Implementation and Evaluation of a Secure and Efficient Web Authentication Scheme using Mozilla Firefox and WAMP

Yassine SADQI, Ahmed ASIMI, Younes ASIMI, Zakaria TBATOU,Azidine GUEZZAZ
Departments of Mathematics and Computer Science, Information Systems and Vision Laboratory (LabSIV), Faculty of Sciences, Ibn Zohr University B.P 8106, City Dakhla, Agadir, Morocco

***Abstract*— User authentication and session management are two of the most critical aspects of computer security and privacy on the web. However, despite their importance, in practice, authentication and session management are implemented through the use of vulnerable techniques. To solve this complex problem, we proposed new authentication architecture, called StrongAuth. Later, we presented an improved version of StrongAuth that includes a secure session management mechanism based on public key cryptography and other cryptographic primitives. In this paper, we present an experimental implementation and evaluation of the proposed scheme to demonstrate its feasibility in real-world scenarios. Specifically, we realize a prototype consisting of two modules: (1) a registration module that implements the registration, and (2) an authentication module integrating both the mutual authentication and the session management phases of the proposed scheme. The experimental results show that in comparison to traditional authentication and session management mechanisms, the proposed prototype presents the lowest total runtime.**

Keywords*— *authentication; session management; web security; cryptographic primitives; computer security and privacy; security implementation; authentication scheme; Mozilla Firefox; WAMP

I. INTRODUCTION (*Heading 1*)

Despite its widely studied security problems [1]–[13], password authentication through an HTML form is the dominant mechanism for authenticating users in modern web applications [4], [14], [15]. More specifically, the lack of authentication standard HTML form and the limited security background of webmasters have created a set of unique design and implementation choices that contain multiple security vulnerabilities [3], [12]. While authentication experts proposed a wide range of secure alternatives [16][17]–[20], Bonneau et al. [4] showed that the majority of these schemes offer more security than passwords, but they are difficult to use and / or expensive to deploy [4].

In [13] we have proposed a new authentication architecture, called StrongAuth, to enhance security without sacrificing usability and deployability. Specifically, our proposal does not require any additional equipment except a modern web browser. Later, we presented an improved version of StrongAuth including a secure session management mechanism [21]. This version covers the complete cycle of authentication in the context of web applications, consisting of mutual authentication phases and the subsequent HTTP requests authentication [21].

In this paper, we realize a prototype consisting of two modules: A registration module that implements the registration phase of the proposed scheme [21], and another authentication module that integrates both the mutual authentication and the session management phases. On the one hand, we integrate the client architecture component as an extension of Mozilla Firefox that can easily install it using Mozilla Firefox Add-ons Manager and the server component as a service of PHP applications within the WAMP platform, which allows us to avoid recompiling the source code of Mozilla Firefox and have an independent server component of the application code; which facilitates the deployment. On the other hand, we evaluate the performance of registration and authentication modules to evaluate the theoretical study presented in [21].

The rest of the paper is organized as follows: in section 2 we briefly review the registration, the authentication and session management phases of the proposed scheme [21]. In Section 3 we are particularly interested in the implementation of our prototype to show the feasibility of the proposed scheme. Section 4 presents our results and discussions of the experimental evaluation of the proposed prototype. Section 5 concluded the paper.

In the rest of this paper, we denoted by:

U_i	ith User.
ID_i	Unique identifier of user U_i.
P_i	Password of user U_i.
$Salt_i$	Cryptographic Salt of user U_i .
d	Web application domain name.
RW_i	Random value used at most once within the scope of a given session generated by the web application for U_i.
RB_i	Random value used at most once within the scope of a given session generated by the browser for U_i .
USK_i,UPK_i	Asymmetric key pair for user U_i generated by

	the browser using a secure asymmetric key generation algorithm.
SSK	Web application Private Key.
SPK	Web application Public Key certificate.
PSK_i	Pre-Session key shared between U_i and the web application using HTTPS.
SK_i	Session Key.
X_i^{new}	Renewal of the parameter X_i.
PBKDF( )	Password-Based Key Derivation Function.
MK_i	The master key that is output from an execution of *PBKDF* by the browser.
SE(k,b)	Encryption of b by k using a secure symmetric encryption algorithm.
AE(k,b)	Encryption of b by k using a secure asymmetric encryption algorithm.
SD(k,b)	Decryption of b by k using a secure symmetric decryption algorithm.
AD(k,b)	Decryption of b by k using a secure asymmetric decryption algorithm.
H( )	Cryptographic one way hash function.
HMAC(K,m)	Compute the keyed-Hash Message Authentication Code of message *m* using the secret key *K*.
A || *B*	The concatenation of binary strings A and B.
$\oplus$	XOR operation.
==	Comparison.
CCS	Client Cryptographic Services.
UACM	User Authentication Credentials Manager.
CS	Client Storage.
SCS	Server Cryptographic Services.
RD	Registration Database.
url	URL of the resource requested by U_i browser.

II. Review of the Proposed Scheme

In this section, we briefly review the registration, the authentication and session management phases of the proposed scheme. For all details see [21].

A. Registration Phase

The registration phase is invoked whenever a new user U_i wants to register within the web application. This registration process does not ask from U_i more than choosing an ID_i and P_i. It is know that password-based authentication presents several security problems. For this, the browser transparently from U_i integrates other cryptographic parameters that are used to strengthen users' authentication. Also in this phase, the proposed scheme relies on HTTPS to protect UPK_i, confidentiality and integrity. This phase proceeds as follows:

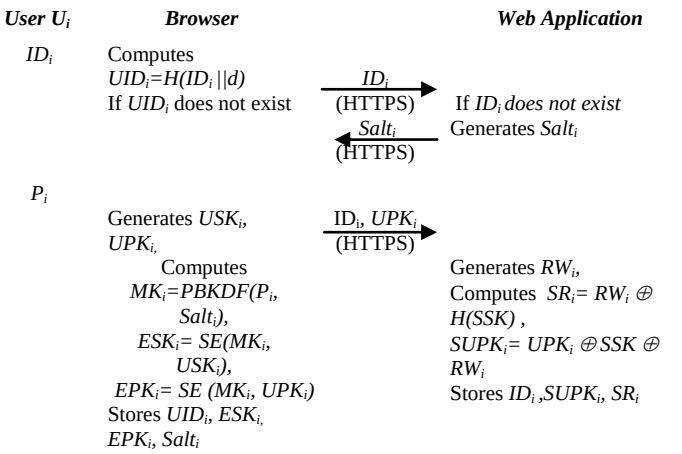


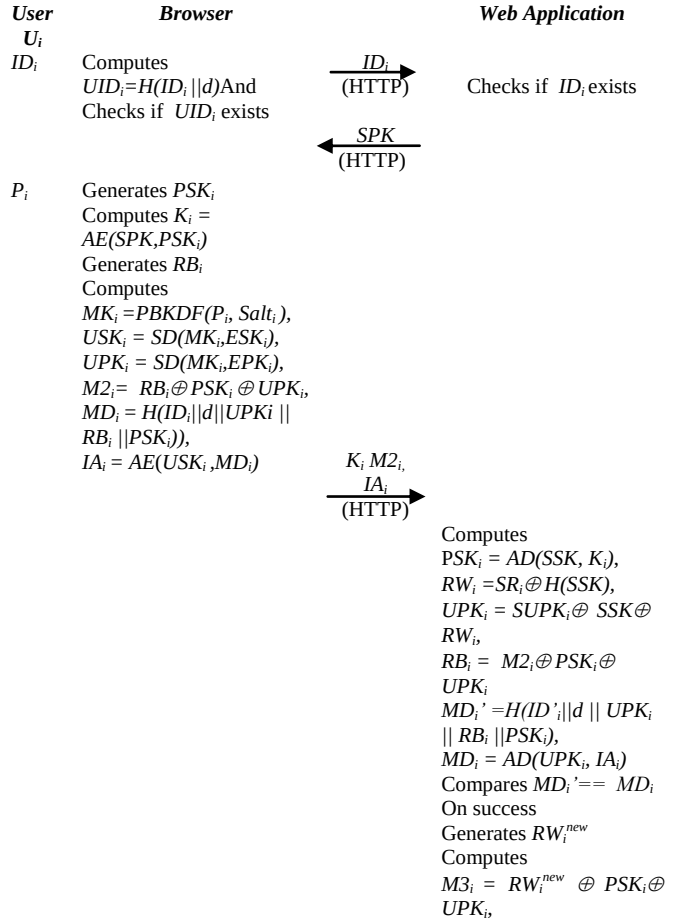
Fig. 1. Registration phase

B. Mutual authentication and session management phase

Primarily, this phase aims to provide:

- A strong mutual authentication between the communicating entities without disclosure of the original authentication settings.
- An agreement on a session key SK_i.
- An HMAC signature in each HTTP request from the browser to the web application using SK_i.

Figure 2 describes the protocol steps



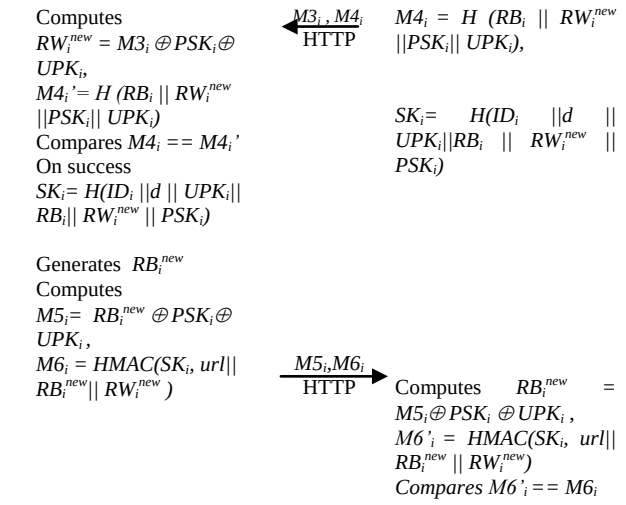


Fig. 2. Mutual authentication and session phases

III. Implementation of the Proposed Prototype

To show the feasibility and effectiveness of the proposed scheme [21], we realize a prototype. Mainly it aims to experience the registration phase, mutual authentication and session management phases. Figure 3 shows the different elements of our prototype, as well as technologies that we used. Tables I and II show the software solutions used in our prototype implementation.

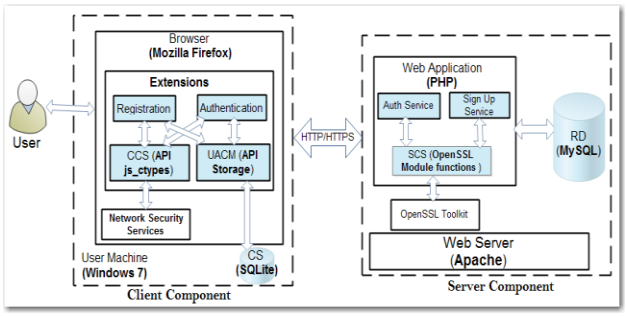


Fig. 3. General architecture of the proposed prototype

TABLE I. The technologies used in the implementation of the client component

Entities	Technology
Browser	*Mozilla Firefox*: The implementation of the client component of our prototype is based on the Mozilla Firefox browser.
Client Cryptographic Services (CCS)	*API js-ctypes* [22]**:** Since we are using Firefox, we choose to use the NSS cryptographic services. To interact with NSS [23] we use the js-ctypes API.
***User Authentication Credentials Manager* (UACM)**	*API Storage* [24]: This interface allows the manipulation of the SQLite database from extensions or internal component of Firefox.
Client Storage (CS)	*SQLite* [25]: The main objective of using the multiplatform engine database SQLite is to overcome any installation or administration, which facilitates deployment. Figure 4 shows the client-side authentication settings.

TABLE II. The technologies used in the implementation of the server component

Entities	Technology
Web Server	*Apache* [26]: Apache HTTP Server with the support of the module Mod_SSL [27] that allows the implementation of HTTPS.
Web Application	Application based on PHP 5.
Server Cryptographic Services (SCS)	*Hash* and *OpenSSL* : PHP platform provides a set of cryptographic extensions either as a part of the core PHP functionalities (without the use of a third-party program), or relies on other cryptographic libraries. In our prototype implementation we use two extensions: • *Hash* [28]**:** For the calculation of hash and message authentication code (HMAC). Hash is a digital hash engine, part of the core of PHP. That means we can use these functions in the web application without installing a third-party library. • *OpenSSL* [29]**:** As opposed to Hash, the use of this module requires the presence of an equal or higher version 0.9.6 of OpenSSL cryptographic library. The purpose of this extension is to present a set of cryptographic functions that can be used easily in a PHP script(e.g. asymmetric/symmetric encryption, generation and verification of digital signatures, etc.).
Registration Database (RD)	The relational database management system MySQL [30]. Figure 5 illustrates the authentication settings on the web application side.

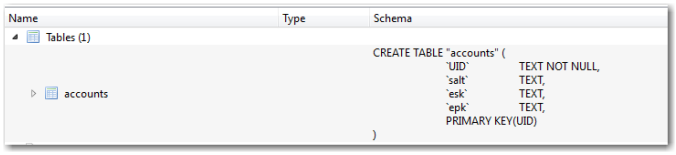


Fig. 4. Client-side authentication settings

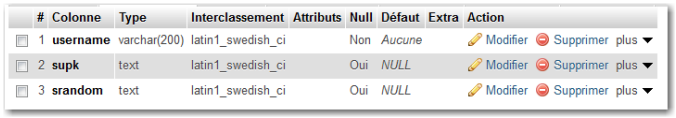


Fig. 5. Web application-side authentication settings

To simplify the implementation and the experimental evaluation of our prototype, we divide the implementation into two modules: (1) registration module, (2) and an authentication module. The following two subsections provide details on each of these two modules.

A. Registration Module

The purpose of this module is to implement the registration phase of the proposed scheme [21]. We develop the client component as an extension of Mozilla Firefox to avoid recompiling the browser source code (no changes were required to the browser source code). Note that instead of HTML password form field (<input type = "password"), the proposed prototype presents the user with a private window to choose securely passwords (Figure 8). On one hand, in the proposed scheme the browser does not send the password to the web application. The password is used only within the client component to generate a symmetric encryption key (the application does not need to know the user's password). On the

other, Sandler and Wallach [7] discussed in detail that the use of password field is a serious problem, facilitating password theft. Our prototype is not the only one using the standard notification API to create a trusted path to the private password window. Menalis et al. [14] also used this concept.

In detail, the implementation of our Mozilla Firefox registration extension required:

- 1020 lines of chrome / privileged JavaScript code: About 600 lines for client cryptographic service implementation based on the js_ctypes API, which provides access to the cryptographic library of Firefox (NSS), and 420 lines to integrate other client component features and the interaction with the server component.
- 85 lines of XUL code: 70 lines of XUL code to define the interface of the private window for password selection and confirmation (Figure 9). Also, we use an overlay file of about 15 lines of XUL code to integrate the extension components into the Firefox user interface.
- Other extension configuration files (chrome.mnifest, install.rdf ....).

On the other hand, web application cryptographic operations side and the communication with the client component have required 130 lines of custom PHP code. Also, to support HTTPS, we add 20 lines of code to the configuration file of the Apache web server. More importantly, our server component is completely independent of the application code.

The steps of the registration module proceed as follows:

1) The application presents a registration form based on HTML and CSS to the user. This preserves the same user experience. Since users are used to complete such information (name, email ...) in the registration phase of current web application. Once the user filled out the form and click on the "Sign Up" button, the extensions sends the information to the web application.

{ ***"salt"***:
"MTQxMDUwLjYwOTk0OTAwIDE0MzAyNTkyODWJlE+EEuJ7aaJTCqE7uwm",
"username": "yassine",
"sessionRegID" :"NjAwMzMzNTE4ODAxNTYwNg==",
"regURL" :"https:\/\/wma.local\/wma_reg2.php",
"sitename" :"Test of the Registration Phase",
"description":"Welcome to the test of the Registration prototype. Please choose a password to finish the registration.",
"imgURL" :"https:\/\/wma.local\/logo.png",
"passwordLabel" :"Password",
"password2Label" :"Confirm your Password",
"failURL" :"https:\/\/wma.local\/registration\/error.php"
}

Fig. 6. JSON response sent by the registration service of the PHP application

2) The web application registration service responds with a JSON object containing the salt generated using the Random Generator of a Safe cryptographic Salt per session (RGSCS [31]) and other settings used thereafter (Figure 6).
3) The extension displays a notification bar that tells the user that the application support the registration protocol (Figure 7).

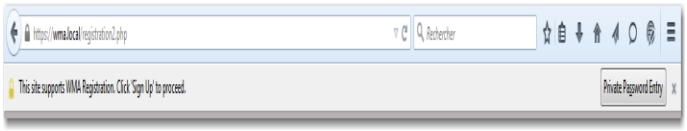


Fig. 7. Notification bar used to create a trusted path to the private password window

4) The user clicks the button bar "Private Password Entry" to display the password private window (Figure 8). The extension uses the settings (site name, description, imgURL, passwordLabel, password2Label, failURL) containing in the JSON object (Figure 6) to customize the information displayed in this window.
5) In this step, the user chooses a password, confirm it, and click on the "OK" button. If there is an error (e.g., the passwords are different, the user clicks the button without entering a password, etc.), the notification bar displays the corresponding error message and a button "Try again "to try again (Figure 9). Otherwise, the protocol proceeds.

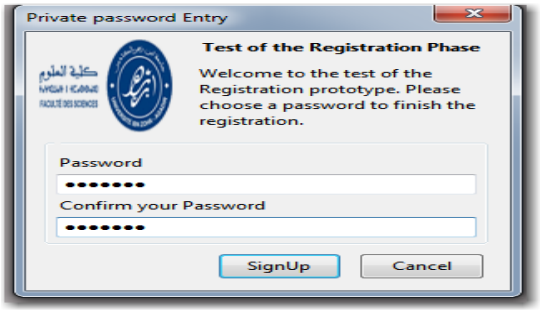


Fig. 8. Private window to choose password safely

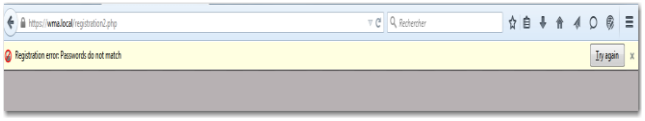


Fig. 9. Error message displayed by our extension using the Mozilla Firefox Notification standard API

6) The CCS extension generates a pair of RSA keys and sends the public key to the web application (Figure 10).

https://wma.local/wma_reg2.php
POST /wma_reg2.php HTTP/1.1
Host: wma.local
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:37.0) Gecko/20100101 Firefox/37.0
….
msg=wmaClientRegistrationExchange&version=wma_v1&sessionRegID=NjAwMzg4MTY0ODY2ODc3MQ%3D%3D&username=ahmed &
clientPublicKey=-----BEGIN%20CERTIFICATE-----

%0D%0AMIICOTCCASGgAwIBAgIBFjANBgkqhkiG9w0BAQUFADAi M
QswCQYDVQQGEwJVUzET%0D%0AMBEGA1UECgwKSlMtVEVTVC1
DQTAeFw0xMzA1MDEyMzU5NTlaFw0yMzA1MDEyMzU5%0D%0ANTl
aMCIxCzAJBgNVBAYTAlVTMRMwEQYDVQQDDApUZXN0IFVzZXIx
MIGfMA0GCSqG%0D%0ASIb3DQEBAQUAA4GNADCBiQKBgQDDiEY
DFtsE196LQRrTnjbaOzw7MsPHwHOK9Rh9%0D%0APgMbswFg3Y4eOr4
P0kDsdzG1X7S1M4guAO6BWGYL32ic4q8wl%2BqEOouMgVacSdah%0D
%0An08bAMpWWik7UjOUfaB6T2JTwL6lsA%2BMA86MPO11764d94%2
BLZaF%2BSs1Pf%2Br7WhrU%0D%0ArUT7QQIDAQABMA0GCSqGSIb3
DQEBBQUAA4IBAQBf3rUVlzLf6huxeNQw8Ju%2BCvpH%0D%0AupdUa
266veguVybpFO5vA1sHeIpCi1W0ew75Mh6kLZz6RUUWcBUePoKAQAcF
RVCI%0D%0A%2BM0tzqUksL7C6NR15UhzNpg8nadyjHx7SxRYkH8v9m
NT%2Fse9WtMQvmSlsZcd1b4k%0D%0AumD9kSictyu%2F2ueM4%2F6nG
xuYb2XDj1iC3MP%2FVNn%2FhYqT%2FJelVPGn%2FVQT1INAqIp4%0
D%0Am0%2Fxkjtmmhoy%2FlUtbO%2FeEuI86MuLchT7V0JgkYoSiLvi8Ss
m0xcZEXScojDCpHsx%0D%0AErMcCoA6DHVX0fgtGzSCQrsuUUekLxm
hA6hdCu7ot6YBTpkDbYihzrbqzV0B%0D%0A-----
END%20CERTIFICATE-----%0D%0A

Fig. 10. Message sent by the registration of extension containing the public key and other parameters related to the current user's session

7) To improve performance, our prototype performs these two operations in parallel:
   - On the one hand, CCS uses the "salt" sent by the application (Step 2) and the password chosen in the previous step, to generate a secret symmetric key via PBKDF. This key is used by the CCS to encrypt RSA keys with AES algorithm (Figures 11 and 12). Subsequently, based on the API of Mozilla Firefox Storage, *UCAM* recorded in the *CS* authentication settings associated with that specific application (Figure 13). The code of the insertion function UCAM can create the file from the embedded SQLite database. This facilitates the use and deployment.
   - On the other, the registration service calculates and stores in the application database the identification information associated with that user (Figure 14).

Fig. 11. The encrypted private key of the user Ahmed

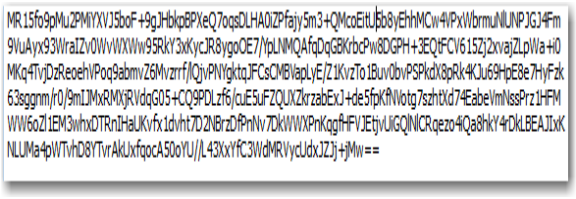


Fig. 12. The encrypted public key of the user Ahmed

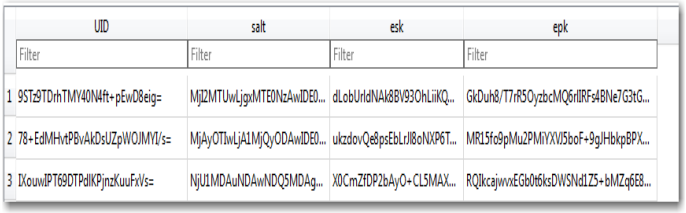


Fig. 13. Client-side authentication settings for three users

Fig. 14. Server side authentication settings for three users

8) After the success of previous operations and before redirecting the user's private application, the extension uses the Mozilla Firefox notification bar to inform the user of the success of the procedure (Figure 15).

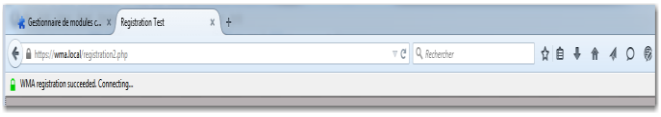


Fig. 15. Successful message displayed using the Mozilla Firefox Notification standard API

B. *Authentication Module*

The authentication module of your prototype integrates two related phases of our architecture: (1) initial connection and mutual authentication phase and (2) HTTP requests authentication or session management phase.

Similar to the registration module, we implement the client component of this module as an extension of Mozilla Firefox. For this, we need:

- 1270 lines of chrome / privileged JavaScript code: Approximately 750 lines for client cryptographic service implementation based on the js_ctypes API, which provides access to Firefox cryptographic library (NSS), and 520 lines to include other client component functionality, as well as interaction with the server component.
- 85 lines of XUL code: 70 lines of XUL code to define the interface of the private window (Figure 17). Also, we use an overlay file of about 15 lines of XUL code

to integrate the components of the extension in the Firefox user interface.

- Other extension configuration files (chrome.mnifest, intall.rdf ....).

In addition, the implementation of server cryptographic services and interaction with the client component has required about 200 lines of PHP code.

In detail, the implementation of the proposed prototype contains the following steps:

1) When our extension detects the support of the proposed protocol. It displays a notification bar asking the user to click on the "Login" button (Figure 16). As shown in Figure 16 unregistered users should first create a new account.

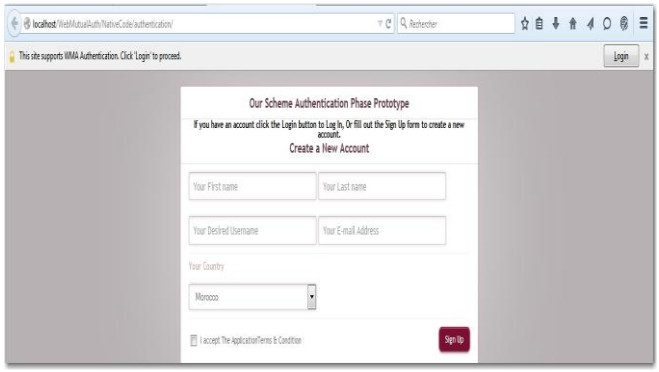


Fig. 16. Notification bar indicating the support of the mutual authentication protocol

2) The user clicks the "Login" button. This displays a private login window (Figure 17) to enter his username and password. As we have explained in the registration module, each application is free to customize the information to display for their users (logo, site name ...).

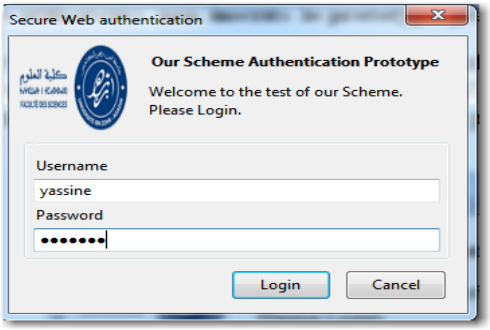


Fig. 17. Private window to safely enter the user's password

3) Once the user enters his/her username / password and click "Login", the UACM checks for authentication parameters associated with that user. In the normal case, the extension retrieves the corresponding user's authentication settings. Otherwise, the extension displays an error message (Figure 18).

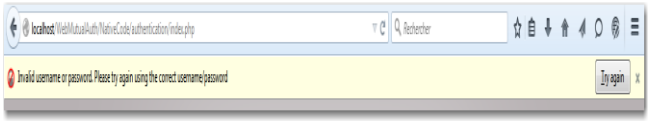


Fig. 18. Example of an error message asking the user to retry the authentication procedure by clicking the button "Try again"

4) The *CCS* execute a series of cryptographic operations to identify the user with authentication service application via a digital signature based on its private key and established a pre-session key (PSK_i). Figure 19 shows a part of the request sent by the extension to the authentication service.

http://localhost/WebMutualAuth/NativeCode/authentication/wma.php

POST /WebMutualAuth/NativeCode/authentication/wma.php HTTP/1.1

Host: localhost

User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:37.0) Gecko/20100101 Firefox/37.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

…

username=yassine **&**

***encryptedPSK*=**
qZgN0nXc5xBYsjy5XQwMjCfu%2Bl%2FNp0sRT7qmcqa8Th9C9Wcog
wlaxoQdk1esMTMi9WLANdT67PM
Hs1yrV4Wa0BdUOWpxTfVRJSg2ewn4aBkpuOX52VuxF7IO%2BLdZo
5caJlDSZDJZOS86kP%2F6%2FOYKI6R0m00AeZwrgSk513ornQebd%
2BWY%2FmqJuVgZD4PQVYHa%2Bjr4E2RffLQ4ILZ%2BFxbOoBL%
2BtTkA3Gc6IsyxqL7I7TYzDwcJWejEaaVGHW%2BwGY%2Blxz2Kdh
sqXgcy8zZz78QTUiPH1maXM6oWbhu0PLBZ6%2FZo3k%2FtFsVJOk
CXzp118g0BO8xMaiZ1QnVqu5CVzwy5Q%3D%3D **&**

IA **=** 616a595676557730655847567 46c4c4d477139586b352b4b36342
f37566a 36796 97 6594865484b374e583438615
749326269316548664c53516238654f5a784652714d65697a766d6432504
76e55704a4246427a527855436f454e655131345a773149356c735761523
264496b45474f676d385135555952764c513866674857614968327a324f4
e6177422f4235716b3959766766763846387078544273 6a7253774d536c4
8756d6d6b622f41347771337756734174 2f784b4c655275455a346751765
13852316675765838584 82b52736a384e753946766a572b79433776534e6
76b4d4b3579415651546a39794230493265494e59796f37612b57776e616
e4232724f5044724b734c536a75413333745a626c3672536a595969526f5
342564c7671446e3239594c5a52504e796d625657726c4f39504459726f5
236757546786c4865734c4b73514a34446c4f5237496848334f413d3d

….

Fig. 19. Part of the HTTP request sent by the authentication extension to the web application authentication's service

5) The *SCS* verifies the cryptographic settings of user authentication, and uses the user public key and other session parameters to authenticate mutually with the browser.
6) The extension checks the authenticity of the web application and displays a message indicating that mutual authentication is successful (Figure 20).

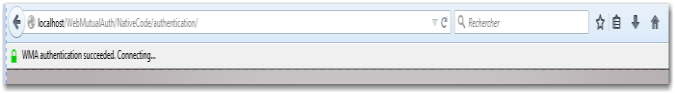


Fig. 20. Message in Mozilla Firefox notification bar tells the user that the mutual authentication is executed successfully

After mutual authentication both parties establish a session key (SK_i). Since this shared key is known only by the authentication extension and the web application, it cannot be obtained by an attacker. This key is used in the session management phase of the proposed scheme [21]. Specifically, our authentication extension includes an HMAC signature [32], [33] in each HTTP request for the web application. Before sending the requested resources, the application must first validate the HMAC signature using SK_i.

Taking inspiration from cookies that use special HTTP headers [34], we decided to create a new HTTP header called "WMA" using the Mozilla Firefox setRequestHeader function. It is a part of the Mozilla Firefox nsIHttpChannel interface [35], which allows to modify the HTTP requests and responses. Figure 21 illustrates an example of a secure session management mechanism. In this example, the user Yassine wants to access the web page "authors.php". For each request that requires authentication, our extension attaches HMAC signature, as well as other parameters in our new HTTP header "WMA". The Web application checks the HMAC signature to authorize or deny access to the requested resources.

Mozilla Firefox HTTP Request

http://auth.local/authors.php

GET / authors.php HTTP/1.1
Host: auth.local
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:37.0) Gecko/20100101 Firefox/37.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
….
WMA:
NjAwNDE2MDM1OTU3ODc5Mg==;yassine;OTMsODksMTAzLDE5MCwxMTEsNDAsMTY4LDYzLDEwOCwyNDksOTIsMjM3LDEwNSwxMzMsMTg4LDMyLDQyLDIzOCwxMzAsMTU3LDE5MCwxNDEsMjEsMTQzLDI1MywwNzgsMTg4LDIwOSw1MCw3NCwxNTQsMTAy;weB92PeFzMz6Y0IfcnPC3g==
Connection: keep-alive
….

PHP Application HTTP Response

HTTP/1.1 200 OK
…
Server: Apache/2.2.21 (Win32) mod_ssl/2.2.21 OpenSSL/0.9.8t PHP/5.3.10
X-Powered-By: PHP/5.3.10
….

Fig. 21. Successful HTTP request authentication

IV. Experimental Evaluation

In this section, we present the results of the experimental evaluation of our prototype. Indeed, in [21], the authors performed a comparative and theoretical evaluation of the proposed scheme regarding computational complexity. Specifically, they showed that in comparison of related schemes, the proposed scheme is efficient and present the lowest computational complexity. On the one hand, HTTPS is only required in the user registration phase. After that, the other phases are running on HTTP. On the other, as we discussed, even if the initial mutual authentication phase requires expensive cryptographic operations (especially asymmetric cryptography) increasing the computing time, the session management phase will need only a negligible overhead. Thus, the main objective of this evaluation is to confirm the results above by conducting performance tests on our prototype. This performance depends on several parameters; including the ability of the processor and memory, the cryptographic algorithms, and the bandwidth of the network.

A. Materials and Algorithms

The material used in our experiments consists of a server; HP Intel (R) Core (TM) i5-3230M CPU 2.60 GHz with 4 GB RAM running on Windows 7 and a client; laptop Accent Genuine Intel (R) CPU 1.3 GHz with 2 GB RAM Windows 7 Professional Version.

TABLE III. Summary of cryptographic algorithms used in our tests at the server, the PHP application, and Mozilla Firefox

	Apache web server (HTTPS)	**Application PHP**	**Mozilla Firefox**
Asymmetric Cryptography	RSA-2048 bits	RSA-2048 bits	RSA-1024 bits
Symmetric Cryptography	AES-128	AES-128	AES-128
Hash function	SHA-256	SHA-256	SHA-256
HMAC Function	HMAC_SHA256	HMAC_SHA1 HMAC_SHA256	HMAC_SHA1 HMAC_SHA256
Key derivation function	-	-	PBKDF 2 with 1000 iterations.

We use a default configuration for all server and client software (i.e., no performance optimizations). All tests are performed in the context of a Fast Ethernet local area network (flow rates up to 100 Mbit / s). The average ping time on the network was 35.25 ms with a standard deviation of ± 20.013 ms. The cost of the cryptographic processing is evaluated by considering an implementation of the algorithms listed in Table III.

- *RSA keys with 1024 bits and 2048 bits as asymmetric cryptography algorithm*: Since RSA is based on the difficult problem of factoring large numbers, RSA key size is often a very controversial subject. Officially the largest factored number is 768 bits. Therefore, the use of 1024-bit RSA key is considered sufficient to guarantee practical security [36]. Nevertheless, not to be placed just outside the known attack capabilities, security agencies such as NIST and ANSSI recommend in their latest reports using RSA with 2048 bit [36], [37]. Therefore, in our prototype, the PHP application, and the web server uses RSA-2048 bit. In return, we chose a RSA-1024 bit for the users. Indeed, in the proposed architecture, the RSA public key of the user is neither transmitted nor stored in clear during registration and user's authentication. In the client and server sides, the public key is stored encrypted, and its transmission to the web application requires a secure connection (HTTPS). This complicates brute force

attacks which require the knowledge of the public part of the RSA key.

- *AES-128 as symmetric encryption algorithm*: Today, AES specified in FIPS 197 [38] is the standard used in most security protocols (TLS, IPsec, etc.). The ANSSI and NIST recommend at least a 100-bit key for data to be protected until 2020, but ANSSI indicates in their report [38] that the use of a 128-bit key is preferable.
- *SHA-256 as a hash function*: Following multiple attacks against the SHA1 algorithm, the majority of applications decided to move to SHA-256 [39].
- *HMAC_SHA1 and HMAC_SHA256 as HMAC function*: While hash functions such as MD5 and SHA-1 are not longer considered safe due to reported collision attacks [40, p. 1], [41]. They may be used in the HMAC functions. HMAC does not require a collision-resistant hash function for its formal security proof [42], [43]. The use of more robust functions like SHA-2 [39] give more security guaranteed, but at a price in the performance level.
- *PBKDF 2 with 1000 iterations as key derivation function*: The use of a key derivation function that requires N iterations to get key increases the calculation cost to perform a dictionary attack on a password with t bits entropy form 2^t operations to 2^t * N operations. Therefore, it makes dictionary attacks and brute force more difficult. However, the computation required for the key derivation by legitimate users also increases with the number of iterations. Thus, there is an obvious compromise: A large number of iterations makes attacks more expensive, but affects performance for the authorized user. PBKDF Version 2 is defined in RFC 2898 [44]. NIST recommends a minimum of 1000 iterations [45].

B. Results and Discussion of the Registration Module

Table IV summarizes the execution time of our registration module, compared with the traditional registration (based on a username and password on HTTPS). The time reported is the average of 10 timings. The total run time includes the time of round trips and networks latency. We calculated the time needed to perform cryptographic operations both client side (Mozilla Firefox registration extension) and PHP applications side. This allowed us to assess the impact of these calculations on performance. As we can see in Table 4, the average total time performance of our proposal is 544.347 ms (with a standard deviation of 113.16 ms). This time is calculated starting the moment the user clicks the Sign up button (Figure 8), after entering their password to the success of this phase (Figure 15). It is clear that our registration module requires more time compared to the traditional registration of users (172.680 ± 32.085 ms). Of course, this is due to the operations integrated into our solution to enhance the security of this phase. Specifically, client-side cryptographic operations require 303.303 ms (± 80.607).

TABLE IV. AVERAGE EXECUTION TIME IN MS (± STANDARD DEVIATION) OF OUR REGISTRATION MODULE, COMPARED WITH THE TRADITIONAL USER'S REGISTRATION

Operation	Our Registration Module	Traditional Registration (Passwords over HTTPS)
Client cryptographic computations	303,303 ± 80,607	-
PHP application cryptographic computations	0,185± 0,018	0,127 ± 0,012
Total runtime	544,347 ± 113,16	172,680 ± 32,085

Figure 22 shows that over 96% of the client time is spent for the RSA key pair generation. Accordingly, the more we increase RSA keys length and the numbers of iterations of PBKDF, performances are affected. At the application side, we obtained almost similar performance to traditional registration. The origin of this small difference is the addition of a generation of a random value and two concatenation operations in our proposal. These are used to avoid storing the unencrypted public key of the user in the application database.

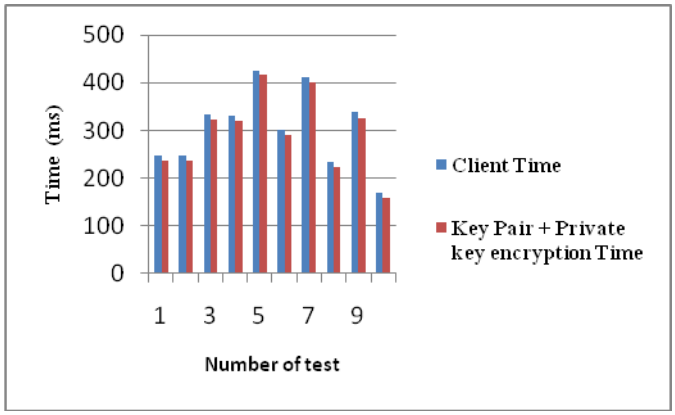


Fig. 22. Comparison of the generation time of 1024 bits RSA keys pair and the time required for other client-side operations

C. Results and Discussion of the Authentication Module

Table V summarizes the results of run time of our authentication module in comparison with traditional authentication (HTML form + HTTPS) and SSL / TLS client certificate authentication. From this table, we can clearly notice that compared to other mechanisms, our authentication module has the lowest total run time (about 148.415 ms 20.315 ms ±) and client certificate authentication SSL / TLS has the highest time (2923.5 ± 350.589). These results confirm those obtained during the theoretical performance analysis in [21].

Indeed, as we have already explained, the authentication phase of the proposed architecture does not require HTTPS, but relies on cryptographic parameters to enhance user authentication over an HTTP connection such as symmetric and asymmetric cryptography. The calculation of these parameters on the client side of our prototype takes 43.862 ms (2,376 ms ± standard deviation) and 15.268 ms (5,701 ms with standard deviation) on the web application side. While the proposed solution has an impact on performance that we think

is acceptable the performance compared to password authentication that requires only 0.065 ms in the application-side and no client-side computing; but security always has a cost, and we believe that the price of insecurity is much higher. Also, these calculations are not likely to affect the user experience.

TABLE V. AVERAGE EXECUTION TIME IN MS (± STANDARD DEVIATION) OF OUR AUTHENTICATION MODULE COMPARED WITH AUTHENTICATION BASED ON AN HTML FORM AND A PASSWORD OVER HTTPS AND WITH THE CLIENT SSL / TLS CERTIFICATE AUTHENTICATION.

Operation	Our Authentication Module	SSL/TLS client certificate authentication
Client cryptographic computations	43,862 ± 2,376	-
PHP application cryptographic computations	15,268 ± 5,701	-
Total runtime	148,415 ± 20,315	2923,5±350,589

Also, as discussed in [21], to create a secure session management mechanism, the proposed scheme attaches an HMAC signature in each HTTP request from the browser to the web application. This ensures the integrity and authenticity of HTTP requests. To assess the impact of this mechanism, we measured the time required to generate an HTTP request signed and time to validate it by the application. Table 6 presents the average time and standard deviations of our prototype in ms when using an HMAC_SHA1 and HMAC_SHA256 functions, compared with traditional session management (the use of cookies sent over HTTPS).

Again, the results in Table VI reaffirm those in [21]. In particular, it is clear to see that the computation time added by generating and validating a HMAC_SHA1 or HMAC_SHA2 is negligible compared to the total execution time (page load time). In other words, the user experience is not affected. Also, despite the cookies by session management requires no cryptographic operations on both the client side and the application side, but the use of HTTPS increases the total execution time required to load a requested page in a user (about 175.680 ms).

TABLE VI. COMPARING THE RESULTS OF BOTH HMAC_SHA1 AND HMAC_SHA256 FUNCTIONS APPLIED DURING THE SESSION MANAGEMENT

	Our authentication Module: Session management phase		HTTP cookies authentication over HTTPS
Operation	**HMAC_SHA1**	**HMAC_SHA256**	
Client cryptographic computations	1,846±0,246	1,974±0,24	-
PHP application cryptographic computations	0,0744± 0,036	0,098±0,0283	-
Total runtime	61,64±19,832	65,3593 ± 33,04	175,680 ± 42,124

V. CONCLUSION

In this paper, we demonstrated the implementation feasibility and experimental evaluation of a secure and efficient authentication scheme. We first presented the details of our proposed prototype implementation. Specifically, we separated the prototype in two modules to simplify the implementation process: A registration module that implements the registration phase and an authentication module which incorporates both mutual authentication and session management phases. In each module, the client component of the proposed prototype is developed as an extension of Mozilla Firefox browser that can easily install and the server component as a service of a PHP web application. This allowed us to avoid recompiling the source code of Mozilla Firefox and have an independent server component of the application code; which also facilitated the deployment procedure. After that, we focused on the experimental evaluation of the proposed prototype. Our experimental results confirmed the proposed scheme-theoretical analysis. In fact, even if the registration phases and mutual authentication of our prototype require expensive cryptographic operations (especially asymmetric cryptography) increasing the computing time, the session management phase will need only a negligible overhead. Compared to the related scheme, we showed that the proposed scheme not only improves the usability and deployability, but also improves the user authentication performances.

REFERENCES

[1] K. Fu, E. Sit, K. Smith, and N. Feamster, “The Dos and Don’ts of Client Authentication on the Web.,” in *USENIX Security Symposium*, 2001, pp. 251–268.

[2] M. Weir, S. Aggarwal, M. Collins, and H. Stern, “Testing metrics for password creation policies by attacking large sets of revealed passwords,” 2010, pp. 162–175.

[3] J. Bonneau and S. Preibusch, “The Password Thicket: Technical and Market Failures in Human Authentication on the Web.,” presented at the The Ninth Workshop on the Economics of Information Security, 2010.

[4] J. Bonneau, C. Herley, P. C. van Oorschot, and F. Stajano, “The quest to replace passwords: A framework for comparative evaluation of web authentication schemes,” presented at the 2012 IEEE Symposium on security and privacy, San Francisco, 2012, pp. 553–567.

[5] J. Bonneau, “The science of guessing: analyzing an anonymized corpus of 70 million passwords,” presented at the 2012 IEEE Symposium on Security and Privacy, San Francisco, 2012, pp. 538–552.

[6] B. Grawemeyer and H. Johnson, “Using and managing multiple passwords: A week to a view,” *Interact. Comput.*, vol. 23, no. 3, pp. 256–267, May 2011.

[7] D. Sandler and D. S. Wallach, “<input type=‘password’> must die,” presented at the Web 2.0 Security & Privacy, 2008.

[8] D. Florencio and C. Herley, “A large-scale study of web password habits,” presented at the the 16th international conference on World Wide Web, New York, 2007, pp. 657–666.

[9] E. Grosse and M. Upadhyay, “Authentication at Scale,” *Secur. Priv. IEEE*, vol. 11, no. 1, pp. 15 – 22, 2013.

[10] D. Florêncio, C. Herley, and B. Coskun, “Do strong web passwords accomplish anything,” presented at the 2nd USENIX Workshop on Hot Topics in Security, Boston, 2007.

[11] S. Grzonkowski, W. Zaremba, M. Zaremba, and B. McDaniel, “Extending web applications with a lightweight zero knowledge proof authentication,” New York, 2008, pp. 65–70.

[12] D. Stuttard and M. Pinto, *The web application hacker’s handbook finding and exploiting security flaws*. Indianapolis: Wiley, 2011.

[13] Y. Sadqi, A. Asimi, and Y. Asimi, “A Cryptographic Mutual Authentication Scheme for Web Applications,” *Int. J. Netw. Secur. Its Appl.*, vol. 6, pp. 1–15, 2014.

[14] M. Manulis, D. Stebila, and N. Denham, "Secure Modular Password Authentication for the Web Using Channel Bindings," in *Security Standardisation Research*, vol. 8893, L. Chen and C. Mitchell, Eds. Springer International Publishing, 2014, pp. 167–189.

[15] Y. Sadqi, A. Asimi, and Y. Asimi, "A Lightweight and Secure Session Management Protocol," *Lect. Notes Comput. Sci.*, vol. 8593, pp. 319–323, 2014.

[16] IETF, "RFC 5246 - The Transport Layer Security (TLS) Protocol Version 1.2," 2008. [Online]. Available: http://tools.ietf.org/html/rfc5246.

[17] M. Wu, S. Garfinkel, and R. Miller, "Secure web authentication with mobile phones," presented at the DIMACS workshop on usable privacy and security software, 2004, pp. 9–10.

[18] B. Parno, C. Kuo, and A. Perrig, "Phoolproof phishing prevention," presented at the Financial Cryptography and Data Security (FC'06), Anguilla, British West Indies, 2006.

[19] Google, "About 2-step verification - Accounts Help." [Online]. Available: https://support.google.com/accounts/answer/180744?hl=en&ref_topic=1099588.

[20] Mozilla, "Mozilla Persona — simple sign-in with email — mozilla.org." [Online]. Available: http://www.mozilla.org/en-US/persona/.

[21] Y. Sadqi, A. Asimi, and Y. Asimi, "A Secure and Efficient User Authentication Scheme for the Web," *Int. J. Internet Technol. Secur. Trans.*, vol. 5, no. 1, pp. 43–63, 2015.

[22] Mozilla, "js-ctypes - Mozilla | MDN." [Online]. Available: https://developer.mozilla.org/en-US/docs/Mozilla/js-ctypes.

[23] Mozilla, "NSS - Mozilla | MDN." [Online]. Available: https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS.

[24] Mozilla, "Storage | MDN." [Online]. Available: https://developer.mozilla.org/en-US/docs/Storage.

[25] SQLite, "SQLite." [Online]. Available: https://www.sqlite.org/.

[26] Apache Foundation, "Apache HTTP Server Project." [Online]. Available: http://httpd.apache.org/.

[27] Apache Foundation, "mod_ssl - Apache HTTP Server Version 2.2." [Online]. Available: http://httpd.apache.org/docs/2.2/mod/mod_ssl.html.

[28] PHP, "Hash: Hash - Manual." [Online]. Available: http://php.net/manual/en/book.hash.php.

[29] PHP, "OpenSSL: PHP OpenSSL - Manual." [Online]. Available: http://php.net/manual/en/book.openssl.php.

[30] Oracle, "MySQL :The world's most popular open source database." [Online]. Available: https://www.mysql.com/.

[31] Y. Asimi, A. Amghar, A. Asimi, and Y. Sadqi, "New Random Generator of a Safe Cryptographic Salt per session (RGSCS)," *Int. J. Netw. Secur.*, vol. 18, no. 3, pp. 445–453, 2016.

[32] M. Bellare, R. Canetti, and H. Krawczyk, "Message authentication using hash functions: The HMAC construction," *RSA Lab. CryptoBytes*, vol. 2, no. 1, pp. 12–15, 1996.

[33] RFC 2104, "HMAC: Keyed-Hashing for Message Authentication," 1997. [Online]. Available: https://www.ietf.org/rfc/rfc2104.txt.

[34] IETF, "RFC 6265 - HTTP State Management Mechanism," 2011. [Online]. Available: http://tools.ietf.org/html/rfc6265. [Accessed: 08-May-2015].

[35] Mozilla, "nsIHttpChannel - Mozilla | MDN." [Online]. Available: https://developer.mozilla.org/en-US/docs/Mozilla/Tech/XPCOM/Reference/Interface/nsIHttpChannel. [Accessed: 08-May-2015].

[36] ANSSI, "Référentiel Général de Sécurité: Règles et recommandations concernant le choix et le dimensionnement des mécanismes cryptographiques," 2014.

[37] E. Barker and A. Roginsky, "Transitions: Recommendation for transitioning the use of cryptographic algorithms and key lengths," *NIST Spec. Publ.*, vol. 800, p. 131A, 2011.

[38] NIST, "FIPS 197: ADVANCED ENCRYPTION STANDARD (AES)," *Process. Stand. Publ.*, Nov. 2001.

[39] NIST, "FIPS 180-4: Secure Hash Standard (SHS)," *Process. Stand. Publ.*, 2012.

[40] X. Wang, Y. L. Yin, and H. Yu, "Finding collisions in the full SHA-1," in *Advances in Cryptology–CRYPTO 2005*, 2005, pp. 17–36.

[41] X. Wang and H. Yu, "How to break MD5 and other hash functions," in *Advances in Cryptology–EUROCRYPT 2005*, Springer, 2005, pp. 19–35.

[42] S. Turner and L. Chen, "RFC 6151 - Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms," 2011. [Online]. Available: https://tools.ietf.org/html/rfc6151.

[43] M. Bellare, "New proofs for NMAC and HMAC: Security without collision-resistance," in *Advances in Cryptology-CRYPTO 2006*, Springer, 2006, pp. 602–619.

[44] RSA Laboratories, "PKCS #5 v2.1: Password-Based Cryptography Standard," 2000. [Online]. Available: https://www.ietf.org/rfc/rfc2898.txt.

[45] M. S. Turan, E. Barker, W. Burr, and L. Chen, "Recommendation for password-based key derivation," *NIST Spec. Publ.*, vol. 800, p. 132, 2010.