A New Security Model for Web Browser Local Storage

Thamer Al-Rousan[1]
Faculty of Information Technology
Isra University
Amman-Jordan

Bassam Al-Shargabi[2]
Faculty of Information Technology
Middle East University
Amman-Jordan

Hasan Abualese[3]
Faculty of Information Technology
Ajloun National University
Ajloun-Jordan

***Abstract*—In recent years, the web browser has taken over many roles of the traditional operating system, such as acting as a host platform for web applications. Web browser storage, where the web applications can save data locally was one of the new functionalities added in HTML5. However, web functionality has increased significantly since HTML5 was introduced. As web functionality increased, so did the threats facing web users. One of the most prevalent threats was the user's privacy violations. This study examines the existing security issues related to the usage of web browser storage and proposes a new model to secure the data saved in the browser's storage. The model was designed and implemented as a web browser extension to secure the saved data. The model was experimentally demonstrated and the result was evaluated.**

***Keywords*—HTML5; security; local storage**

I. Introduction

The internet and its applications have critically influenced us and becoming the main part of daily life. The internet provides information and opportunities, which were not accessible previously. The digital population has risen in the last few years; it reached 4.087 billion users in April 2018 [1]. With an increase in dependency on the internet, the importance of privacy protection has become significant.

When browsing, users expose themselves and their personal information to several threats, such as malware or phishing, which directly disrupt privacy [2]. A user's profile has been the target of many attacks. There are different reasons for an attack on a user's profile, including identifying the characteristics of users to meet their requirements or selling this information for surveillance or advertising [3]. Over the years, many technologies have been utilized to track a user's profile; the most well-known is cookies.

HTML is a standard language used to develop web applications. HTML5 is the latest version of HTML that the W3C standard introduced as a written language for web pages and associated applications (APIs). HTML5 brings a set of new advancements to the browser, which didn't exist previously, including enhanced XMLHttpRequest (XHR) and webSocket [4]. Web-based applications can be run offline via local storage, meaning the data will be saved in the user-side storage and it can be accessed without having to connect to the network [5]. User -side storage as indexed database APIs, web SQL databases, and web storage revolutionized the web and minimized the differences between native applications and web-based applications. Consequently, network connections' lack of effectivity on the server-side and the visual uncertainty affected by refreshes could be avoided. Besides, HTML5 offers functionality that can be utilized by web-based applications on any platform and on different browsers. Thus, modern web browsers support the plurality of new HTML5 features [6].

In spite of the advantages, user-side storage did not come without a price. The data saved by user-side storage is unencrypted, so users may experience privacy violations, such as a tracking vector. The literature in [7], [8], [9] revealed that the data created by web applications and saved in user-side storage cannot be completely deleted, even when the users tried. Thus, there are dangerous privacy implications, such as the details of a user's bank account or credit cards.

Previous studies to protect user-side storage were rather limited. This study proposes a new model that aims to protect the user's privacy. The proposed security model, which was based on the JavaScript encryption library (JSEL), was implemented as a web browser extension. The browser extension offered complete security protection, as the data were saved in encrypted form.

The remainder of the paper is arranged as follows. The theoretical background is presented in Section 2. Section 3 presents an overview of possible technical attacks associated with local browser storage and the current security solutions. Section 4 presents a new security model for browser-based storage. Related works are discussed in Section 5. The experimental study and the evaluation are presented in Sections 6 and 7, respectively. Finally, the conclusions and future work are presented in Section 8.

II. Background

The most famous user-side database is HTTP cookies. Cookies are small packets of data (normally 4KB) sent from websites servers to users through HTTP headers or by utilizing user-side scripting. Cookies saved in a user's browser are on the user's machine. Every cookie is related to a source, i.e., a port number, HTTP protocols, and the hostname. It based on a security mechanism termed same-origin policy (SOP). Website servers use cookies to remember a user's stateful information or to trace the browsing activity of the user. Security and privacy are the major challenges, as an attacker can steal data from cookies [10].

Over the years, many technologies appeared to save structured data in user-side storage. The majority appeared through third-party plugins, such as Oracle Java, Object of Adobe Flash, Google Gears, and Microsoft Silverlight [7]. As HTML5 appeared, browsers started to replace third-party plugins with built-in functionalities and new user-side storage technologies were established, such as “indexed database API, web SQL database, and web storage” [11].

Web storage provides a way for web applications to save data locally in the user’s browser without affecting website performance and provides more storage capacity than cookies. Depending on the user's browser, web storage capacity can be anywhere from 5MB to 25MB [11]. Web storage saves data in key/value pairs in the browser, and there two types. First is session storage data, which will be lost when the user browser is closed. Second is local storage data that are maintained even when the browser is closed [12]. Web storage is dependent on user-side scripting, such that the web application can retrieve locally stored data by utilizing a user-side JavaScript API even when the user’s browser is disconnected [11]. Besides, web storage differs from cookies, since the saved data cannot be transferred through HTTP headers. The security policy of web storage is the same as for cookies since each source is assigned to a unique web storage object. Therefore, using web storage on websites that do not support HTTPS or that use hostname sharing is not recommended [6].

A WebSQL database is another way for web applications to save huge amounts of information in the user’s browser, which can be queried by using standard SQL syntax. A WebSQL database is similar to Google Gears, and both depend on SQLite [13]. The main weakness of a WebSQL database is that it is not supported by W3C, and many browsers, for example, Mozilla, have decided to stop supporting WebSQL databases. Despite the W3C decision, three main browser vendors (Opera, Safari, and Chrome) still support WebSQL databases [11].

Indexed database API (Indexed DB) is a transactional database system that came from the W3C specification in 2009. Indexed DB is a substitute for the deprecated WebSQL database [14]. Indexed DB has the same accessing mechanism as web storage, but the scale and structure of the saved data are different. Web storage saves data in key/value pairs and is suitable when the dataset is simple, whereas Indexed DB allows for a large amount of structured data to be saved in storage [15].

Compared to SQL-based relational database management system (RDBMS), which uses tables to save data, Indexed DB is an object-oriented database. Indexed DB allows for objects to be saved/retrieved with keys [16]. Web applications can only access and operate saved structured data through the API provided by Indexed DB. Indexed DB is built on a transactional database mode and is typically key-value asynchronous storage. It provides rapid access to a lot of organized information [14]. In contrast, the security mechanism for Indexed DB is no different than web storage. The Indexed DB security mechanism is based on the principles of the SOP [13].

III. Security Issues

With the new functionalities of HTML5, which increased the accessibility to a user’s computer resources, including offline caching and local storage, new security issues arose. Most web browser security methods were not upgraded to the new technologies of HTML5. The only browser security method in place against possible risks was SOP [9]. SOP links the saved information to a specific domain, so the information can only be retrieved from the original domain. When SOP is applied, the browser tests the port number, name of host, and protocol against the source record of the saved data [2].

HTML5 has numerous new modules, such as “XMLHttpRequest (XHR), the document object model (DOM), and cross-origin resource sharing (CORS).” HTML5 brought new technologies, such as local storage, web Socket, and enhanced XHR. With these technologies and modules, it has improved the surface of the attack and added new risks to the end-user, which means that the SOP will not help [2]. The weaknesses of SOP led to many aggressions, such as “cross-site scripting (XSS), cross-site request forgeries (CSRF), cross-origin resource sharing (CORS) attacks, social engineering, and physical access” [17].

CORS is a technique that lets JavaScript make an XHR request from other domains outside the original domain. XHR is used by JavaScript to perform transfers between the server and the user. Cross-domain requests are normally prevented by web browsers, so CORS adds additional information to HTTP headers to permit the request [17]. A CORS technique allows for several domains (cross-domain requests) between the user and the server. However, hackers can make cross-site request forgeries by bypassing SOP and creating “cross-domain connections” to permit the deployment of a CORS attack vector [3].

XSS is one the most common attacks in a web-based application. Based on the open software security community (OWASP) list, XSS ranks third place in OWASP list [18]. XSS is a security weakness, where malicious code (generally JavaScript) is injected into a web-based application and is comprised of dynamic content sent to a victim's browser. The victim’s browser cannot identify that the script is infected, and will execute the malicious script without being validated because it appears to have arrived from a trusted source. The malicious script can change or transmit session tokens, cookies, or any information via the victim’s browser [19]. There are many techniques to prevent XSS attacks, such as encoding output (stop executing URL links that include binary encoded characters), filtering (the input data is filtered before it saved in the database), and authenticating user input (checking the format of the user) [6].

Social engineering is a way of fooling and cheating users so that personal data is shared, such as bank accounts or passwords. The hackers can install malicious code to access data by controlling the user’s computer. Social engineering techniques are easier than other hacking techniques and can include calling by phone, sending an email, and real-life chats. An email from a known sender is a popular example of a social engineering technique. Hackers send an email message with a link to the victim contact list members. The link may contain a picture, audio, movie, or file that the malicious software is inserted in. If the member clicks on this link, that member could then be the next victim [20].

CSRF is also known as session riding or a one-click attack. An attacker could trick the victim into unintentionally performing an undesirable action in the web application [21]. CSRF is normally performed with the assistance of social engineering techniques. As the victim is currently authenticated by the web application, it is difficult for the server to differentiate between legal requests and fake requests [15]. CSRF attacks can harm both users and businesses. It can result in damaged user relationships, illegal money transfers, changed passwords, and stealing of data, including the session cookies [17].

Physical access is another security problem that occurs when the attacker has access to the user's computer. When this happens, the saved data and even the deleted data can be stolen. Recovering deleted data is primarily concerned for developers. The experiment conducted by Shahryar [22] showed that the location of the deleted data within a database was still physically reserved. The deleted data was only marked as deleted ,but the data existed in storage and could be retrieved with forensic tools. In most web browsers, the non-deleted data will not be overwritten by newly saved data [9]. The data is saved in storage as non-encrypted, so it is not protected and presents a possible security issue. Encryption will prevent data from being attacked. User-side encryption is important since it offers a security mechanism for saved data and blocks unauthorized persons from stealing the user's data. Unfortunately, user-side encryption is not yet mature. In the next suction, this study proposes a new encryption model to protect a user's data.

IV. Security Model

This study proposes a new security model to address issues caused by saving data in an unsafe manner. In the proposed model, an additional layer was added among the web browser storage and the web browser itself. The model contains a new algorithmic framework for improving security against threats. The model uses JavaScript encryption library, which was applied to the web browser as “an extension.” The extension was located at the top of the web browser storage, such that whenever writing or reading information, it is encrypted.

Encryption/decryption can be implemented inside browsers via the” JavaScript crypto library.” There are a small number of JavaScript encryption libraries that may be applied to encrypt data in the client’s browser, such as WebCryptoAPI, PolyCrypt, Cryptos, Jscrypto, and SJCL [21]. Many factors affect the selected encryption library, including library size, hashing functions, key lengths, keyed-hash message authentication codes (HMAC), salt, and availability of multi-platform and multi-browser options [11].

This study selected the Stanford JavaScript crypto library (JSCL) to implement the encryption/decryption on the user’s browser. The SJCL library was chosen because it is a secure, small, multi-platform, and powerful library [23]. SJCL uses the advanced encryption standard (AES) to encrypt data at key sizes of 128/192/256 bits. It also uses an HMAC validation code, the SHA256 hash function, OCB and CCM authenticated modes, and the PBKDF2 to strengthen the password. BKDF2 applies a pseudorandom function to the entered password together with a salt value and reiterates the process several times to generate a derived key, which then can be utilized as a crypto key. PBKDF2 verifies each message it sends to avoid any changes. Furthermore, the tests conducted in several browsers on Windows, Linux, and Mac have shown that the SJCL was faster than any existing encryption library [23].

As the browser extension used the SJCL library, the same shared key was applied for both encryption and decryption, which made the data prone to malicious attacks. To overcome the limitations of the SJCL library, the study combined the RSA standard [9] with SJCL library. The combination resulted in a hybrid encryption algorithm comprised of both RSA and AES to guarantee data integrity.

When the browser storage received a request from a web application to save data, the data was encrypted by the proposed browser extension. This ensured that the data was safe and secured against illegal access even if the attacker gained physical access to the machine. The data encryption/decryption steps are as follows:

- Get Login: The initial step is to afford a secure log into the system. Web applications can handle this step for the users by using the login process.
- Data Encryption: When a new request from a web application arises to save data in the database, the plain text data (PTORG) is encrypted by the designed extension and a public key (KSJCL) is generated. The KSJCL will be used when the SJCL library encrypting the data. The encrypted data is named ENORG. Simultaneously, when the encryption process is enabled in user-side, the RSA standard is generated. RSA standard is used to encrypt the ENORG by using the private key (PSJCL) to produce a “digital signature.”
- Data Decryption: When a request from an authorized user to read data from the browser’s storage arises, the key (K SJCL) is encrypted using an RSA standard with the requester public key (RKPB). The encrypted key is named (RKEN SJCL). An XML file is then created, including ENORG and RKEN SJCL. Finally, the ENORG is decrypted using KSJCL.
- Deletion of Data: The data in the session storage is safely deleted - that is, it was replaced with zeros. Thus, the deleted data cannot be read again, since the original data was set to zero.

The goal of this study is to protect saved data on client-side databases against illegitimate access. The study had many choices for encryption, but most tended to save encryption keys on the server-side and not on the user-side. One limitation of utilizing the server-side was associated with offline storage, the client won't almost certainly get to the information. Besides, if the webserver is vulnerable to any attack, then the encryption key will be perilous and thus, the encrypted information will be defenseless against unapproved access from attackers. Also, the W3C recommendation is to save the encryption keys on the client-side [24].

Browser security models depend on an SOP mechanism. However, an SOP mechanism alone is not enough for sophisticated web-based applications' local storage security. The proposed model differs in the security mechanism. Where the browser security models attempt to secure data amongst the user's browser and web-based applications, the proposed model secures data saved in the user's database. Thus, users can visit other websites without worrying about the databases.

V. Related Work

Previous studies regarding securing web browser storage and its effectiveness are still in early stages and are limited. Aggarwal et al. [25] conducted one of the first studies to analyze browsing security vulnerabilities. The results revealed that there was an insufficient implementation of the security mechanism in different web browsers, which pointed to user activities. Additionally, security control for Firefox was proposed, which protected users after private mode was enabled. In 2011, Oh et al. [26] analyzed the log files generated by a web browser, concentrating on search history, timeline analysis, and URL encoding. A Classification of Web Browser Log (CWB) tool was proposed to prove the analysis. Unfortunately, in the experiments, old versions of browsers were used that are currently outdated.

Ohana and Shashidhar [27]studied portable browsers to determine if data still existed after the browser session stopped. Similarly, Said et al. [28] analyzed the RAM in the browser sessions, including authorizations, history, images, and videos. Satvat et al. [29] extended the work by analyzing the file system and the network, which exposed significant contradictions in the browsing implementation that violated a client's privacy. Heule et al. [30]developed a security access control to protect confidential data that could be accessed and used by attackers, focusing on JavaScript extensions in Chrome. In the same manner, Lerner et al. [31] studied JavaScript extensions in Firefox from different perspectives, such as social, safety, and debugging, to determine which may be malicious.

Ruiz et al. [32] concentrated on recovery techniques created during browsing. Experiments within four personal levels showed how the browser could be stopped, namely shutdown, power down, freeze, and kill processes. The results revealed that all levels included user privacy violations in terms of obtaining browsing data. In the same manner, Montasari and Peltola [33] studied both file systems and RAM in different browsers. The results revealed that the most secure browser was Chrome and in second place was Firefox.

A survey conducted by Gao et al. [34] concentrated on awareness from a user's viewpoint. Authors surveyed more than 200 users regarding the security and privacy mechanisms provided by the most common web browsers on smartphone and desktop platforms. The results revealed that better security guarantees were required concerning a user's privacy. Tsalis et al. [35] studied the protection provided by different popular web browsers. A set of web data created in a usual browsing session was used to determine where these data were saved after the session was stopped. The results revealed that the deleted data after the browsing session were discovered straightforward or in a roundabout way in the database. Subsequently, any person who had "physical access" to the user's machine with adequate IT abilities could access these browsing data. In addition, nearly all browsers provided a similar protection level, and only Chrome in the guest mode provided better protection. Belloro and Mylonas [30] investigated the most common user storage methods, namely "Web SQL database, web storage, and indexed database." The outcomes revealed that web storage was the most utilized. Belloro and Mylonas also surveyed whether well-known mobile and desktop browsers that used "indexed database API, web SQL database and web storage" could defend clients from privacy violations. The results showed that the maturity of the security controls was inadequate to avoid privacy violations.

VI. Experiment

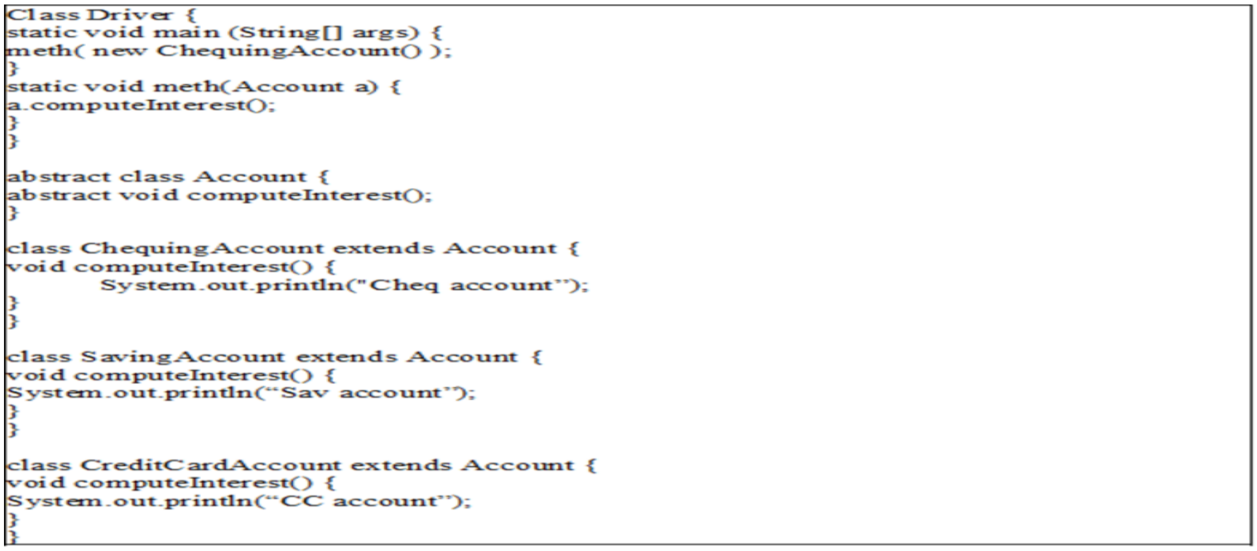
The experiment performed in this study was based on performance. The study tested the speed of encryption/decryption, which affected the performance of the model. Thus, the study tested the performance on the user-side to prevent the network and server latency impacting the results. The study selected indexed database as a representative for web browser local storage.

Before running the test, the code was executed for a compatibility test in diverse browsers, including Firefox, Google Chrome, Safari, Mozilla, and Opera. The latest Google Chrome browser hosted by a Windows10 server was chosen as the test environment. The server was an HP with Intel I7 8550U, and 8 GB DDR4 RAM. The test executed a small JavaScript by dexie.js, which is a powerful library for an indexed database and it can accelerate performance. Google private mode was used to the grantee that the test was executed without any additional load of scripts or any extension. Fig. 1 show the code used.

The test was performed in steps. The first step was to save the data without encryption (in a standard manner). The second step was to save data with encryption, as seen in Fig. 2. The same password and username were saved with encryption.

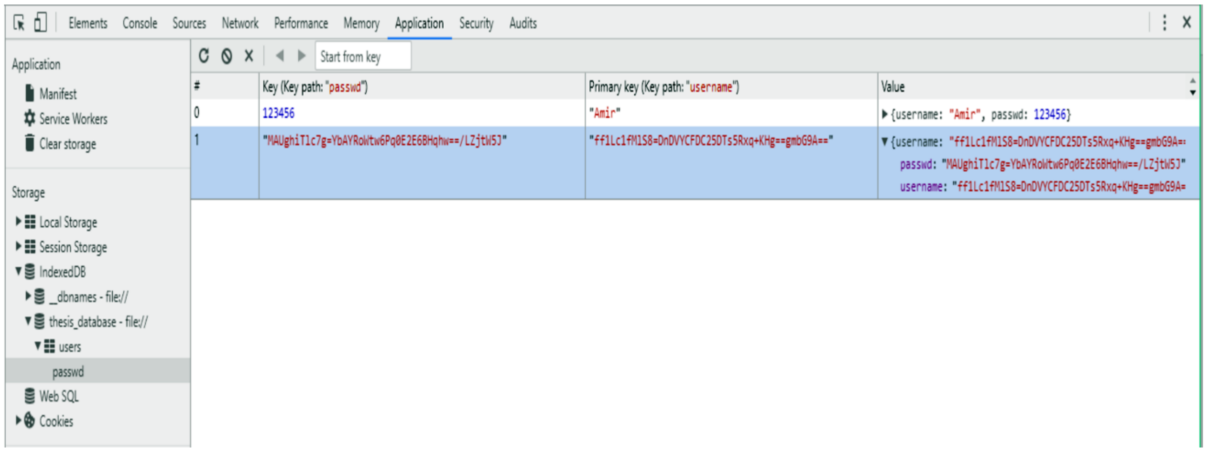
The test was executed without any user intervention. JavaScript read the entries of the database, placed the data into the table, and read it as a console output (console.log). All references employed in this study were based on Google performance analysis [17]. Fig. 3 shows the results of the performance test, it shows the top 10 bottom-up of processes that most consuming time.

Because the indexed database is a NoSQL database, the script run time had to be tested independently of the transaction speed of the database. Consequently, the performance test depended on the reading and writing of database entries in a loop. The time was measured before the loop started and after the loop was finished. Finally, the data entries contained "Password + I and User + i" where "I" was the loop counter. The time in milliseconds to insert database entries with and without encryption in Google Chrome is shown in Table I.



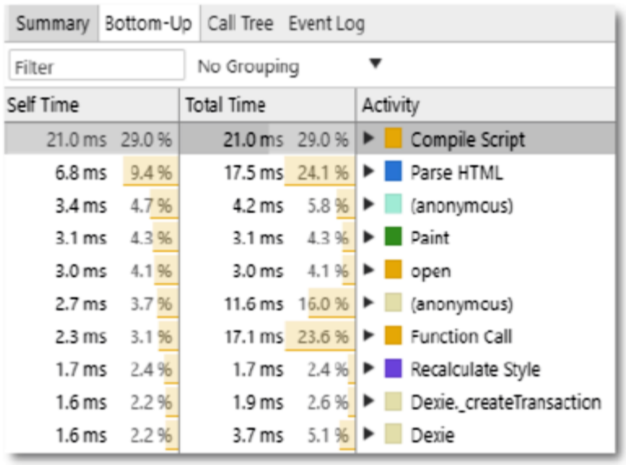
```
Class Driver {
static void main (String[] args) {
meth( new ChequingAccount() );
}
static void meth(Account a) {
a.computeInterest();
}
}

abstract class Account {
abstract void computeInterest();
}

class ChequingAccount extends Account {
void computeInterest() {
        System.out.println("Cheq account");
}
}

class SavingAccount extends Account {
void computeInterest() {
System.out.println("Sav account");
}
}

class CreditCardAccount extends Account {
void computeInterest() {
System.out.println("CC account");
}
}
```

Fig. 1. AES Sample.

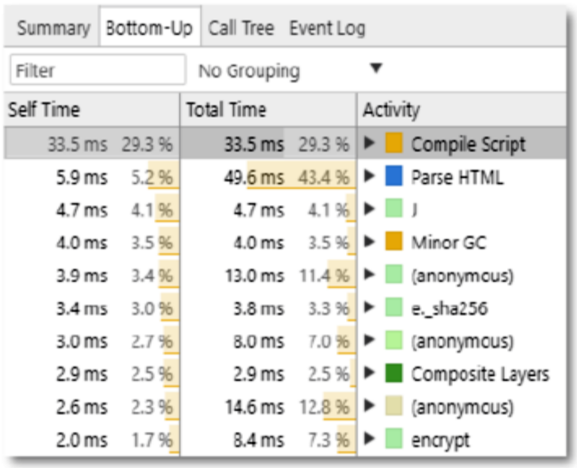


#	Key (Key path: "passwd")	Primary key (Key path: "username")	Value
0	123456	"Amir"	{username: "Amir", passwd: 123456}
1	"MAUghiT1c7g=YbAYRoWtw6Pq0E2E6BHqhw==/LZjtW5J"	"ff1Lc1fM1S8=DnDVYCFDC25DTs5Rxq+KHg==gmbG9A=="	{username: "ff1Lc1fM1S8=DnDVYCFDC25DTs5Rxq+KHg==gmbG9A= passwd: "MAUghiT1c7g=YbAYRoWtw6Pq0E2E6BHqhw==/LZjtW5J" username: "ff1Lc1fM1S8=DnDVYCFDC25DTs5Rxq+KHg==gmbG9A=

Fig. 2. Key and Values after Plain Text Encryption.



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



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

Fig. 3. The Results in the Performance Test; Left the Plain Text Entries, Right Encrypted Entries.

TABLE. I. THE TIME TO INSERT DATA INTO INDEXEDDB

Test Number	Record Number	Without Encryption (ms)	With Encryption (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

VII. EVALUATION

Big O notation was used to evaluate the proposed model. It compares the effectiveness of different algorithms by revealing the time that the algorithm took to run. The runtime can be expressed with Big O Notation as how fast the runtime grew relative to the size of the input "n" (denoted O (n)).

AES allows for three diverse key sizes of 128, 192, or 256 bits. The processing required 10 rounds when encrypting with the 128-bit key, 12 rounds for the 192-bit key, and 14 rounds for the 256-bit key. When ciphering with the 128 bit key, all 2^{128} keys mixtures must be inspected by decrypting the encrypted text with every one of those values [13].

For decryption and encryption, every round had four functions, excluding the final round with three. Encryption had the following functions: SubByte, ShiftRows, MixColumn, and AddRoundKey. A similar number of round functions were used for decryption, but with the opposite transformation.

The algorithm takes into account the diverse runtime periods with the same inputs, based on the speed of the processor, disk speed, instruction set, and type of compiler. The measured runtime "T (n)" is the amount of primary steps, taking into account that every progression step requires steady time. Since the inner loop iterates, the runtime was computed as:

$$T(n) = O(n^2) \qquad (1)$$

Where **n** is the size of the input data. Thus, the performance of an algorithm is proportional to the square of **n**. The study tested the performance impact of including encryption steps to the key-value database of web storage. Fig. 4 shows the results.

The results indicate that the process with encryption steps became slower by 10-40%. Therefore, applying encryption/decryption increased security but decreased response time. The process of encryption/decryption depended on hardware optimization and configuration. Devices with superior processors and greater memory storage sped up the process. AES functionality is now integrated into many processors, which helps to reduce encryption\decryption time.

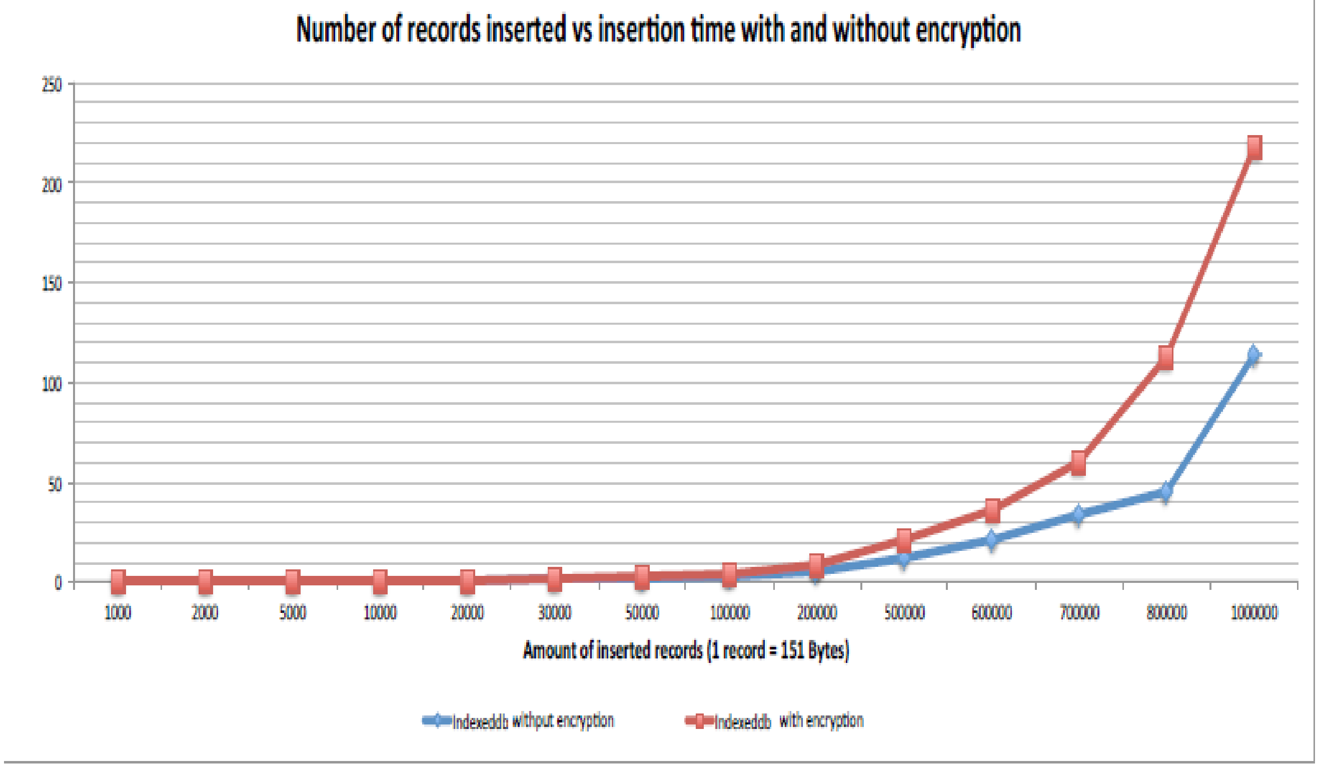


Fig. 4. The Time to Insert Data with and without Encryption into IndexedDB.

VIII. Conclusion and Future Work

Local browser storage has important advantages, when compared with a server-side database, including fast response, offline usage, and decreased network latency. However, currently, local browser storage is not secure. The main security concern is that data saved locally is unencrypted. The literature confirmed that the unencrypted data was not the only problem facing local browser storage. Data recovery of previously deleted data was another serious problem. Current security methods do not provide adequate security defenses for data saved locally, particularly data that may contain private information.

This study proposed and implemented a new security model for local browser storage. The new model added an additional layer amongst local browser storage and the web browser itself. The model contained an algorithmic framework for improving security against vulnerabilities identified in this study. JSCL was used in the proposed algorithm and was applied to the web browser as an extension. The SJCL library was chosen because SJCL is a secure, small, fast, multi-platform and powerful library for cryptography in JavaScript. The study combined the RSA standard with an SJCL library, which resulted in a hybrid encryption algorithm to guarantee data integrity.

When browser storage received a request from a web application to save data, the data was encrypted by the proposed browser extension. Encrypting data in browser storage kept data from being undermined regardless of whether the attackers acquired physical access to the machine, which may occur if a phone, tablet, or computer is lost or stolen.

Although the proposed model has been effectively applied to local browser storage, additional enhancements could be made in the future by extending the performance and security model. For example, a complete security library could be written to utilize diverse JavaScript crypto libraries. Further experiments with different crypto libraries could also be performed to compare the results.

Acknowledgments

We would like to express our deep appreciation to the people at Isra University who have helped us throughout our research.

References

[1] S.Sheikh, and M. AdeelPasha,"Energy-Efficient Multicore Scheduling for Hard Real-Time Systems: A Survey," ACM Transactions on Embedded Computing Systems (TECS), vol. 17, no. 6, 2018.

[2] N.Tahmasbi, and E. Rastegari, "A Socio-Contextual Approach in Automated Detection of Public Cyberbullying on Twitter," ACM Transactions on Social Computing, vol. 11, no. 4, 2018.

[3] P.Gradinger, D. Strohmeierand C. Christiane, "Definition and measurement of cyberbullying," Journal of Psychosocial Research on Cyberspace , vol. 4, no. 2, 2015.A. Robert, and J. Ravenscroft, "Dynamic data structures, a web based tool for teaching linked lists and binary trees," Journal of Computing Sciences in Colleges, vol. 33, no. 6, pp. 97-106, 2018.

[4] T. Al-Rousan, and H. Al Ese, "Impact of Cloud Computing on Educational Institutions: A Case Study," Recent Patents on Computer Science, vol. 8, no. 2, pp. 106-111, 2015.

[5] M. Díaz, M.Martín, and B.Rubio, "State-of-the-art, challenges, and open issues in the integration of Internet of things and Cloud Computing,"

[6] Journal of Network and Computer Applications, vol. 67, no. 1, pp. 99-117, 2016.

[7] V. Karavirta, and C. Shaffe, "Creating engaging online learning material with the JSAV JavaScript algorithm visualization," IEEE Transactions on Learning Technologies, vol. 9, no. 2, pp. 171--183, 2016.

[8] I. Koren, and R. Klamma, "The Exploitation of OpenAPI Documentation for the Generation of Web Frontends," in Proceedings of the The Web Conference 2018, Lyon, France, 2018.

[9] N. Correia, and J. Kleimola, "Web browser as platform for audiovisual performances," in Proceedings of the 11th Conference on Advances in Computer Entertainment Technology, Funchal, Portugal, 2017.

[10] A. McDonald, "Cookie confusion: do browser interfaces undermine understanding?," in Proceeding on Human Factors in Computing Systems, Atlanta, Georgia, USA, 2015.

[11] W.Kuang, L. Wu, and Y.Liu, "Key Selection for Multilevel Indices of Large-scale Service Repositories," in Proceedings of the 10th International Conference on Utility and Cloud Computing, Austin, Texas, USA, 2016,pp. 139-144.

[12] T. Al-Rousan, " An Investigation of User Privacy and Data Protection on User-Side Storage," International Journal of Online Engineering, 2019, vol. 15 no. 9, pp.17-30. 14, 2019.

[13] J.Hamilton, "Internet scale storage," in Proceedings of the 5th ACM SIGMOD International Conference on Management of data, Athens, Greece, 2018.

[14] M. Arenas, "Database Theory Column Report on PODS 2018," ACM SIGACT News, vol. 49, no. 4, pp. 55-57 , 2018.

[15] H. Oosterhuis, J. Culpepper, and M. Rijke, "The Potential of Learned Index Structures for Index Compression," in Proceedings of the 23rd Australasian Document Computing Symposium, Dunedin, New Zealand, 2018.

[16] R.Brunel, N. May, and A.Kemper, "Index-assisted hierarchical computations in main-memory RDBMS," VLDB Endowmen, vol. 9, no. 12, pp. 1065-1076 , 2016.

[17] L. Kim, and H. Lee , "Web-in-the-loop simulation framework for supporting CORS-based development," in Proceedings of the Poster Session and Student Colloquium Symposium, Alexandria, Virginia, 2017.

[18] J. Laassiri , "Data Security and risks for IoT in intercommunicating objects," in Proceedings of the 2nd international Conference on Big Data, Cloud and Applications, Tetouan, Morocco, 2017.

[19] J. Bozic, and F. Wotawa, "XSS pattern for attack modeling in testing," in Proceedings of the 8th International Workshop on Automation of Software Test, San Francisco, California, 2016.

[20] T. Al-Rousan, "Cloud Computing for Global Software Development: Opportunities and Challenges," in ransportation Systems and Engineering: Concepts, Methodologies, Tools, and Applications, IGI Global, 2015, pp. 897-908.

[21] N. Meng, S. Nagy, D. Yao, and D. Zhuang, "Secure coding practices in Java: challenges and vulnerabilitie," in Proceedings of the 40th International Conference on Software Engineering, Gothenburg, Sweden, 2018.

[22] H.Shahriar, "Security vulnerabilities and mitigation techniques of web applications," in Proceedings of the 6th International Conference on Security of Information and Networks, Aksaray, Turkey , 2016.

[23] W. Groef, F. Massacci, and T. Piessens, "NodeSentry: least-privilege library integration for server-side JavaScript," in Proceedings of the 30th Annual Computer Security Applications Conference, Louisiana, USA , 2017.

[24] M. Olan, "HTML5 jumpstart," Journal of Computing Sciences in Colleges, vol. 28, no. 3, pp. 35-36 , 2016.

[25] C. Aggarwal, Y. Li, P. Yu, and R. Jin, "On dense pattern mining in graph streams," VLDB Endowment, vol. 3, no. 1-2, pp. 975-984 , 2010.

[26] J. Oh, N. Son, and S. Lee, "A Study for Classification of Web Browser Log and Timeline Visualization," Lecture Notes in Computer Science , 2011.

[27] D. Ohana, and N. Shashidhar, "Do Private and Portable Web Browsers Leave Incriminating Evidence? A Forensic Analysis of Residual Artifacts from Private and Portable Web Browsing Sessions," Journal on Information Security, vol. 10, no. 1, pp. 135-142, 2013.

[28] H. Said, N. Al Mutawa, and I. Al Awadhi, "Forensic analysis of private browsing artifact, "Proceedings of the 11th Conference on Forensic analysis of privatbrowsing artifacts," 2011.

[29] K. Satvat, M. Forshaw, F.Hao, and E.Toreini, "On the privacy of private browsing –A forensic approach. ,," Journal of Information Security and Applications, vol. 19, no. 1, pp. 88-100, 2014.

[30] D. Ruiz, F. Amatte, and K. Park, "Overconfidence: Personal Behaviors Regarding Privacy that Allows the Leakage of Information in Private Browsing Mode," International Journal of Cyber-Security and Digital Forensics , vol. 4, no. 36, pp. 104-416, 2015.

[31] R. Montasari, and P. Peltola, "Computer Forensic Analysis of Private Browsing Modes," in Proceedings of the 10th Conference on Global Security, Safety and Sustainability: Tomorrow's Challenges of Cyber Security, Springer International Publishing, 2015.

[32] X. Gao, Y. Yang, H. Fu, and J. Lindqvist, "Private Browsing: an Inquiry on Usability and Privacy Protection," in Proceedings of the 15th Conference on Privacy in the Electronic Society, ACM.

[33] N.Tsalis , N. Virvilis, and A. Mylonas, "Browser Blacklists: A utopia of phishing protection. Security and Cryptography," in Security and Cryptography (Eds.), Lecture Notes (CCIS), Springer, 2017.

[34] S. Belloro, and A. Mylonas, "Security considerations around the usage of client-side storage APIs," Technical Report No. BUCSR-2018-01, 2018.

[35] K. Basques, "Tools for Web Developers - Performance Analysis Reference," Google, [Online]. Available: Performance Analysis Reference. [Online]. [Accessed 3-12-2018].