A PAKE – SRP6 BROWSER EXTENSION

Alexandru Gavril Bardas[1]

ABSTRACT

The username/password paradigm is a well-known authentication mechanism. Probably the most common version in use is the password authentication via an HTML form. The user has to type his/her password directly into a web page from the site to which he/she wishes to authenticate himself/herself. The problem with using this approach is that it relies on the user to determine when it is safe to enter his/her password. If the user authenticates himself/herself to a phishing website by disclosing his/her password, the password is stolen even though the session is fully encrypted. In other words in traditional password authentication, passwords are used only for client-side authentication. Password-authenticated key exchange (PAKE) on the other hand, offers password-based mutual authentication. This mutual authentication is different because its client-side authentication cannot be separated from its server-side authentication part.

This paper shows that PAKE can represent a practical alternative approach to protect passwords without relying on a Public Key Infrastructure (PKI). Therefore, the goal of this work was to study how to integrate PAKE into web applications, not to develop a standalone PAKE implementation. We analyzed the PAKE client-side implementation within a web browser and tested it with a server-side implementation on a web server. The developed extension is a Mozilla Firefox web browser extension. The implementation is just a proof of concept that shows that a password authenticated key exchange can be done over HTTP and can be used against phishing attacks.

1. INTRODUCTION

The username/password paradigm is by far the most commonly used authentication mechanism [3]. From on-line banking and different web accounts (university account, Netflix account etc.) to personal emails the use of passwords is present everywhere. There are additional or alternative authentication mechanisms (tokens, biometrics), but they often require additional hardware, which is most of the time considered too expensive for an application.

Passwords are weak secrets because their security is limited by its low-entropy. A carefully chosen password has only about 20-30 bits entropy [3][5]. Therefore passwords are subject to dictionary attacks or even to a simple exhaustive search. Forcing users to choose and remember cryptographically strong passwords may cause more problems than it solves [3][5]. Since they are weak secrets, passwords have to be protected during transmission. SSL/TLS is the widely deployed method used for this purpose. However this method requires a Public Key Infrastructure in place and may be subject to a main-in-the-middle attack [3].

[1] bardasag@ksu.edu, Kansas State University

Probably the most common authentication mechanism in use is the password authentication via an HTML form [4]. The user has to type his/her password directly into a web page from the site to which he/she wishes to authenticate himself/herself. The problem with using this approach is that it relies on the user to determine when it is safe to enter his/her password. If the user authenticates himself/herself to a phishing website by disclosing his/her password, the password is stolen even though the session is fully encrypted [3].

Most of the time in order to resist phishing and other social engineering attacks, a user uses and relies on the browser's security indicators and warning messages. For instance the URL bar and the site's SSL certificate should be used to authenticate the website and determine when it is safe to enter the password. However studies suggest that many users habitually click through SSL certificate warnings and misunderstand or ignore browser security indicators [4]. In other words the client-side does not authenticate the server-side.

In traditional password authentication, passwords are used only for client-side authentication. Password-authenticated key exchange (PAKE) on the other side, offers password-based mutual authentication. This mutual authentication is different because its client-side authentication cannot be separated from its server-side authentication part. If one of these two authentication processes fails, the whole authentication protocol will fail and the session will be aborted.

The fact that these two parts are not separable is not very common. For instance, as stated before, the combination of passwords for client-side authentication and certificates for server-side authentication in SSL/TLS is being widely used on the web. However, this combination can be separated: the client-side password-based authentication assumes a pre-existing secure channel to the server, which is accomplished through prior server-side authentication. If the server-side authentication is bypassed due to an incorrect checking, the client-side authentication will be insecure. As a result, this approach is vulnerable to phishing attacks. On the other hand, if implemented correctly, PAKE, which does not assume any pre-existing secure channels, will not be vulnerable to this kind of phishing attacks.

This work describes the details of integrating PAKE in to the Web. We did not develop a stand-alone implementation of PAKE. Our goal was to study how to integrate PAKE into web applications. In other words we analyzed the PAKE client-side implementation within a web browser and tested it with a server-side implementation on a web server. SRP version 6 is the authentication protocol used in PAKE. The developed extension is a Mozilla Firefox web browser (latest tested version 16.0.2) extension and was tested using a PHP server-side implementation on Apache. In case the authentication succeeds the traffic between the web server and the client will be encrypted using a common shared session key.

2. BACKGROUND

A **P**assword-**A**uthenticated **K**ey **E**xchange (PAKE) protocol is a cryptographic protocol that allows two parties that share a common secret (a password) to mutually authenticate each other and establish a shared key, without explicitly revealing the password in the process [4].

The PAKE research represents an alternative approach to protect passwords without relying on a Public Key Infrastructure (PKI). This research aims to achieve two important goals. First, it allows zero-knowledge proof of the password without revealing it to the other party. Second, it performs authenticated key exchange [3]. In case the password is correct, both sides will be able to establish a common session key that no one else is able to compute.

In 1992 the first important milestone was reached in PAKE research. Bellovin and Merrit[5] introduced the Encypted Key Exchange (EKE) protocol. Even though this protocol had some reported weaknesses [3] it showed that the PAKE problem was solvable. In the following years a number of PAKE protocols have been proposed. Many of these protocols were variants of EKE.

The few techniques that claim to resist known attacks have been patented by different companies or by Stanford University: EKE was patented by Lucent Technologies, SPEKE by Phoenix Technologies and SRP by Stanford University [3].

The Secure Remote Password (SRP) protocol was first proposed by Tom Wu in 1998. The protocol has been revised several times and is currently at version 6. Previous versions of the protocol are: SRP 3 in Wu98, SRP in RFC 2945 and SRP 5. The SRP protocol fulfills the following properties: it allows mutual authentication between the server and the client (server explicitly authenticates the client and vice-versa), it is resistant to online and offline dictionary attacks mounted by an eavesdropper, and it does not require a trusted third party (no PKI infrastructure is needed). The protocol creates a large private key shared between the two parties in a manner similar to Diffie–Hellman. Furthermore, it verifies to both parties that the two keys are identical and that both sides have the correct password. SRP version 6 is used for strong password authentication in TLS-SRP and other standards such as EAP [8], and is being standardized in IEEE P1363 [12] and ISO/IEC 11770-4 [13].

Hao and Ryan proposed a protocol called J-PAKE, which authenticates a password with zero-knowledge and then subsequently creates a strong session key if the password is correct. The authors claim that the protocol does not require PKI deployment and protects users from leaking passwords. Moreover they claim that the protocol has computational efficiency comparable to the EKE and SPEKE schemes with clear advantages on security. However they are not comparing J-PAKE with SRP due to different design goals.

The goal of using PAKE protocols for web authentication is to help make it easier for users to authenticate websites and reduce the attack surface of social engineering based attacks against their accounts [4]. When using a PAKE protocol, in case the user mistakenly attempts to authenticate himself/herself to a phishing website, the protocol will fail and the user password will remain safe. The phishing website does not have the correct password in order to successfully complete the protocol and establish a session key with the user.

Engler et all perform a systematic investigation of various practical issues and challenges in deploying PAKE for web authentication. They claim that although many PAKE protocols have been proposed, there is a little momentum for integrating PAKE protocols

into web authentication. They investigated two categories of issues: 1) security issues related to user interface (UI) design and 2) deployment and integration hurdles [4].

Currently there are two proposals for PAKE-based web authentication: TLS-SRP (operates at the transport layer), and HTTPS-PAKE (operates at the application layer).

The TLS-SRP extension proposes an integration of PAKE into TLS. Although it can be a good solution for closed applications like VPN or IPP, it is not convenient for general web systems [1]. TLS server authentication is not very powerful because certificates (especially so-called "class-1" certificates) can be acquired by any party including phishers. Phishing using HTTPS certified by publicly-accepted PKI is already a real issue, because users do not always pay attention to the subject of issued certificates [1]. Moreover TLS-SRP might not work properly in sites that employ load balancing HTTPS front-ends. Deploying TLS-SRP in this type of architecture will require new network protocols to manage user sessions between HTTPS front-ends and back-end application servers. Another problem with TLS-SRP is that it does not explicitly support authentication to multiple realms within a single domain [4].

Oiwa et. all address the drawbacks of TLS-SRP and propose to implement PAKE at the application layer, over HTTPS. Similar to *Engler et all*, I will refer to this approach as HTTPS-PAKE. The HTTPS-PAKE protocol design includes using the Public Key Infrastructure. The protocol is inspired by HTTP Digest Authentication, which authenticates users within the HTTP protocol. HTTPS-PAKE works by tunneling the PAKE protocol messages over HTTP, in additional HTTP headers. This approach gives the application layer control over authentication policies without any interaction with TLS. However, HTTPS-PAKE still relies on TLS to provide a secure channel between browsers and servers. When using HTTPS-PAKE authentication, a user's browser first establishes a TLS connection with the server. Then, over the encrypted TLS session, the browser and the server perform a PAKE handshake and establish a PAKE key derived from the user's password [1][4].

HTTPS-PAKE might be vulnerable to stronger threats such as pharmers and network attackers in case the user ignores certificate warnings (as many studies suggest) [4].

3. PAKE-SRP 6 EXTENSION

I. Goals

Our primary goal was to implement PAKE at the application layer using SRP 6 as the PAKE protocol (We will refer to this approach as PAKE-SRP 6). We created an extension for a web browser (Mozilla Firefox latest tested version is 16.0.2) that uses PAKE-SRP 6. In order to test the browser extension we used a web server (Apache 2.2.21) and PHP to write the server-side PAKE-SRP 6 part. In case the authentication succeeds and the two sides share a common session key, the traffic between the web server and the client will be encrypted.

Design goals:

- The SRP-6 protocol should be strictly followed
- The authentication will fail in case one of the sides (client or server) is fake
- Implement a mechanism for users to detect spoofing windows
- Web traffic will be protected after mutual authentication using K (a secret session key that only the client and the server know)
- A usual browser will not be able to know and use the key in order to decrypt the content of the page before displaying it

We are not claiming that this extension is suited for all needs and computer users.

II. Comparison

An alternative solution to my PAKE-SRP 6 browser extension is presented by *Oiwa et. all.* The extensions that they propose for IE and Firefox are using HTTPS-PAKE [11]. HTTPS-PAKE implies using the Public Key Infrastructure (PKI).

The Web Wallet approach proposed in *Wu et. al*. is slightly different from my PAKE-SRP 6 browser extension. The Web Wallet is a browser sidebar, which users can use to submit their sensitive information online [9]. Wu's approach displays the password box only when the browser detects a login page or a user requests to login. Our extension will appear on the screen only if the user clicks the extension button. In this way we are hoping to minimize the effects of a mimicry attack. Such an attack has proved to be efficient on the web wallet approach.

Currently Mozilla Firefox developers work on a project that uses the J-PAKE protocol. They are describing J-PAKE as being a technique that allows Alice and Bob to mutually authenticate and agree upon an encryption key, using only a pre-shared secret value. It is a zero-knowledge protocol, meaning this can be done without ever exposing the secret value to the other party [10]. This project might also constitute an alternative to PAKE-SRP 6. However the PAKE-SRP 6 extension will open a separate browser window designed to communicate with websites that support SRP 6. The extension will be launched by the user and not automatically by any applications.

4. SECURE REMOTE PASSWORD PROTOCOL VERSION 6 (SRPV6) OVERVIEW

The latest version of the SRP protocol is version 6 and as previously mentioned it is used for strong password authentication in TLS-SRP and other standards such as EAP [8]. Moreover it is being standardized in IEEE P1363 [12] and ISO/IEC 11770-4 [13]. The protocol can be summarized as follows:

	CLIENT (w)		SERVER (U, s, v)
1		U $\Rightarrow$	Look up s, v
2	$x = h(s, w)$	$s \Leftarrow$	
3	$a \in_R [1, p]$, $A = g^a \bmod p$	A $\Rightarrow$	If A≡0 quit
4	If B≡0 quit	B $\Leftarrow$	$b \in_R [1, p]$, $B = 3v + g^b \bmod p$ $u = h(A \parallel B)$
5	$S_C = (B - 3g^x)^{a+ux} \bmod p$		$S_S = (Av^u)^b \bmod p$
6	$K_C = h(S_C)$		$K_S = h(S_S)$
7	$M_1 = h(A, B, S_C)$	$M_1 \Rightarrow$	(verify M_1)
8	(verify M_2)	$M_2 \Leftarrow$	$M_2 = h(A, M_1, K_S)$

System parameters: ***p***, ***g***

w stands for the user's password; ***U*** for user ID; ***s*** for salt

v for Password Verification Data (***v*** = g^x mod p, where x = *hash* of *s* and *w*)

In practice ***v*** = $g^x \bmod p$ where x = SHA(<salt> | SHA(<username>)|":"|<raw_password>)

If the authentication succeeds than in Step 6 S_c will be equal to S_s, that means that the client and the server share a common session key. Step 7 explicitly authenticates the client. The server calculates $M_1' = h(A, B, S_s)$ and checks whether $M_1 = M_1'$; if it's not the case than it quits. On the other hand Step 8 explicitly authenticates the server. The client calculates $M_2' = h(A, M_1, K_c)$ and checks whether $M_2 = M_2'$. The client quits in case the equality does not hold.

5. IMPLEMENTATION

The PAKE-SRP 6 extension is a Mozilla Firefox browser extension. The extension installation file format is *.xpi* and can be installed in Firefox by opening the Add-on menu and choosing the option "Install from a file" (more details on installing and testing the extension are presented in the Appendix). After the installation is completed, an icon will show up in the main browser next to the *Mozilla Firefox Start Page* icon. In case the user clicks on the PAKE-SRP 6 icon a new bowser window will open. This browser window is designed to support the SRP 6 protocol. The browser window has an URL address bar, a username field, a password field and a Login button. In order to be able to test the extension we also developed a "test" (toy) server that supports the SRP 6 protocol and is written in PHP. The extension can also load web pages that do not support PAKE-SRP 6

authentication but a normal browser will not be able to load a webpage from a server that supports PAKE-SRP 6.

PAKE-SRP 6 Firefox Extension

Several tools for developing Firefox extensions proved to be very helpful in the development process of the PAKE-SRP 6 extension:

- XUL Explorer 1.0a1pre (used for designing the graphical interface of the extension)
- Developer Assistant 0.3.0.20110927 (Firefox extension that contains a suite of tools for developers: an interactive JavaScript shell, HTML and XUL editors with live preview etc.)
- Firebug 1.8.3 (Firefox extension that enables users to edit, debug, and monitor CSS, HTML, and JavaScript live in web pages)

Primarily we used XUL Explorer to design the graphical interface and Javascript to specify the actions of the extension. Text editors like TextWrangler for Mac and Notepad++ for Windows were helpful in managing the Javascript code. The interactive JavaScript shell included in the Developer Assistant and Firebug were very useful for testing certain code snippets before integrating them with the rest of the code.

The Firefox Extension ***PAKE-SRP6.xpi*** is actually a *.zip* file and it contains the following folders and files:

- chrome:
  - content:
    - images
    - extension_icon.png
    - BigInt.js *(Big Integer Library v. 5.4 created by Leemon Baird)*
    - browser.xul
    - extension_interface.xul
    - extension_actions.js
    - hex.js
    - rc4.js
    - webtoolkit.sha1.js *(Secure Hash Algorithm (SHA1) - http://www.webtoolkit.info/)*
- chrome.manifest
- install.rdf

The structure of the extension file is specific to Firefox Extensions: a folder named *chrome* and two files *chrome.maifest* and *install.rdf*. At installation time, the two files contain important information for the Firefox browser. The location of the extension files, the supported browser versions and author name are some of the details included in these files.

The *content* folder located in *chrome* contains the files that build the actual extension:

- Graphical interface:
  - *browser.xul* (graphical interface in the main browser, the PAKE-SRP 6 icon that appears in the main browser),
  - *images folder* contains the icon image that is used in the main browser,
  - *extension_interface.xul* (extension GUI – a new browser window with a URL address bar, a username field, a password field and a Login button)
- Actions (Javascript):
  - *extension_actions.js* (performs the client side SRP 6 protocol computations and manages the communication with the server side)
  - *BigInt.js* (Big Integer library is used for the computations performed in *extension_actions.js*)
  - *hex.js* (used for encoding/decoding data to/from hexadecimal format from/to bytes sequence)
  - *rc4.js* (used for encrypting/decrypting text given a key using the RC4 stream cipher)
  - *webtoolkit.sha1.js* (secure hash function, SHA1, used for hashing strings)

In order to establish and maintain a communication with a server that supports the PAKE-SRP 6 extension, we used *XPCOM* for Firefox.

XPCOM is a cross platform component object model. It has multiple language bindings, letting the XPCOM components be used and implemented in JavaScript, Java, and Python in addition to C++. Interfaces in XPCOM are defined in a dialect of IDL called XPIDL. XPCOM itself provides a set of core components and classes, e.g. file and memory management, threads, basic data structures (strings, arrays, variants), etc.[20]. For the PAKE-SRP 6 extension, creating sandboxed HTTP connections [22] and File I/O [23] were two valuable resources during the development process. Using the examples and code snippets provided, we were able to send and receive data over HTTP to/from the server side. In this way the extension is able to follow closely the client-side SRP 6 protocol description provided in Section 4.

Server supporting PAKE-SRP 6

The server application for testing the Firefox PAKE-SRP 6 extension is written in PHP with the *php_gmp* extension installed and enabled. This PHP extension includes functions that allow users to work with arbitrary-length integers using the GNU MP library. This is very helpful for performing the server side computations of the SRP 6 protocol, especially when 1024-bit numbers like the system-wide parameter *p* are used in the calculations. For more information on how PHP was installed and configured, follow the directions provided in the Appendix section.

The test server **Server-*PAKE-SRP6*** contains the following folders and files:

- Server-PAKE-SRP6:
  - Crypt:
    - RC4.php
  - PAKE-SRP6-server.php

The *RC4.php* file is part of the PHP library, *phpseclib* [26], and implements the RC4 stream cipher. In case the authentication succeeded and a common session key is shared between the two sides, this file is used to encrypt the content that will be sent to the client.

The *PAKE-SRP-server.php* file is the actual test server for the PAKE-SRP 6 extension. This file contains the computation details of the server side SRP 6 authentication protocol and also manages the communication with the client side. Furthermore in order to make the client side aware that it supports PAKE-SRP 6, the server sends the following unencrypted content the first time the client side loads the webpage:

```
<PAKE-SRP6><html><body> … <body></html>
(For instance: <PAKE-SRP6><html><body>This page requires support for
PAKE-SRP 6 over HTTP.</body></html>)
```

By attaching the <*PAKE-SRP6*> before the HTML content, the client side extension will know that the SRP 6 authentication protocol is supported and therefore it can start the authentication process. In case the authentication process succeeds, the client side will receive the encrypted content from the server. The server will use the RC4 stream cipher and the session key that it shares with the client to encrypt the webpage content.

Interaction between the Server Side and the Client Side

The client side is initiating the communication with the server. The user opens the PAKE-SRP 6 extension and types the URL of the server, and the user's credentials (username and password). After clicking on the *Login* button, the extension will initiate a connection to the server. Upon receiving this request, the server will send the unencrypted HTML content to the client. As mentioned before, the unencrypted content is preceded by <*PAKE-SRP6*>. When the client receives the server's response, it knows that the server-side supports PAKE-SRP 6 authentication. Therefore it will not display the unencrypted content and will follow the protocol steps stated in Section 4. If the received content does not contain the PAKE-SRP 6 header (<PAKE-SRP6>), the browser extension will display an alert telling the user that the SRP 6 authentication protocol is not supported and the received unencrypted content will be displayed in the extension window.

In case PAKE-SRP is supported, the first step is sending the username that the user entered. When the server receives this information it will send back the salt for that specific username. In case the username does not exist, the test server will send a fake salt number back. Upon receiving the salt, the client is able to calculate *A* specified in the protocol description and send it to the server. In case the received *A* is not empty or zero, the server computes *B* and sends it back to the client. Now both sides are able to calculate the common session key and exchange M_1 and M_2 in order to verify each other's authenticity. If the authentication process fails at any point, the extension will display an alert informing the user that the authentication failed and the most recent HTML unencrypted content that was received will be displayed in the browser extension (*"This page requires support for PAKE-SRP 6 over HTTP."*). Some examples why the authentication process might fail: *A* or *B* are empty or zero, M_1 or M_2 cannot be successfully verified or unexpected communication

errors. In case the authentication process is successfully completed, the server will encrypt the webpage content using the common session key and send it to the client-side. The client's extension will decrypt the content and display it in the extension window.

6. EVALUATION

Security of the Implementation

The implementation follows the SRP 6 protocol specification; the authentication process will fail in case one of the sides (client or server) is fake. Furthermore the user is able to detect spoofing windows. If the user doesn't click on the *PAKE-SRP 6* extension icon, the extension window will not pop up. In case the same icon is used for a spoofing window then this means that another extension had to be installed by the user on her/his system. Therefore a similar icon will not appear in Firefox unless the user installs another extension.

Web traffic is protected after mutual authentication using a secret session key that only the client and the server know. In the implementation, the shared key is used only once by the server and by the client side (only one encrypted webpage content is sent to the client). Therefore the RC4 stream cipher is used in a secure fashion. Attackers will not be able to know and use the key in order to decrypt the content of the webpage. If the user and the server want to use the same session key for multiple encrypted contents then a block cipher like AES should be used. This will slightly complicate the implementation when it comes to using an encrypted channel (most probably the initialization vector, IV, and possible other information has to be sent between the two parties).

Limitations

SRP 6 alone does not provide a way to establish a common long-term secret between the client and the server side. The two sides have to share a common long-term secret (user's password in this case) that is used every time to generate a new session key. Obviously in case an attacker obtains the password for a specific username then she/he can impersonate both the client and the server side. However if the attacker steals the PVD, *v*, then she/he can mount an off-line dictionary attack. SRP 6 is subject to server compromise based dictionary attack. Moreover, in case the attacker obtains the username and the salt (which are sent in plain text) for a certain PVD that it obtained from a compromised legitimate server then it can impersonate that server.

The provided Firefox browser extension and the test server are not meant to be used for cryptographic purposes beyond testing. First of all, the libraries and classes used for big integers, random numbers and encryption were not rigorously verified. In other words the random number generators may not be secure and may leak information or the big integer classes may behave unexpectedly in certain situations.

This implementation is just a proof of concept that shows that a password authenticated key exchange (using the SRP version 6 authentication protocol) can be done over HTTP and can be used against phishing attacks.

7. SUMMARY

Our primary goal was to integrate PAKE into a web browser by using SRP 6 as the PAKE protocol. There are several web browser extensions available but none of them uses SRP 6. We developed an extension for Mozilla Firefox web browser that uses PAKE-SRP 6. In order to test the browser extension we used a web server (Apache) and PHP to write the server-side PAKE-SRP 6 part. In case the authentication succeeds and the two sides share a common session key, the subsequent communication is done over an encrypted channel.

8. ACKNOWLEDGEMENT

I would like to thank Haroun Macalisang for his contribution in the very first version of the implementation and Dr. Eugene Vasserman for his feedback on this work.

REFERENCES

1. Y. Oiwa, H. Takagi, H. Watanabe, PAKE-based Mutual HTTP Authentication for Preventing Phishing Attacks, Madrid, Spain, 2009
2. R. Gennaro and Y. Lindell, A Framework for Password-Based Authenticated Key Exchange, Advances in Cryptology — EUROCRYPT 2003
3. F. Hao, P. Ryan, J-PAKE: Authenticated Key Exchange Without PKI, Transactions on Computational Science XI Volume 6480/2010
4. J. Engler, C. Karlof, E. Shi, D. Song, Is it too late for PAKE?, W2SP Oakland, 2009
5. R.J Anderson, Security Engineering: A Guide to Building Dependable Distributed Systems, New York, Wiley 2001
6. S. Bellovin and M. Merritt, Encrypted Key Exchange: password-based protocols secure against dictionary attacks, IEEE Symposium on Research in Security and Privacy, May 1992
7. T. Wu, The Secure Remote Password Protocol, Internet Society Symposium on Network and Distributed Systems, 1998
8. J. Carlson, B. Aboba, H. Haverinen, EAP SRP-SHA1 Authentication Protocol, IETF Draft, 2001
9. M. Wu, R.C. Miller, G. Little, Web Wallet: Preventing phishing attacks by revealing user intentions, SOUPS, 2006
10. Mozilla Wiki - https://wiki.mozilla.org/Firefox/Projects/J-PAKE
11. Research Center for Information Security - https://www.rcis.aist.go.jp/special/MutualAuth/software/MutualTestFox/index-en.html
12. IEEE P1363 - http://grouper.ieee.org/groups/1363/
13. ISO/IEC 11770-4 (Information technology, Security techniques, Key management, Part 4: Mechanisms based on weak secrets) - www.nhzjj.com/asp/admin/editor/newsfile/2010318165259678.pdf
14. The How-to on developing Firefox Extensions - http://www.rietta.com/firefox/Tutorial/prefs.html
15. XUL Explorer- https://developer.mozilla.org/en/XUL_Explorer
16. Developer Assistant - https://addons.mozilla.org/en-US/firefox/addon/extension-developer/
17. FireBug - https://addons.mozilla.org/en-US/firefox/addon/firebug/
18. DOM Inspector - https://developer.mozilla.org/En/DOM_Inspector
19. Dr. Xunhua Wang –Slides: Password-authenticated Key Exchange: SRP and SPEKE
20. XPCOM API Reference - https://developer.mozilla.org/en/XPCOM_API_Reference

21. XPCOM Interfaces - https://developer.mozilla.org/en/XUL_Tutorial/XPCOM_Interfaces
22. MDN Creating Sandboxed HTTP Connections - https://developer.mozilla.org/en/Creating_Sandboxed_HTTP_Connections
23. MDN File I/O - https://developer.mozilla.org/en/Code_snippets/File_I%2F%2FO
24. MDN Miscellaneous - https://developer.mozilla.org/en/Code_snippets/Miscellaneous
25. MozillaZine - http://forums.mozillazine.org/viewtopic.php?f=27&t=448212
26. Phpseclib - http://phpseclib.sourceforge.net/
27. WAMP Server - http://www.wampserver.com/en/
28. Firefox - http://www.mozilla.org/en-US/firefox/new/

APPENDIX

Installing Apache, PHP, Firefox and the PAKE-SRP 6 Extension on Windows (tested on 7 and XP SP3)

Step 1

In case you don't have Apache and PHP (with the *php_gmp* extension enabled) installed on your system, install *WampServer2.2* [27] with default settings.

Enable *php_gmp* extension (library) in WAMP (Figure 1):

1. Start WAMP (double click on the desktop icon)
2. Click on the Wamp icon taskbar icon
3. Click on *PHP*
4. Click on *PHP extensions*
5. Click on *php_gmp*

Figure 1 – Enable php_gmp extension

Copy the **Server-PAKE-SRP6** folder in C:\wamp\www

Step 2

In case you don't have Firefox installed on your computer, download Firefox [29] and install it using default settings. The PAKE-SRP 6 extension was **tested and is working** on version **16.0.2**

Step 3

Install the Firefox extension by dragging and dropping the ***PAKE-SRP6.xpi*** file on the opened Firefox window (Figure 2).

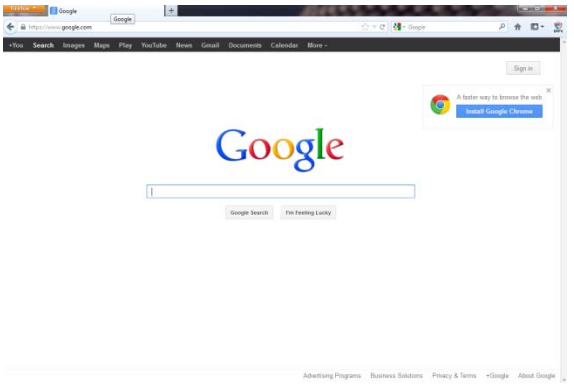

Figure 2 – Correctly installed Firefox Extension

Running the SRP-6 over HTTP implementation

Click on the *PAKE-SRP6* toolbar icon in the Firefox window. A new window will be opened where the user has to type the URL of the server that supports PAKE-SRP6 over HTTP, the username and the password.

The test implementation supports two accounts that are stored on the server. Use:

Usename: **alex** Password: **password**	or	Usename: **bardas** Password: **password1**

Testing the PAKE-SRP 6 extension:

Open Firefox
Click on the *PAKE-SRP 6* extension
Type in the URL of the server that supports the PAKE-SRP 6 (for testing purposes the URL is *http://localhost/Server-PAKE-SRP6/PAKE-SRP6-server.php*)
Type in the username and password (**alex** and **password** or **bardas** and **password1**)
Click Login
If everything goes well you will have to get something similar to Figure 3. This message will appear only if the content of the page was decrypted. Therefore a browser that does support PAKE-SRP 6 will not be able to know the key and use this key to decrypt the message before displaying it.

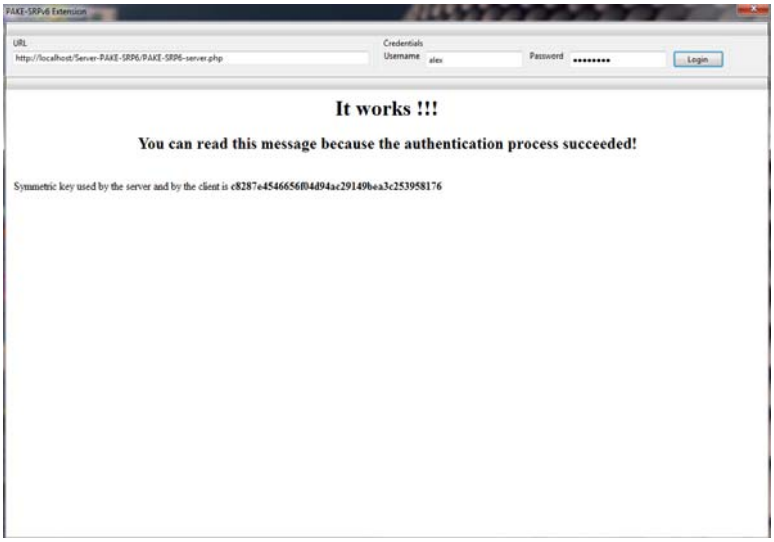

Figure 3 – Working SRP6 over HTTP implementation