TALLINN UNIVERSITY OF TECHNOLOGY

Faculty of Information Technology

Department of Software Science

TUT Centre for Digital Forensics and Cyber Security

ITC70LT

Amirhossein Akbari 156312IVCM

A NOVEL APPROACH FOR SECURING HTML5 CLIENT-SIDE DATABASE, INDEXEDDB

Master thesis

Olaf M. Maennel

PhD

Professor at Tallinn University of Technology

Tallinn 2018

Author’s declaration of originality

Author’s declaration of originality is an essential and compulsory part of every thesis. It always follows the title page. The statement of author’s declaration of originality is presented as follows:

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Amirhossein Akbari

05.05.18

Abstract

In recent years, call for killing the browser extensions is arising. Many of the client-based software are immigrating to the web applications. HTML5 is not only a simple markup language anymore. It is regularly developing and bringing new marvelous features. One of this new feature is client-side database IndexedDB. It is a powerful database technology which has brilliant expandability and can handle a significant amount of data within the browser. IndexedDB is plain text. Therefore, it is vulnerable to unauthorized access to its content.

This thesis proposes a method for securely saving IndexedDB database contents. The solution focuses on three different abilities. First the use of the offline and the online mode, second being a cross-browser and third being only a native web (Pure JavaScript) without installation of any third-party extension. IndexedDB is a quite young API, and just a few studies have been done about its security (none of them is available now). This research, first, opens the initial discussion by pointing out the problems and giving an overview of other client-side databases. Then, reviewing the earlier research around IndexedDB and address the security problems. After that showing the design concepts by UML security extension and implementing the security framework with verification by illustrating the result. Finally, checking the proposed solution performance and discuss the future of the research.

This thesis is written in English and is 63 pages long, including 5 chapters, 28 figures and 3 tables.

Keywords: HTML5, JavaScript, web, security, client-side database, IndexedDB, NoSQL, encryption

Annotatsioon

Uuenduslik lähenemine HTML5 kliendipoolse andmebaasi (IndexedDB) turvalisuse tagamiseks.

Viimastel aastatel on brauseri laienduste kasutamine suurenenud. Paljud kliendipoolsed tarkvarad on sisenenud veebirakendustesse. HTML5 pole enam ainult lihtne märgistussüsteem. See on pidevas arengus ning toob uusi imelisi funktsioone. Üheks neist on kliendipoolne andmebaas IndexedDB. See on võimas andmebaasi tehnoloogia, mida saab suurepäraselt laiendada ja mis suudab brauseris hallata märkimisväärset mahtu andmeid. IndexedDB on kõigest tekst, seetõttu on selle sisu haavatav volitusteta juurdepääsu puhul.

Käesolev magistritöö pakub välja meetodi turvaliselt salvestada IndexedDB andmebaasi sisu. Lahendus keskendub kolmele erinevale oskusele. Esiteks offline ja online režiimi kasutamine, teiseks cross-browseri funktsioon ning kolmandaks olemaks lisanditeta veeb (Pure JavaScript), millele pole installeeritud kolmanda osapoole poolt laiendusi. IndexedDB on küllaltki noor API ning seetõttu on selle turvalisusest tehtud vaid mõned uurimistööd (ükski neist pole praegu saadaval). Käesolev töö algab esialgse aruteluga tuues välja probleemid ning andes ülevaate teistest kliendipoolsetest andmebaasidest. Seejärel antakse ülevaade varasematest uuringutest, mille keskmeks on IndexedDB, ning tuuakse välja turvalisuse probleemid. Järgnevalt keskendub magistritöö disaini kontseptsioonidele UML turvalisuse laiendamise abil, teostatakse turvalisuse raamistik koos tõestusega. Lõpetuseks kontrollitakse välja pakutud lahenduse teostumist ning arutletakse magistritöö tuleviku üle.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 63 leheküljel, 5 peatükki, 28 joonist, 3 tabelit.

Table of abbreviations and terms

2FA	Two-Factor Authentication
ACID	Atomic, Consistent, Isolated, Durable of the database
API	Application Programming Interface
CORS	Cross-Origin Resource Sharing
CRUD	create, read, update, and delete
CS	Computer Science
DNS	Domain Name System
DOM	Document Object Model
eID	Electronic Identity
HTML	Hypertext Markup Language
iOS	iPhone Operating System (Apple mobile operating system)
IoT	Internet of Things
JS	JavaScript
LAN	Local Area Network
MFA	Multi-Factor Authentication
MITM	Man-in-the-middle attack
ms	milliseconds
NoSQL	non- SQL or non- relational
OpenSSL	a Toolkit for the Secure Sockets Layer(SSL) and TLS
PGP	Pretty Good Privacy cryptography
RDBMS	Relational Database Management System
SHA	Secure Hash Algorithms
sjcl	Stanford JavaScript Crypto Library
SOP	Same-Origin Policy
SQL	Structured Query Language
W3C	The World Wide Web Consortium
WAN	Wide Area Network
XSS	Cross-Site Scripting

Table of contents

List of figures

List of tables

1. Introduction

Hypertext Markup Language, HTML is the "Web's core language for creating content for everyone to use anywhere. " [1]. From 1989 when Tim Berners-Lee[1] since now, 2018 HTML is developing and changing drastically, and it has never been that reach to handle most of the user dynamic and static needs in web pages.

The last version of it is 5.2 (fifth major version and second minor version) which released by the end of 2017. HTML 5 introduces great new features which make HTML not only a simple markup language to work with text. New elements and features which either bring new functionality to HTML or improve it for making different, dynamic and robust websites without using extra plugins. New features in semantics which specify precisely what is the content of the website. MathML is one of the examples of HTML5 new semantics' feature which helps to work directly with mathematical formulas. In connectivity, Web Socket and WebRTC announced to work with non-HTML data with the server and having real-time communications and video calls. Furthermore, multimedia on the web is more capable and advanced than ever by using new elements such as Camera API and track elements which enables to work with subtitles and chapters. The canvas element is supporting to illustrate graphs and other objects. Performance and integration. Web-based protocol handlers and online and offline events are optimizing the use of hardware and make the websites respond faster. Features such as Geolocation and device orientation are granting the use of different devices and their functionalities. [2]

There are many other new features which have been provided, or they are under development by W3C. However, client-side storages, in particular, IndexedDB which can handle more complex data and has higher performance and abilities than any other client-based database is the focus of this research.

This research starts with introductory topics about the importance of HTML5 and its current key role in the web. After this, it covers the different type of databases such as client-side, server-side, SQL and NO-SQL databases. Henceforward, it would review IndexedDB in detail. Which browsers are supporting this new API and how is it is

[1] Tim Berners-Lee: https://www.w3.org/People/Berners-Lee/

possible to store data by IndexedDB. Following it will examine the security of IndexedDB, and there will be a comparison with all other client-based databases. Above all, the proposed security framework and different options for doing encryption will be covered. Finally, verification of the implemented method and the effects on the performance of storing data securely with all results will be shown.

1.1. Research context

“Science is an organized or systematic body of knowledge.” [3] There has been endless discussion about Computer Science itself. Is it a science or just an “empirical discipline”? [4] It is still indeed an ambiguous area to do research especially for junior researchers. When junior researcher starts to find, develop, and do the real research area, there are many confusions in the research procedure, especially for more practical approaches.

Researching computer science is more challenging than other areas, not only because many enterprise companies are investing huge amount money and doing constant research in this field but also the nature of computer science itself. It is tightly connected to other sciences, and often researcher should combine different research approaches to achieve the goals. As this research is in the more practical computer science field, it does not consider as a classical way of doing research. However, it tries to identify the problems and solve them by combining the robust theory and action.

A methodology is “a system of principles, practices, and procedures applied to a specific branch of knowledge.” It is a “scientific approach that investigates, compares, contrasts, and explains” the diverse ways that research could be conducted. This happens alongside different methods that could be used in these processes. Whereas method is as an “approach, procedure, and guidelines that are used in conducting research. A method might require different tools, instruments, equipment, and such.” [5] [6]

In this research, a methodology is identifying the weaknesses by the systematic approach and develop an experimental a proof of concept. One of the best ways to demonstrate the mitigation of security flaws is to show the applied proof of concept solution on the existing vulnerable system before and after the solution.

Research problem: Since IndexedDB is one of the newest recommended APIs of W3C, just a little research has been done in this field. Besides, the only developed tool as a browser extension is not available. [7] The nature of the web application in the client-side makes everything transparent to everyone. For this reason, it is impossible to entirely protect the front-end code as the back-end. Even by using the obfuscation and encoding of the code, still, it is possible to abuse the client-side by understanding its functionality.

> *The HTML5 client-side database, IndexedDB is saving the data in plain text. Hence, it is vulnerable to unauthorized access to the client's storage.*

The primary **research question** is:

> *How to protect the user's information in the IndexedDB API, against a different type of attacks which triggers to access to the user's data in plain-text which ends with sensitive data exposure?*

A possible answer would be to have a full disk or storage encryption. Yes, it can be the answer but just for the third-party application access, not those flaws or attack vector which is inside the browser. Also, most of the typical users do not have any tendency to apply any cryptography on their device's storage, especially on mobile devices. The next solution would be applying the encryption on database records, which would discuss in this research in detail. The next challenge is (**minor questions**):

> *What kind of cryptography should be used for the development of the solution?*
>
> *In case of using the asymmetrical cryptography, where should the private and public keys store?*
>
> *Is there any other security measure which can help to add an extra layer of security to insure the valid input into the database?*

Also:

> *How to apply the proposed solution on the complete offline web application which is using IndexedDB for saving data?*

These **Hypotheses** follow the earlier questions**:**

> *What kind of risk remains unsolved even after applying the solution?*
>
> *Are there any new risks which appear after applying the implemented method?*

Finding the security vulnerabilities with automatic scanners like Acunetix[1], Nessus[2] and more advanced one like Burp Suite[3] on the client-side is a very challenging task. It needs a manual double check due to the false positive findings. Correspondingly there is no way to have an automatic or versatile method to secure the client-side application. The only way is just following the best practices based on the knowledge and experiences to prevent the security problems.

It should be clarified that sometimes correlation of the leaked information could help an attacker to meet the desired goal, not only the leaked information itself.

1.2. Motivation

The role of the client-side data storages is getting increasingly vital, especially when there is a need to save various and complex data such as different media, video and any other more complex data than text. Having offline storage within the browser also would help developers to focus on the web platform of their product instead of developing for different platforms.

Furthermore, most of the browser vendors as well as browser users, who has the website or who uses it, all of them are desperate for having none or free plugin browsing. Currently, when someone wants to use plugins such Adobe player or Java in the browser, either they are not supported by the browser at all or the security warning will pop up to inform the user the risk of using it.

The other primary reason is the significant increase of browsing the Internet on gadgets. In October 2016, browsing the Internet on mobile devices become more than desktop

[1] https://www.acunetix.com/

[2] https://www.tenable.com/products/nessus/nessus-professional

[3] https://portswigger.net/burp

usage for the first time. Therefore, any solution which is connected to the browsers should support most of the current web browsing on the mobile devices.

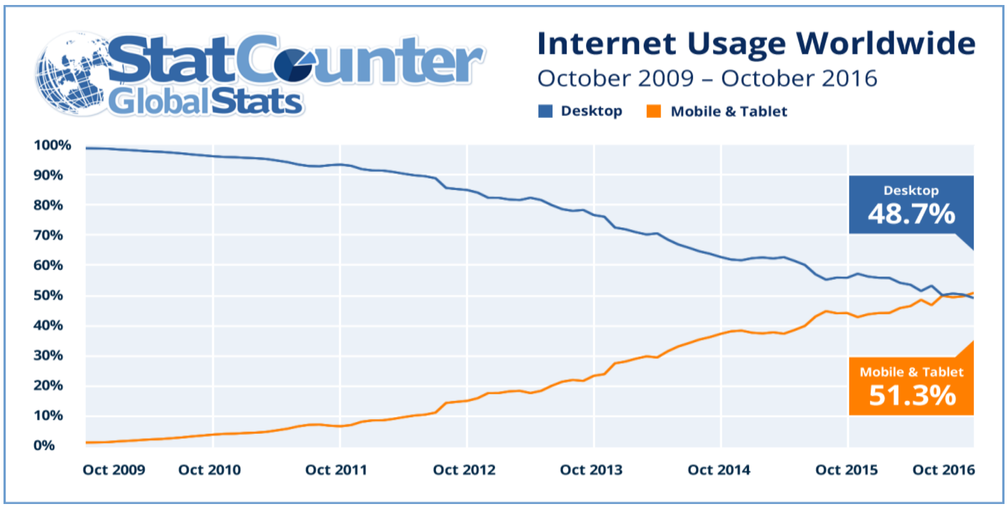


Figure 1: Mobile and tablet Internet usage, worldwide. [8]

1.3. Scope

The idea of this research is implementing the solution for saving data in IndexedDB in a secure way. The solution focuses on usability and compatibility on different well-known platforms such as Windows, Linux, iOS, and Android without installing any extension or add-ons.

The scope narrowed down to the possibility to have unauthorized access to the user's storage which is in plain text. Technical attacks such as code injections, Cross-origin resource sharing (CORS), Cross-site scripting(XSS) as well as social engineering attacks and unauthorized physical access to the user machine or device storage, which tends to have illegitimate access to the user's data.

Vulnerabilities on the server side including protection of private key are out of the scope of this research. This includes all miss-configured and mistakes which have been done by the developer. Moreover, all other types of attacks, such as brute force attack, cross-directly attacks, DNS spoofing, and MITM attacks to steal the private key are not covered by this research.

The focus of the current research is to make sure that the valid data in client side is storing in not plain text mode and to bring essential security for saving these data with encryption.

This research uses one of the existence JavaScript encryption libraries, and implementation of a new encryption method is out of the scope.

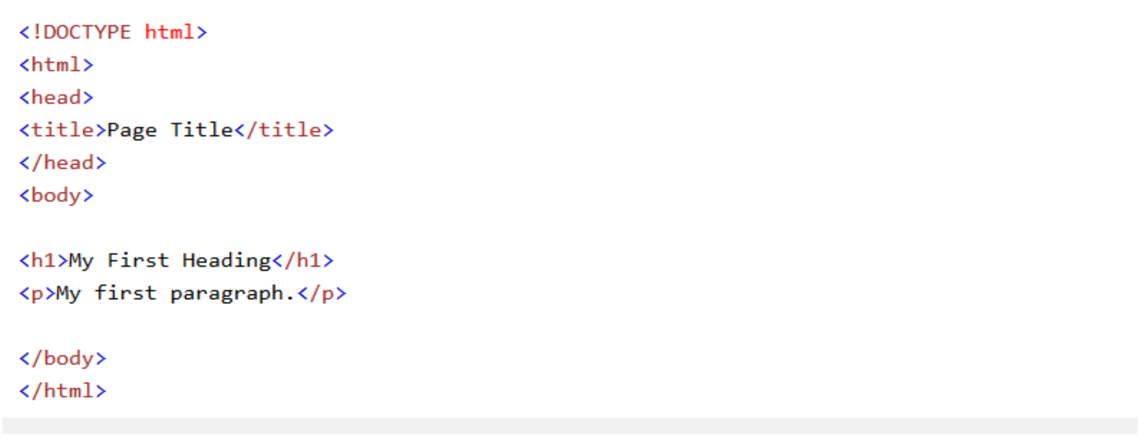
2. Related work

A network is the construction of information and services which are shared among devices so-called nodes or hosts. These hosts can send and receive data, the one which is giving a service named as a server and the other one which using those services called client. Networks are divided into different classes based on the distance and number of devices which can handle them. On the top of all networks, local area network or LAN, and wide area network exists. The latter refers to coverage for the wider area. The ultimate WAN is the Internet which connects incalculable devices such as computers, servers, all kind of gadgets including smartphones, and the Internet of things or IoTs. [9]

2.1. HTML, the WEB's core language

The story of HTML is stared when Tim Berners-Lee and other researchers were looking for a solution of the problem of information accessibility and sharing at the CERN nuclear research facility, Geneva, Switzerland in 1989. They introduced hypertext which can enable connection among diverse sources with hyperlinks. After a while, this method became popular among other researchers in the world and the whole connection among these pages on the Internet named as World Wide Web. [9]

HTML structure is based on the elements which are represented by tags.



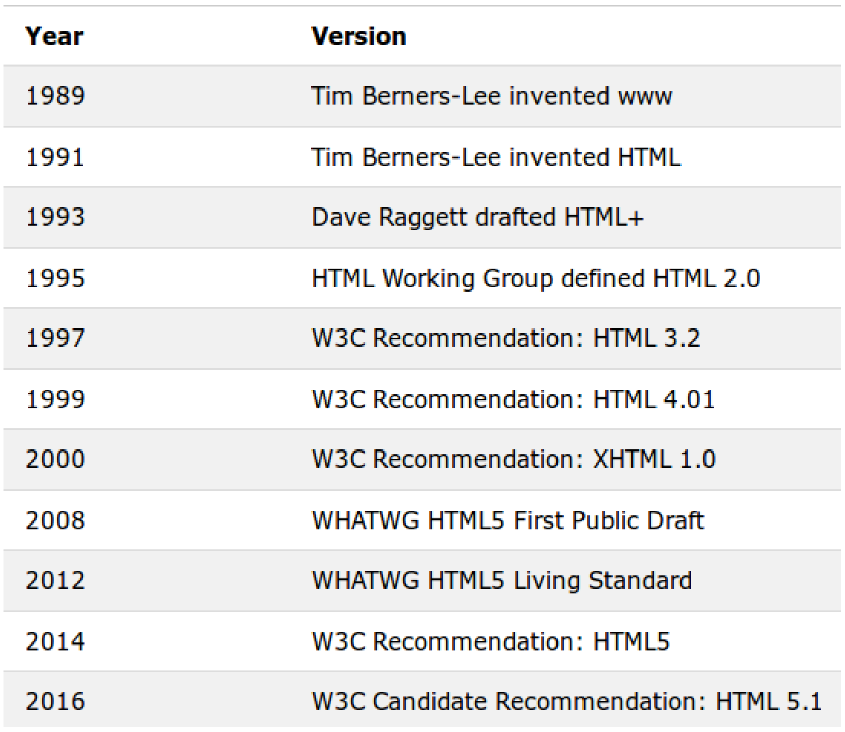
```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>

<h1>My First Heading</h1>
<p>My first paragraph.</p>

</body>
</html>
```

Figure 2: A simple HTML document [10]

In the above picture, <HTML>, <head>, <title>, and <head> are the tags which normally appear in pairs [10]

Since the invention of the WEB, there have been different versions of HTML:

Table 1: HTML versions [10]



Year	Version
1989	Tim Berners-Lee invented www
1991	Tim Berners-Lee invented HTML
1993	Dave Raggett drafted HTML+
1995	HTML Working Group defined HTML 2.0
1997	W3C Recommendation: HTML 3.2
1999	W3C Recommendation: HTML 4.01
2000	W3C Recommendation: XHTML 1.0
2008	WHATWG HTML5 First Public Draft
2012	WHATWG HTML5 Living Standard
2014	W3C Recommendation: HTML5
2016	W3C Candidate Recommendation: HTML 5.1

2.2. Database

"Database is a collection of data." [11] In general, there is a various form of databases available such as rational databases, object, and object-rational databases, etc...

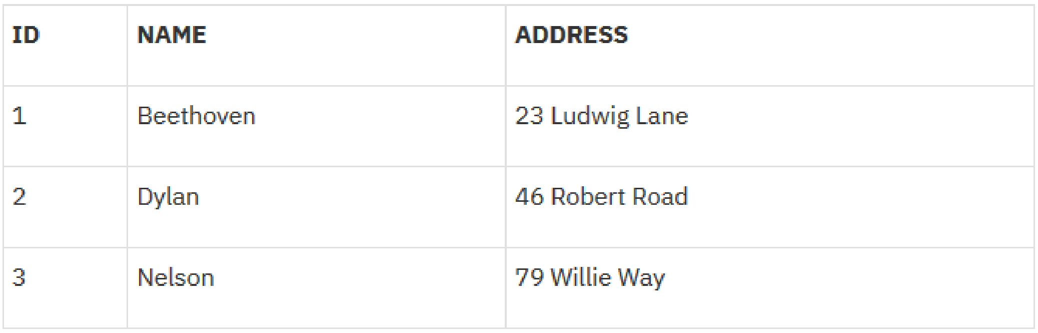
Based on the usage and the type of data, we need to use different databases. This research cannot afford to discuss all of them. Silberschatz gave a comprehensive overview of databases concepts.[1] However, before going further, we need to understand two different main type of databases in General: SQL and NoSQL.

[1] - Abraham Silberschatz, Henry F. Korth, S. Sudarshan, Database System Concepts, 6th Edition, 2011, McGraw-Hill Education

2.2.1. SQL and NoSQL database

SQL or structured query language is a programming language which has been designed for working with relational database management system or RDBMS. SQL structure consists of tables and “tables are divided into rows and columns.” [12]

Table 2: Database table example [12]



ID	NAME	ADDRESS
1	Beethoven	23 Ludwig Lane
2	Dylan	46 Robert Road
3	Nelson	79 Willie Way

NoSQL or non- SQL is a database which enables saving and retrieving data not only with tables, it could also refer to Not only SQL meaning that it may have similar queries which are using in SQL. NoSQL is playing a leading role in the database systems when high scalability and working with big data is demanding. [13] NoSQL databases are using different methods for saving and retrieving the data. One of them is key/value pairs. It is unstructured, non-rational and non- object-oriented. The key is the attribute for the data, and the value could be the data or pointer to any file itself or its address. [14]

Based on the Stack Overflow’s 2018 survey about databases, RDBMS are still the most common than NoSQL databases such as MongoDB. [15]

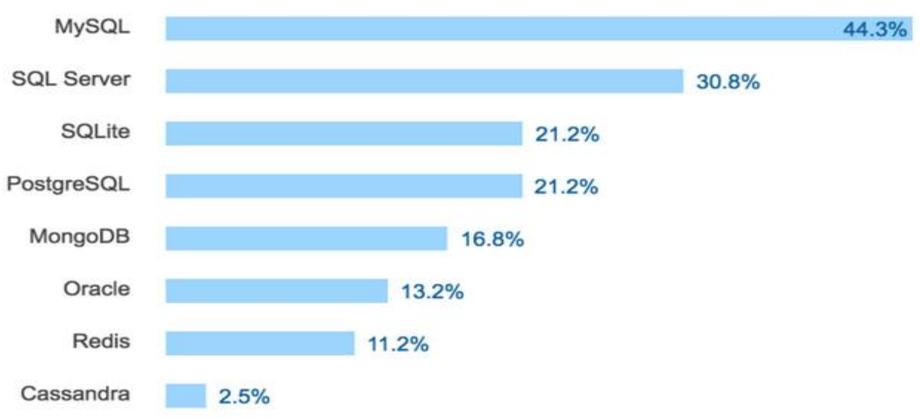


Figure 3: Most popular databases in 2018 according to Stack Overflow survey [15]

However, NoSQL databases are emerging significantly because they can provide impressive performance in the speed and the size. This “high availability” has an adverse effect on ACID (Atomic, Consistent, Isolated, Durable) of the database. [16] Furthermore, when large transactions are needed considering enterprise companies such as Google, Amazon, Microsoft, and Facebook, it is also the first choice of saving data. Being suitable for small and enterprise companies with no worries for the design and the type of the data which will be saved are other exciting points to use NoSQL instead of SQL database. [17]

Some of the popular NoSQL databases are MongoDB, Google’s Big Table, CouchDB, and Cassandra. Considering the drawbacks of NoSQL database over Rational one, immaturity, no availability of standard query language, “no standard interface” and challenging maintenance should be mentioned. [18]

2.2.2. Server-side and client-side database

The WEB’s databases are divided into two main categories: server-side and client-side. Server-side which is the processing and storing data occurs on the server, and client-side is the opposite, meaning that it happens on the user’s machine. The other way to classify the web’s databases are online (mostly as a server-side) and offline ones (often as a client-side). The former needs a connection to the Internet, and the latter can be accessed without any connection.

2.2.3. Overview of the client-side databases

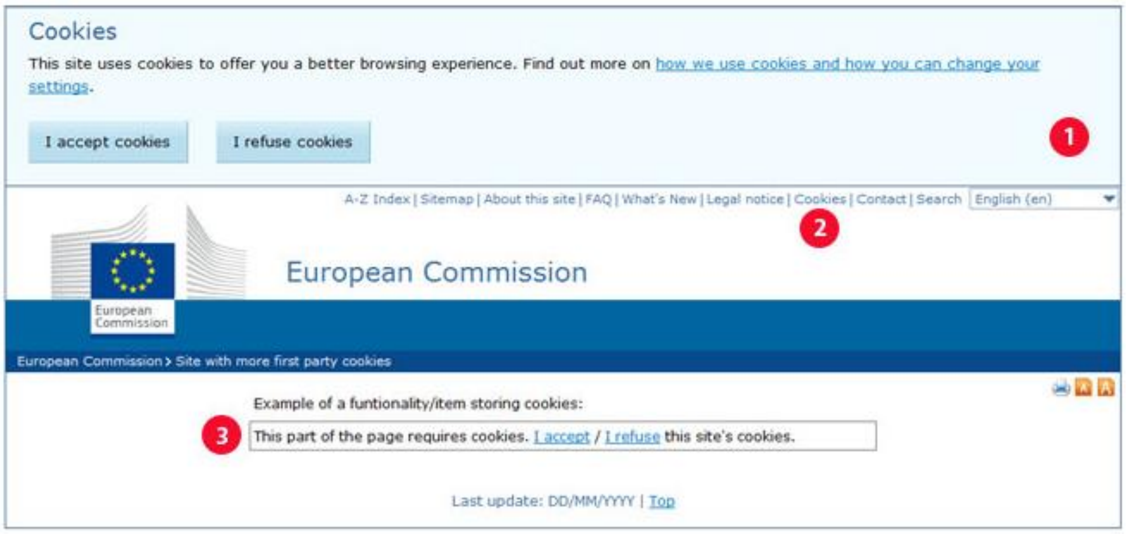
As mentioned before, based on the usage and the data type we can choose which database to use. There are many reasons to consider a client-side database such as:

- They are more responsive than server-side. It will be quicker to access and save the data locally. Hence it is helpful to increase the performance in comparison with the server-side.
- They can be used in an offline mode. It will also make the web application useful when there is a weak connection, accessibility and availability improvement.
- They will reduce the server load as well as bandwidth usage.

- They give us a chance for saving previous activity of the user on the website and customization of the website such a widgets and color scheme or font size. [19]

HTTP Cookie is the most popular client-side database. It uses the stateful session to connect the server and the client for their needs. Each Cookie is 4 KB, and it stores in a user’s browser.

Although, it is a simple and well-supported mechanism; it is lack of basic privacy and security requirements. The European Commission has defined some basics to how to use Cookies. [20]


Cookies

This site uses cookies to offer you a better browsing experience. Find out more on how we use cookies and how you can change your settings.

I accept cookies
I refuse cookies
1
A-Z Index | Sitemap | About this site | FAQ | What's New | Legal notice | Cookies | Contact | Search
English (en)
2
European Commission
European Commission
European Commission > Site with more first party cookies

Example of a funtionality/item storing cookies:

This part of the page requires cookies. I accept / I refuse this site's cookies.

Last update: DD/MM/YYYY | Top

Figure 4: An example of EU commission Cookie information to the user [20]

Cookie poisoning and cookie injection are some examples of security issues. The former is a type of attack which attacker manipulate the cookie contents and bypass the security system. The latter happens if an attacker can inject code or string to HTTP header to execute the commands on the server. [21] [22]

The productivity and flexibility are too weak in comparison with other competitors. It also has extra overhead since it should react by each HTTP requests between server and client. [23] [24]

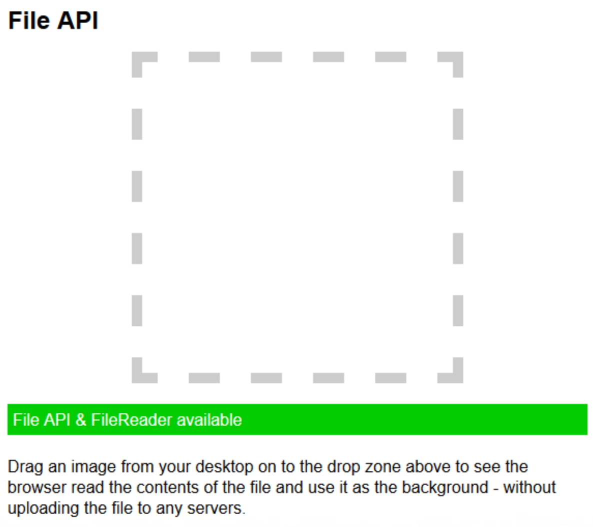
Flash Cookie is a type of cookie which is no longer in use due to plugin-installation. As mentioned in an Introduction part, most of the browsers already stopped to support Flash plugins or just give limit access to them. [25]

Google Gears is another type which Google stopped its development to focus more on HTML5 APIs such as IndexedDB, File API. [26]

Web SQL Database is no longer in maintenance by Web Application Working Group, and they recommend using Web Storage and IndexedDB. [27]

JavaScript Variables considered as a most simple choice of storing data on the client-side. However, it has not enough features with too many limitations to consider as a proper database. This story applies to **windows.name** property too. [28]

File API is currently a W3C Working Draft. It presents File interface which "provides information about files and allows JavaScript in a web page to access their content. " [29] The Blob (Binary Large Object) interface is an „immutable "raw data. File and Blob both should occur asynchronously; this prevents web applications blocking and freezing. [30]



Drag an image from your desktop on to the drop zone above to see the browser read the contents of the file and use it as the background - without uploading the file to any servers.

Figure 5: File API example [31]

```
var holder = document.getElementById('holder'),
    state = document.getElementById('status');

if (typeof window.FileReader === 'undefined') {
  state.className = 'fail';
} else {
  state.className = 'success';
  state.innerHTML = 'File API & FileReader available';
}

holder.ondragover = function () { this.className = 'hover'; return false; };
holder.ondragend = function () { this.className = ''; return false; };
holder.ondrop = function (e) {
  this.className = '';
  e.preventDefault();

  var file = e.dataTransfer.files[0],
      reader = new FileReader();
  reader.onload = function (event) {
    console.log(event.target);
    holder.style.background = 'url(' + event.target.result + ') no-repeat
center';
  };
  console.log(file);
  reader.readAsDataURL(file);

  return false;
};
</script>
```

Figure 6: File API source code [31]

Web storage API is W3C recommendation which represents two similar mechanisms as HTTP session cookies.

- `sessionStorage` saves data for just one session, and after closing the browser tab, data is lost.

```
if (sessionStorage.clickcount) {
    sessionStorage.clickcount =
Number(sessionStorage.clickcount) + 1;
} else {
    sessionStorage.clickcount = 1;
}
document.getElementById("result").innerHTML = "
Number of the clicks on the button is " +
sessionStorage.clickcount + " time(s) in this
session.";
```

Figure 7: sessionStorage example, counting user's clicks on the button in the current session [32]

localStorage stores data without expiry even if the user closes the tab.

```
// Store
localStorage.setItem("lastname", "Smith");
// Retrieve
document.getElementById("result").innerHTML =
localStorage.getItem("lastname");
```

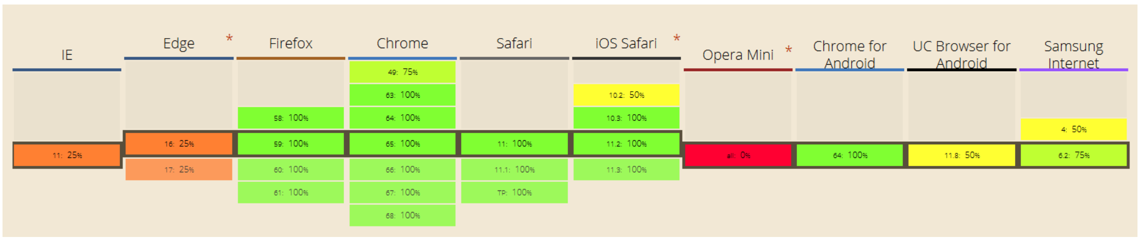
Figure 8: localStorage example, creating and retrieving name/value pair [32]

2.3. Indexed Database API 2.0

Indexed Database API 2.0 or **IndexedDB** is the most exciting recommended candidate among all the databases, when it comes to large-scale data, without limitation. It enables the developer to build web applications with powerful query capabilities. IndexedDB is suitable for storing a considerable amount of data such as online labs, libraries, or any use of a web application without the need for permanent internet connection like widget, to-do lists. [33]

IndexedDB uses many transactions for to store and retrieve data. It is an object-oriented JavaScript database. The objects which support by IndexedDB are all structured close algorithm objects including All primitive types, Boolean object, String object, Date, Blob, File, FileList, ImageData, Array, ObjectMap, etc. „The structured clone algorithm is an algorithm defined by the HTML5 specification for copying complex JavaScript objects. " [34] Furthermore the efficiency of saving data in IndexedDB is the same as storing the file in the OS. [21]

Browser support of two different version of IndexedDB could be found in the following reference, here is the summary:



IE	Edge *	Firefox	Chrome	Safari	iOS Safari *	Opera Mini *	Chrome for Android	UC Browser for Android	Samsung Internet
			49: 75%						
			63: 100%		10.2: 50%				
		58: 100%	64: 100%		10.3: 100%				4: 50%
11: 25%	16: 25%	59: 100%	65: 100%	11: 100%	11.2: 100%	all: 0%	64: 100%	11.8: 50%	6.2: 75%
	17: 25%	60: 100%	66: 100%	11.1: 100%	11.3: 100%				
		61: 100%	67: 100%	TP: 100%					
			68: 100%						

Figure 9: IndexedDB browser support: supported, not supported, and Partial support [35]

2.3.1. IndexedDB basic pattern

The IndexedDB basic pattern is:

1. Open a database.
2. Create an Object Store in the database.
3. Start a transaction and request a database operation, like add or retrieve.
4. Waiting for the completion of the operation by listening to the DOM event.
5. Do something with the results. [36]

First, we need to add the following prefixes:

```
var indexedDB = window.indexedDB ||
window.mozIndexedDB || window.webkitIndexedDB ||
window.msIndexedDB;


//prefixes of window.IDB objects
var IDBTransaction = window.IDBTransaction ||
window.webkitIDBTransaction ||
window.msIDBTransaction;
var IDBKeyRange = window.IDBKeyRange ||
window.webkitIDBKeyRange || window.msIDBKeyRange


if (!indexedDB) {
    alert("Your browser does’nt support a stable
version of IndexedDB.")
}
```

Figure 10: IndexedDB prefixes of implementation **[37]**

```
var userData = [
  { id: "1", name: "Tapas", age: 33, email:
"tapas@example.com" },
  { id: "2", name: "Bidulata", age: 55, email:
"bidu@home.com" }

            ];
```

Figure 11: Sample Data [37]

Then Open the database, and after that create and add the data to the table:

```
var db;
var request = indexedDB.open("databaseName", 1);

request.onerror = function(e) {
  console.log("error: ", e);
};

request.onsuccess = function(e) {
  db = request.result;
  console.log("success: "+ db);
};
request.onupgradeneeded = function(e) {

}
request.onupgradeneeded = function(event) {
        var objectStore =
event.target.result.createObjectStore("users",
{keyPath: "id"});
        for (var i in userData) {
                objectStore.add(userData[i]);
        }
}
```

Figure 12: Open the database, create, and add table [37]

2.3.2. IndexedDB security

Each browser saves the local databases in different directories, but the critical problem is even after deleting the browsers cash, it is possible to recover the database file itself and have access to its data.

W3C has already defined some privacy and security requirements and information in IndexedDB API 2.0 documentation:

- **User tracking:** By blocking third-party storage, expiring stored data, treating persistent storage as cookies, site-specific safe-listing of access to databases, Origin-tracking of stored data, and shard blocklist.

- **Cookie resurrection:** IndexedDB should consider separately from HTTP session cookies, which enables the use of both features as an extra backup for each other.
- **The sensitivity of data:** All stored data should consider as a sensitive data. Information such as emails, appointments in the calendar, to-do lists, „health records or other confidential documents" could be possibly saved on the user machine by this API.
- **DNS spoofing attacks:** By using Transport Layer Security or TLS certificates it is possible to mitigate this type of attacks to make sure about the domain of the host.
- **Cross-directory attacks:** „There is no feature to restrict the access by pathname even if a path-restriction feature was made available "
- **Implementation risks:** Allowing third-party website to read and write the data could cause information leakage, and spoofing correspondingly. Hence W3C is highly recommended to follow the original model which not lets adversary access to the user's information. [38]

IndexedDB is based on same-origin policy or SOP. SOP forces origins (port and host) of document or script to have a connection with a specific domain, hence the data is not accessible by other sources. [39]

2.3.3. IndexedDB potential attacks:

Any attack on the client-side which triggers to get access to the user's storage considers as a potential attack since IndexedDB is saving the data in plain text without any encryption. Therefore, it is necessary to add an extra security layer on it.

Examples of attacks which may lead to information leakage or data disclosure in IndexedDB mentioned by Kimak such as Cross-origin resource sharing or CORS, Cross-site scripting or XSS, social engineering, and physical access. [40]

Cross-origin resource sharing or **CORS** is a method which adds extra information to HTTP headers to permit the user to access to different origins (domain) which currently is in use. [41] It is regularly forbidden to interact with cross-domain requests than same-origin requests. However, this feature allows handling of different origins (cross-domain requests) between the client and the server.

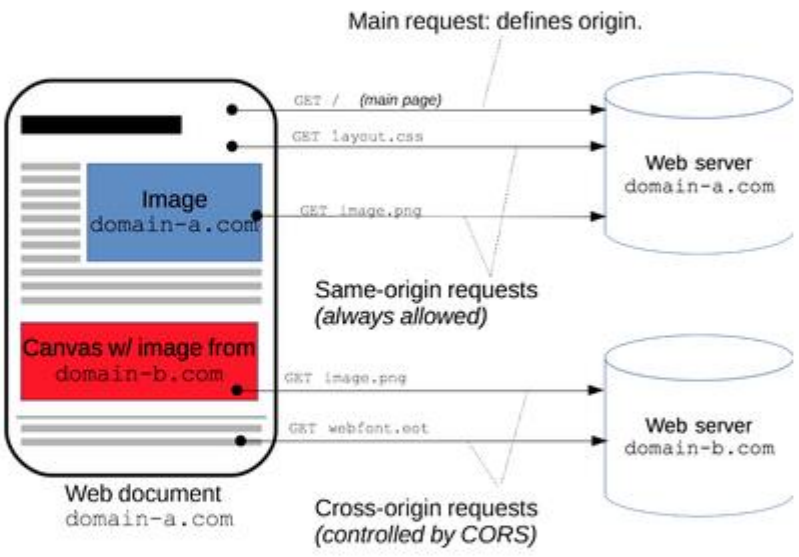

Main request: defines origin.
GET / (main page)
GET layout.css
GET image.png
Image domain-a.com
Web server domain-a.com
Same-origin requests (always allowed)
Canvas w/ image from domain-b.com
GET image.png
GET webfont.eot
Web server domain-b.com
Web document domain-a.com
Cross-origin requests (controlled by CORS)

Figure 13: The CORS mechanism [41]

An attacker could change the value of requested HTTP header Origin and gain unauthorized access to the web client. [42]

Cross-site scripting or **XSS** is a type of code or script injection to the web applications. Three common types of these attacks are stored XSS, reflected XSS and DOM-based[1] XSS. Stored XSS is an injected malicious code which is “permanently stored on the target servers.” Reflected XSS needs user interaction such as responding to the error message or any other response from the user to input to the server. DOM-based XSS which “is executed because of modifying the DOM environment.”, meaning that attacker could manipulate any objects in the DOM to execute his malicious script in the user’s browser. [43] [44] [45]

Social engineering attack where attacker tried to deceive the target to achieve his goal and finally gain unauthorized access to the system. It could accomplish by technical or

[1] Document Object Model or DOM. “When a web page is loaded, the browser creates a Document Object Model of the page.” [67]

non-technical ways. Sending phishing email, calling by phone, real-life chats are most common examples of social engineering attacks.

Physical access attack could be stolen user's machine such as a laptop, mobile phones, or other any gadget, unattended logged in the machine.

Browser and third-party application zero-day attacks: zero-days are unrevealed vulnerabilities which can be abused by an attacker. In the browsers, there have always been some zero-days since they are the primary bridge between user's machine or device and the Internet. Some vendors already understood the criticality of the zero-days. Thus, they proposed bug bounties. Giving the example of one of the freshest zero-day was LinkedIn Autofill[1] feature bug. [46]

2.3.4. IndexedDB security proposed solutions

There has been very little research about IndexedDB until now. This section reviews those available ones to the public.

First, the research includes of reviewing the role of IndexedDB in the past, present and future. It States the role of Cookies, eCommerce, and mobile eCommerce as a motivation for the development of IndexedDB in the past. Current situation of IndexedDB has been analyzed by a different point of views such as Improving "Cookies functionality," usage in mobile devices and offline-usage. The future of IndexedDB showed as not a suitable storage for storing personal information. Besides of encryption using multifactor authentication (MFA) or two-factor authentication (2FA) proposed as solutions. [21]

Second, the proposed security model consists different layers which means even if "authentication process comprised" it is not possible to intercept and read the data. The model is using hashing and encryption by Stanford JavaScript Library as an extension of the browser. [47] It gets a secure login, encrypt the data, store the public key on both client and server side and private key in the server, decryption of data, and finally secure deletion. Evaluation of security model had done by performing XSS attack scenario before and after adding security model. [40]

[1] https://www.linkedin.com/help/lms/answer/65688/linkedin-autofill-setup-guide?lang=en

The solution is a theoretical framework proposed in the development of IndexedDB. It divides into distinct parts such as:

- Client-side data encryption
- Code analysis
- Input validation
- SOP (same-origin policy)

The framework is a browser extension. Using JavaScript encryption which is based on EAS or SHA-256 with verification hash. [48] Code Analysis consist of static and dynamic analysis. “Input and output validation” have done with taking strings and returning the proper value. [40]

Third, the solution consists of two parts, writing and reading. Authentication has done by using OAuth [49], OpenSSL [50] , Crypto++ [51] crypto libraries for encryption. [52]

In the writing phase, ensuring that the secure connection is established, open a connection to the database, encryption with a library which generates a public key (on both sides), and a private key on the server side, and finally saving the file and closing the connection.

In the reading phase, checking user credentials, get the public key from a user and do the key pairing, decrypt the data, show the data, and close the connection. [52]

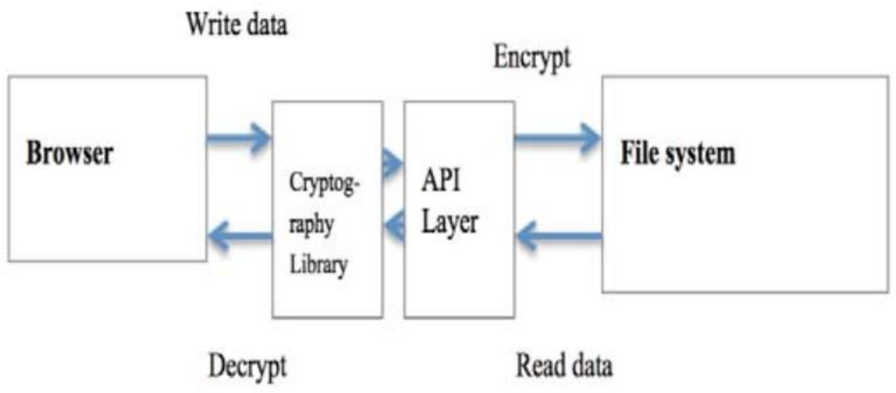


Figure 14: Proposed encryption library [52]

3. Solution (proof of concept)

This section explains how the solution is defined and designed. After that have an overview of JavaScript cryptography solution and input validation on the client-side.

3.1. Design concepts

Confidentiality (Just authorized users can access to the information), integrity (Preventing authorized changes and manipulation), and availability (assuring that the services are available and users have access to their information) are three principal concepts in information security.

Confidentiality
Availability
Integrity

Figure 15: The CIA triad [53]

Since this research is based on the client-side database, the CIA triad impact would be:

- **Confidentiality:** Information disclosure from the user's device. It could be the browsers history, clipboard, cookies, files, social engineering and finally all plain-text client-side databases.

- **Availability:** Any attacks or security flaw which leads to lack of access to the information by an authorized user for legitimate access, such as taking advantage of browser vulneraries, popup floods, spam pages.

- **Integrity:** Integrity could threaten by any attacks which lead to inject any code on the user's browser or manipulate the data. XSS, CORS and all type of injection in client-side could consider as a negative impact on integrity.

The framework would cover the confidentiality impact; however as shown in (figure 15), (the CIA triad), they are overlapping in some areas.

The attack in information security is defined as “attempt to destroy, expose, alter, disable, steal or gain unauthorized access to or make unauthorized use of an asset (anything valuable for an organization).” [54]

OWASP has an extensive list of the different attack scenarios. [1] Most of the references divide them into two main categories, active and passive. However, one conceivable way to categorize the web attack scenarios is to divide them into to category: server-side and client-side.

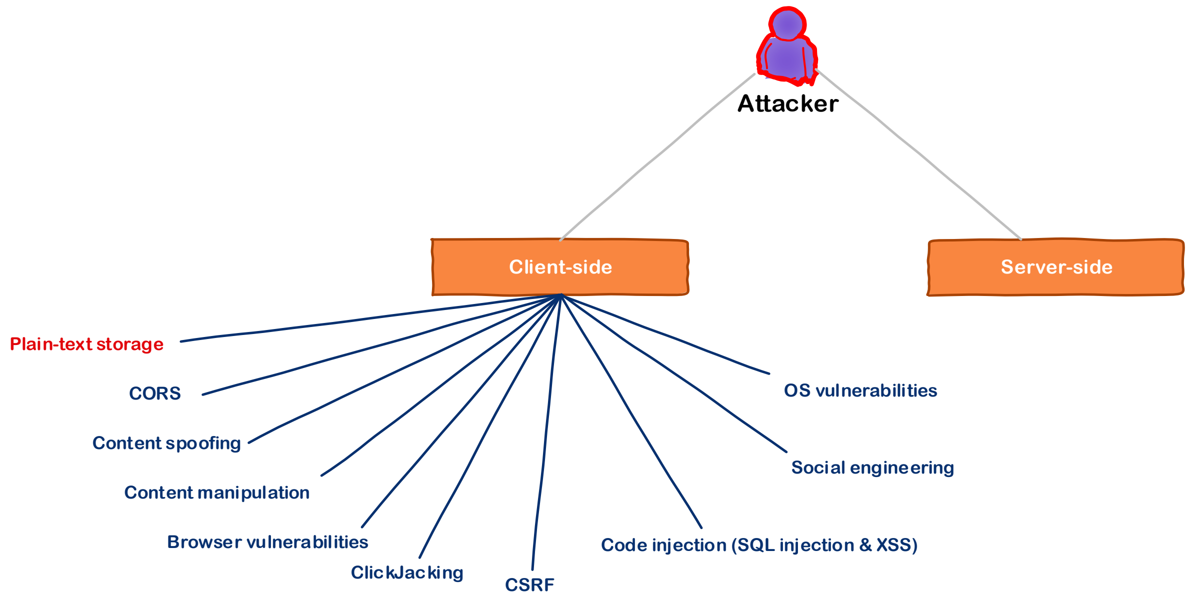


Figure 16: Typical attacks on the client-side

(Figure 16) Shows the common attacks on the client-side. Storing sensitive information on the client side is not recommended at all, however correlation of information can help an attacker to establish the attack. The simple example of this type of attacks is when the attacker had already some specific information about the victim or targeted system, and

[1] https://www.owasp.org/index.php/Category:Attack

getting access to the insensitive information can help to have a better estimate about other information or vice versa.

As it is explicitly highlighted in the (figure 16), the purpose of this research is **mitigation** against **unauthorized access to the plain-text storage** client-side database, IndexedDB. However, the **other type of technical and non-technical** ones such as code injections, contents manipulation and even device theft which also can trigger to have illegitimate access to the data on the client storage could be mitigated by this solution.

There are also some risks which remain without any solution. These types of residual risks are inevitable due to the nature of the client-side. The best example is cross-directory attacks and DNS spoofing attacks. About the latter, even the solution checks the secure connection no one can "guarantee that a host claiming to be in a certain domain really is from that domain." [38]

Use cases are used to show the "functional requirements" in software development and design. It helps to clearly define and demonstrate the communication between so-called actors, which are the roles or users of the system. [55] "A **Misuse case** is the inverse of a use case." [56]

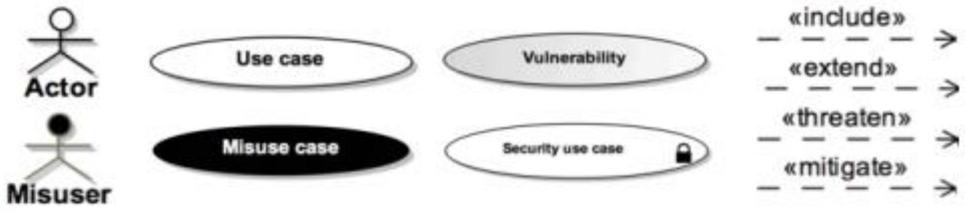


Figure 17: "Graphical Misuse case constructs." [57]

The system is a proof of concept security framework, the misuse case diagram is:

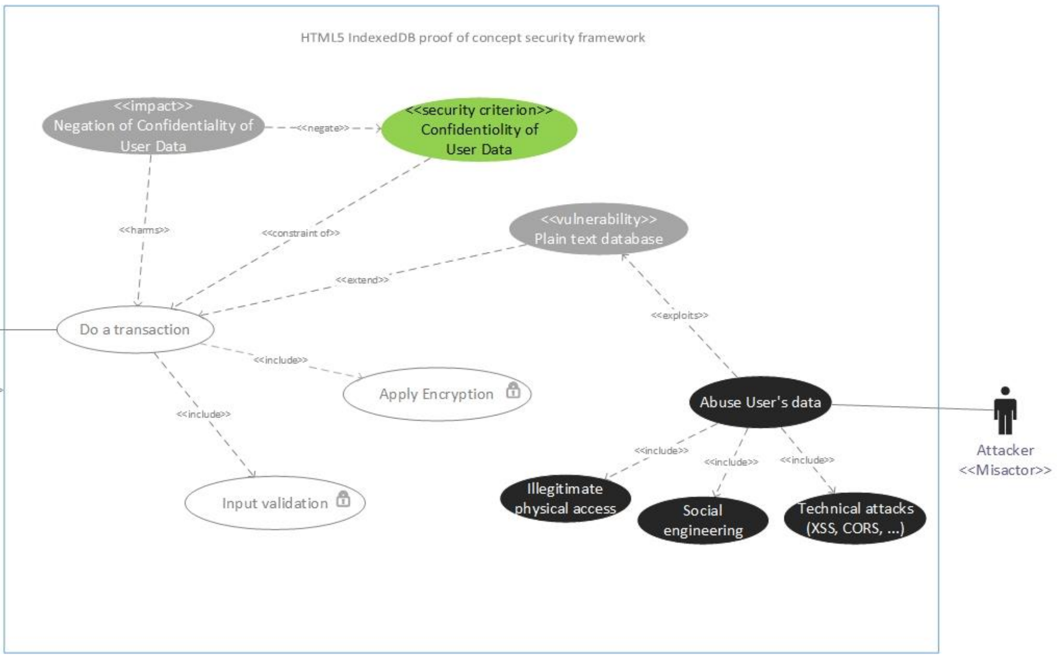

HTML5 IndexedDB proof of concept security framework
<<impact>> Negation of Confidentiality of User Data
<<negate>>
<<security criterion>> Confidentiolity of User Data
<<harms>>
<<constraint of>>
<<vulnerability>> Plain text database
<<extend>>
<<exploits>>
User <<actor>>
Do a transaction
<<include>>
Apply Encryption
<<include>>
Input validation
Abuse User's data
Attacker <<Misactor>>
<<include>>
<<include>>
<<include>>
Illegitimate physical access
Social engineering
Technical attacks (XSS, CORS, ...)

Figure 18: Misuse case diagram of the framework

- The **actor** is the user of the browser.
- The **misactor** is an attacker who is abusing the vulnerability.
- The **vulnerability** is saving plain text data in client browsers' database.
- The **use case** is the transactions in IndexedDB.[1]
- The **misuse cases** Illegitimate physical access, social engineering, and technical attacks.
- The **security criterion** is the confidentiality of the user data.
- The **security use cases** are validation the input and applying cryptography.
- The **impact** is a negation of the confidentiality of the user data.

- [1] (As explained previously in **2.3.1** any interaction with IndexedDB needs transactions, so it is not the database transaction)

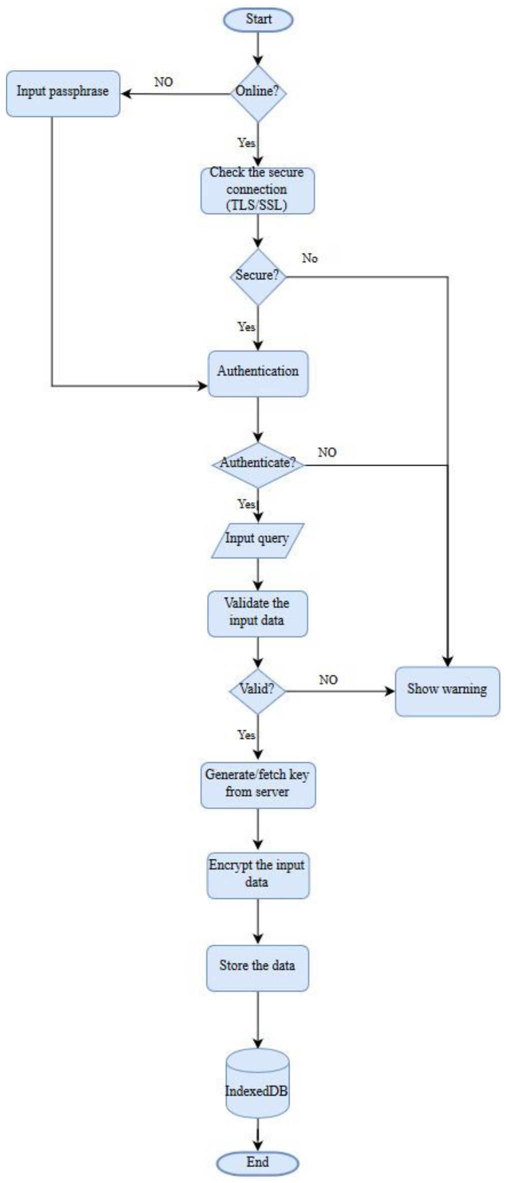


Figure 19: Flowchart of the implemented security framework

(Figure 19) shows the encryption phase of the security framework. In decryption phase, everything is the opposite of the flowchart. Since one of the important advantages of IndexedDB is running in offline mode, therefore the framework is supporting both online and offline mode. However, in the offline mode user needs to input the passphrase because saving the private key in the client-side is not secure.

3.2. JavaScript crypto libraries and input validation

This section discusses different options of JavaScript cryptography. Following that the input validation in the client-side will be explained.

3.2.1. JavaScript encryption libraries

The next challenge is choosing among different JavaScript encryption libraries. Most of the JavaScript encryption libraries are not up to date. Therefore, they have not mentioned in the list above.

Comparing several aspects of various client-side encryption libraries such as performance, flexibility, and reliability needs another research to be done. Furthermore, even trying each of them takes considerable time and often it is unsuccessful due to not being up to date.

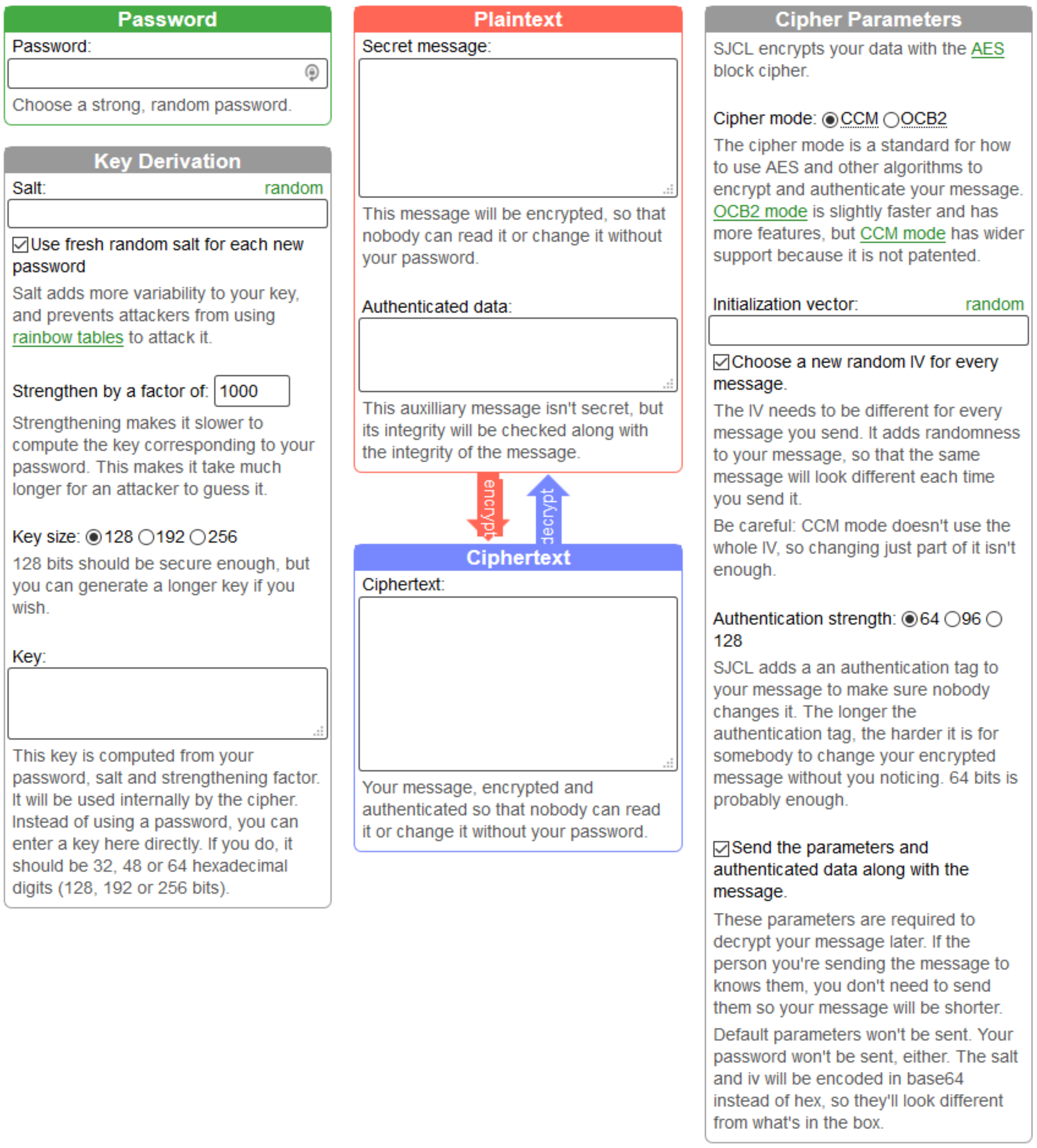
Below there is a list of the most well-known ones among developers with the brief description:

Table 3: JavaScript encryption libraries

JavaScript Encryption library name	Description
Web Cryptography API	JavaScript API which handles basic cryptographic operations. However, it cannot handle electronic identity (eID) or online banking which are using specific certificates and hardware cryptography. https://www.w3.org/TR/WebCryptoAPI/
(sjcl) Stanford JavaScript Crypto Library	JavaScript Crypto library introduced by Stanford Computer Security Lab. It is designed for having a rapid, robust, easy and cross-platform library. https://bitwiseshiftleft.github.io/sjcl/
crypto-js	The simple interface of JavaScript cryptographic implementation. Not up to date anymore, recommends forge. https://code.google.com/archive/p/crypto-js/
Forge	An implementation of TLS in JavaScript. https://github.com/digitalbazaar/forge
Ohdave	Complete RSA in JavaScript. http://ohdave.com/
OpenPGP in JavaScript	Public key encryption in JavaScript. https://www.hanewin.net/encrypt/
jsbin	RSA and ECC in pure JavaScript. http://www-cs-students.stanford.edu/%7Etjw/jsbn/
js-nacl	JavaScript high-level API wrapper based on NaCL.[1] https://github.com/tonyg/js-nacl https://cr.yp.to/highspeed/coolnacl-20120725.pdf

Considering the speed, flexibility, and ease of use of sjcl Stanford JavaScript Crypto library, this framework is based on sjcl or Stanford JavaScript Crypto library. It is optimized and compressed to under 6.4 KB with remarkable performance. The industry-standard AES algorithm is used at 128, 192 or 256 bits; the hash function is SHA256; the authentication code is HMAC; the authentication code is PBKDF2, and the authenticated-encryption modes are CCM and OCB. [58] The library also has a demo and the browser test:

[1] Networking and Cryptography library, http://nacl.cr.yp.to/



Password

Password:

Choose a strong, random password.

Key Derivation

Salt: random

☑ Use fresh random salt for each new password

Salt adds more variability to your key, and prevents attackers from using rainbow tables to attack it.

Strengthen by a factor of: 1000

Strengthening makes it slower to compute the key corresponding to your password. This makes it take much longer for an attacker to guess it.

Key size: [✓ 128] 192 256

128 bits should be secure enough, but you can generate a longer key if you wish.

Key:

This key is computed from your password, salt and strengthening factor. It will be used internally by the cipher. Instead of using a password, you can enter a key here directly. If you do, it should be 32, 48 or 64 hexadecimal digits (128, 192 or 256 bits).

Plaintext

Secret message:

This message will be encrypted, so that nobody can read it or change it without your password.

Authenticated data:

This auxilliary message isn't secret, but its integrity will be checked along with the integrity of the message.

Ciphertext

Ciphertext:

Your message, encrypted and authenticated so that nobody can read it or change it without your password.

Cipher Parameters

SJCL encrypts your data with the AES block cipher.

Cipher mode: [✓ CCM] OCB2

The cipher mode is a standard for how to use AES and other algorithms to encrypt and authenticate your message. OCB2 mode is slightly faster and has more features, but CCM mode has wider support because it is not patented.

Initialization vector: random

☑ Choose a new random IV for every message.

The IV needs to be different for every message you send. It adds randomness to your message, so that the same message will look different each time you send it.

Be careful: CCM mode doesn't use the whole IV, so changing just part of it isn't enough.

Authentication strength: [✓ 64] 96 128

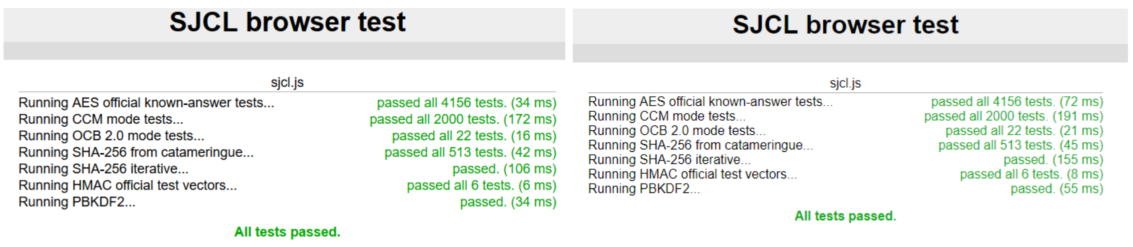
SJCL adds a an authentication tag to your message to make sure nobody changes it. The longer the authentication tag, the harder it is for somebody to change your encrypted message without you noticing. 64 bits is probably enough.

☑ Send the parameters and authenticated data along with the message.

These parameters are required to decrypt your message later. If the person you're sending the message to knows them, you don't need to send them so your message will be shorter.

Default parameters won't be sent. Your password won't be sent, either. The salt and iv will be encoded in base64 instead of hex, so they'll look different from what's in the box.

Figure 20: Stanford JS Crypto library demo page [59]



SJCL browser test

```
                         sjcl.js
Running AES official known-answer tests...     passed all 4156 tests. (34 ms)
Running CCM mode tests...                      passed all 2000 tests. (172 ms)
Running OCB 2.0 mode tests...                     passed all 22 tests. (16 ms)
Running SHA-256 from catameringue...             passed all 513 tests. (42 ms)
Running SHA-256 iterative...                              passed. (106 ms)
Running HMAC official test vectors...              passed all 6 tests. (6 ms)
Running PBKDF2...                                          passed. (34 ms)

                    All tests passed.
```

SJCL browser test

```
                         sjcl.js
Running AES official known-answer tests...     passed all 4156 tests. (72 ms)
Running CCM mode tests...                      passed all 2000 tests. (191 ms)
Running OCB 2.0 mode tests...                     passed all 22 tests. (21 ms)
Running SHA-256 from catameringue...             passed all 513 tests. (45 ms)
Running SHA-256 iterative...                              passed. (155 ms)
Running HMAC official test vectors...              passed all 6 tests. (8 ms)
Running PBKDF2...                                          passed. (55 ms)

                    All tests passed.
```

Figure 21: Stanford JS Crypto library browser test (left Firefox 59.0.2, right Chrome 66.0.3359.117) [60]

3.2.2. Input validation

Input validation is a mechanism which helps to guarantee the input of the proper form of data in the information system. It is highly recommended to confirm the input in the first steps by input validation technics exactly after the system received any data (received any input). However, it should not be the main security consideration for code injections and other types of attack which are used by inputting invalid data into the system. In web applications, all of the non-trustworthy inputs should handle the input validation such as HTTP and XML requests, HTTP POST, and database queries. Input validation could be a simple string check, enforcing regular expressions, whitelisting and JavaScript input validation which happens in the client-side. [61] [62] For instance, it could be checking the input for a proper pattern such as a date, email, and … below there is a simple input validation which requires entering a username between 4 to 8 characters:

```
<form>
  <div>
    <label for="uname">Choose a username:
</label>
    <input type="text" id="uname" name="name"
required size="10"
          placeholder="Username"
          minlength="4" maxlength="8">
    <span class="validity"></span>
  </div>
  <div>
    <button>Submit</button>
  </div>
</form>
```

Figure 22: A simple input validation example [63]

Must be noted there is no such a general input validation which can validate all type of inputs. Thus, it is a part of the front-end developer responsibility to confirm the input before allowing the user to input any data into the system.

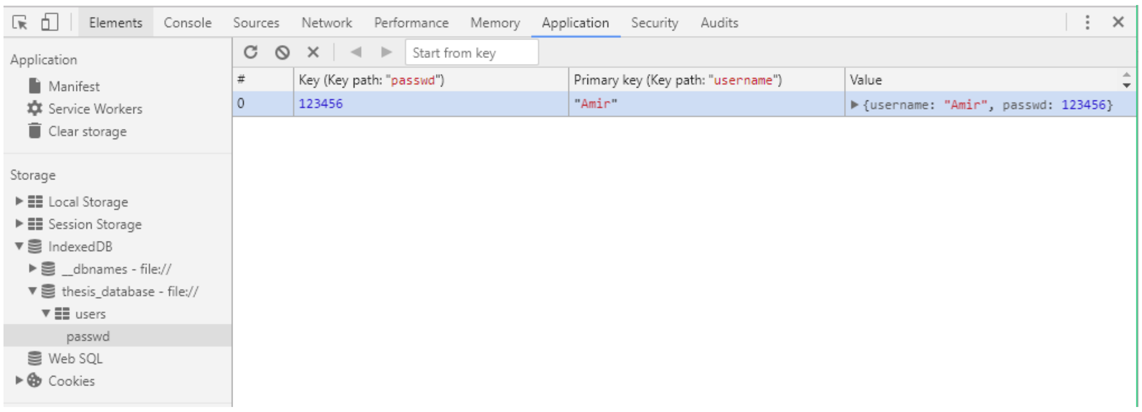
The server-side input validation applies to the server-side which is not the focus of this research.

4. Experiment and performance

This section shows the results of the applied framework. Before going into details and starting this research, the idea was to use Intel XDK [1] as an IDE and environment to develop the code. However, the idea of not using anything than the browser and text editor became more interesting. For testing the compatibility of the code, the code ran on different browsers. Finally, Google Chrome, Mozilla Firefox, Opera, and Safari have been chosen for the compatibility test. The test environment is the latest Chrome browser (Version 66.0.3359.117) hosting by windows 10 machine. The machine is Lenovo T460S with Intel i5 5300, 8 Gigabyte DDR4 RAM and TOSHIBA THNSFJ256GDNU A SSD hard disk. The server simulation is simple npm http-server. [2] However, the testing and the performance check is just on the client-side to omit the effect of the server and network latency.

4.1. Experiment

The experiment runs the small script[3] by dexie.js. It is a minimalistic, lightweight wrapper for IndexedDB which has near-native performance. [64] The first script stores the data in a standard way without encryption.



#	Key (Key path: "passwd")	Primary key (Key path: "username")	Value
0	123456	"Amir"	▶{username: "Amir", passwd: 123456}

Figure 23: IndexedDB database plain text contents

[1] https://software.intel.com/en-us/xdk/docs/release-notes-information-intel-xdk

[2] https://www.npmjs.com/package/http-server

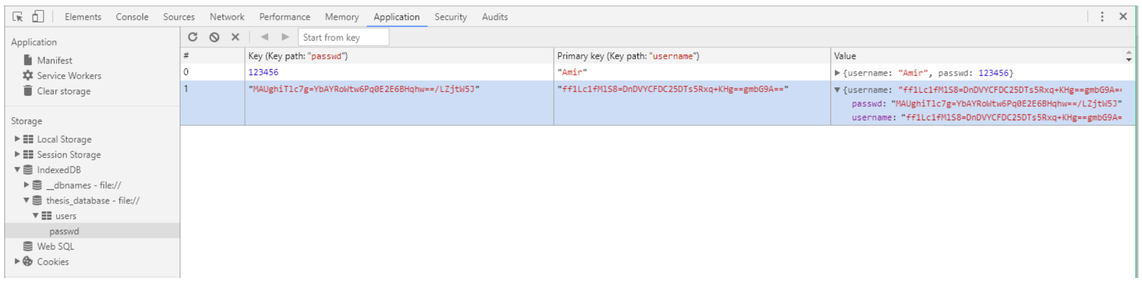
[3] https://github.com/amirgole6

The attack vector can be any mentioned attack scenarios, however for having clearer snapshot the XSS in IndexedDB which does not have a proper input validation consider as follow:

```
function user(){

var resource = location.hash.substring(1);

db.users.get("user",resource);

example = db.users.get("user");
document.getElementById("div1").innerHTML=user;
}
</script>

<body onload="action()">
<div id="div1"></div>
</body>
```

Figure 24: XSS simple example attack scenario [1]

The same username and password have been stored with encryption (Appendix III). It should be noted that this is just an example and saving credentials on the client-side is against the security considerations, however, in some cases if the developer does not have any other option (complete offline application) it might be used.



#	Key (Key path: "passwd")	Primary key (Key path: "username")	Value
0	123456	"Amir"	▶ {username: "Amir", passwd: 123456}
1	"MAUghiTlc7g=YbAYRoWtw6Pq0E2E6BHqhw==/LZjtW5J"	"ff1Lc1fMlS8=DnDVYCFDC25DTs5Rxq+KHg==gmbG9A=="	▼ {username: "ff1Lc1fMlS8=DnDVYCFDC25DTs5Rxq+KHg==gmbG9A= passwd: "MAUghiTlc7g=YbAYRoWtw6Pq0E2E6BHqhw==/LZjtW5J" username: "ff1Lc1fMlS8=DnDVYCFDC25DTs5Rxq+KHg==gmbG9A=

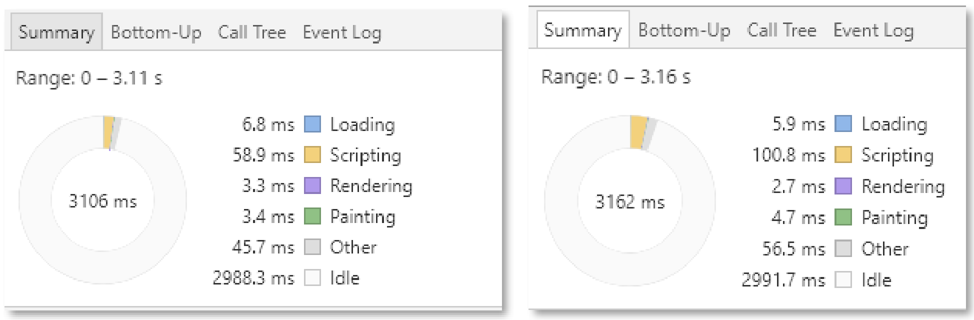
Figure 25: IndexedDB database key and values after encryption (same data)

[1] The idea is from OWASP local storage testing.

It is also possible to see the tables and its entries (even after deleting them and recover back) with third party application such as SQLite manager. A possible solution for mitigating this risk would be using the full disk encryption. However, the database content is readable and accessible by the browser itself, therefore, even after full disk encryption, the risk of attacks from the browser side like code injections remains. However, after applying this solution on the database keys and values, the risk of illegitimate access inside the browser is mitigated too.

4.2. Analyzing the performance

Performance check cannot be done by just showing the result of inserting and reading data into the database. The running and parsing the script itself also should consider in performance analysis. Therefore, the best way of doing it is checking the performance by the browser. Almost all the current browsers have recording the performance option in DevTools (Inspect Element). Using Incognito Mode (Private Browsing) is ensuring the clean run without extra load of any extension or script. The experiment runs without any user interaction. Hence the database entries are reading by JavaScript and put in the table and read it as an output of the console (console.log). All used libraries run locally on the machine too. The reference which is used in this research is based on Performance Analysis Reference by Google. [65]



Range: 0 – 3.11 s

3106 ms

6.8 ms	🟦 Loading
58.9 ms	🟨 Scripting
3.3 ms	🟪 Rendering
3.4 ms	🟩 Painting
45.7 ms	⬜ Other
2988.3 ms	☐ Idle

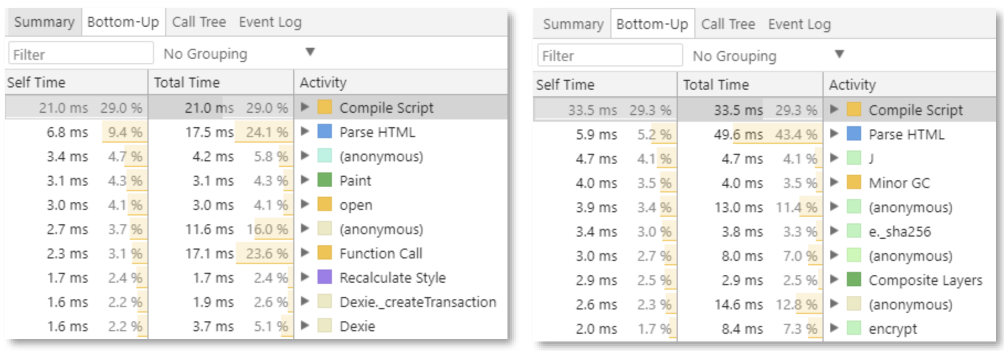
Range: 0 – 3.16 s

3162 ms

5.9 ms	🟦 Loading
100.8 ms	🟨 Scripting
2.7 ms	🟪 Rendering
4.7 ms	🟩 Painting
56.5 ms	⬜ Other
2991.7 ms	☐ Idle

Figure 26: Summary of the page load left plain text entries, right encrypted entries

As shown in (figure 26) applying the encryption puts an extra 56 ms load in the whole process of storing and reading database entry plus the parsing and running the whole script.


Summary | Bottom-Up | Call Tree | Event Log — Filter — No Grouping

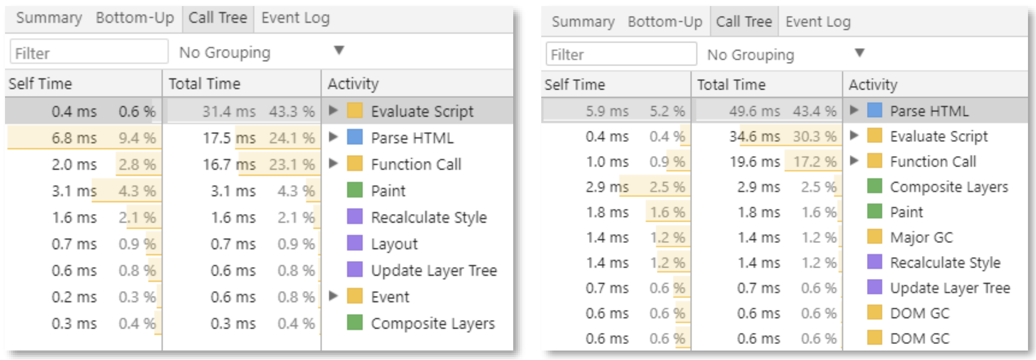
Self Time		Total Time		Activity
21.0 ms	29.0 %	21.0 ms	29.0 %	Compile Script
6.8 ms	9.4 %	17.5 ms	24.1 %	Parse HTML
3.4 ms	4.7 %	4.2 ms	5.8 %	(anonymous)
3.1 ms	4.3 %	3.1 ms	4.3 %	Paint
3.0 ms	4.1 %	3.0 ms	4.1 %	open
2.7 ms	3.7 %	11.6 ms	16.0 %	(anonymous)
2.3 ms	3.1 %	17.1 ms	23.6 %	Function Call
1.7 ms	2.4 %	1.7 ms	2.4 %	Recalculate Style
1.6 ms	2.2 %	1.9 ms	2.6 %	Dexie._createTransaction
1.6 ms	2.2 %	3.7 ms	5.1 %	Dexie

Summary | Bottom-Up | Call Tree | Event Log — Filter — No Grouping

Self Time		Total Time		Activity
33.5 ms	29.3 %	33.5 ms	29.3 %	Compile Script
5.9 ms	5.2 %	49.6 ms	43.4 %	Parse HTML
4.7 ms	4.1 %	4.7 ms	4.1 %	J
4.0 ms	3.5 %	4.0 ms	3.5 %	Minor GC
3.9 ms	3.4 %	13.0 ms	11.4 %	(anonymous)
3.4 ms	3.0 %	3.8 ms	3.3 %	e._sha256
3.0 ms	2.7 %	8.0 ms	7.0 %	(anonymous)
2.9 ms	2.5 %	2.9 ms	2.5 %	Composite Layers
2.6 ms	2.3 %	14.6 ms	12.8 %	(anonymous)
2.0 ms	1.7 %	8.4 ms	7.3 %	encrypt

Figure 27: Top 10 Bottom-UP in the performance test, left plain text entries, right encrypted entries

The Bottom-Up tab shows those activities "where the most time was directly spent." [65]


Summary | Bottom-Up | Call Tree | Event Log — Filter — No Grouping

Self Time		Total Time		Activity
0.4 ms	0.6 %	31.4 ms	43.3 %	Evaluate Script
6.8 ms	9.4 %	17.5 ms	24.1 %	Parse HTML
2.0 ms	2.8 %	16.7 ms	23.1 %	Function Call
3.1 ms	4.3 %	3.1 ms	4.3 %	Paint
1.6 ms	2.1 %	1.6 ms	2.1 %	Recalculate Style
0.7 ms	0.9 %	0.7 ms	0.9 %	Layout
0.6 ms	0.8 %	0.6 ms	0.8 %	Update Layer Tree
0.2 ms	0.3 %	0.6 ms	0.8 %	Event
0.3 ms	0.4 %	0.3 ms	0.4 %	Composite Layers

Summary | Bottom-Up | Call Tree | Event Log — Filter — No Grouping

Self Time		Total Time		Activity
5.9 ms	5.2 %	49.6 ms	43.4 %	Parse HTML
0.4 ms	0.4 %	34.6 ms	30.3 %	Evaluate Script
1.0 ms	0.9 %	19.6 ms	17.2 %	Function Call
2.9 ms	2.5 %	2.9 ms	2.5 %	Composite Layers
1.8 ms	1.6 %	1.8 ms	1.6 %	Paint
1.4 ms	1.2 %	1.4 ms	1.2 %	Major GC
1.4 ms	1.2 %	1.4 ms	1.2 %	Recalculate Style
0.7 ms	0.6 %	0.7 ms	0.6 %	Update Layer Tree
0.6 ms	0.6 %	0.6 ms	0.6 %	DOM GC
0.6 ms	0.6 %	0.6 ms	0.6 %	DOM GC

Figure 28: Top 10 Call Tree in the performance test, left plain text entries, right encrypted entries

The Call Tree tab illustrates "the root activities that cause the most work." Root activities are those makes the browser to do something such as execution of the handler. [65]

Most of the performance checks and benchmarks concepts are applicable to SQL databases. One of the most famous ones is checking the performance of the CRUD (create, read, update, and delete) on the database. IndexedDB is a NoSQL database, and it has its own way of doing the database transactions, therefore, performing the CRUD and checking the performance was not possible. Furthermore, performance analysis in details needs another research to be done.

The performance check of the proposed solution is a bit challenging because not only the speed of database transaction should be monitored but also the script running time. Therefore, the performance experiment is based on writing and reading the database entries in a loop. The time has been measured before starting and after finishing the loop.

The sample entries are including "User + i" and "Password + i" which i is the counter of the loop.

Table 4: Write and read records time in milliseconds

Test No.	Number of the records	Plain text (ms)	Proposed solution (ms)
1	1	2	5
2	100	5	53
3	500	14	109
4	1000	21	202
5	5000	87	515
6	10000	126	1185
7	50000	554	4760
8	100000	956	9008
9	500000	6350	66187

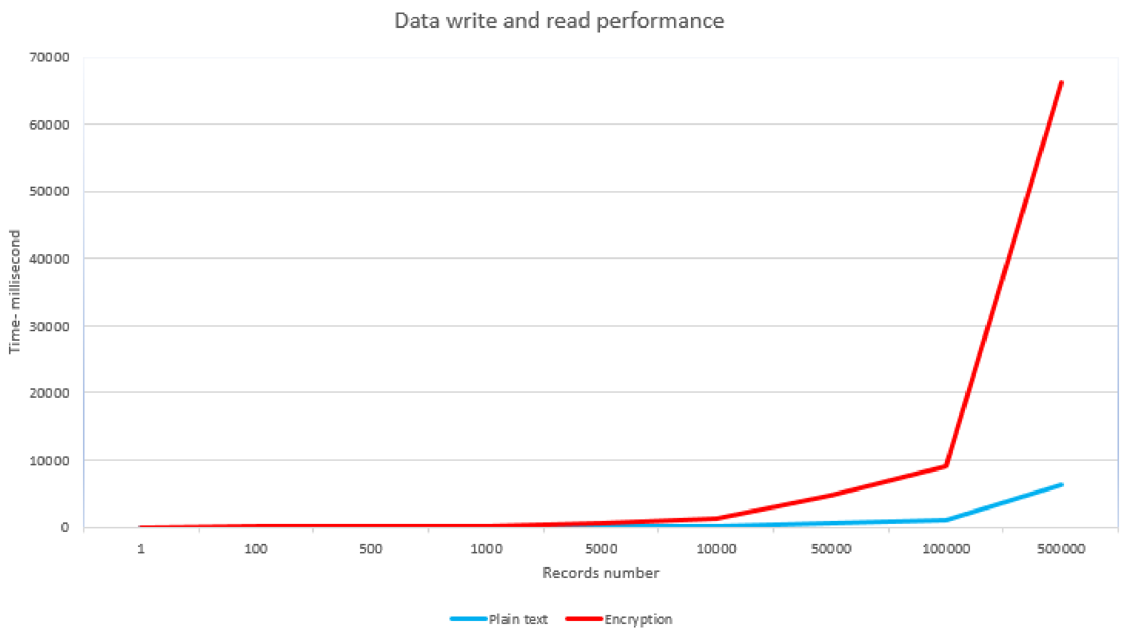


Figure 29: Data write and read performance benchmark

As it was anticipated, the proposed solution has an extra overhead on database transactions. However, the question of how optimized is this solution remains unanswered since comparing the performance with other solutions is not possible due to unavailability of them.

5. Final discussion

This part discusses the last thoughts and future ideas.

5.1. Conclusion

There is no general answer to the primary question "*How to protect the user's information in the IndexedDB API, against a different type of attacks which triggers to access to the user's data in plain-text which ends with sensitive data exposure?"*. However. This research has shown with the help of applying an extra layer of security such as validating the input, checking the secure connection and applying the cryptography, it is possible to add extra security to IndexedDB. The research had plenty options for applying cryptography, but most of the options can be used in the server-side and not the client-side. For online application it uses private and public key pairing way. The private key is stored on the server side and each time user logins it fetches the key from the server. For offline application generating the key based on the passphrase or only using the passphrase being used. Even though the W3C recommendation is to store the keys in IndexedDB, the risk of the data loss remains unanswered if user or developer decides to remove the database which key has been stored there. [66]

The idea behind of the research was to understand whether it is possible to protect the user's data in client-side storage IndexedDB and how it could be accomplished to protect the user's information against technical and non-technical attacks which leads to illegal access to the sensitive or insensitive information. Even accessing the insensitive information could help an adversary to correlate exposed information to establish more complex attacks.

The only earlier work did not focus on the cross-platform (cross-browser) without installation of any extension and also did not support both offline and online applications. This research would be a significant solution which can support different browsers and both offline and online usage. In verification part, it has been shown that the database contents are storing in the user's machine in a secure way. Often applying security leads to having an overload on the existing system. However, by using the proposed security platform, securing the database entries happened with a little overload on the performance. Due to nature of the client-side application, there are still some risks which cannot be mitigated by not only this framework but also any other available solutions, such as abusing the pathname to get unauthorized access to the database contents. The ideal situation for all data which stored on the client-side would be applying complete security layer on it, giving the example of how https (HTTP secure over TLS or SSL) was first an option but nowadays it is a must for the websites. In the final word, every single effort matters when there is no standard way to protect user's data in this database. This work could consider as an example for developers as well as other researchers to get some ideas, use it, expand it, and finally have better security in their applications.

5.2. Future work

The future work for the proposed solution is to write a complete security wrapper library for IndexedDB which can handle different JavaScript crypto libraries. Considering use of other types of authentication and security elements such as eID (the best example would be Estonian ID card) could be another possibility for further research. In the end, considering the optimization, the solution could use different JavaScript crypto libraries and compare the results. A current active research which can consider as future work is writing a script as an extension for Burp Suite[1] to evaluate IndexedDB security by this scanner.

[1] https://portswigger.net/burp

References

[1] "W3C," [Online]. Available: https://www.w3.org/html/. [Accessed March 2018].

[2] "HTML5," Mozilla MSN Web Docs, [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5. [Accessed March 2018].

[3] R. K. Jain and H. C. Triandis, Management of research and development organizations: managing the unmanageable, vol. 27, John Wiley & Sons, 1997.

[4] A. Newell and H. A. Simon, Computer science as empirical inquiry: Symbols and search, ACM, 2007.

[5] H. Hassani, "Research Methods in Computer Science: The Challenges and Issues," *arXiv preprint arXiv:1703.04080,* 2017.

[6] T. X. Bui, "Decision support systems for sustainable development," in *Decision Support Systems for Sustainable Development*, Springer, 2002, pp. 1-10.

[7] "IDB-Encryption," Github, 25 October 2016. [Online]. Available: https://github.com/stefankim/IDB-Encryption/blob/master/README.md. [Accessed April 2018].

[8] "Mobile and tablet internet usage exceeds desktop for first time worldwide," StatCounter GlobalStats, 1 November 2016. [Online]. Available: http://gs.statcounter.com/press/mobile-and-tablet-internet-usage-exceeds-desktop-for-first-time-worldwide. [Accessed March 2018].

[9] P. M. Carey, New Perspectives HTML5 and CSS3 7th edition, Boston: Cengage Learning, 2017.

[10] W3C, "HTML Introduction," W3C, [Online]. Available: https://www.w3schools.com/html/html_intro.asp. [Accessed March 2018].

[11] S. Kedar, Database Management Systems, Technical Publications, 2009.

[12] "Tables, rows, and columns," IBM, [Online]. Available: https://www.ibm.com/support/knowledgecenter/SSPK3V_6.3.0/com.ibm.swg.im.soliddb.sql.doc/doc/tables.rows.and.columns.html. [Accessed April 2018].

[13] J. Lourenço, B. Cabral, J. Bernardino and M. Vieira, "Comparing NoSQL Databases with a Relational Database: Performance and Space," vol. 2, pp. 1-14, 9 2015.

[14] P. Atzeni, F. Bugiotti and L. Rossi, "Uniform access to NoSQL systems," *Information Systems,* vol. 43, pp. 117-133, 2014.

[15] E. Team, "Most popular databases in 2017 according to StackOverflow survey," EverSQL, 26 June 2017. [Online]. Available: https://www.eversql.com/most-popular-databases-in-2017-according-to-stackoverflow-survey/. [Accessed March 2018].

[16] B. G. Tudorica and B. Cristian, *A comparison between several NoSQL databases with comments and notes,* 2011, pp. 1-5.

[17] P. Emmons, "5 Reasons to Switch to a NoSQL Database," DragonSpears, 25 May 2016. [Online]. Available: https://www.dragonspears.com/blog/5-reasons-to-switch-to-a-nosql-database. [Accessed March 2018].

[18] A. Nayak, A. Poriya and D. Poojary, "Type of NOSQL databases and its comparison with relational databases," *International Journal of Applied Information Systems,* vol. 5, pp. 16-19, 2013.

[19] "Client-side storage," Mozilla, [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Client-side_storage. [Accessed March 2018].

[20] E. Commission, "Cookies," European Commission, [Online]. Available: http://ec.europa.eu/ipg/basics/legal/cookies/index_en.htm. [Accessed April 2018].

[21] S. Kimak and J. Ellman, "The role of HTML5 IndexedDB, the past, present and future," in *Internet Technology and Secured Transactions (ICITST), 2015 10th International Conference for,* 2015.

[22] G. Buja, K. B. A. Jalil, F. B. H. M. Ali and T. F. A. Rahman, "Detection model for SQL injection attack: An approach for preventing a web application from the SQL injection attack," in *Computer Applications and Industrial Electronics (ISCAIE), 2014 IEEE Symposium on,* 2014.

[23] Z. Kessin, Programming HTML5 Applications: Building Powerful Cross-Platform Environments in JavaScript, O'Reilly Media, 2011.

[24] M. MacDonald, HTML5: The Missing Manual (1st ed.), O'Reilly Media, 2011.

[25] G. Keizer, "How Apple, Google, Microsoft and Mozilla will eliminate Adobe Flash," Computerworld, 31 July 2017. [Online]. Available: https://www.computerworld.com/article/3211437/web-browsers/faq-how-apple-google-microsoft-and-mozilla-will-eliminate-adobe-flash.html. [Accessed April 2018].

[26] A. Boodman, "Stopping the Gears," Google, 03 11 2011. [Online]. Available: http://gearsblog.blogspot.de/2011/03/stopping-gears.html. [Accessed April 2018].

[27] W3C, "Web SQL Database," W3C, 18 November 2018. [Online]. Available: https://www.w3.org/TR/webdatabase/. [Accessed April 2018].

[28] C. Buckler, "HTML5 Browser Storage: the Past, Present and Future," SitePoint, 09 October 2013. [Online]. Available: https://www.sitepoint.com/html5-browser-storage-past-present-future/. [Accessed April 2018].

[29] sideshowbarker, "File," Mozilla, 02 March 2018. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/File. [Accessed April 2018].

[30] W. W. Draft, "File API," W3C, 26 October 2017. [Online]. Available: https://www.w3.org/TR/FileAPI/#intro. [Accessed April 2018].

[31] H. demos, "File API," HTML5 demos, [Online]. Available: https://html5demos.com/file-api/. [Accessed April 2018].

[32] W3schools, "HTML5 Web Storage," W3C, [Online]. Available: https://www.w3schools.com/html/html5_webstorage.asp. [Accessed April 2018].

[33] M. W. Docs, "IndexedDB API Basic concepts," Mozilla, [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API/Basic_Concepts_Behind_IndexedDB. [Accessed April 2018].

[34] MDN, "The structured clone algorithm," Mozilla, [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Structured_clone_algorithm. [Accessed April 2018].

[35] C. I. use, "Can I use IndexedDB," Can I use, [Online]. Available: https://caniuse.com/#search=indexedb. [Accessed April 2018].

[36] M. Contributors, "Using IndexedDB," Mozilla, [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API/Using_IndexedDB. [Accessed April 2018].

[37] M. K. Swain, "Html5 IndexedDB," The Code Project, [Online]. Available: https://www.codeproject.com/Articles/757939/Html-IndexedDB. [Accessed April 2018].

[38] W3C, "Indexed Database API 2.0," W3C, [Online]. Available: https://www.w3.org/TR/IndexedDB-2/#persistence-risks. [Accessed April 2018].

[39] MDN, "Same-origin policy," MDN, [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy. [Accessed April 2018].

[40] S. Kimak and J. Ellman, "HTML5 IndexedDB Encryption: Prevention against Potential Attacks," *International Journal of Intelligent Computing Research*, vol. 6, pp. 621-630, 2015.

[41] MDN, "Cross-Origin Resource Sharing (CORS)," Mozilla, [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS. [Accessed April 2018].

[42] OWASP, "CORS OriginHeaderScrutiny," OWASP, 16 08 2016. [Online]. Available: https://www.owasp.org/index.php/CORS_OriginHeaderScrutiny. [Accessed April 2018].

[43] OWASP, "Cross-site Scripting (XSS)," OWASP, 06 03 2018. [Online]. Available: https://www.owasp.org/index.php/Cross-site_Scripting_(XSS). [Accessed April 2018].

[44] OWASP, "DOM Based XSS," OWASP, [Online]. Available: https://www.owasp.org/index.php/DOM_Based_XSS. [Accessed April 2018].

[45] MDN, "Cross-site scripting," Mozilla, [Online]. Available: https://developer.mozilla.org/en-US/docs/Glossary/Cross-site_scripting. [Accessed April 2018].

[46] A. Burlacu, "LinkedIn Bug Could Have Compromised Users' Personal Data, Including Emails And Phone Numbers," Tech Times, 21 April 2018. [Online]. Available: http://www.techtimes.com/articles/225782/20180421/linkedin-bug-could-have-compromised-users-personal-data-including-emails-and-phone-numbers.htm. [Accessed April 2018].

[47] S. C. S. Lab, "Stanford Javascript Crypto Library," Stanford Computer Security Lab, [Online]. Available: https://bitwiseshiftleft.github.io/sjcl/. [Accessed April 2018].

[48] R. E. Morse, P. Nadkarni, D. A. Schoenfeld and D. M. Finkelstein, "Web-browser encryption of personal health information," *BMC medical informatics and decision making,* vol. 11, p. 70, 2011.

[49] A. Parecki, "The OAuth 2.0 authorization framework," The OAuth 2.0, [Online]. Available: https://oauth.net/. [Accessed April 2018].

[50] O. S. Foundation, "Welcome to OpenSSL!," OpenSSL Software Foundation, [Online]. Available: https://www.openssl.org/. [Accessed April 2018].

[51] W. D. a. J. Walton, "Crypto++® Library 6.1," Crypto++ community, 22 01 2018. [Online]. Available: https://www.cryptopp.com/. [Accessed April 2018].

[52] S. Kimak, J. Ellman and C. Laing, "Some potential issues with the security of HTML5 indexedDB," 2014.

[53] J. Andress, The Basics of Information Security second edition, Syngress, 2014.

[54] I. O. f. Standardization, "ISO/IEC 27000," [Online]. Available: http://standards.iso.org/ittf/PubliclyAvailableStandards/c041933_ISO_IEC_27000_2009.zip. [Accessed April 2018].

[55] I. Jacobson, Object-oriented software engineering: a use case driven approach, Pearson Education India, 1993.

[56] G. Sindre and A. L. Opdahl, "Eliciting security requirements with misuse cases," *Requirements engineering,* vol. 10, pp. 34-44, 2005.

[57] R. Matulevicius, "Unpublished Draft," in *Fundamentals of Secure System Modelling*, Springer International Publishing, 2017, pp. 105-116.

[58] E. Stark, M. Hamburg and D. Boneh, "Symmetric cryptography in javascript," in *Computer Security Applications Conference, 2009. ACSAC'09. Annual*, 2009.

[59] M. H. a. D. B. Emily Stark, "https://bitwiseshiftleft.github.io/sjcl/demo/," Stanford Computer Security Lab, [Online]. Available: https://bitwiseshiftleft.github.io/sjcl/demo/. [Accessed April 2018].

[60] "SJCL browser test," Stanford Computer Security Lab, [Online]. Available: https://bitwiseshiftleft.github.io/sjcl/browserTest/. [Accessed April 2018].

[61] OWASP, "Input Validation Cheat Sheet," OWASP, [Online]. Available: https://www.owasp.org/index.php/Input_Validation_Cheat_Sheet. [Accessed April 2018].

[62] T. Scholte, W. Robertson, D. Balzarotti and E. Kirda, "Preventing input validation vulnerabilities in web applications through automated type analysis,"

in *Computer Software and Applications Conference (COMPSAC), 2012 IEEE 36th Annual*, 2012.

[63] MDN, "Input value length," Mozilla, [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/text. [Accessed April 2018].

[64] D. Fahlander, "Dexie.js," Dexie.js , [Online]. Available: http://dexie.org/. [Accessed April 2018].

[65] K. Basques, "Tools for Web Developers - Performance Analysis Reference," Google, [Online]. Available: Performance Analysis Reference. [Accessed April 2018].

[66] W3C, "Key Storage," W3C, [Online]. Available: https://www.w3.org/TR/WebCryptoAPI/. [Accessed April 2018].

[67] W3C, "JavaScript HTML DOM," W3C, [Online]. Available: https://www.w3schools.com/js/js_htmldom.asp. [Accessed April 2018].

[68] M. Rosica, "Example of authentication between client(js) and server(php)," [Online]. Available: http://cryptojs.altervista.org/js-php/. [Accessed April 2018].

Appendix I – Assay on HTML5 IndexedDB API security

HTML5 Local Storage IndexedDB Security

Amirhossein Akbari
Department of information Technology
Tallinn University of Technology
Tallinn, Estonia
akbari6@outlook.com

***Abstract*— In recent years, the call for killing browsers extension or add-ons such as Adobe flash player and Java plugins is arising. In the most cases, browser attacks happened due to the vulnerability of extensions. HTML5 is considered the best replacement for all extensions and browsers are utterly desperate for removing these extensions from their application and move to HTML5 instead. This made HTML5 an exciting place for Hackers as well. Furthermore, HTML5 changed the traditional way of storing data just only on server-side. By this developer can decide based on the performance and other measurements to put this load on the client-side. Hence there is a notable security concern in this field.**

Keywords*— Web Secuirty, HTML5, cross-platform, Local Sotrage, Client-Side, IndexedDB,eCoomerce, Cookie, Encyption, W3C(Web Consortium)

I. Introduction

HTML is a markup language for describing web documents. It is a set of markup tags. HTML tags describe various document content. [1] After almost three decades of the invention of HTML by Tim Berners-Lee, HTML is still the main language of the web. [2]

HTML5 is the last version of HTML which developers using for making web pages. HTML5.1 is a new working draft of HTML5. HTML5 introduces new features such as new attributes, messaging enhancements, and other important compatibility features for mobile phone browsers. [3] Besides of patching the bugs and auditing, most browser companies claim that they will not support Adobe and Java plug-ins by the end of 2016 due to the vulnerability of these plug-ins. [4], [5]

Surprisingly HTML5 is not only the simple markup language anymore however it is a combination of different components such as XMLHttpRequest (XHR), Document Object Model (DOM), Cross-Origin Resource Sharing (CORS), enhanced HTML/Browser rendering, localstorage, webSQL, websocket, webworkers, enhanced XHR, DOM. [6] These new features of HTML5 are interesting enough for attackers to start to investigate security issues and abuse them.

The process of HTML5 development was more than seven years from the first public working draft in 2008 to complete the standardization of it. So, that majority of security issues which published during these years are already solved, but still, some parts remain without an entirely proper countermeasure.

Storing data in local machine database or client side is still one of a remained issue since all data is saving in plain text.

II. Background

A. Problem domain

IndexedDB is a plain text client-side storage. Correlation of user saved data could be triggers to identify theft.

Considering the mobile usage of IndexedDB, data is saved in mobile internal memory. In case of losing the device or theft, retrieving removed files is possible. This can lead to data exposure. [7]

B. Storing data on the client-side

Reasons for using client-side for saving the data are different:

- More responsive – Increasing the performance
- Offline-mode usage- Increasing Accessibility
- Make website usable when connection lost
- Reduce server-side load
- Reduce bandwidth usage

Even with the best Internet connection still saving and retrieving data on local machine is more responsive and faster. In October 2016, mobile Internet usage exceeds Desktop hence websites should consider their design with mobile usability. [8] This trigger to having website with capability to responds to user needs even in offline mode in case of losing connection for a short period of times.

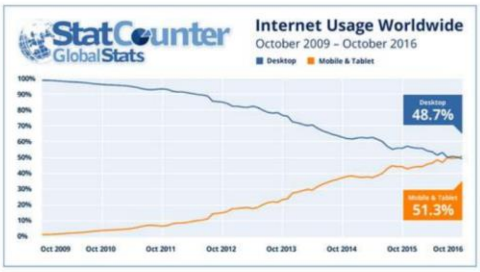


Figure 1 – Interner Usage Worldwide [8]

Another advantage of using HTML5 itself is, there is no need for installation of any extra application and user can just visit the website and run the application on their device's browsers. [9]

C. *Different type of client-side storages*

There are various technologies available for storing the data in client side. From simple JavaScript variables, Cookies and HTML5 Web SQL to HTML5 web storage, HTML5 file API and HTML5 Indexed database.

JavaScript variables are not the proper way of saving user's data since they will wipe by closing the tabs or refreshing the page. Cookies are still widely using in websites. Some cons such as: storing data as strings only, limitation in storage and all other Internet privacy issues around them still exist.

Web SQL database was deprecated from 2010. [10]

Table 1 is showing the security considerations for other three client-side storages which are recommended by W3C (Web consortium).

Candidate APIs by W3C	**Security Consideration by W3C**			
	DNS spoofing attacks	Cross-directory attacks	Preventing selection looping	System-sensitive files
Web Storage	✓	✓	-	-
File API	-	-	✓	✓
IndexedDB	✓	✓	-	-

Table 1 – Security Considerations by W3C on client-side

None of these technologies has an option for data encryption by default. Social engineering and physical access to the user machine could be potential attacks in this scope. [7]

D. *Related work*

Currently HTML5.1 draft is ongoing. Many research and analysis about HTML5 are happening every day.

With the focus of client-side storage, research on web storage has been published. Some existing security prevention like Transport Layer Security(TLS), input validation and client-side encryption proposed for mitigating of the possible attacks. [22] Since there are a few works published about IndexedDB itself which will cover in more details in security countermeasures section.

III. INDEXEDDB

Indexed Database also is known as IndexedDB API is currently recommended, W3C candidate. [11] IndexedDB is storing data in plain text with key-value pairs. It is NoSQL (not only SQL) [12], "asynchronous, browser-based data store". It provides "fast access to unlimited amount of data". These features make IndexedDB as a suitable data store on client-side. However, lack of encryption and ability to retrieve deleted data are the main issue with this innovative technology. [13]

IndexedDB restricts data to the single origin which could be scheme, host, port, or domain. This helps browser to implement sandboxed and avoid accessing data by other origins. [11], [14]

Lack of automatic Internationalized sorting, synchronizing with server-side database and full-text searching are some drawbacks of IndexedDB. [21]

The basic pattern for using IndexedDB is first to open a database, create an object store in a database, start transactions like adding or retrieving data, waiting for the operation to complete (it is happening in the background because it is asynchronous) and finally use the result. [15]

A. *Importance of IndexedDB*

The main motivation behind of designing IndexedDB is "to perform advanced key-value data management." [11]

Besides of bringing extra features for client-side storage, eCommerce and Cookies which are IndexedDB's "intellectual antecedents" are main points of developing IndexedDB. [13]

considering eCommerce brings many benefits over offline stores such as: terminating of the third party "middleman cost, lower operational costs, providing search functionality, browsing through large amounts of products "and .... [13]

Cookies are strings for saving user information in web pages. Cookies were invented to "remember information about the user." They are limited to 4KB. [16] They widely used in eCommerce and online marketing. There are some security problems related to cookies. One the possibilities could be stealing user identity and bypass the security with Cookie poisoning attacks. [17] Another possible attack would be Cookie Injection attacks which may cause SQL injection attacks. [18] Also regarding privacy, Cookie law legislate by EU directive to make users aware and agree with storing and retrieving data on a user's machine or device. [19]

Limitation in size and flexibility (storing all data as strings) of Cookies have motivated developing IndexedDB. [13]

B. *IndexedDB potential attacks*

HTML5 has been analyzed for different attacks and exploits over a period of standardization. S. Shah demonstrates top ten "attack surface and possible threats in details. S. Shah stated that security countermeasures are needed during the time to overcome these issues. [20]

IndexedDB is one of those new features of HTML5 with its security concerns. Cross-origin resource sharing (CORS), Cross-site scripting(XSS), social engineering and physical access are potential attacks on IndexedDB. [13]

IV. SECURITY COUNTERMEASURES

A few research has done about IndexedDB security. All four-related works will discuss in detail in this section.

First, a theoretical framework proposed in the development of IndexedDB. It divided into various parts such as [21]

- Client-side data encryption
- Code analysis
- Input validation
- SOP (same-origin policy)

The framework will be as a browser extension. Using JavaScript encryption which is based on EAS or SHA-256 with verification hash. [23] Code Analysis consist of static and dynamic analysis. "Input and output validation" have done with taking strings and return the proper value. [21]

Second, the analysis showed that deleted entries just "marked as deleted" and it is possible to retrieve the data from database after deleting them. To solve the problem, this solution is proposed, in writing phase: [24]

1. Ensuring secure connection is established
2. Open a connection to the database
3. Encryption with a library which generates a public and a private key
4. Saving the file and closing the connection

In reading phase:

1. Checking user Credentials
2. Get the key
3. Decrypt data
4. Show data
5. Close the connection

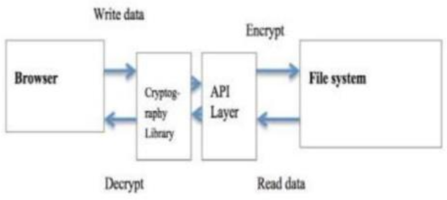


Figure 2 – Proposed Encryption library [24]

Authentication have done using by OAuth [25], OpenSSL [26], Crypto++ [27] crypto libraries for encryption. [24]

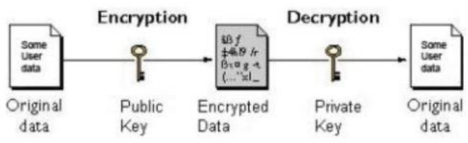


Figure 3 - Encryption and Decryption using keys [24]

Third, the proposal security model consists layers which means even if "authentication process comprised" it is not possible to read the data. The model is using hashing and encryption by Stanford JavaScript Library [28] as an extension of the browser. It "gets a secure login, encrypt the data, store public and private key, decryption of data, and finally secure deletion." [13]

Evaluation of security model had done by performing XSS attack scenario before and after adding security model. [13]

Fourth, the research includes of reviewing the role of IndexedDB in the past, present and future. Stating the role of Cookies, eCommerce, and mobile eCommerce as motivation for the development of IndexedDB in the past. Current situation of IndexedDB has been analyzed by a different point of views such as Improving "Cookies functionality," usage in mobile devices and offline-usage. The future of IndexedDB showed as not a suitable storage for storing personal information. Besides of encryption using multifactor authentication (MFA) or two-factor authentication (2FA) proposed as solutions. [29]

V. CONCLUSION

Apart from all new features which HTML5 is providing, it has some potential vulnerabilities. Main advantages of HTML5 are being a cross-platform and ability to run in browsers without installing the extra application efficiently. Storing data into Client-side storage brings better performance, increases availability and accessibility, reducing bandwidth usage and server-side load. IndexedDB API is suffering from not having enough security consideration. Protecting user's security and privacy is possible with the implementation of encryption, input validation and applying additional security policies on SOP.

REFERENCES

[1] 'HTML Introduction'. Available: http://www.w3schools.com/html/html_intro.asp Accessed [25.11.2016]

[2] 'A history of HTML', Available: https://www.w3.org/People/Raggett/book4/ch02.htm, Accessed [25.11.2016]

[3] 'HTML5 Working Draft' Available: https://www.w3.org/TR/html/introduction.html Accessed [25.11.2016]

[4] 'NPAPI Plugins in Firefox', Available: https://blog.mozilla.org/futurereleases/2015/10/08/npapi-plugins-in-firefox/ Accessed on [25.11.2016]

[5] 'Moving to a Plugin-Free' Available: https://blogs.oracle.com/java-platform-group/entry/moving_to_a_plugin_free Accessed [25.11.2016]

[6] 'HTML5 Differences from HTML4', Available: https://www.w3.org/TR/html5-diff/ Accessed [25.11.2016]

[7] Stefan Kimak, Jeremy Ellman, "Some potential issues with the security of HTML5 indexedDB", Internet Technology and Secured Transactions (ICITST) 2015 10th International Conference for, pp. 379-383, 2015

[8] 'Mobile and tablet internet usage exceeds desktop for first time worldwide' Available: http://gs.statcounter.com/press/mobile-and-tablet-

internet-usage-exceeds-desktop-for-first-time-worldwide Accessed [25.11.2016]

[9] Harjono, J. Ng, G. Kong, D. Lo, J. (2011) Building smarter web applications with HTML5. Conference of the Center for Advanced Studies on Collaborative Research

[10] 'Web SQL Database' Available: https://www.w3.org/TR/webdatabase/ Accessed [25.11.2016]

[11] Ali Alabbas, Joshua Bell, 'Indexed Database API 2.0', Available: https://w3c.github.io/IndexedDB/ Accessed [25.11.2016]

[12] Strozzi, C. (1998) NoSQL A Relational Database Management System. Available: http://www. strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/Home% Accessed [25.11.2016]

[13] Stefan Kimak, Jeremy Ellman, 'HTML5 IndexedDB Encryption: Prevention against Potential Attacks.' - International Journal of Intelligent Computing Research (IJICR), Volume 6, Issue 4, December 2015

[14] 'HTML Living Standard — Last Updated 26 November 2016' Available: https://html.spec.whatwg.org/multipage/browsers.html#concept-origin Accessed [26.11.2016]

[15] 'Using IndexedDB' Available: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API/Using_IndexedDB Accessed [26.11.2016]

[16] 'JavaScript Cookies', Available: http://www.w3schools.com/js/js_cookies.asp Accessed on [26.11.2016]

[17] Buja, G., Jalil, K. B. A., Ali, F. B., Mohd, H., & Rahman, T. F. A. (2014). Detection model for SQL injection attack: An approach for preventing a web application from the SQL injection attack. In Computer Applications and Industrial Electronics (ISCAIE), 2014 IEEE Symposium on (pp. 60-64). IEEE

[18] Appelt, D., Nguyen, C. D., Briand, L. C., & Alshahwan, N. (2014). Automated testing for SQL injection Vulnerabilities: An input mutation approach. In Proceedings of the 2014 International Symposium on Software Testing and Analysis (pp. 259-269). ACM

[19] 'Cookies, The EU Internet Handbook', Available: http://ec.europa.eu/ipg/basics/legal/cookies/index_en.htm Accessed [26.11.2016]

[20] Shah, Shreeraj. "HTML5 Top 10 Threats Stealth Attacks and Silent Exploits." BlackHat Europe (2012).

[21] Stefan Kimak, Jeremy Ellman, and Christopher Laing. "An investigation into possible attacks on html5 indexeddb and their prevention." The 13th Annual Post-Graduate Symposium on The Convergence of Telecommunications, Networking and Broadcasting (PGNet 2012), Liverpool, UK. 2012.

[22] West, W. and Pulimood, M. (2012) ANALYSIS OF PRIVACY AND SECURITY IN HTML5 WEB STORAGE. ACM digital library. Journal of Computing Sciences in College. Volume 27 Issue 3

[23] Morse, R. Nadkarni , P. Schoenfeld, D. Finkelstein, D. (2011) WebBrowser encryption for personal health information. BMC Medical Informatics and Decision Making Volume 11

[24] Stefan Kimak, Jeremy Ellman, "Some potential issues with the security of HTML5 indexedDB", Internet Technology and Secured Transactions (ICITST) 2015 10th International Conference for, pp. 379-383, 2015.

[25] 'OAuth 2.0' Available: https://oauth.net/about/ Accessed: [27.11.2016]

[26] Engelschall, R. S. (1999). About the OpenSSL Project Available: https://www.openssl.org Accessed: [27.11.2016]

[27] Dai, W. (2004) Cryptoo++ Library. Available: http://www.cryptopp.com Accessed: [27.11.2016]

[28] 'The Stanford Javascript Crypto Library', Available: http://bitwiseshiftleft.github.io/sjcl/ Accessed: [27.11.2016]

[29] S. Kimak and J. Ellman, "The role of HTML5 IndexedDB, the past, present and future," 2015 10th International Conference for Internet Technology and Secured Transactions (ICITST), London, 2015, pp. 379-383.

Appendix II – sjcl demo source code **[59]**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Strict//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>SJCL demo</title>
  <link rel="stylesheet" type="text/css"
href="example.css"/>
  <script type="text/javascript"
src="../sjcl.js"></script>
  <script type="text/javascript"
src="form.js"></script>
  <script type="text/javascript"
src="example.js"></script>
</head>
<body onload="loaded()">
  <h1>SJCL demo</h1>

  <div class="header">
  <p>This page is a demo of the Stanford Javascript
Crypto Library. To get started, just type in a
password in the left pane and a secret message in
the middle pane, then click "encrypt". Encryption
takes place in your browser and we never see the
plaintext.</p>

  <p>SJCL has lots of other options, many of which
are shown in the grey boxes.</p>
  </div>

  <form id="theForm" onsubmit="return false;">
  <div class="column" id="ckey">
    <!-- Password and pbkdf2 parameters -->
    <div class="box" id="ppassword">
      <h2>Password</h2>
      <div class="section">
        <label for="password">Password:</label>
        <input type="password" class="wide"
name="password" id="password" autocomplete="off"
tabindex="1"/>
        <p class="explanation">
          Choose a strong, random password.
        </p>
      </div>
    </div>
    <div class="box" id="pkey">
      <h2>Key Derivation</h2>
      <div class="section">
        <div>
          <label for="salt"">Salt:</label>
          <a class="random floatright"
href="javascript:randomize('salt',2,0)">random</a>
        </div>
        <input type="text" id="salt" class="wide
hex" autocomplete="off" size="17" maxlength="35"/>
        <input type="checkbox" name="freshsalt"
id="freshsalt" autocomplete="off"
checked="checked"/>
        <label for="freshsalt">Use fresh random
salt for each new password</label>
        <p class="explanation">
          Salt adds more variability to your key,
and prevents attackers
          from using <a
href="http://en.wikipedia.org/wiki/Rainbow_table">r
ainbow tables</a> to attack it.
        </p>
      </div>

      <div class="section">
        <label for="iter">Strengthen by a factor
of:</label>
        <input type="text" name="iter" id="iter"
value="1000" class="numeric" size="5" maxlength="5"
autocomplete="off"/>
        <p class="explanation">
          Strengthening makes it slower to compute
the key corresponding to your
          password.  This makes it take much longer
for an attacker to guess it.
        </p>
      </div>

      <div class="section">
        Key size:
        <input type="radio" name="keysize"
value="128" id="key128" checked="checked"
autocomplete="off" onclick="extendKey(4)"/>
        <label for="key128">128</label>
        <input type="radio" name="keysize"
value="192" id="key192" autocomplete="off"
onclick="extendKey(6)"/>
        <label for="key192">192</label>
        <input type="radio" name="keysize"
value="256" id="key256" autocomplete="off"
onclick="extendKey(8)"/>
        <label for="key256">256</label>
        <p class="explanation">
          128 bits should be secure enough, but you
can generate a longer
          key if you wish.
        </p>
      </div>

      <!-- cipher key -->
      <div class="section">
        <div>
          <label for="key">Key:</label>
          <!--
          <a class="random floatright"
href="javascript:randomizeKey()">random</a>
          -->
        </div>
        <textarea id="key" name="key" class="hex"
rows="2" autocomplete="off"></textarea>
        <p class="explanation">
          This key is computed from your password,
salt and strengthening factor.  It
          will be used internally by the cipher.
Instead of using a password, you can
          enter a key here directly.  If you do, it
should be 32, 48 or 64 hexadecimal
          digits (128, 192 or 256 bits).
        </p>
      </div>

    </div>
  </div>

    <!-- mode controls -->
  <div class="column" id="cmode">
    <div class="box">
      <h2>Cipher Parameters</h2>
      <p class="explanation">
        SJCL encrypts your data with the <a
href="http://en.wikipedia.org/wiki/Advanced_Encrypt
ion_Standard"><acronym title="Advanced Encryption
Standard">AES</acronym></a> block cipher.
      </p>
      <div class="section">
```

```
        Cipher mode:
        <input type="radio" name="mode" value="ccm"
id="ccm" checked="checked" autocomplete="off"/>
        <label for="ccm"><acronym title="Counter
mode with Cipher block chaining Message
authentication code">CCM</acronym></label>
        <input type="radio" name="mode"
value="ocb2" id="ocb2" autocomplete="off"/>
        <label for="ocb2"><acronym title="Offset
CodeBook mode, version 2.0">OCB2</acronym></label>
        <p class="explanation">
          The cipher mode is a standard for how to
use AES and other
          algorithms to encrypt and authenticate
your message.
          <a
href="http://en.wikipedia.org/wiki/OCB_mode">OCB2
mode</a>
          is slightly faster and has more features,
but
          <a
href="http://en.wikipedia.org/wiki/CCM_mode">CCM
mode</a> has wider
          support because it is not patented.
        </p>
      </div>


      <div class="section">
        <div>
          <label for="iv">Initialization
vector:</label>
          <a class="random floatright"
href="javascript:randomize('iv',4,0)">random</a>
        </div>
        <input type="text" name="iv" id="iv"
class="wide hex" size="32" maxlength="35"
autocomplete="off"/>
        <input type="checkbox" id="freshiv"
autocomplete="off" checked="checked"/>
        <label for="freshiv">Choose a new random IV
for every message.</label>
        <p class="explanation">
          The IV needs to be different for every
message you send.  It adds
          randomness to your message, so that the
same message will look
          different each time you send it.
        </p>
        <p class="explanation">
```

```
          Be careful: CCM mode doesn't use
          the whole IV, so changing just part of it
isn't enough.
        </p>
      </div>

      <div class="section">
        Authentication strength:
        <input type="radio" name="tag" value="64"
id="tag64" autocomplete="off" checked="checked"/>
        <label for="tag64">64</label>
        <input type="radio" name="tag" value="96"
id="tag96" autocomplete="off"/>
        <label for="tag96">96</label>
        <input type="radio" name="tag" value="128"
id="tag128" autocomplete="off"/>
        <label for="tag128">128</label>
        <p class="explanation">
          SJCL adds a an authentication tag to your
message to make sure
          nobody changes it.  The longer the
authentication tag, the harder it is
          for somebody to change your encrypted
message without you noticing.  64
          bits is probably enough.
        </p>
      </div>

      <div class="section">
        <input type="checkbox" name="json"
id="json" autocomplete="off" checked="checked"/>
        <label for="json">Send the parameters and
authenticated data along
          with the message.</label>
         <p class="explanation">
           These parameters are required to decrypt
your message later.  If the
           person you're sending the message to
knows them, you don't need to send
           them so your message will be shorter.
         </p>
         <p class="explanation">
           Default parameters won't be sent.  Your
password won't be sent, either.
           The salt and iv will be encoded in
base64 instead of hex, so they'll
           look different from what's in the box.
         </p>
      </div>
    </div>
  </div>

  <div class="column" id="ctexts">
    <div id="pplaintext" class="box">
      <h2>Plaintext</h2>
      <div class="section">
        <label for="plaintext">Secret
message:</label>
        <textarea id="plaintext" autocomplete="off"
rows="5" tabindex="2"></textarea>
        <div class="explanation">
          This message will be encrypted, so that
nobody can read it or change it
          without your password.
        </div>
      </div>

      <div class="section">
        <label for="adata">Authenticated
data:</label>
        <textarea id="adata" autocomplete="off"
tabindex="3"></textarea>
        <div class="explanation">
          This auxilliary message isn't secret, but
its integrity will be checked
          along with the integrity of the message.
        </div>
      </div>
    </div>

    <div id="buttons">
      <a href="javascript:doEncrypt()" id="encrypt"
tabindex="4"><span
class="turnDown">encrypt</span></a>
      <a href="javascript:doDecrypt()" id="decrypt"
tabindex="6"><span
class="turnUp">decrypt</span></a>
    </div>

    <div id="pciphertext" class="box">
      <h2>Ciphertext</h2>
      <label for="ciphertext">Ciphertext:</label>
      <textarea id="ciphertext" autocomplete="off"
rows="7" tabindex="5"></textarea>
      <div class="explanation">
      </div>
    </div>
  </div>

  <div class="column" id="ctexts">
    <div id="pplaintext" class="box">
      <h2>Plaintext</h2>
      <div class="section">
        <label for="plaintext">Secret
message:</label>
        <textarea id="plaintext" autocomplete="off"
rows="5" tabindex="2"></textarea>
        <div class="explanation">
          This message will be encrypted, so that
nobody can read it or change it
          without your password.
        </div>
      </div>

      <div class="section">
        <label for="adata">Authenticated
data:</label>
        <textarea id="adata" autocomplete="off"
tabindex="3"></textarea>
        <div class="explanation">
          This auxilliary message isn't secret, but
its integrity will be checked
          along with the integrity of the message.
        </div>
      </div>
    </div>

    <div id="buttons">
      <a href="javascript:doEncrypt()" id="encrypt"
tabindex="4"><span
class="turnDown">encrypt</span></a>
      <a href="javascript:doDecrypt()" id="decrypt"
tabindex="6"><span
class="turnUp">decrypt</span></a>
    </div>

    <div id="pciphertext" class="box">
      <h2>Ciphertext</h2>
      <label for="ciphertext">Ciphertext:</label>
      <textarea id="ciphertext" autocomplete="off"
rows="7" tabindex="5"></textarea>
      <div class="explanation">
```

```
        Your message, encrypted and authenticated
so that nobody can read it
        or change it without your password.
      </div>
    </div>
  </form>
</body>
</html>
```

Appendix III – AES sample

```
<!doctype html>
<html>
  <head>
      <!-- Include Dexie -->
      <script src="./dexie.js"></script>
           <script type="text/javascript" src="./cryptography.js"></script>
      <script>
          //
          // Define the database
          //
          var db = new Dexie("thesis_database");
          db.version(1).stores({
              users: 'username,passwd'
          });
                     var Crypt = new Crypt();  // constructor
          //
          // Put some data into it
          //
                     var ciphertext = Crypt.AES.encrypt("Amir");
                     var passwdtext = Crypt.AES.encrypt("123456");
          db.users.put({username: ciphertext, passwd: passwdtext}).then (function(){
              //
              // stored and read
              //
              return db.users.get(ciphertext);
          }).then(function (user) {
              //
              // Display the result
              //
                              var plaintext  = Crypt.AES.decrypt(user.username);
                          var passwdplain = Crypt.AES.decrypt(user.passwd);
             //alert (plaintext + " password is " + passwdplain);
                console.log (plaintext + "password is " + passwdplain);
          }).catch(function(error) {
             //
             // catch any error
             //
             alert ("Error": " + error);
          });
      </script>
  </head>

</html>
```