de sécurité du protocole de synchronisation notamment l'intégrité des fichiers, la non-répudiation, l'authentification, la synchronisation sécurisée et le contrôle d'accès. Nous nous concentrons principalement sur le contrôle d'accès, dans le contexte de partage des données entre différents utilisateurs.

## 3.1 Timely-CPAB: Contrôle d'accès avec dimension temporelle

Pour le partage des données, nous distinguons deux types d'utilisateurs: le propriétaire et le consommateur des données. Le propriétaire, outre la création, le stockage et le partage de données, il est responsable de l'imposition de la politique d'accès. Le consommateur est l'utilisateur qui télécharge et déchiffre les fichiers partagés par le propriétaire.

Plusieurs enjeux majeurs sont soulevés par le partage des données dans le contexte du Coffre Fort Numérique. Tout d'abord, la synchronisation des données qui sont déjà chiffrées par les consommateurs vers des destinations multiples introduit des problématiques de gestion de clés et de contrôle d'accès. Il est essentiel aussi de mettre en évidence que ni le propriétaire qui a partagé les données ni le consommateur ne sont nécessairement connectés en même temps. Pour l'enjeux de la sécurité, il est indispensable d'assurer la confidentialité des données contre l'accès non autorisé des consommateurs ainsi que du serveur de synchronisation. De plus, nous ajoutons aussi la dimension temporelle pour un accès restreint dans le temps. Les données sont accessibles alors par les consommateurs uniquement pendant une période de temps définie par le propriétaire dans sa politique de sécurité.

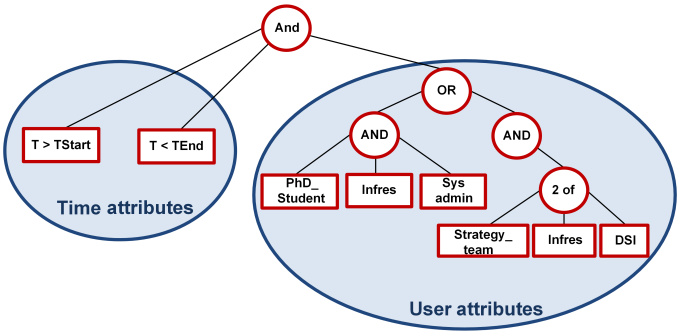



**Figure 16:** *Contrôle d'accès avec une dimension temporelle Timely-CP-ABE*

Pour soulever tous ces enjeux, nous proposons d'utiliser le chiffrement basé sur les attributs, Ciphertext Policy Attribute Based Encryption (CP-ABE) pour la gestion du contrôle d'accès. Nous ajoutons à CP-ABE, l'horodatage pour ajouter une dimension temporelle à l'accès ainsi que la division de la clé de chiffrement, en deux parties. Une partie est envoyée directement et seulement au consommateur en toute sécurité. La deuxième partie de la clé

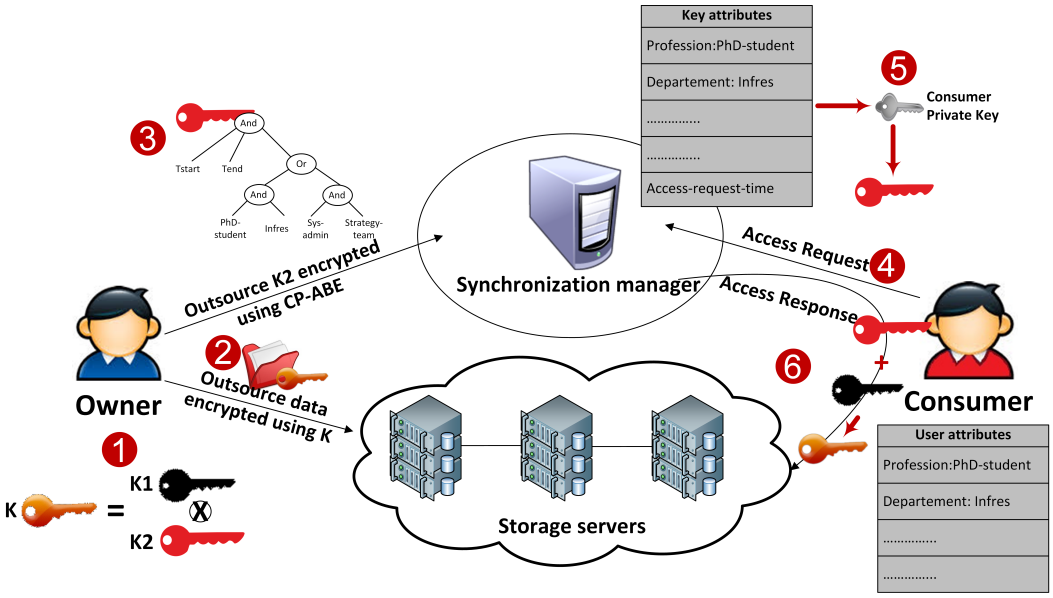



**Figure 17:** *Protocole de partage des fichiers avec Timely-CP-ABE*

est chiffrée en utilisant CP-ABE et envoyée au serveur de synchronisation pour vérifier la légitimité du consommateur.

La figure 17 illustre, notre protocole de partage de fichiers en considérant le Timely-CP-ABE. Nous considérons que le Gestionnaire de synchronisation est un serveur semi-fiable. Cela signifie, d'abord, que le serveur est honnête et suit les exigences de l'architecture et du protocole. Deuxièmement, le serveur est curieux, donc il essaye de recueillir autant d'informations et de données que possible. Comme les données sont sauvegardées chiffrées du côté du serveur, la clé du chiffrement du fichier ne doit pas être récupérée par le serveur. Sinon, le serveur peut avoir un accès complet aux données des utilisateurs.

Pour cette raison, nous divisons la clé de chiffrement en deux clés $K_1$ et $K_2$.

- La première clé $K_1$ est envoyée chiffrée au consommateur en utilisant sa clé publique. Ainsi, le consommateur est l'entité unique qui peut la récupérer. Cette clé n'est envoyée qu'une seule fois.
- La seconde clé $K_2$ est envoyée cryptée à l'aide du Timely-CP-ABE. Elle est ensuite déchiffrée par le serveur pour vérifier le droit d'accès du consommateur.

Avec cette division des clés de chiffrement et pour accéder aux données, le consommateur doit obtenir:

- **du propriétaire** la clé $K_1$ lors de la première demande de synchronisation;
- **du Gestionnaire de Synchronisation** la clé $K_2$ et l'autorisation de télécharger le fichier chiffré tant que ses attributs obéissent à la politique de contrôle d'accès du propriétaire et que l'accès est demandé pendant la période légitime.

Avec $K_1$ et $K_2$, le consommateur calcule la clé du déchiffrement K et peut déchiffrer le fichier.

## 3.2 Analyse de la sécurité du protocole

Les services de sécurité sont répartis entre les composants d'architecture en fonction de leurs rôles dans la synchronisation comme illustré dans la figure 18. Les fonctionnalités détaillées sont décrites comme suit:

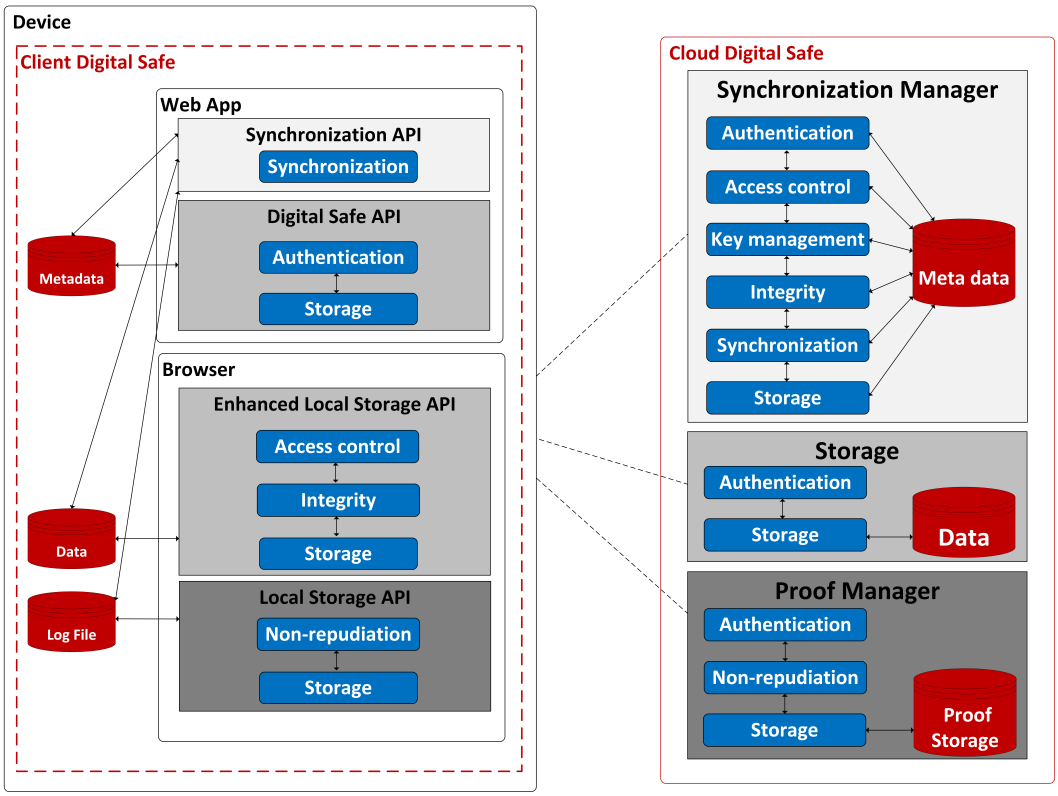



**Figure 18:** *Aspects de sécurité dans l'architecture de SyncDS*

- **Stockage sécurisé des données**:
  Pour gagner la confiance des utilisateurs, il est essentiel de donner une haute priorité à la sécurité des données à la fois stockées localement et stockées dans le Cloud. Comme l'utilisateur ne fait pas confiance aux serveurs distants, il est réconfortant pour lui d'envoyer ses données chiffrées. Le Cloud est donc considéré comme un serveur de stockage de blob chiffré. Ainsi, le fournisseur de stockage ne peut pas accéder à ces données.

- **Integrité des fichiers**:
  Dans notre architecture, les haches des fichiers et des répertoires sont introduits dans les métadonnées du fichier. En fait, ce mécanisme préserve l'intégrité du fichier. Avec l'introduction de l'arbre de hachage hiérarchique (HHT), vérifier la valeur du hash du répertoire racine du système de fichiers vérifie l'intégrité de tout le système de fichiers.

Il détecte systématiquement si un objet a été modifié par un tiers, ou le système de fichiers conserve sa version d'origine. Dans ce cas, il n'est plus nécessaire de vérifier le hash des fichiers un par un pour vérifier leur intégrité.

- **Non-repudiation**:
  Dans le Coffre Fort Cloud, un Gestionnaire de preuves est l'entité qui stocke les métadonnées signées par la clé privée de l'utilisateur. Ainsi, le stockage d'une preuve garantit la non-répudiation. Par conséquent, ni le client ni le fournisseur de services ne peuvent nier avoir participé au processus de stockage.

- **Authentifiation**:
  Elle consiste à vérifier l'identité de l'utilisateur par le Gestionnaire de synchronisation avant tout accès aux services de stockage et de synchronisation. Pour la sauvegarde des données et des épreuves, l'authentification de l'utilisateur par les serveurs de stockage et le gestionnaire des preuves sont également nécessaires.

- **Synchronisation sécurisée**:
  Sécuriser les messages de synchronisation entre les différentes entités d'architecture est crucial dans notre architecture. Le chiffrement des messages échangés à l'aide d'une clé de session mutuellement échangée évite la récupération d'informations confidentielles en cas d'interception de session. Dans la conception de notre protocole de synchronisation, les fichiers des utilisateurs sont envoyés chiffrés. Pour éviter les chiffrements multiples du même message envoyé, le chiffrement se produira au niveau de la couche application au lieu de la couche transport. Dans ce cas, seuls les messages à haut niveau de confidentialité sont chiffrés.

- **Contrôle d'accès**:
  Dans le cas du partage de données entre différents utilisateurs, seul le consommateur autorisé peut accéder aux données. Même le fournisseur de Cloud est concerné par le contrôle d'accès et ne doit pas avoir le droit d'accéder aux données. Le gestionnaire de synchronisation, en tant que composant frontal de l'architecture de synchronisation, est le premier responsable du contrôle d'accès. Dans notre proposition, le propriétaire des données peut partager des données avec les consommateurs sur une période de temps spécifique. Au-delà de cette période, l'accès est refusé. Ceci conduit à proposer un CP-ABE avec une dimension temporelle et ajouter l'horodatage dans la vérification du contrôle d'accès.

    **- Confidentialité contre le Gestionnaire de synchronisation**:
  Bien que ce serveur est censé nuire à la confidentialité des données, il doit être honnête de suivre les données et les instructions de gestion des clés et de traiter la demande

d'accès aux données. Même si le Gestionnaire de synchronisation gère les clés et vérifie la légitimité d'accès, il ne détient qu'une partie du secret. Ce secret est utilisé uniquement pour vérifier que le consommateur, au moment de la demande d'accès, obéit à la politique d'accès imposée par le propriétaire. Ainsi, il ne peut pas récupérer les données de l'utilisateur avec cette clé.

**- Confidentialité de l'accès des consommateurs non autorisés**:

Dans notre protocole proposé, le consommateur légitime qui a le privilège d'accès est l'entité unique qui peut récupérer et déchiffrer les données. Même si le Gestionnaire de synchronisation dispose de la clé $K_2$, la clé $K_1$ manque pour calculer la clé de chiffrement K. En outre, même si le consommateur a la clé $K_1$ et il n'a pas le privilège d'accéder aux données du propriétaire. Il ne peut pas récupérer la clé $K_2$ du Gestionnaire de synchronisation.

Un autre cas est envisageable. Le consommateur a récupéré précédemment les deux clés $K_1$ et $K_2$ et il n'a plus le privilège d'accéder aux données en raison de l'expiration du temps ou des changements de politique. Dans ce cas, le Gestionnaire de synchronisation, lors du calcul de la nouvelle clé $K_2$, ne lui donne pas l'autorisation de récupérer et de synchroniser la nouvelle version des données.

## 3.3 Étude des performance du Timely-CP-ABE

Pour l'évaluation des performances, nous comparons l'introduction du partage de fichiers avec la dimension temporelle lorsqu'il est basé sur notre Timely-CP-ABE et lorsqu'il utilise une simple liste de contrôle d'accès ACL. Nous considérons, dans notre analyse, plusieurs schémas de politiques d'accès aléatoire et des attributs d'utilisateurs qui peuvent répondre aux véritables cas de partage de fichiers dans le contexte du Coffre Fort numérique.

Dans la figure 19, les résultats montrent qu'il est plus efficace d'adopter le CP-ABE amélioré dans l'architecture de synchronisation. Plus précisément, le CP-ABE amélioré lorsqu'il est adopté pour la synchronisation de partage de fichiers réduit le coût de vérification au cinquième par rapport à l'ACL. Il est encore plus efficace car il introduit la gestion des clés en plus de la vérification d'accès; chose qui n'est pas effectuée par l'ACL.

# 4 Conclusion

Trois contributions principales font l'objet de notre thèse de recherche. Dans la première, nous proposons un Coffre Fort Client standardisé basé sur HTML5. Nous améliorons, par conséquent, la sécurité des APIs de stockage de HTML5. Ces APIs améliorées sont la base de notre Coffre Fort client. L'évaluation de la performance prouve que l'amélioration des APIs de HTML5 est crucial et efficace. Dans la deuxième contribution, nous proposons un protocole de synchronisation efficace et sécurisé entre les Coffres Forts. Nous avons de même

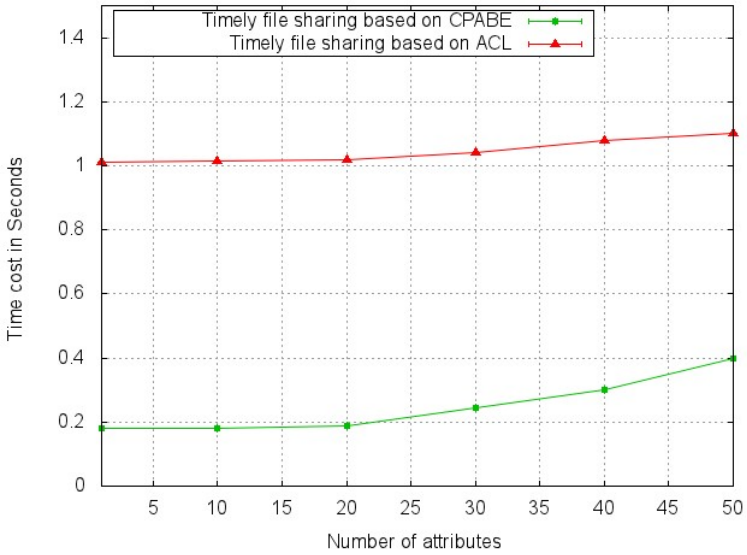



**Figure 19:** *Performances du Timely-CP-ABE*

démontrer l'efficacité de notre solution. La troisième contribution se concentre sur la sécurité du partage des fichiers. Nous proposons ainsi un mécanisme de contrôle d'accès avec une dimension temporelle qui obéit aux exigences du Coffre Fort Numérique. Notre mécanisme fait preuve d'efficacité et sécurité.

Comme de travaux future, il est intéressant d'adopter les APIS de HTML5 dans le domaine de l'internet des objets et de se focaliser sur le partage collaboratif des données dans le contexte du Coffre Fort numérique.

# Appendix A

# HTML5 API classification

## A.1 HTML5 APIs

| Communication and networking | |
|---|---|
| **WebSocket** | Allows the establishment of a bidirectional and asynchronous communication between the browser and the client to exchange informations |
| **HXMLHttpRequest** | - Ensures an asynchronous communication between the client and the server<br>-In its new version, enhancement were added for a better interoperability with binary data, blobs and files<br>- Adopts the Cross Origin Ressource Sharing CORS |
| **WebRTC** | introduces the possibility to the browser the possibility to communicate with an other browser based on P2P communication |
| **Web Messaging** | Ensures the communication between two documents with different origins within the same browser |
| **Notification** | Ensures the display of notifications at the user's computer |
| **Server Sent Events** | Opens an HTTP connection to receive Push notifications from the server in the form of DOM events. The API is designated to work with other pushing systems like SMS Push. |
| **Network information** | Retrieves informations about the network connection |
| **Device** | |
| **Geolocalisation** | Retrieves informations about the position of the device using GPS, WIFI or cellular network |
| **Battery status** | Explores informations about the battery status of a device |
| **Screen orientation** | Retrieves the orientation of the device's screen |
| **Vibration** | Accesses au vibration mechanism of the device. Its is a tactile feedback |

| | |
|---|---|
| **Motion Sensors** | Provides the developer information about the orientation, the movement and acceleration of the device. |
| **Web Audio** | Allows the generation and the process of the sound. It supports 3 types of audio formats: MP3, WAV and OGG. |
| **Media capture** | Captures the sound and the video with the hardware of the user( (microphone or webcam)). |
| **video** | Manages the play of the video on a web page. It supports 3 types of video formats: MP4, WebM and OGG. |
| **Data** | |
| **Indexed database** | Offers the possibility to store data in a structured database. |
| **Application Offline** | Stores different elements needed from the web application to work in the offline mode. |
| **File** | Manipulates files stored locally in the user machine. |
| **FileSystem** | Simulates a local file system used by the web application to manages files and repositories. |
| **WebStorage** | Stores data in the user's device as a key/value pair. |
| **WebSQL** | Stores the data in SQL databases. This API has been abandoned in favor of the indexed database API. |
| **User Interface** | |
| **SVG** | Scalable Vector Graphic displays vectorial graphic objects based on XML. |
| **WebGL** | Brings 3D graphics to a web page. It is closely conform with the OpenGL ES 2.0 API. |
| **Drag and Drog** | Allows the user to click on an item and move it to another location while holding down the mouse button. |
| **Fullscreen** | Puts in full screen several web application elements such as videos and images. |
| **Performances optimization** | |
| **Web Workers** | Allows developers to use the multi-threading and to run background scripts that run at the same time with the main web page. The script communicates with the main web page via the web Messaging API. |
| **Session history and navigation** | Allows you to change URL without reloading the entire web page. |
| **Page visibility** | detects if the page appears to the user. This avoid for example, periodic actions such as updating messages while the user does not see the page. This allows better CPU consumption and energy. |

| | |
|---|---|
| **Animation timing** | detects the best animation rate based on the visibility state of the page for better CPU utilization. |
| **Navigation timing** | Measures the web page loading speed. |
| **Security** | |
| **Web cryptographies** | Ensures security features such as the hash, the generation and the verification of signature, the encryption and the decryption. |

## A.2 HTML5 APIs classifications

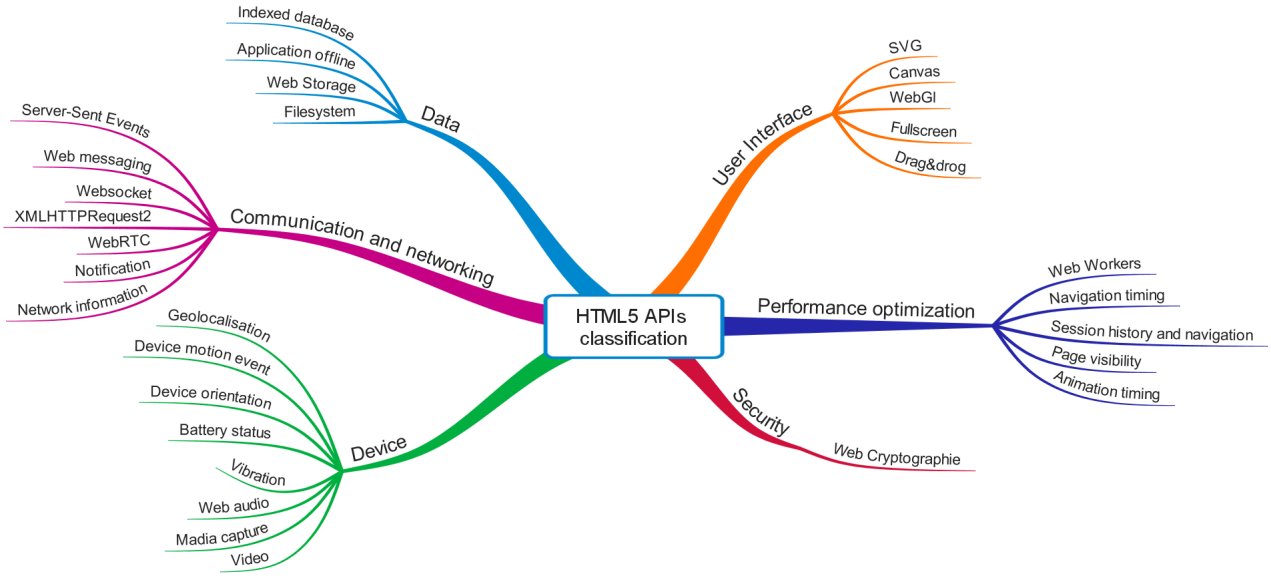



**Figure A.1:** *Classification according to the nature of managed elements 1*

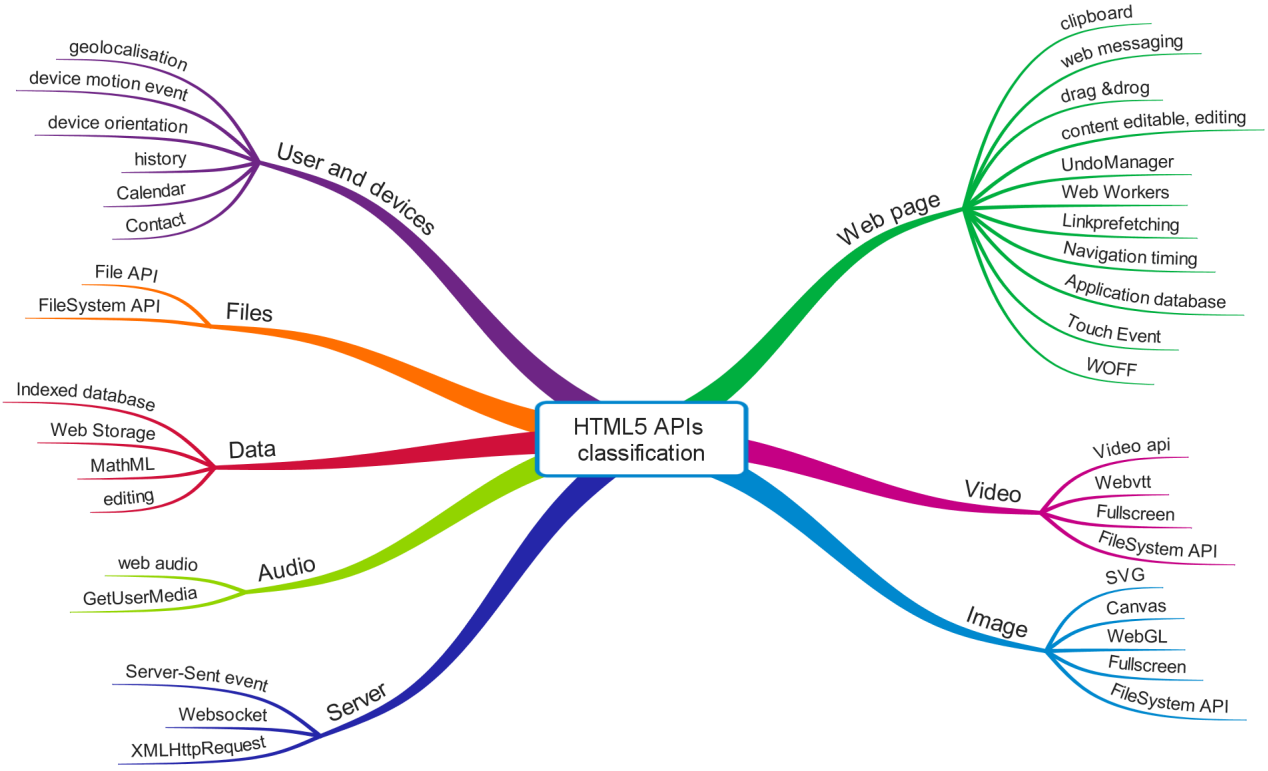



**Figure A.2:** *Classification according to the nature of managed elements 2*

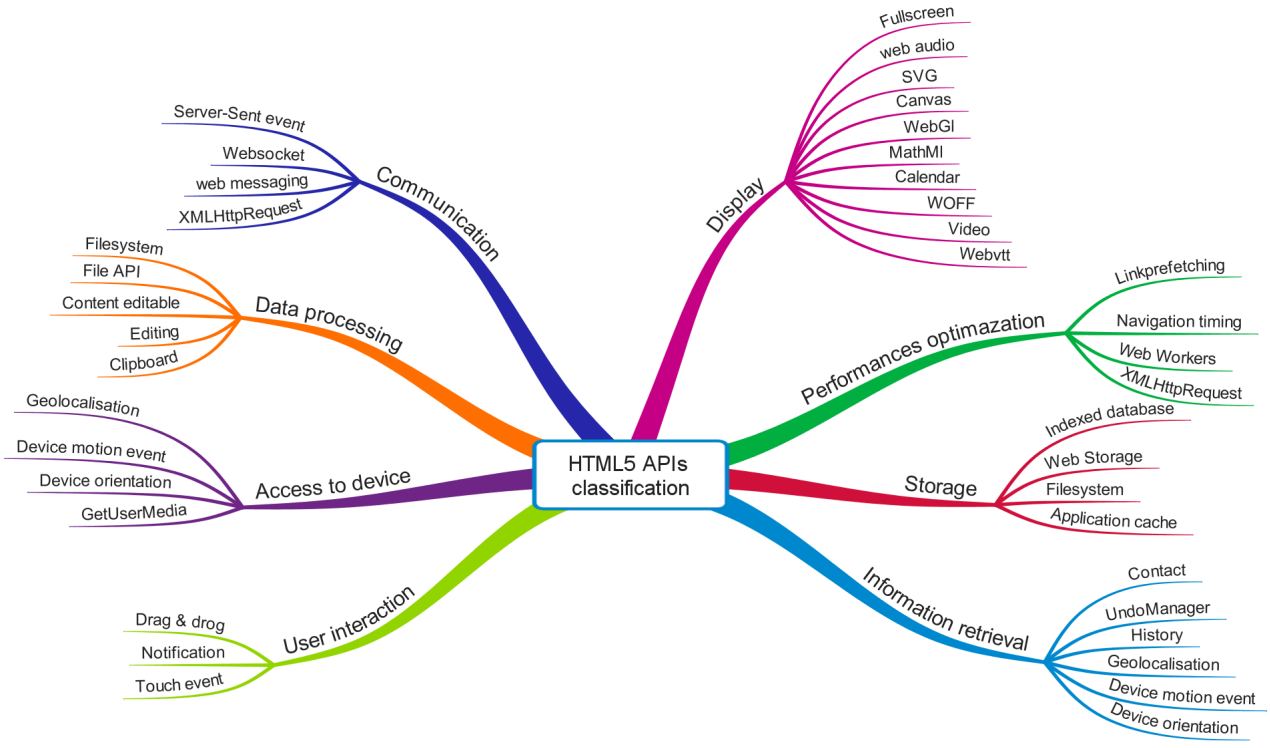



**Figure A.3:** *Classification according to the actions on managed elements*

# Appendix B

# HTML5 Local Storage APIs

## B.1 WebStorade API

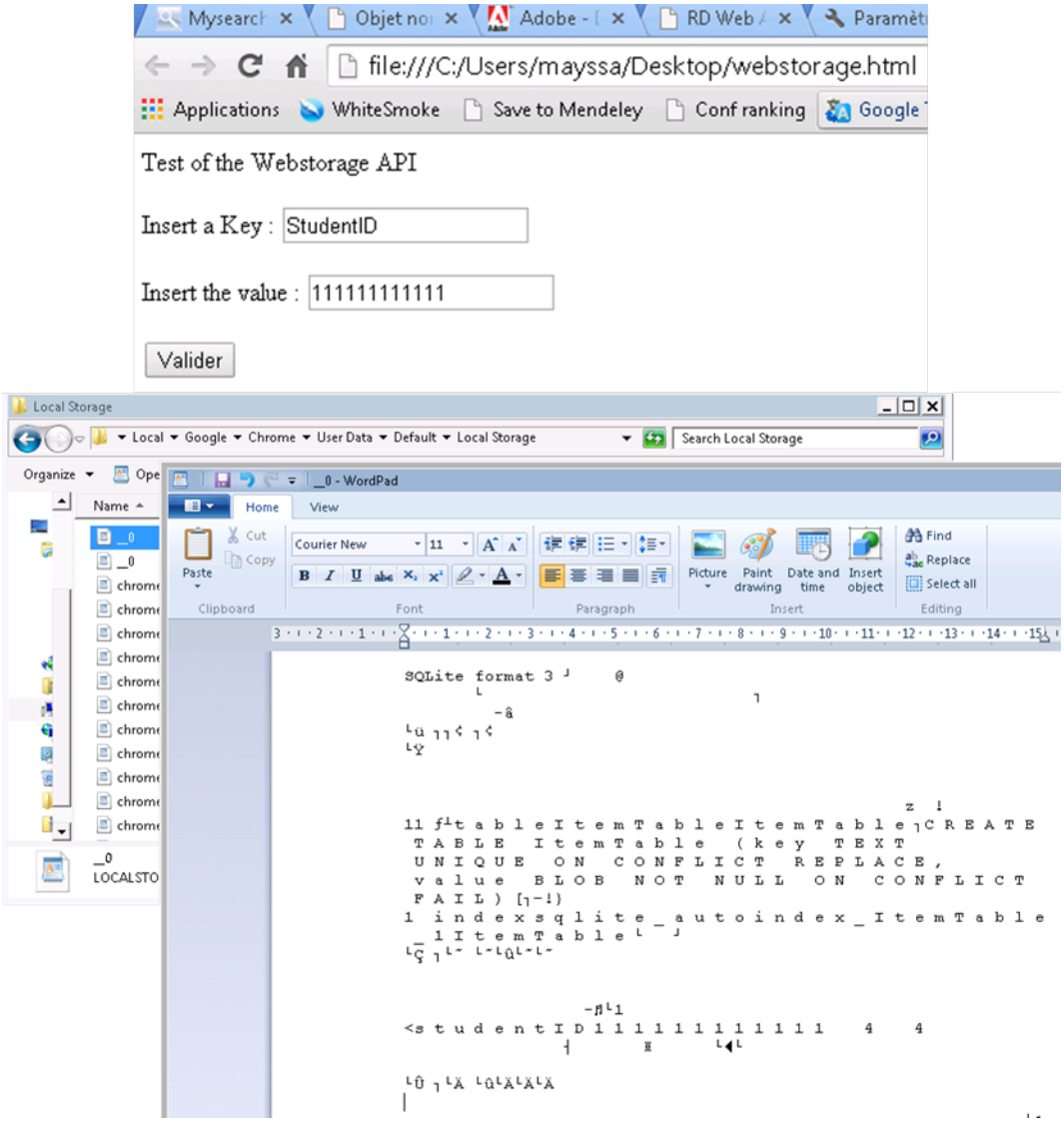



**Figure B.1:** *WebStorage API example*

```
<script>
        var key;
        var value;
        document.getElementById("validate").onclick = function () {
            key = document.getElementById("key").value;
            value = document.getElementById("value").value;
            window.localStorage.setItem(key, value);
            alert("Data are stored");
        }
</script>
```

**Figure B.2:** *Javascript of the WebStorage API*

## B.2 IndexedDB API

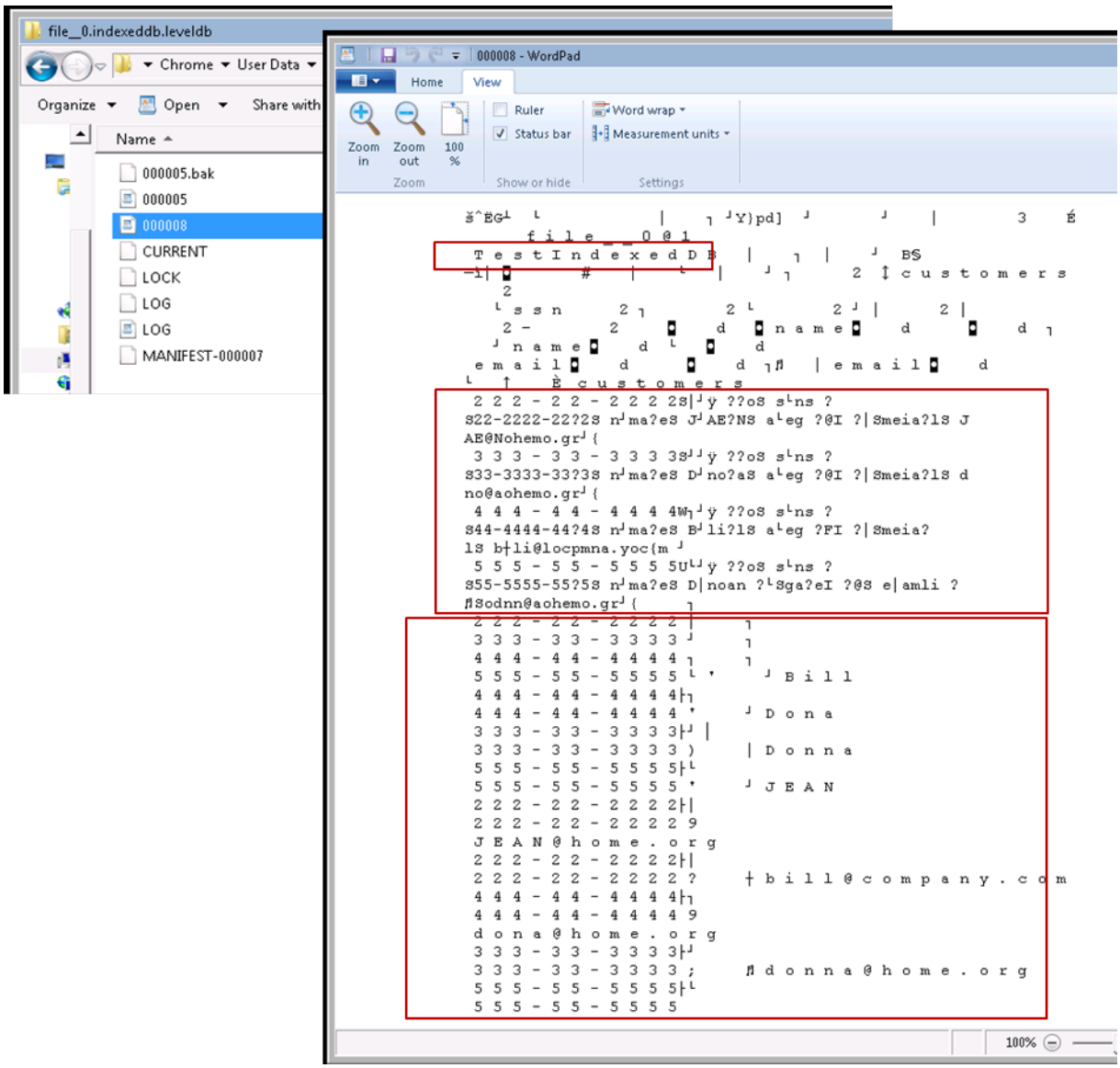



**Figure B.3:** *IndexedDB API files*

```
const customerData = [
    { ssn: "444-44-4444", name: "Bill", age: 35, email: "bill@company.com" },
    { ssn: "555-55-5555", name: "Donna", age: 32, email: "donna@home.org" },
    { ssn: "333-33-3333", name: "Dona", age: 32, email: "dona@home.org" },
    { ssn: "222-22-2222", name: "JEAN", age: 32, email: "JEAN@home.org" }
];
const dbName = "TestIndexedDB";

var request = indexedDB.open(dbName, 2);
var db;
request.onerror = function(event) {
  // Handle errors.
};
request.onupgradeneeded = function(event) {
    var db = event.target.result;
    var objectStore = db.createObjectStore("customers", { keyPath: "ssn" });
    objectStore.createIndex("name", "name", { unique: false });
    objectStore.createIndex("email", "email", { unique: true });
    for (var i in customerData) {
        objectStore.add(customerData[i]);
    }
};
```

**Figure B.4:** *Javascript example of the IndexedDB API*

## B.3 FileSystem API

```
navigator.webkitPersistentStorage.requestQuota(1024* 1024 * 1024, function (grantedBytes) {
window.requestFileSystem(PERSISTENT, grantedBytes, onInitFs, errorHandler);
}, function (e) {
    console.log('Error', e);
});

function onInitFs(fs) {
    fs.root.getFile('test.txt', { create: true }, function (fileEntry) {
        // Create a FileWriter object for our FileEntry (test.txt).
        fileEntry.createWriter(function (fileWriter) {

            fileWriter.onwriteend = function (e) {
                console.log('Write completed.');
            };
            fileWriter.onerror = function (e) {
                console.log('Write failed: ' + e.toString());
            };
            // Create a new Blob and write it to test.txt.
            var blob = new Blob(['This text is inserted into the file test.txt'], { type: 'text/plain' });

            fileWriter.write(blob);

        }, errorHandler);

    }, errorHandler);
```

**Figure B.5:** *Javascript example of the FileSystem API*

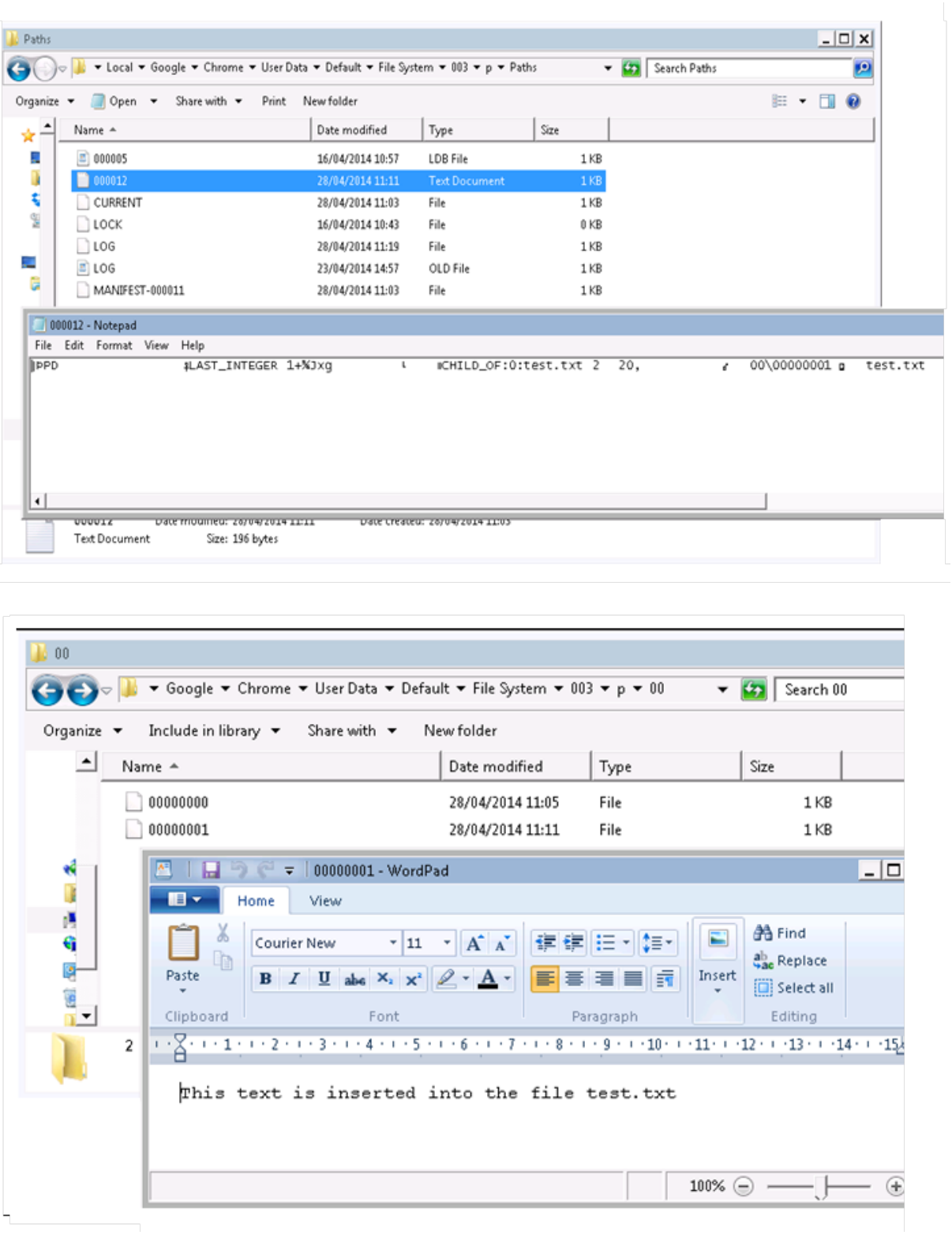



**Figure B.6:** *FileSystem API example*

# Bibliography


[1] 1password: Put passwords in their place. https://agilebits.com/.

[2] Afnor groups: Delivering forceful, impartial services.

[3] Amazon simple storage service:using client-side encryption. http://docs.aws.amazon.com/AmazonS3/latest/dev/UsingClientSideEncryption.html.

[4] The chromium projects: For developers sync diagnostics.

[5] Cloudfogger - free file encryption for dropbox and the cloud. http://www.cloudfogger.com/en/.

[6] Devicejs. http://devicejs.org/.

[7] Dojo documentation: dojox.storage. http://dojotoolkit.org/reference-guide/1.9/dojox/storage.html#dojox-storage.

[8] Google drive: A safe place for all your files. https://www.google.com/intl/en-en/drive/.

[9] Google gears api. https://developers.google.com/gears/?hl=fr&csw=1.

[10] Gsafe: Government safe.

[11] Html5 security cheat sheet. https://www.owasp.org/index.php/HTML5_Security_Cheat_Sheet.

[12] Iframe element. w3c working draft. http://www.w3.org/TR/2011/WD-html5-20110525/the-iframe-element.html.

[13] The last password you'll have to remember. https://lastpass.com/.

[14] Microsoft developer network: Userdata behavior. http://msdn.microsoft.com/en-us/library/ms531424(v=vs.85).aspx.

[15] noduino. http://semu.github.io/noduino/.

[16] Safenet kmip and amazon s3 integration guide. http://www2.safenet-inc.com/AWS-guides.

[17] Secure storage in the cloud. https://www.wuala.com/fr/.

[18] Security in google driver. https://support.google.com/drive/answer/141702?hl=en.

[19] The thing system. http://thethingsystem.com/.

[20] Using the html5 fullscreen api for phishing attacks. http://feross.org/html5-fullscreen-api-attack/.

[21] Webrtc 1.0: Real-time communication between browsers w3c editor's draft 10 april 2014.

[22] World wide web consortium. http://www.w3.org/.

[23] Your stuff is safe with dropbox. https://www.dropbox.com/security.

[24] Indexed database api. w3c candidate recommendation. 04 July 2013.

[25] Top 10 Mobile Technologies and Capabilities for 2015 and 2016, 12 February 2014. [Online. Last accessed: 2014-06-01].

[26] Iso7498-2 : 1989. information processing systems-open systems interconnection. 1989.

[27] The dublin core metadata element set. https://www.ietf.org/rfc/rfc5013.txt, August 2007.

[28] Whatwg specification of html5 client-side session and persistent storage of name/value pairs. https://whatwg.org/specs/web-apps/2007-10-26/multipage/, 2007.

[29] Advanced crypto software collection: Ciphertext-policy attribute-based encryption. http://acsc.cs.utexas.edu/cpabe/, 2011.

[30] Nf z42-020: Functional specifications of a digital safe component to preserve digital information in conditions that ensure their integrity over time, July 2012.

[31] The websocket api, w3c candidate recommendation, 2012.

[32] File api. w3c last call working draft, 2013.

[33] How to build efss plans to address current and future business requirements. https://www.gartner.com/doc/2948535/build-efss-plans-address-current, 2014.

[34] Trusted computing group. tpm main specification. 2014.

[35] Magic quadrant for enterprise file synchronization and sharing. https://www.gartner.com/doc/3098819/magic-quadrant-enterprise-file-synchronization, 2015.

[36] Reddit. syncthing: Open source bittorrent sync alternative (p2p sync tool). http://www.webupd8.org/2014/06/syncthing-open-source-bittorrentsync.html, April 2015.

[37] Security considerations for webrtc draft-ietf-rtcweb-security-08. http://tools.ietf.org/html/draft-ietf-rtcweb-security-08, Aug 2015.

[38] Web real-time communication use cases and requirements. https://tools.ietf.org/html/rfc7478, March 2015.

[39] Web storage editor's draft. http://dev.w3.org/html5/webstorage/, 3 August 2013.

[40] Offline web applications, w3c working group note. http://www.w3.org/TR/offline-webapps/, 30 May 2008.

[41] Web intents, w3c working group note. http://www.w3.org/TR/web-intents/, May, 2013.

[42] Shamir Adi. Identity-based cyrptosystems and signature schemes. In *Advances in Cryptology-CRYPTO, Springer-Verlag Press*, 1984.

[43] S. Agarwal, V. Chauhan, and A. Trachtenberg. Bandwidth efficient string reconciliation using puzzles. In *IEEE Transactions on Parallel and Distributed Systems*, 2006.

[44] S. Agarwal, D. Starobinski, and A. Trachtenberg. On the scalability of data synchronization protocols for pdas and mobile devices. *IEEE Network*, Jul 2002.

[45] Raihan Al-Ekram, Archana Adma, and Olga Baysal. diffx: An algorithm to detect changes in multi-version xml documents. In *Proceedings of the 2005 Conference of the Centre for Advanced Studies on Collaborative Research*, CASCON '05, 2005.

[46] Bertrand Alain. Web hypertext application technology working group- html living standard. http://www.whatwg.org/specs/web-apps/current-work/multipage/.

[47] J C. Anderson, J. Lehnardt, and N. Slater. Couchdb the definitive guide.

[48] Quentin Mayo Lauren Thomas Ashkan Soltani, Shannon Canty and Chris Jay Hoofnagle. Flash Cookies and Privacy. *AAAI Spring Symposium on Intelligent Information Privacy Management*, 2010.

[49] G. Ateniese, K. Fu, M. Green, and S. Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. In *Proc. Network and Distributed Systems Security Symp. (NDSS)*, 2005.

[50] Steven M. Bellovin. Problem areas for the ip security protocols. In *In proceeding of the Sixth Usenix Unix Security Symposium*, July 1996.

[51] Juan Benet. Ipfs - content addressed, versioned, P2P file system. *CoRR*, abs/1407.3561, 2014.

[52] Benjamin, C.Pierce, and Jerome Vouillon. What's in unison? a formal specification and reference implementation of a file synchronizer. In *Tech. rep. MS-CIS-03-36, Department of Computer and Information Science, University of Pennsylvania*, 2004.

[53] K. Bhargavan and A.Delignat-Lavaud. Web-based attacks on host-proof encrypted storage . *Proc of USENIX WOOT*, 2012.

[54] D. Borthakur. The hadoop distributed file system: Architecture and design, 2007.

[55] Sonja Buchegger, Doris Schiöberg, Le hung Vu, and Anwitaman Datta. Peerson: P2p social networking – early experiences and insights. In *In Proc. ACM Workshop on Social Network Systems*, 2009.

[56] C. Schmidt C. Vogt, M.Jonas Werner. Content-centric user networks: Webrtc as a path to name-based publishing. pages 1–3. 2013 21st IEEE International Conference on Network Protocols (ICNP), October.

[57] Yinzhi Cao, Vaibhav Rastogi, Zhichun Li, Yan Chen, and Alexander Moshchuk. Redefining web browser principals with a configurable origin policy. 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2013.

[58] Melissa Chase and Sherman S. M. Chow. Improving privacy and security in multi-authority attribute-based encryption. In *In Proceedings of the 16th ACM Conference on Computer and Communications Security*, pages 121–130, 2009.

[59] Sudarshan S. Chawathe and Hector Garcia-Molina. Meaningful change detection in structured data. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, SIGMOD '97, 1997.

[60] Sudarshan S. Chawathe, Anand Rajaraman, Hector Garcia-Molina, and Jennifer Widom. Change detection in hierarchically structured information. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, SIGMOD '96, 1996.

[61] Sanghong An; Sangmin Park; Hyeontaek Oh; Jinhong Yang; Hyojin Park; Junkyun Choi. Lightweight web-based communication interface design for web of objects. January.

[62] Richard S. Cox, Steven D. Gribble, Henry M. Levy, and Jacob Gorm Hansen. A safety-oriented platform for web applications. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, pages 350–364, 2006.

[63] Idilio Drago, Marco Mellia, Maurizio M. Munafo, Anna Sperotto, Ramin Sadre, and Aiko Pras. Inside dropbox: Understanding personal cloud storage services. In *Proceedings of the 2012 ACM Conference on Internet Measurement Conference*, IMC '12, 2012.

[64] Bonomi Flavio, Milito Rodolfo, Zhu Jiang, and Addepalli Sateesh. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, 2012.

[65] E. Goh, H. Shacham, N. Modadugu, and D. Boneh. Sirius: Securing remote untrusted storage. In *Proc. Network and Distributed Systems Security Symp. (NDSS)*, 2003.

[66] Matthew Green, Susan Hohenberger, and Brent Waters. Outsourcing the decryption of abe ciphertexts. In *Proceedings of the 20th USENIX Conference on Security*, 2011.

[67] Robert Haas. KMIP – Key Management Interoperability Protocol. *IBM Zurich Research Laboratory, Tech. Comm.*, 2009.

[68] Navendu Jain, Mike Dahlin, and Renu Tewari. Taper: Tiered approach for eliminating redundancy in replica synchronization. In *In Proc. of the USENIX Conference on File And Storage Systems*, 2005.

[69] Mark Scanlon Jason Farina and M. Tahar Kechadi. Bittorrent sync: First impressions and digital forensic implications. *Digital Investigation*, 2014.

[70] K. Jayaraman, Wenliang Du, B. Rajagopalan, and S.J. Chapin. Escudo: A fine-grained protection model for web browsers. In *Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on*, June 2010.

[71] J.Bethencourt and and B.Waters A. Sahai. Ciphertext-policy attribute-based encryption. In *IEEE Symposium on Security and Privacy, IEEE Computer Society, Los Alam*, 2007.

[72] Steven C. Chapra Jeffrey D. Walker. A client-side web application for interactive environmental simulation modeling . *Environmental Modelling and Software*, 2014.

[73] Mayssa jemel and Ahmed serhrouchni. Content protection and secure synchronization of html5 local storage data. In *IEEE 11th Consumer Communications and Networking Conference (CCNC)*, Jan 2014.

[74] Bo Zhu Jin Li, Kui Ren and Zhiguo Wan. Privacy-aware attribute-based encryption with user accountability. In *Proceedings of Information Security ISC*, Jan 2009.

[75] S. Josefsson. PKCS 5: Password-Based Key Derivation Function 2 (PBKDF2) Test Vectors. RFC 6070 (Informational. Jan. 2011.

[76] M. Kallahalla, E. Riedel, Q. Wang R. Swaminathan, and K. Fu. Plutus: Scalable secure file sharing on untrusted storage. In *Proc. USENIX Conf. File and Storage Technologies*, 2003.

[77] Chris Karlof, J. D. Tygar, David Wagner, and Umesh Shankar. Dynamic pharming attacks and locked same-origin policies for web browsers. In *the Fourteenth ACM Conference on Computer and Communications Security (CCS 2007*, 2007.

[78] Stefan Kimak and Jeremy Ellman. Performance Testing and Comparison of Client Side Databases Versus Server Side. *14th Annu. Postgrad. Symp. Converg. Telecommun. Netw. Broadcast. Liverpool, UK*, 2013.

[79] Stefan Kimak, Jeremy Ellman, and Christopher Laing. An Investigation into Possible Attacks on HTML5 IndexedDB and their Prevention. *13th Annu. Postgrad. Symp. Converg. Telecommun. Netw. Broadcast. Liverpool, UK*, 2012.

[80] A. Barthy E. Rescorlaz L. Huang, E. Y. Chen and C. Jackson. Talking to yourself for fun and profi. In *Web 2.0 Security and Privacy, Oakland, CA, USA*, 2011.

[81] Junzuo Lai, Robert H. Deng, and Yingjiu Li. Fully secure cipertext-policy hiding cp-abe. In *Proceedings of the 7th International Conference on Information Security Practice and Experience*, 2011.

[82] Junzuo Lai, Robert H. Deng, and Yingjiu Li. Expressive cp-abe with partially hidden access structures. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '12, 2012.

[83] Allison Lewko and Brent Waters. Decentralizing attribute-based encryption. In *Proceedings of the 30th Annual International Conference on Theory and Applications of Cryptographic Techniques: Advances in Cryptology*, EUROCRYPT'11, 2011.

[84] Chao Liang, Luokai Hu, Zhou Lei, and Jushu Wang. Synccs: A cloud storage based file synchronization approach. Jul 2014.

[85] Zhenghong Liu, Yuhong Jia, and Tieli Sun. Study on role-based access control with time constraint. In *Seventh International Conference on Computational Intelligence and Security (CIS), 2011*, Dec 2011.

[86] Tongbo Luo and Wenliang Du. Contego: Capability-based access control for web browsers. In *In Proceedings of the 4th International Conference on Trust and Trustworthy Computing*, 2011.

[87] F. Zavoral M. Krulis, Z.Falt. Catalysing the success of webrtc for the provision of advanced multimedia real-time communication services. In *17th International Conference on Intelligence in Next Generation Networks (ICIN)*, 2013.

[88] F. Zavoral M. Krulis, Z.Falt. Catalysing the success of webrtc for the provision of advanced multimedia real-time communication services. 2013.

[89] M. Mandviwalla, S. D. Clark, and K. Sandoe. Collaborative writing as a process of formalizing group memory. In *System Sciences, 1995. Proceedings of the Twenty-Eighth Hawaii International Conference on*, volume 4, pages 342–350 vol.4, Jan 1995.

[90] Santiago Melia, Jaime Gomez, Sandy Perez, and Oscar Diaz. A model-driven development for gwt-based rich internet applications with ooh4ria. In *Proceedings of the 2008 Eighth International Conference on Web Engineering*, ICWE '08, 2008.

[91] P. Mell and T. Grance. The nist definition of cloud computing. *Technical Report 6, National Institute of Standards and Technology*, 2011.

[92] Y. Minsky, A. Trachtenberg, and R. Zippel. Set reconciliation with nearly optimal communication complexity. *Information Theory, IEEE Transactions on*, Sept 2003.

[93] Mounira Msahli and Ahmed Serhrouchni. Sbaas: Safe box as a service. In *9th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing*, Nov 2013.

[94] Martin Mulazzani, Sebastian Schrittwieser, Edgar Weippl, Manuel Leithner, and Markus Huber. Dark Clouds on the Horizon : Using Cloud Storage as Attack Vector and Online Slack Space. *Proc. USENIX Secur. Conf. Secur. , CA, USA*.

[95] M. Dahlin N. Jain and R. Tewari. Taper: Tiered approach for eliminating redundancy in replica synchronization. In . *In Proc. of the USENIX Conference on File And Storage Systems*, 2005.

[96] BRAD NEUBERG. Coding In Paradise Now in a Browser Near You Offline Access and Permanent, Client-Side Storage, Thanks to Dojo.

[97] Terri Oda, Glenn Wurster, P. C. van Oorschot, and Anil Somayaji. Soma: Mutual approval for included content in web pages. In *Proceedings of the 15th ACM Conference on Computer and Communications Security*, CCS '08, 2008.

[98] Meulen R Pettey C. Gartner's 2012 hype cacle for emerging technologies identifies "tipping point" technologies that will unlock long-awaited technology. 2012.

[99] Xiulei Qin, Wenbo Zhang, Wei Wang, Jun Wei, Hua Zhong, and Tao Huang. A comparative evaluation of cache strategies for elastic caching platforms. In *11th International Conference on Quality Software*, July 2011.

[100] Alex Russell. Comet: Low latency data for the browser. 2006.

[101] Howard Gobioff Sanjay Ghemawat and Shun-Tak Leung. The google file system, 2003.

[102] P. Saxena, D. Akhawe, S. Hanna, Feng Mao, S. McCamant, and D. Song. A symbolic execution framework for javascript. In *Security and Privacy (SP), 2010 IEEE Symposium on*, May 2010.

[103] R. Sharma and A Datta. Supernova: Super-peers based architecture for decentralized online social networks. In *Fourth International Conference on Communication Systems and Networks*, Jan 2012.

[104] Ravi S.Sandhu. Role-based access control. In *Advances in computers*, 1998.

[105] Torsten Suel, P. Noel, and D. Trendafilov. Improved file synchronization techniques for maintaining large replicated collections over slow networks. In *20th International Conference on Data Engineering, 2004. Proceedings.*, March 2004.

[106] Kazuki Yoneyama Takashi Nishide and Kazuo Ohta. Attribute-based encryption with partially hidden encryptor-specified access structures. In *ACNS, volume 5037 of Lecture Notes in Computer Science*, 2008.

[107] A. Trachtenberg, D. Starobinski, and S. Agarwal. Fast pda synchronization using characteristic polynomial interpolation. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, 2002.

[108] A. Tridgell and P Mackerras. The rsync algorithm. technical report tr-cs-96-05, department of computer science. In *The Australian National University, Canberra, Australia*, 1996.

[109] Ashok Kumar Das Vanga Odelu and Adrijit Goswami. An efficient cp-abe with constant size secret keys using ecc for lightweight devices. In *IACR Cryptology ePrint Archive*, 2015.

[110] V.Goyal, A.Sahai O.Pandey and, and B. Waters. Attribute based encryption for fine-grained access control of encrypted data. In *In Proceedings of ACM Conference on Computer and Communications Security (ACM CCS)*, 2006.

[111] Dakuo Wang, Judith S. Olson, Jingwen Zhang, Trung Nguyen, and Gary M. Olson. Docuviz: Visualizing collaborative writing. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 1865–1874.

[112] Helen J. Wang, Chris Grier, Alexander Moshchuk, Samuel T. King, Piali Choudhury, and Herman Venter. The multi-principal os construction of the gazelle web browser. In *Proceedings of the 18th Conference on USENIX Security Symposium*, pages 417–432. USENIX Association, 2009.

[113] Brent Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization, 2008.

[114] Hales Wesley. Html5 Architecture. *O'Reilly Media, Sebastopol*, 2012.

[115] Bao Xianqiang, Xiao Nong, Shi Weisong, Liu Fang, Mao Huajian, and Zhang Hang. Syncviews: Toward consistent user views in cloud-based file synchronization services. In *Sixth Annual Chinagrid Conference (ChinaGrid),*, Aug 2011.

[116] Kailiang Ying Wenliang Du Heng Yin Gautam Nagesh Peri Xing Jin, Xunchao Hu. Code injection attacks on html5-based mobile apps: Characterization, detection and mitigation. In *21st ACM Conference on Computer and Communications Security*, 2014.

[117] Hao Yan, U. Irmak, and T. Suel. Algorithms for low-latency remote file synchronization. In *The 27th Conference on Computer Communications. IEEE INFOCOM 2008*, April 2008.

[118] Yanjiang Yang and Youcheng Zhang. A generic scheme for secure data sharing in cloud. In *Parallel Processing Workshops (ICPPW), 2011 40th International Conference on*, Sept 2011.

[119] E. Yuan and J. Tong. Attributed based access control (abac) for web services. In *2005 IEEE International Conference on Web Services, 2005. ICWS 2005. Proceedings.*, July 2005.

[120] Nicholas C. Zakas. Learning from xauth: Cross-domain localstorage. In *http://www.nczonline.net/blog/2010/09/07/learning-from-xauth-cross-domain-localstorage/*, September 2010.

[121] Qi Zhao, Xuanzhe Liu, Xingrun Chen, Jiyu Huang, Teng Teng, and Yong Zhang. Towards a data access framework for service-oriented rich clients. *2010 IEEE Int. Conf. Serv. Comput. Appl.*, dec 2010.

[122] Rui Zhao and Chuan Yue. All your browser-saved passwords could belong to us: A security analysis and a cloud-based new design. In *Proceedings of the Third ACM Conference on Data and Application Security and Privacy*, CODASPY '13, pages 333–340. ACM, 2013.

# Local data storage : security and availability

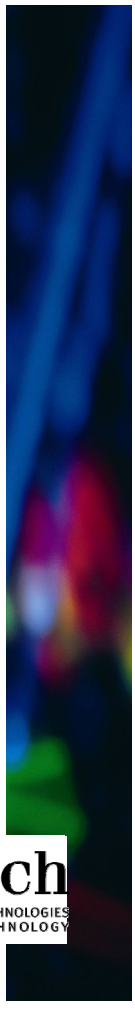
**Mayssa JEMEL**

**ABSTRACT :** Due to technological advancements, people are constantly manipulating multiple connected and smart devices in their daily lives. Cross-device data management, therefore, remains the concern of several academic and industrial studies. The proposed frameworks are mainly based on proprietary solutions called private or closed solutions. This strategy has shown its deficiency on security issues, cost, developer support and customization. In recent years, however, the Web has faced a revolution in developing standardized solutions triggered by the significant improvements of HTML5. With this new version, innovative features and APIs are introduced to follow business and user requirements. The main purpose is to provide the web developer with a vendor-neutral language that enables the implementation of competing application with lower cost. These applications are related neither to the used devices nor to the installed software.

The main motivation of this PhD thesis is to migrate towards the adoption of standardized solutions to ensure secure and reliable cross-device data management in both the client and server side. There is already a proposed standardized Cloud Digital Safe on the server side storage that follows the AFNOR specification while there is no standardized solution yet on the client-side. This thesis is focused on two main areas : 1) the proposal of a standardized Client Digital Safe where user data are stored locally and 2) the synchronization of these data between the Client and the Cloud Digital Safe and between the different user devices.

We contribute in this research area in three ways. First, we propose a Client Digital Safe based on HTML5 Local Storage APIs. We start by strengthening the security of these APIs to be used by our Client Digital Safe. Second, we propose an efficient synchronization protocol called SyncDS with minimum resource consumption that ensures the synchronization of user data between the Client and the Cloud Digital Safe. Finally, we address security concerns, in particular, the access control on data sharing following the Digital Safe requirements.

**MOTS-CLEFS :** Digital Safe, HTML APIs, Local Storage APIs, syncrhonization protocol, Hierarchical Hash Tree, WebSocket, CP-ABE, time based access control.



TELECOM ParisTech

EDiTE

ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY