INCORPORATING PRIVACY AND SECURITY FEATURES IN AN OPEN SOURCE SEARCH ENGINE

A Project Report

Presented to

The faculty of Department of Computer Science

San Jose State University

In Partial fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Akash Patel

May 2014

SAN JOSE STATE UNIVERSITY

The Undersigned Project Committee Approves the Project Titled

INCORPORATING PRIVACY AND SECURITY FEATURES IN AN OPEN SOURCE SEARCH ENGINE

by
Akash Patel

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Chris Pollett, Department of Computer Science Date

Dr. Sami Khuri, Department of Computer Science Date

Dr. Chris Tseng, Department of Computer Science Date

APPROVED FOR THE UNIVERSITY

Associate Dean Office of Graduate Studies and Research Date

ABSTRACT

INCORPORATING PRIVACY AND SECURITY FEATURES IN AN OPEN SOURCE SEARCH ENGINE

The aim of this project was to explore and implement various privacy and security features in an open-source search engine and enhance the security and privacy capabilities of Yioop. Yioop, an open-source PHP search engine based on GPLv3 license, is designed and developed by Dr. Chris Pollett. We have enabled a crawl, search and index mechanism for hidden services by execution of codes, which has facilitated access of the Tor network in Yioop. We have diversified the ability of the previously supported text CAPTCHA functionality in Yioop by implementing hash CAPTCHA and provided feasibility to toggle between text CAPTCHA and hash CAPTCHA. To enable the user to log in to his or her respective Yioop account without sharing the password over the network, we have incorporated zero knowledge authentications in which Yioop does not store the user's real password, but it stores the numerical password, which is derived from the user's original password.

ACKNOWLEDGEMENTS

I would like to thank Dr. Chris Pollett for being my advisor and guiding me throughout the project. I express my gratitude towards Dr. Sami Khuri and Dr. Chris Tseng for their time and feedback. I thank Ms. Binal Thakkar and Mr.Sanket Sheth for their advice in technical writing. Finally, I would also like to convey my special thanks to my family, and friends for their help and support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

Most modern search engines record a user's IP address, the time of visit and the user's tracking cookies to keep track of searched terms in order to improve efficiency of the search results and to show user specific advertisements. This information can be used to reveal personal information and hence, many people object to its collection.

The goal of this project was to learn about various security protocols and implement security and privacy features in an open-source search engine. We have explained Perfect Forward Secrecy (PFS) and steps to implement it. A PFS checker script written by us can be used to test whether a website supports PFS or not. We have gained knowledge of the Tor network and Tor hidden service, and also covered proxy settings in the Tor network to enable crawl of the Tor hidden service. We have designed hash CAPTCHA, which can be used as a substitute for text CAPTCHA. We incorporated a zero knowledge authentication system, which allows a system to authenticate users without any need for sharing an actual password over the network.

The report is sorted into chapters. Chapter 2 briefly describes Yioop and its features. Chapter 3 explains Perfect Forward Secrecy (PFS) and detailed steps to implement and verify PFS. Chapter 4 presents an overview of the Tor network and Tor hidden service. Chapter 5 illustrates existing CAPTCHA system and implementation of the hash CAPTCHA. Chapter 6 briefs about zero knowledge authentication system. Chapter 7 gives the conclusion based on the results obtains in the previous chapters followed by the references.

CHAPTER 2
SUMMARY OF BACKGROUND WORK IN YIOOP

Yioop, an open source PHP Search engine created by Dr. Chris Pollett. It allows user to index a website or a collection of websites. In Yioop, the user has control over the exact sites which are being indexed with Yioop. Yioop supports the indexing of many different file types including: HTML, Atom, BMP, DOC, ePub, GIF, JPG, PDF, PPT, PPTX, PNG, RSS, RTF, Sitemaps, SVG, XLSX, and XML. It has a web interface for controlling which, amongst these file types (or all of them) user want to index. It supports also attempting to extract information from unknown file types. . It can index page ranges up to tens or hundreds of millions. It is designed to work on PC, Smartphone and tablet.

Yioop supports crawling, indexing and searching on either single machine and or several machines. Yioop crawler can crawl websites and non-web archives. It has two important processes: fetcher and queue server. A fetcher downloads the contents from crawled web pages or non-web archives and processes them. The queue server performs of index building and scheduling activities. It is written using its own model-view-controller framework with a user interface that is simple and user-friendly For further details on Yioop's features user can refer to SeekQuarry website [1].

Dr. Pollett has started developing Yioop in 2010. He is the primary author of Yioop. In the past, few students had incorporated various features in Yioop. Students had implemented new features such as autosuggest and spell check, text summarization, source-code searching capability and geographical location local search, access control in a social networking environment and wordnet feature.

Prior to this project, Yioop could not crawl the Tor network; hence it was not possible to index the Tor hidden services. We modified Fetch URL function of Yioop to provide capability of accessing the Tor network. New code was added in the controllers, models and views to support the hash CAPTCHA and zero knowledge authentication. To calculate SHA1 and Fiat-Shamir parameters new JavaScripts have been implemented. “SHA1_test” case is also written to verify the output of SHA1 JavaScript.

CHAPTER 3
PERFECT FORWARD SECRECY

Perfect Forward Secrecy (PFS) is a property of a key-agreement protocol which guarantees that a session key used to encrypt the data will never be compromised even though a private key is compromised in the future [2].

3.1 Overview of PFS

Let's assume a user accesses a website which uses HTTPS protocol for secure communication. In the beginning of this process, the client requests the secure connection to which the server sends back the certificate detail which contains its public key. The server also has a private key which can be used to decrypt the data encrypted using its public key. The client browser generates a session key and encrypts the data using it. It then encrypts the session key using the public key provided by the server. Along with this data, the client also shares the encrypted session key with the server. The server uses its private key to decrypt the encrypted session key and this key is used to decrypt all the communication messages. In this setup, the session key is known only to the client and the private key is known only to the server. Figure 1 shows the steps involved in SSL handshaking. An attacker can sniff over the network and capture this encrypted data. The attacker needs either the session key which is generated by the client or a private key of the server to decrypt this data. As he is not aware of either of these keys, he will not be able to decrypt the data. However, he can still store all the intercepted messages and later on if he obtains the private key by some technique like breaking into the server computers, he can derive the session key and decrypt all the stored messages. Not all decrypted information is important, but some information like credit card and address details are important even after a few years. This spot is the motivation factor for the PFS.

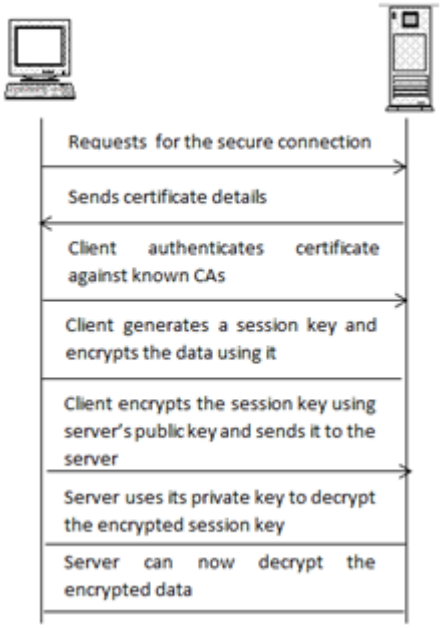


Figure 1: SSL handshaking

In PFS, instead of using the single session key to encrypt all the communication messages, a client (browser) periodically creates a new session key. Both client and server contribute to the key generation and use a random value for each key. The client and server forget this random value after the key creation. Even if an attacker managed to obtain a private key, he would not know the random value generated by the client and the server so could not reconstruct the session key [3].

Derivation of the session key is the key factor to determine whether a connection has PFS or not. While creating a secure connection, the Transport Layer Security (TLS) handshake protocol is responsible for the authentication and key exchange. The TLS protocol essentially performs three key things: cipher suite negotiation, authentication of a server and key exchange [4]. The cipher suite is responsible for the derivation of the session key.

In the beginning of the SSL handshake, the client browser sends a list of supported cipher suites to the server. The server then picks one of the cipher suites, based on the

ranking defined in the SSL configuration file of the server, and then informs the client about the usage of the cipher suite in future communication. Cipher suites, which use ephemeral Diffie-Hellman (DHE) or the elliptic-curve Diffie-Hellman (ECDHE) for the key generation, give PFS.

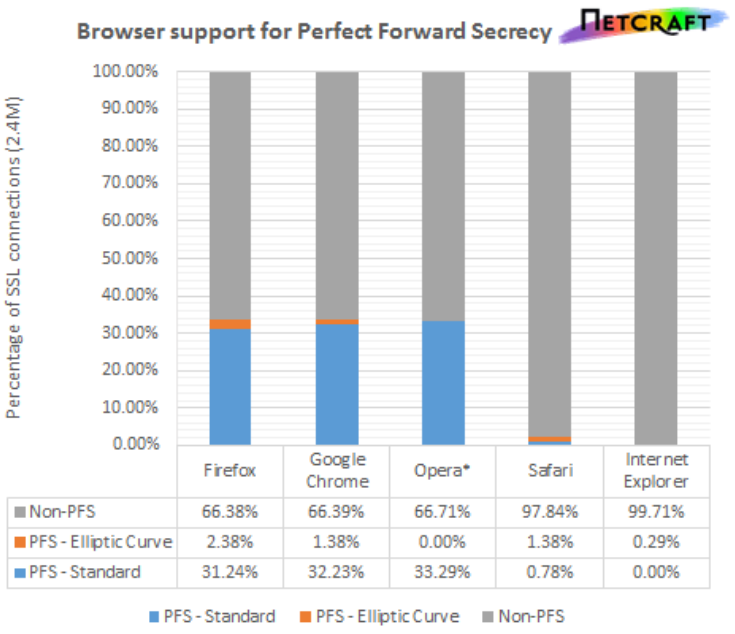
A webserver needs to be configured to enable PFS. Usually the SSL configuration file on the web server has information about a list of cipher suites which are supported by the server. While determining the cipher suite for the connection, the web server starts scanning the list of cipher suites in order and returns the first match which is sent by the client. Thus the order of cipher suites is also important. In order to enable PFS, DHE and ECDHE, cipher suites should be given higher order in the configuration file.

For Apache server, PFS requires Apache version 2.3.3 or higher. Figure 2 shows the configuration setting in the "httpd-ssl.config" file to enable PFS on Apache server.

```
SSLProtocol +TLSv1 +TLSv1.1 +TLSv1.2
SSLCipherSuite ECDHE-RSA-RC4-SHA:ECDHE-RSA-AES128-SHA:RC4-SHA
SSLHonorCipherOrder on
```

Figure 2: Settings in httpd-ssl.config file to enable PFS

There are certain challenges in enabling PFS, the first being that it uses Diffie-Hellman key exchange. In this exchange a new session key is generated for each request. This key generation takes significant time, thus PFS can increase the server response time. The second challenge is from the client's side, which is that not all browsers support necessary cipher suites. Most browsers assign less priority to the cipher suites that support PFS. Figure 3 shows the browser support for PFS (Data is taken from the Netcraft's SSL survey conducted in September 2013).



*Opera does not include its TLS 1.2 cipher suites.

Figure 3: Browser support for PFS [5]

3.2 PFS verification and PFS Checker Script

There are many ways to verify whether a website supports PFS or not. The SSL scan utility and SSL lab website are good sources to verify it. I have used the SSL scan utility to demonstrate that Yioop supports PFS. Figure 4 shows PFS verification of Yioop using the SSL scan utility. As shown in Figure, Yioop prefers a cipher suite DHE_RSA_AES256-SHA for the connection and hence it supports PFS.

PFS checker is a PHP script written by us to verify whether a website supports PFS or not. The input to the script is a URL of the website. The script tries to make the connection with the website using different cipher suites and verifies whether the website supports PFS or not. In the results, all the ciphers marked with the blue color are the ones which support PFS and are supported by the website. The ciphers in red support PFS but are not supported by the website. The “PFS Support” field indicates whether websites

support PFS or not. The snapshot of the result of PFS verification of Yioop is shown in Figure 5.

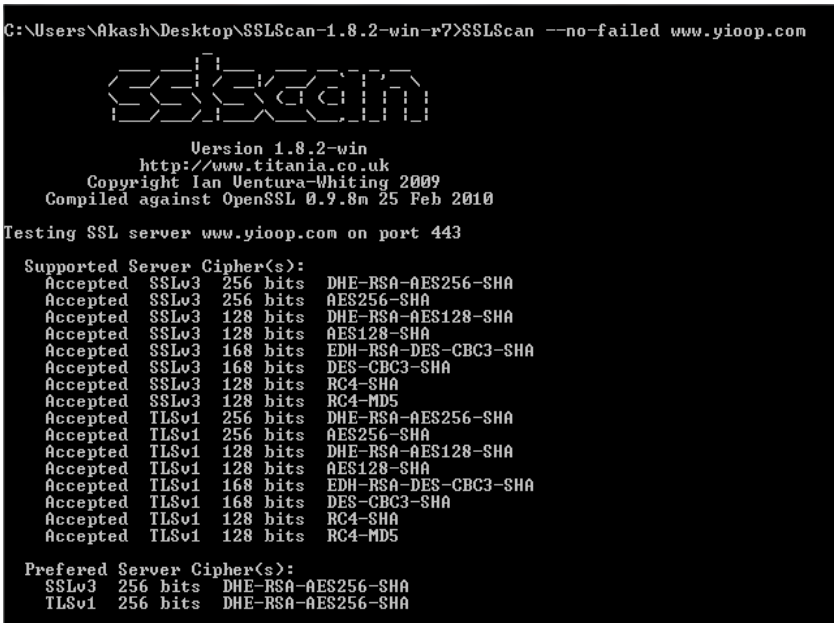
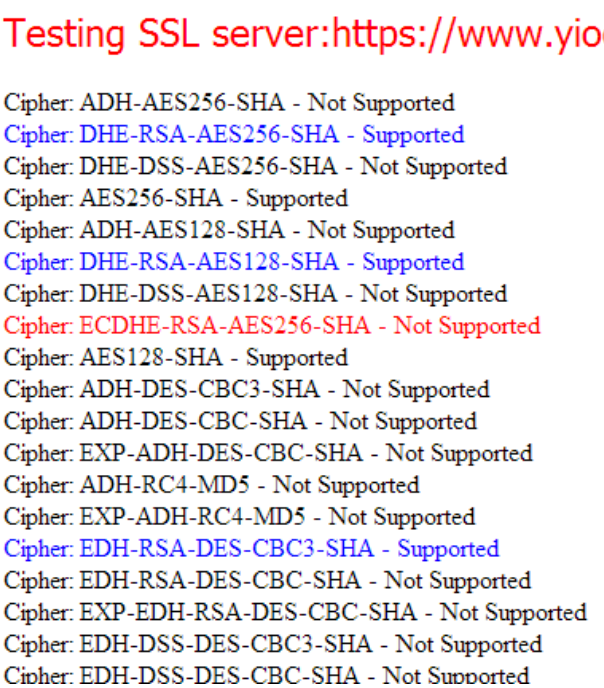


Figure 4: PFS verification of Yioop using SSL scan utility



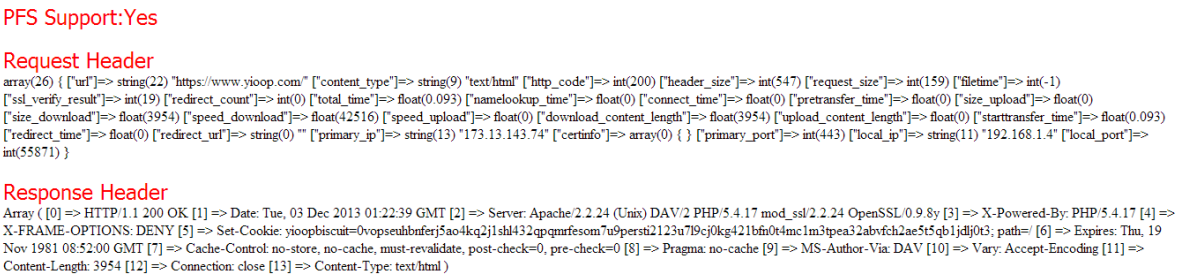


Figure 5: PFS verification of Yioop using PFS checker script

I have used the "SSL Lab Test" website to test which search engines support PFS and the result for the same is as shown in table 1.

Table 1: PFS test on various search engines [6].

Website Name	**Supports PFS or Not**
Yioop.com	Yes
Google.com	Yes
Yahoo.com	Yes
Bing.com	No
Duckduckgo.com	Yes
AOL.com	No

3.3 Conclusion

PFS provides long-term security to the data communicated over the network. It protects previous communications from retrospective decryption. On April 7, 2014 a serious security bug was found in the OpenSSL (which is widely used to implement Transport Layer Security Protocol). It is estimated that half a million sites were affected by this bug and the data of millions was compromised [7]. The impact of the Heartbleed bug could have been alleviated if most of the servers and clients had enabled the PFS.

CHAPTER 4
CRAWLING TOR NETWORK

The Onion Router (Tor) is an open source software used by journalists, the military, corporates, and many others for anonymous internet surfing. Originally it was designed and developed to protect the U.S. navy's confidential communication [8].

Tor network prevents the "Traffic analysis" attack which is a special kind of attack to deduce the pattern information from the communication pattern. In this attack, the attacker observes the flow of data packets between two entities in a system. This attack can be used to identify who is talking with whom on the internet.

4.1 Overview of the Tor network

Tor uses an Onion routing system. It maintains a list of the available machines (a.k.a nodes) and uses them to direct traffic over the Internet. Tor distributes the transaction among different nodes so there is no single point connection between the sender and receiver. Since the transaction is distributed among many nodes, it is difficult to do traffic analysis [9].

For example, User A uses Tor network to send data to user B. User A's Tor client obtains a list of the available nodes from the Tor server and randomly selects three nodes (e.g. Node 1, node 2 and node 3). Then it applies multiple layer of encryption on the data using the public key of the selected nodes so each of the selected nodes can decrypt the incoming packet using their private keys. Instead of sending data packet directly from A to B, Tor network sends encrypted packet from A to Node 1. Tor client on Node 1 decrypts the first layer of encryption and identifies the next node where it needs to send the packet. This process continues till Node 3, where it receives the location of the user B and finally transmits the unencrypted message. Each node in the Tor circuit has

knowledge of only two nodes: the node from which it receives the packet, and the node to which it sends the packet. Randomness in the node selection process is very important. For each request from A to B, Tor client selects different nodes so it would be very difficult for the network interceptor to associate data packets with any node.

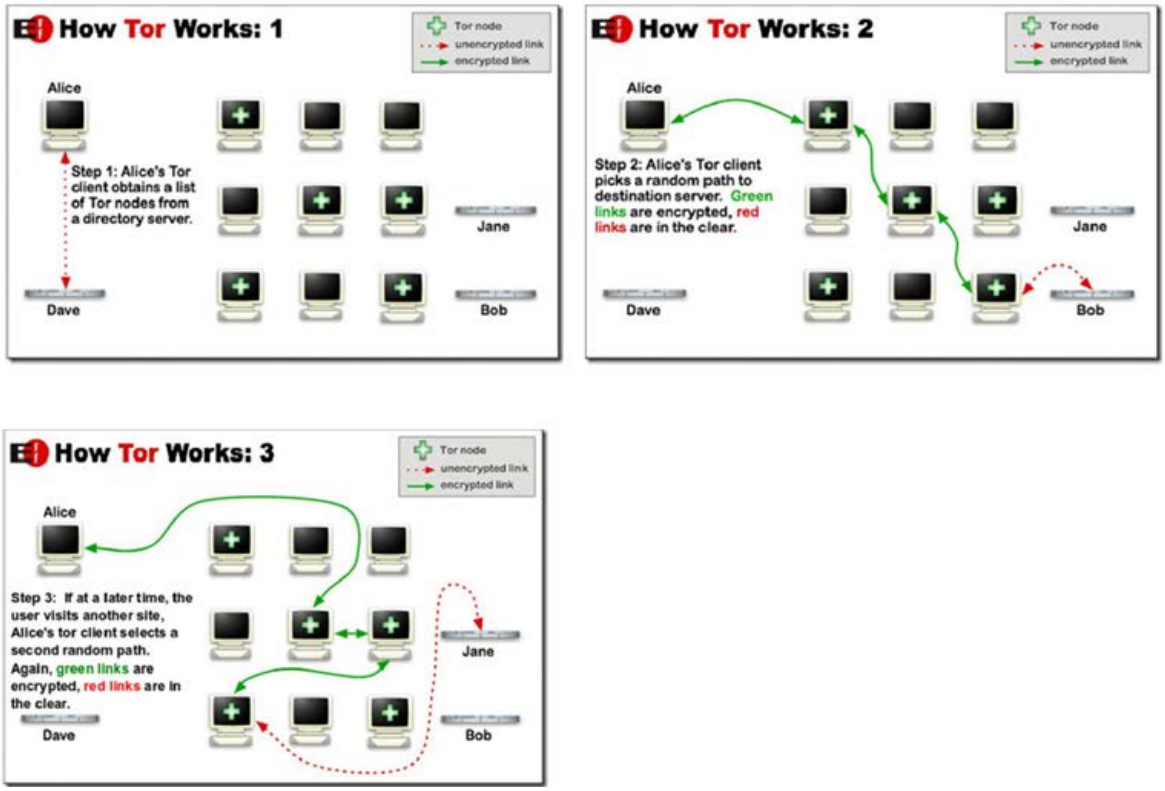


Figure 6: working of the Tor network [10]

4.2 Yioop as a Tor hidden service

Tor hidden service allows publishers to publish the service without revealing their identity (IP address). Users can connect to a hidden service without knowing publisher of the service using the rendezvous point. This type of anonymity provides protection against distributed DoS attacks as an attacker would not know the IP address of a service [11].

We have published Yioop running on localhost as a hidden service and the steps to create it are stated below:

1. Install a web server locally
2. Configure the hidden service to point to the local web server
3. Open the torrc file located at: \Tor\Tor Browser\Data\Tor \torrc
4. Now add an entry like below in the torrc file
    a. HiddenServiceDir D:\Tor\HiddenService
    b. HiddenServicePort 80 127.0.0.1:80
5. Save the torrc file and restart the Tor client

HiddenServiceDir is a place where the Tor stores the information about a hidden service. Tor will generate a hostname file in this folder, which has an Onion URL for the service. Tor has generated the onion URL for this service, which is http://x2emztb4ndxvhzt6.onion/. We have accessed this hidden service using the Tor browser and the result page is as shown in Figure 7.

Figure 7: Example of Tor hidden service

4.3 Crawling Tor network

Our goal was to implement the code in Yioop so it can crawl the Tor network. To achieve this, the first thing, we did was to gain knowledge of the Tor network and Tor hidden service. The next step was to install the Tor client. The Tor bundle is available for all major operating systems. We downloaded the Tor bundle for Windows 7 and configured the proxy settings in the Tor browser. Figure 8 shows various proxy setting options available in Tor.

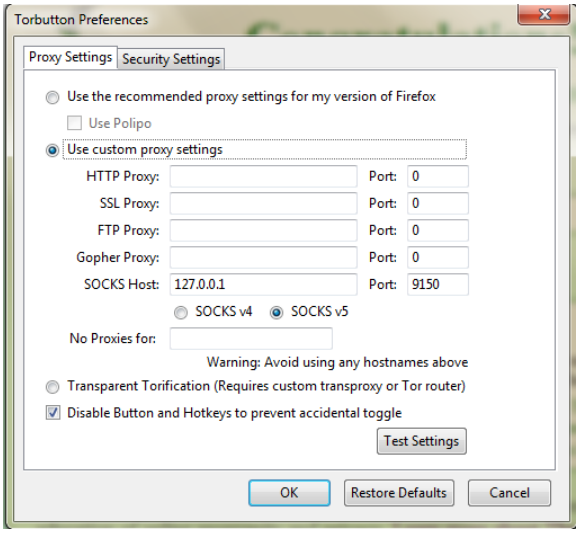


Figure 8: Proxy settings in Tor browser

Once we understood the working of the Tor network, the next step was to implement the code in Yioop so it can crawl the Tor network. We have added code in the functions: “getPages”, “prepareUrlHeaders”, and “getPage” of the fetch_url.php. Tor network has an issue that it increases the access time of the website as it needs to bounce the internet packet among Tor nodes. We have written code in such a manner that during crawl, fetcher will use Tor network only when it is crawling an Onion URL. Figure 9 shows proxy server settings in Yioop. Admin user can configure these settings to use the Tor network during the crawl.

To verify whether Yioop can actually crawl the Onion URL, we did crawl on the onion URL and Yioop was able to crawl it successfully. Figure 10 shows the snapshot of the crawl. After the crawl, we had set up the crawl data as search index and performed the search query. Figure 11 shows the snapshot of the search query and result.

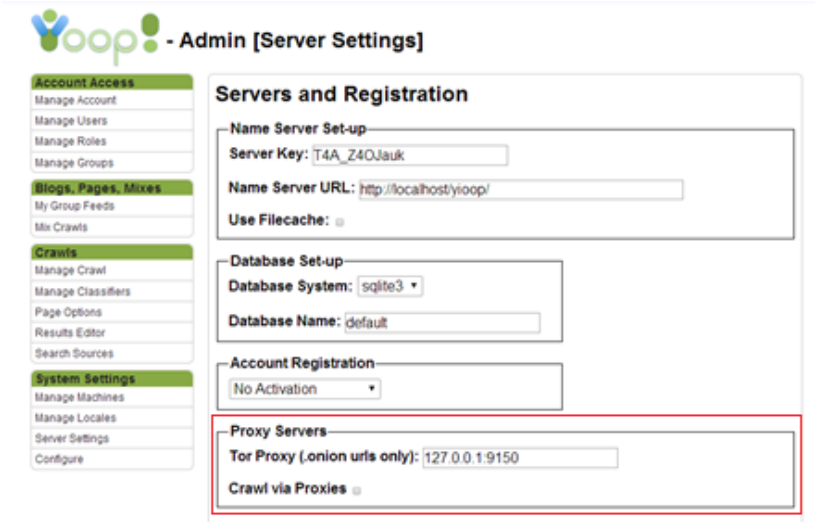


Figure 9: Tor proxy settings in Yioop

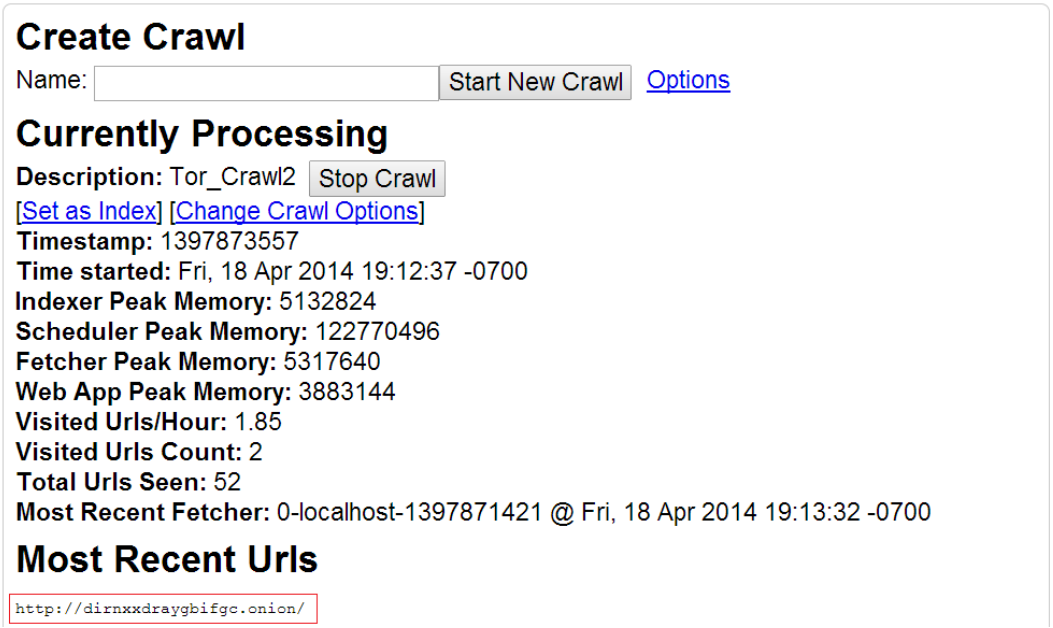


Figure 10: Yioop crawling an Onion URL

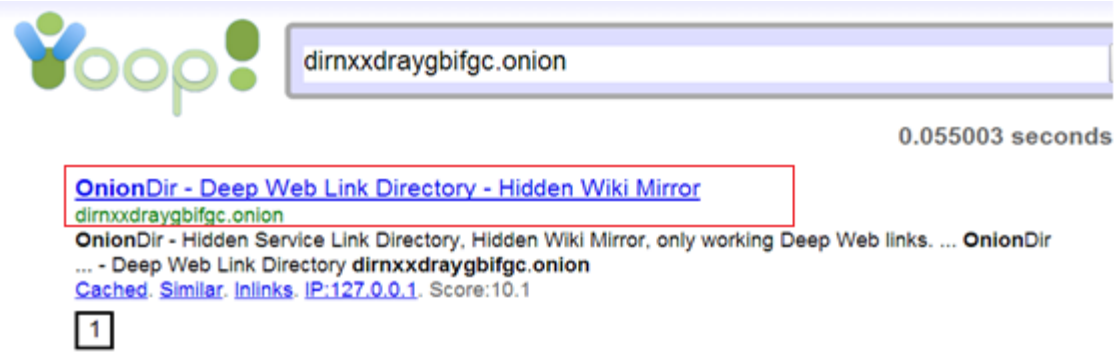


Figure 11: Onion URL search query on Yioop

4.4 Tor proxy link

In the client-server communication, a proxy server acts as an intermediate machine which hides the internal clients from an external network. It is a substitute for connecting directly to the web.

Ixquick is a popular search engine which provides proxy links for each query result, so the users can surf websites anonymously. We have decided to implement a similar kind of feature in Yioop so it can generate a proxy link for each query result. When user clicks on the proxy link, request first comes to the Yioop server, which in turn forwards the request to the website and returns the result page to the user. In this case, Yioop server acts as a forward proxy server, which gets the HTTP request from a client and forwards it to the requested website, thus provides anonymity to the client IP. In addition to that, Yioop server uses Tor connection to make a request to website, so Yioop's identity can be kept hidden.

We have implemented code in "search_view.php" file to create a dynamic proxy link. Also, we have added class "ProxyURL" to receive a URL from the user and to make a request to the website through the Tor network. The snapshot of the proxy link feature is as shown in Figure 12.

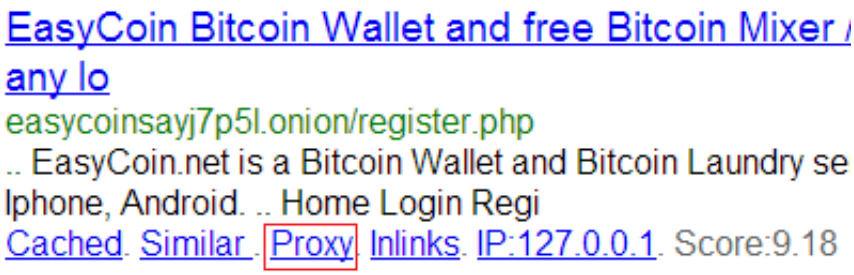


Figure 12: Proxy link feature in Yioop

4.5 Conclusion

Tor network and Tor hidden service have gained popularity in recent times. Duck Duck Go is a popular search engine, which is published as the Tor hidden service [12] and its onion URL is https://3g2upl4pq6kufc4m.onion/. The ability to crawl the Tor network will allow Yioop to index the Tor hidden services and display them in the search result. The proxy link feature will provide anonymous web surfing experience to the user.

CHAPTER 5
HASH CAPTCHA

CAPTCHA stands for Completely Automated Public Turing test to tell Computers and Humans Apart. It is a type of a challenge response test to find whether a user is human or machine [13]. CAPTCHA prevents spam submission by automated software thus improving the quality of the software, and is mainly used in registration functionality, forgot password functionality, and in the comment field for a blog post.

5.1 Existing CAPTCHA system

In today's world, there are different types of CAPTCHAs available. The first and a very popular type of CAPTCHA is a standard distorted word with an audio option. This CAPTHCA uses twisted letters and a background color gradient to hide the message. From a security point of view, this CAPTCHA is reliable as the distorted word is hard to crack by machine. However, sometimes legitimate users also face difficulty in deciphering it. The picture identification CAPTCHA is also very popular. In this CAPTCHA, a few images and a question are shown to a user where he needs to identify the correct image based on the question. This CAPTCHA is very user-friendly. The math-solving CAPTCHA and 3D CAPTCHA are also popular [14].

CAPTCHA implementation is very important factor from a security point of view. In earlier CAPTCHA implementation, the hidden fields were used to store the questions and answers of CAPTCHA on the client side where it was verified using JavaScript. A smart machine can easily read the values of these hidden fields, thus can solve the CAPTCHA. Therefore, any implementation which stores CAPTCHA's data on the client side is highly vulnerable.

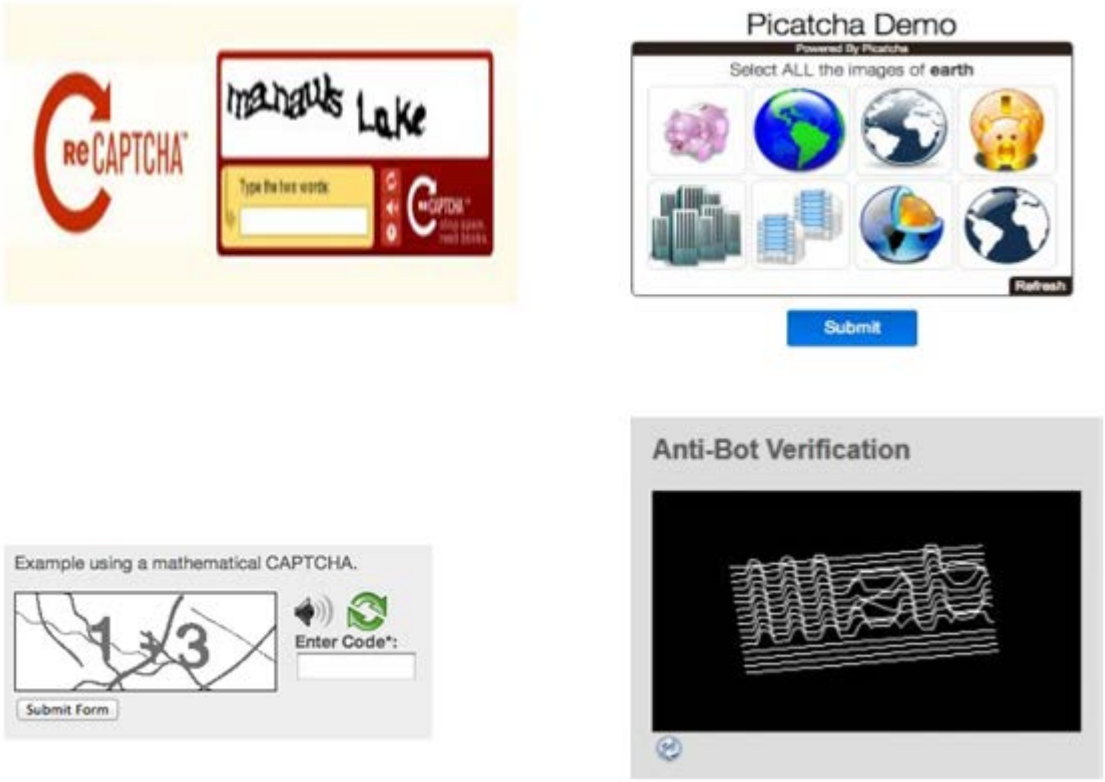


Figure 13: Different types of CAPTCHAs [14]

Recently a few attacks exploited the server-side weakness of CAPTCHA implementation. Randomness in CAPTCHA generation is very important. Theoretically server should not utilize the fixed set of CAPTCHA; it should generate a random CAPTCHA for each new request. Many of the sites use predefined sets of CAPTCHA. Each CAPTCHA is associated with an identifier which can either be a numeric or a fixed-length character string. The rainbow attack can exploit websites which use a static identifier. In this type of attack, the attacker creates the lookup table of CAPTCHA identifiers and its solution. The attacker solves all the CAPTCHAs manually and at runtime uses this table to find the CAPTCHA's answer [15]. Table 2 shows an example of rainbow table.

Table 2: Example of rainbow table [15]

Identifier	CAPTCHA	Solution
0	95C7A	95C7A
1	58412	58412
2	9038F	9038F
3	49F1C	49F1C
4	A8887	A8887
5	K89D	K89D
998	IOP9	IOP9
999	KLO7	KLO7

Figure 14 illustrates users' responses on CAPTCHA. Most of the users bother by the CAPTCHA. Another issue with difficult CAPTCHA is that it not only prevents spam, but also prevents search engine bots. The search engine bot will not be able to parse the web page, thus the page will not appear in the search results. This way it can reduce the popularity of the website.

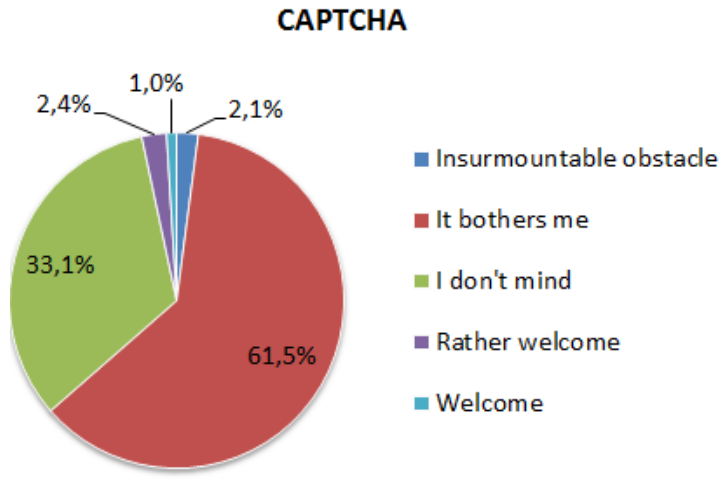


Figure 14: User's response on CAPTCHA

We were looking for some technique that is user-friendly and at the same time does not compromise security. Bitcoin's hashcash implementation has motivated us to implement hash CAPTCHA.

5.2 Cryptographic HASH function:

Hash CAPTCHA uses the concept of a cryptographic hash function. A hash function takes an arbitrary length string as input and returns a fixed-size string as output. Any minor change in the input should result in a completely different output string. Figure 15 shows example of cryptographic hash function. The ideal cryptographic hash function has four main properties: First, for any given string it should be easy to calculate the hash. The input string may contain alphabets, digits or special characters. Second, it is infeasible to generate a message that has a given hash. Third, even a slight change in the input string should result in a completely different hash. Fourth, two different input strings should not produce the same hash value [17].

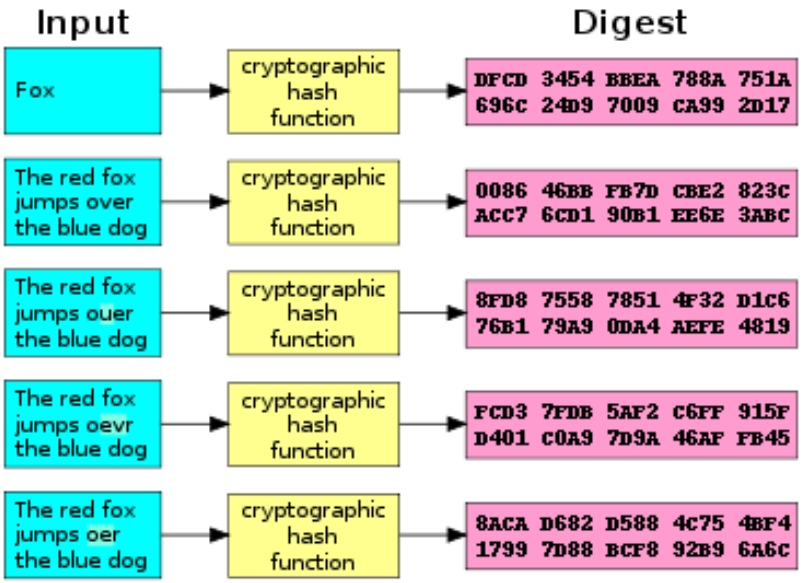


Figure 15: Example of cryptographic hash function [18]

SHA-1 and SHA-256 are very popular hash functions. They were designed by United States National Security Agency. SHA-1 generates 160-bit (20-byte) hash value and SHA-256 generates 256-bit (32-byte) hash value [19].

Table 3: Example of a cryptographic SHA1 function

Input String	**Output (Hexadecimal Value)**
Abcde	03de6c570bfe24bfc328ccd7ca46b76eadaf4334
Abcdx	a96e144dfdc6380c8a4ae43bea1c81cb01215020
Abcdo	debc9595cdf63a07efb18141692dcaab67f115fc
ABCDO	58ba5a974fa794df712866f355c9c63ac162d4d2
1x%&b!!B	52f4190f30e51d55bb03220f9bf47fae538dc1f8

Table 4: Example of a cryptographic SHA256 function

I/P	**Output: Hexadecimal Value**
abcde	36bbe50ed96841d10443bcb670d6554f0a34b761be67ec9c4a8ad2c0c44ca42c
abcdx	f2d58ae536dc5d52ff1c83332ea1184be67febd39eaab1c98118a6cb33b9aa2a
abcdo	ffb2fa8d1f9c02b9b3dfb3a465399a294533f7f131e6d3d773b2be786918b377
ABCDO	d8d219fd83565b7d122b0c7ac7629478649a429118d94dbd0c983ac054e7a262
1x%&b!!B	6b1d09646c61379166432352cfdec05c839dc2cb37df910742171b3edeb68799

So we had two choices, either use SHA1 or SHA256 to generate a hash value for the input string. We have written a JavaScript to compare the execution time of SHA1 and SHA256 functions and CryptoJS library to calculate the SHA1 and SHA256 values. Figure 16 shows an execution time comparison of the SHA1 and SHA256 function. We did 50 iterations for each input so the execution time can be compared in milliseconds.

After the experiment, we have realized that SHA1 and SHA256 calculations take almost same amount of time in hash generation, so I have decided to use SHA1 as it is very popular.

5.3 Hash CAPTCHA

Hash CAPTCHA is based on the hash cash proof of work concept. A proof of work is a piece of data which is difficult to produce as it must satisfy certain requirements. A client needs to perform some operations on the data at client side. This operation costs certain CPU cycles to the client. On the server side, it should be very easy to check whether the data sent by the client satisfies certain requirements or not. This setup prevents spam submission and denial attack by the client as for each submission, the client needs to carry out certain calculations which will cost some CPU cycles.

Table 5: Execution time comparison of the SHA1 and SHA256 function

#	**Random String**	**SHA1 Time (ms)**	**SHA256 Time (ms)**
1	041006c58b8168eb4096e3f34d0a16e4	11	8
2	1503130d07c2933db297e5fec1cb016b	7	3
3	1cea132c84b7311467e9d47f187af16c	2	7
4	c74b803684c192bb487137c2739cc0c7	3	3
5	e62e08814a6242ee7081c52c6f18a500	2	1
6	6f3174ec55e7499c676dfbd21dff31a1	3	3
7	295e64d28a763357da183c00421f0ec8	1	3
8	6e482115606c2377abbcc4545be62556	2	4
9	99621044dd037e730b9f96c344949413	1	2

#	Random String	SHA1 Time (ms)	SHA256 Time (ms)
10	d01a5d7696818fa79e0a7101c0b76c48	3	2
11	dcc73ab38e6ef10083de0617edd9c425	3	2
12	2b4babddd3511e2d4fcf5158d9805a14	1	5
13	a81664ba0b59b14cadad4f9bffebf77a	4	2
14	40eb5e28864484711466dfc0604f3037	3	2
15	a5ce82caf8066931746e49c4ec28b313	4	1

In our implementation, the server sends a random string to the client. JavaScript attached to the requested page calculates SHA1 on this random string. The script takes the random string sent by the server as an input and concatenates the integer value known as nonce at the end of the input string. The script then calculates SHA1 on the resultant string. If the resultant hash begins with '00' then script returns nonce value; otherwise, nonce is incremented by 1 and the entire process starting from string concatenation is repeated. For example, we have taken "Test" as the random string and started nonce from 0. For the string "Test: 135," SHA1 produces a hash that starts with the two leading zeroes. The script stores the nonce value in the hidden text field and it is sent to the server when the client submits the request.

Table 6: Example of hash CAPTCHA proof of work

Input String	SHA1 hash
Test:0	0b96f625c18f94dcebf01ab6bf84413743952f10
Test:1	6e8125d1846308969fb7cd4694b096333f310c4b
Test:2	b2da2df3450dab8c0797d9e7da357d1bde454c07

Input String	SHA1 hash
Test:3	0deb0e1be1b9889bddd4778f71eaa1c8c84845dd
Test:133	7985064947063366a79195bc11a423a491ff542f
Test:134	8575c8682f9fbb79034370fcfd0b17d5253a9ee2
Test:135	0022bd9826082dd8af48c412dde45beba7a1c3d3

On the server side, it is very easy to check whether the concatenation of the input string and nonce produces the desired number of leading zeroes or not. In modern computers, 135 calculation of SHA1 function does not take much time. To generate a hash of two leading zeroes, it generally takes fractions of a second. So this calculation would not affect the genuine client, but for the spammers it would be a highly CPU-intensive task.

One advantage of this system is that it is very simple to increase the client side calculation by simply varying the desired number of leading zeroes of the generated hash value. By simply changing this integer variable from 2 to 34, the client side calculation time will increase from approximately 1 second to 4 hours.

5.4 Hash CAPTCHA in Yioop

Yioop is using text CAPTCHA for “create an account” and “suggest URL” functionality. In the admin panel, we have added an option to toggle between text CAPTCHA and hash CAPTCHA. Figure 16 shows CAPTHCA mode option in Yioop.

When a user requests for CAPTCHA enabled page, the server first determines the CAPTCHA mode and in the case of hash CAPTCHA, the server generates a random string and returns the requested page to the client. Once the page is loaded on the client browser, hash CAPTCHA script is called, which performs hash cash proof of works on

the string sent by the server. It finds the nonce value for which the hash of the given string produces the desired number of leading zeroes.

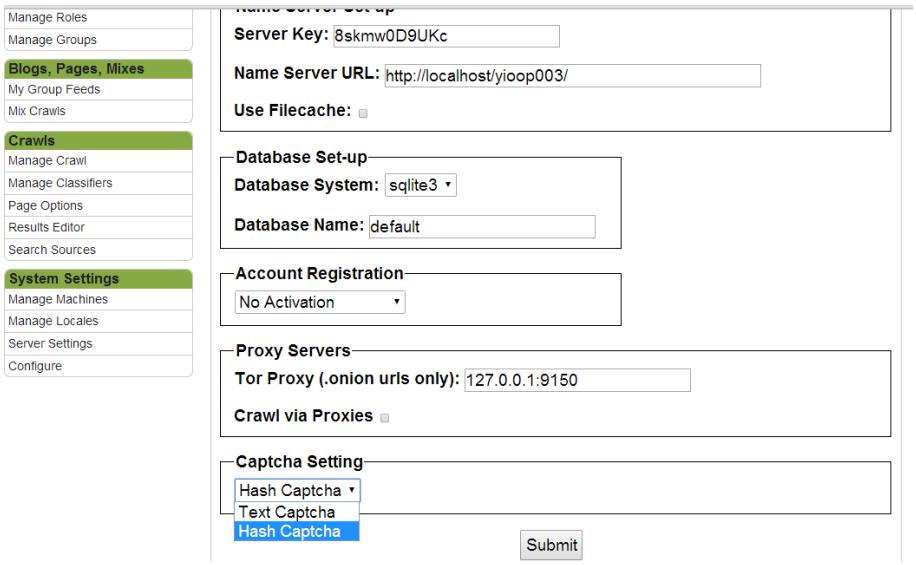


Figure 16: CAPTCHA mode option in Yioop

When a client submits the form, the server checks that for the given nonce passed, and the random string produces the desired number of leading zeroes or not. If the validation passes, then the server proceeds with the requested action; otherwise, it gives a “hash code validation failed” error and sends the page back to the client with a new random string. On the server side, I have used PHP’s SHA1 function for verification.

Ideally, the server should generate a random string for each new request. If the server uses a predefined set of random strings or very common algorithm to generate a random string, then an attacker may do some pre-calculations on the input strings and can tweak the JavaScript to avoid client-side calculation. To generate a random string, we have concatenated two parameters: the time of request and the auth_key of Yioop and performed md5 hash on the resultant script. Figure 17 shows a snapshot of “Suggest

URL" module and Figure 18 shows a snapshot of "Create Account" module after hash CAPTCHA implementation.

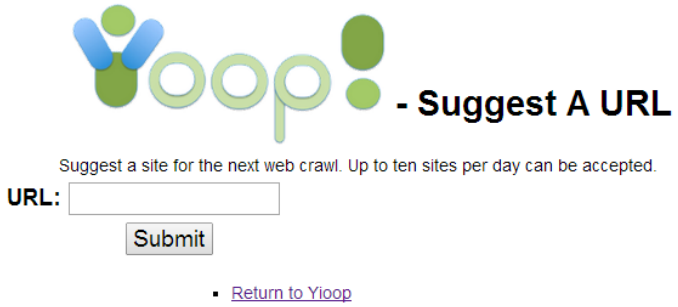


Figure 17: Hash CAPTCHA in "Create Account" module

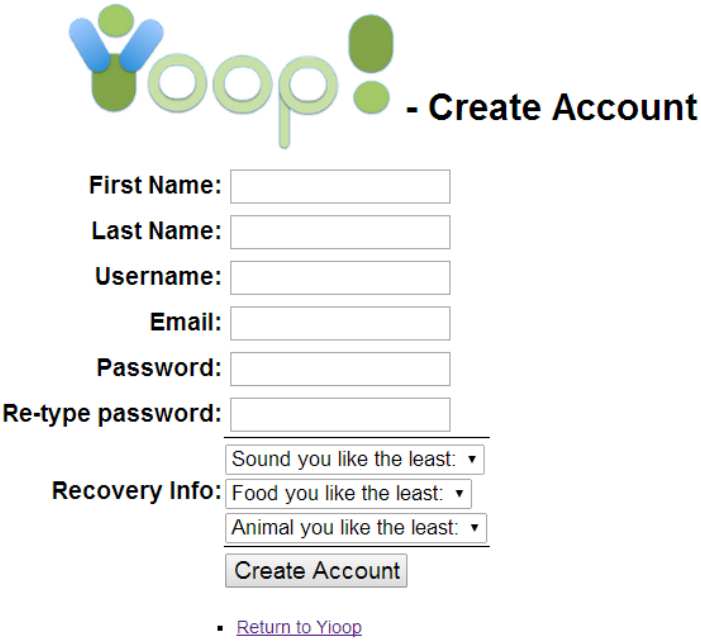


Figure 18: Hash CAPTCHA in "Create Account" module

In hash CAPTCHA, JavaScript on the client side calculates SHA1 hash on the input string and it is important to verify whether JavaScript calculates correct SHA1 or not, otherwise it will never pass the CAPTCHA check at the server side. On the server we have used PHP's SHA1 function for verification.

We have decided to write the test cases for SHA1. One challenging task was to test the JavaScript function on the server side. PHP is a server-side scripting language,

whereas JavaScript is a client-side scripting language. I have found one external PHP library called V8Js class. This library allows testing the JavaScript function at the server side.

The primary goal of Yioop's design is that it should run on default PHP configuration and it should not have any dependency on the external library. Therefore, we have decided not to use any external library and write our own class to test the SHA1 function. I have written a class called JavaScriptTest which extends the existing UnitTest class. It has a blank implementation of all the required functions of the Unit test framework. I have added a class called SHA1Test, which extends a class JavaScriptTest and compares the result of a JavaScript's SHA1 function to the PHP's SHA1 function. Figure 19 shows the output of the test run.

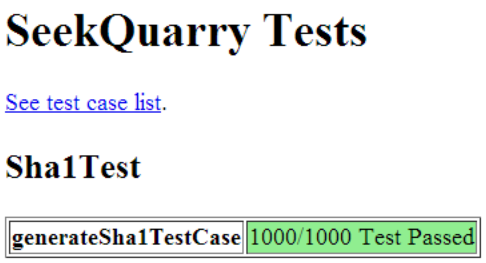


Figure 19: Test run of SHA1 test case

5.5 Conclusion

Text CAPTCHA is good from a security point of view, but it is not very user-friendly. Hash CAPTCHA can be used to replace the text CAPTCHA as it is very easy to implement and user-friendly. In Yioop, the admin has an option to toggle between text CAPTCHA and hash CAPTCHA.

CHAPTER 6
ZERO KNOWLEDGE AUTHENTICATION SYSTEM

Authentication is a process of verifying a user's identity and it plays an important role in a security of web applications. There are many ways by which users can provide their identity, such as by providing a username and password, using a biometric card, or swiping the card.

6.1 Traditional Authentication System

In the typical web authentication system, the user provides their username and password and the system displays a results page according to the user's privileges. Figure 20 illustrates the workflow of the traditional login system. In the past few years, a number of attacks have been made to sniff usernames and passwords. The eavesdropper attack is the most common attack in which the attacker intercepts the authentication data transfer over the network and if the data is not encrypted then the attacker can easily retrieve the username and password. In the man in the middle attack, the attacker stands between the prover and the verifier and acts as the prover to the verifier and the verifier to the prover. In the replay attack, the attacker records the successful authentication data and can use it later on for the authentication. The easiest thing an attacker can do is try a few common passwords such as "12345", "password", or "admin" to login to the specific account.

It is unsafe to send a plain text password or hash of a password over the network. Sometimes the attacker gets access to a password file stored on a server and if the server has stored passwords in the simple text file then the attacker can easily get all the credentials within.

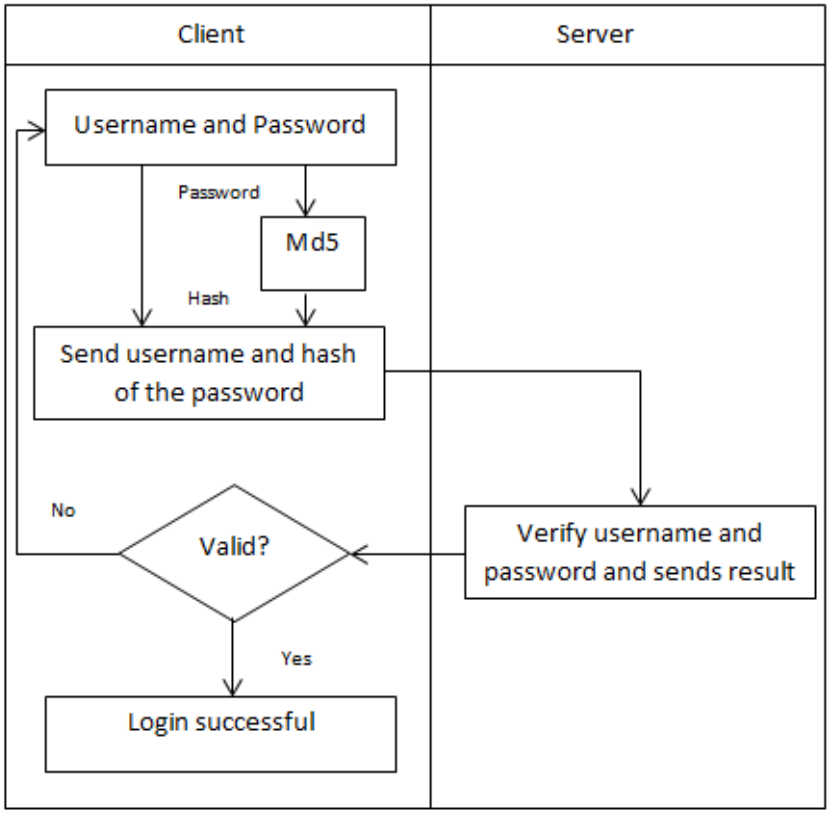


Figure 20: Workflow of the traditional login system [20]

Figure 21 shows a summary of the various authentication systems and possible attacks on the authentication systems

Attack/Authentication Method	Static Password	Soft-token Certificate/ SSL-TLS	Hard-token Certificate/ SSL-TLS	One-time Password/ Time-based Code Generator	Challenge-response	Biometrics
UT/U1a: User surveillance	A	X	X	A	X	X
UT/U1b: Token/notes theft	A	X	A	A	X	X
UT/U2a: Hidden code	A	A	A	A	X	A
UT/U2b: Worms	A	A	A	A	X	A
UT/U2c: E-mails with malicious code	A	A	A	A	X	A
UT/U3a: Smartcard analyzers	X	X	A	A	X	X
UT/U3b: Smartcard reader manipulator	X	X	A	X	X	X
UT/U3c: Brute-force attacks with PIN calculators	X	X	A	A	X	X
UT/U4a: Social engineering	A	X	X	X	X	A
UT/U4b: Web page obfuscation	A	X	X	X	X	A
CC1: Pharming	A	X	X	A	A	A
CC2: Sniffing	A	X	X	A	A	A
CC3: Active man-in-the-middle attacks	A	X	X	A	A	A
CC4: Session hijacking	A	X	X	A	A	A
IBS1: Brute-force attacks	A	X	X	A	X	A
IBS2: Security policy violation	A	A	A	A	A	A
IBS3: Web site manipulation	A	X	X	A	X	A

Legend
A: Applicable
X: Not Applicable

Figure 21: Possible attacks on various authentication systems [21]

The above problems are motivational factors to implement the zero knowledge authentication system. The zero knowledge system allows user to prove that they know the secret (i.e. password) without revealing the actual secret. In this system, a user does not need to share the actual password over a network. The server also does not store the password or the hash of the password, instead it stores the numerical password derived from the user's actual password. The idea of the zero knowledge authentication system is based on an authentication scheme developed by Fiege, Fiat and Shamir, usually known as simply Fiat-Shamir.

6.2 Fiat-Shamir Protocol

Let's say Alice wants to convince Bob that she knows the password without revealing the actual password to him. It sounds impossible, but there is a probabilistic process by which Bob can verify that Alice knows the password to an arbitrarily high probability [22].

Fiat-Shamir has developed a protocol for the authentication that relies on the fact that finding a square root modulo N is comparable to the difficulty of factoring. The complete protocol is as described below [22].

One time setup:

1. In this protocol, the trusted center selects modulus $N = p.q$ where p and q are secret large prime numbers and N is public.
2. Alice knows secret S such that S is co-prime to N and $1 \leq S \leq N - 1$. She computes $= S^2 \ mod \ N$. Here, S is private and V is public.

Protocol:

1. Alice selects random number r and sends $x = \ r^2 \ mod \ N$ to Bob. This step is called a commitment phase.

2. Bob sends either 0 or 1 to Alice. This step is called a challenge phase.

3. Alice sends $y = r.S^e \ mod \ N$ to Bob. This step is called a response phase.

Verification:

1. Bob verifies $y^2 = x.V^e \ mod \ N$. Here $V = S^2 \ mod \ N$ and $x = r^2 \ mod \ N$

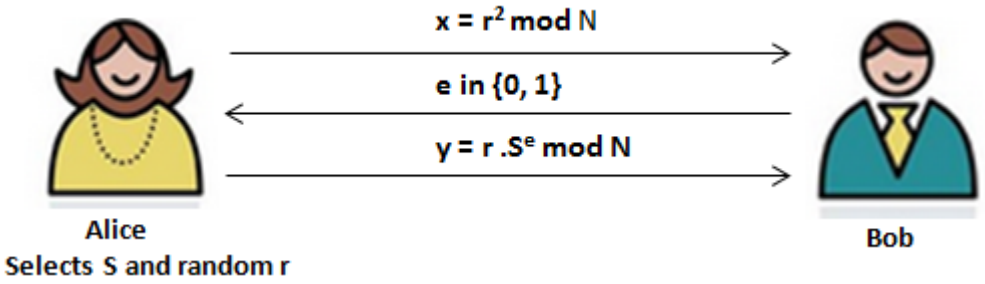


Figure 22: Fiat-Shamir protocol

In this protocol, a random value is used by Alice during a commitment phase and by Bob during a challenge phase. Let's say Bob does not use a random value for e in the second message and uses a fixed value e of either 0 or 1. In the case of $e = 0$, Trudy (attacker) sends $x = r^2 \ mod \ N$ in the first message and $y = r \ mod \ N$ in the third message. In this case, Trudy isn't required to know the secret so she can generate any random number and convince Bob that she knows the secret. In the case of $e = 1$, Trudy will send $x = r^2.V^{-1} \ mod \ N$ in the first message and $y = r \ mod \ N$ in the third message. Bob tries to calculate $y^2 = x \ .V^e \ mod \ N$ so the left part of an equation would be $r^2 \ mod \ N$ and right part would be $x.V^e \ mod \ N = r^2.V^{-1}.V \ mod \ N = r^2 \ mod \ N$ thus Bob is convinced that Trudy knows the secret. Thus, it is necessary for Bob to choose a random value for e. Trudy can only fool Bob with a probability of ½ and, after n iterations, Trudy can fool Bob by the probability: $(1/2)^n$ [21].

Alice also needs to choose the random value r for each iteration. Suppose Alice chooses a constant value for all iterations. In the case of $e = 0$ and Alice sends $r \ mod \ N$

in the third message. And in the case of $e = 1$, Alice sends $r.S \ mod \ N$. Here Trudy can record all the communication and learn about $r \ mod \ N$ and $r.S \ mod \ N$. It is very easy for Trudy to derive S from $r \ mod \ N$ and $r.S \ mod \ N$ [21].

Figure 23 shows an example of Fiat-Shamir protocol. For this example, I have assumed the password $S = 5$ and random value $r = 3$. I have also assumed that Bob sends $e = 1$ in the third message, so without knowing r and S, Bob can actually authenticate Alice using V, y and N.

6.3 Zero Knowledge Authentication in Yioop

Yioop has functionality for creating user accounts. Once the account is created for the user, it can login to the system using login credentials. We have decided to implement zero knowledge authentications for the login module and also want to provide an option of toggling between normal authentication and ZKP authentication. The admin user has an option to select the authentication mode. Figure 24 displays authentication mode options in an admin panel of Yioop.

On the server side, the first step was to implement database-related code. In the existing system Yioop uses a USERS table to store the login credentials. In Yioop, "profile_model.php" has all the create table queries so we have added one more create table query to it to create the USERS_ZKP table in database. Whenever a Yioop is set up for the first time, it accesses "createdb.php" to create database tables and create a default root user. We added a query in "createdb.php" to insert the root user in the USERS_ZKP table.

Once the database is set up, the next step was to implement the code in the controller and models to support this feature. In Yioop admin controller contains the entry point function "processRequest" for the sign in module. I have rewritten this function to

support both authentication modes. This function calls “checkValidSignin” function of the signIn model to validate the user credential

Suppose we have two prime numbers

$p = 11$

$q = 13$

One time setup:

$N = p.q$

$N = 11 . 13 = 143$

$V = s^2 mod\ N$

$V = 25\ mod\ 143 = 25$

Alice Sends:

$x = r^2\ mod\ N$

$x = 9\ mod\ 143 = 9$

$y = r.s\ mod\ N$

$y = 3 . 5\ mod\ 143 = 15$

Bob Verifies:

$y^2 = x.v^e\ mod\ N$

$y^2 = 15^2 = 225$

$x.v^e\ mod\ N = 9 . 25^1 = 225$

Figure 23: Example of Fiat-Shamir protocol

We have added a function “checkValidSigninForZKP” to validate the user in case of the ZKP authentication. The user model also has a few functions which access and alter the various information of the USERS table. In this implementation, all the model class needs to identify is on which table it needs to perform the requested operation. I have written a function “getUserTableName” which reads the user profile file and determines the authentication mode. If the authentication mode is “Normal

Authentication" then it sets the user table name as USERS and if the mode is "ZKP Authentication" it sets the table name as USERS_ZKP. Instead of copying this function to the every model class I have kept it in the model class file as every class in the models directory extends it.

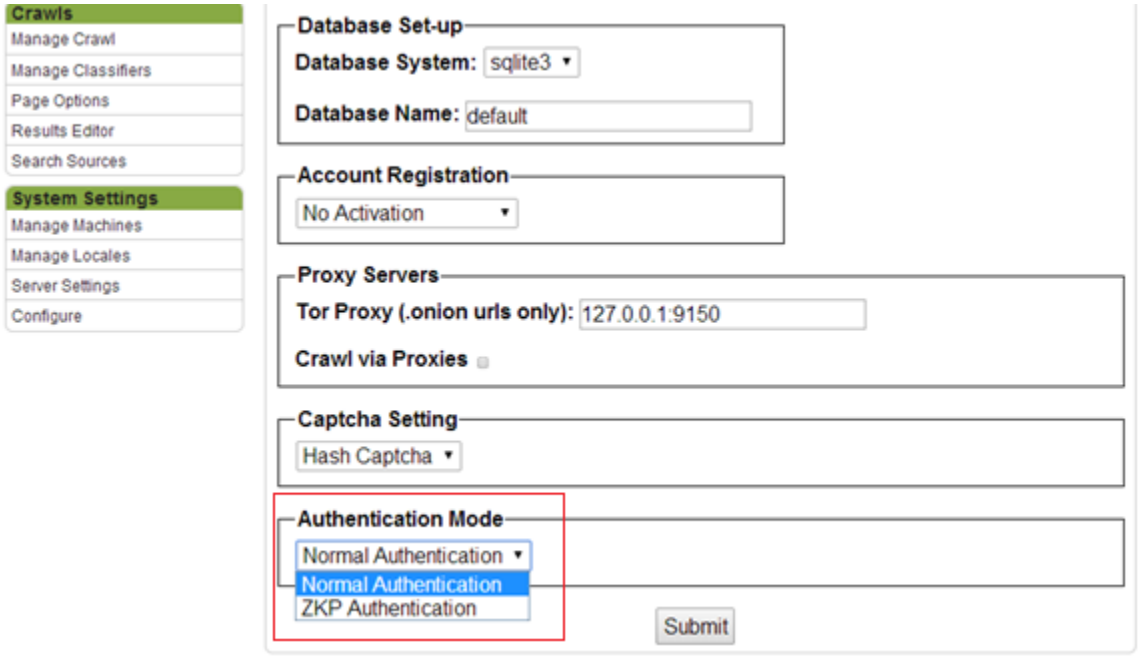


Figure 24: Authentication mode option in Yioop

One of the limitations of the zero knowledge system is that it can only work with numbers as it is based on mathematical expressions, therefore we need to find some technique to convert the user's password to numbers. We have implemented SHA1 JavaScript that has a function to calculate a 40 bit hexadecimal value from the input string, so we have decided to use the SHA1 function to convert the user's password to a numeric value.

The biggest challenge in the implementation was to perform the mathematical operations like multiplication, power and modulo on very large number at the client's side. The zero knowledge authentication system was difficult to break only when we

used very large prime numbers (e.g. 160 bits or more). We have thus used jsbn library which is a fast and portable implementation of large-number mathematical operations in pure JavaScript.

The user can create the account using the “create account” module. In the case of the ZKP system the registration module will be as shown in Figure 25. When the user clicks on the submit button after filling in all the details of the create account page, the JavaScript “zkp.js” attached to the page calculates Fiat-Shamir parameter V and submits the username and parameter V along with other form details. The server then stores the username and parameter V in the USERS_ZKP table.

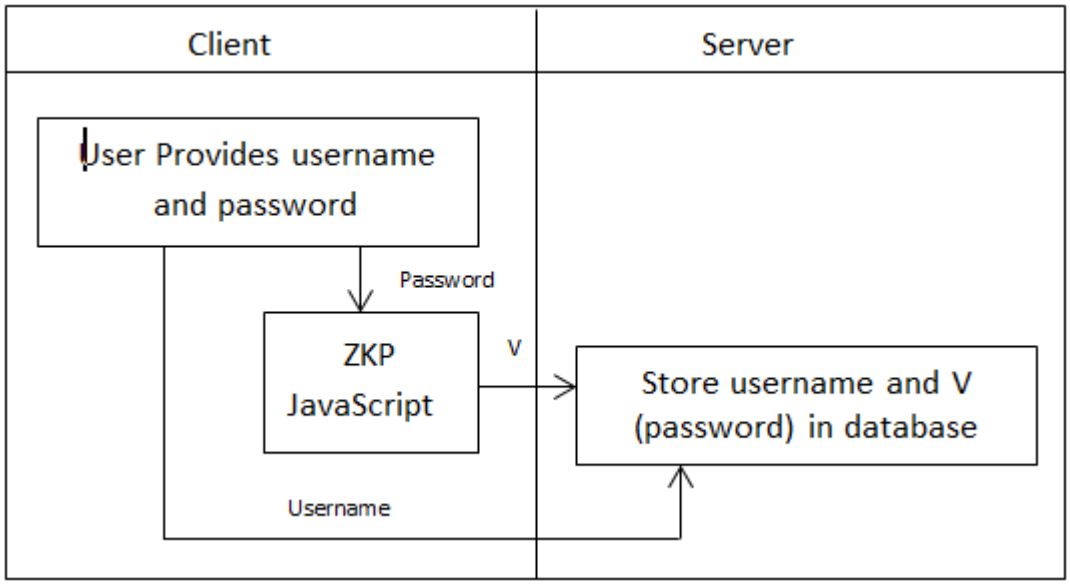


Figure 25: Registration module in Yioop

The Fiat-Shamir protocol is an example of an interactive proof system. In order to authenticate the user with a very low probability of error, there should be at least 20 iterations between the server and client. So if we use this protocol for the sign in module then the user needs to provide the password 20 times.

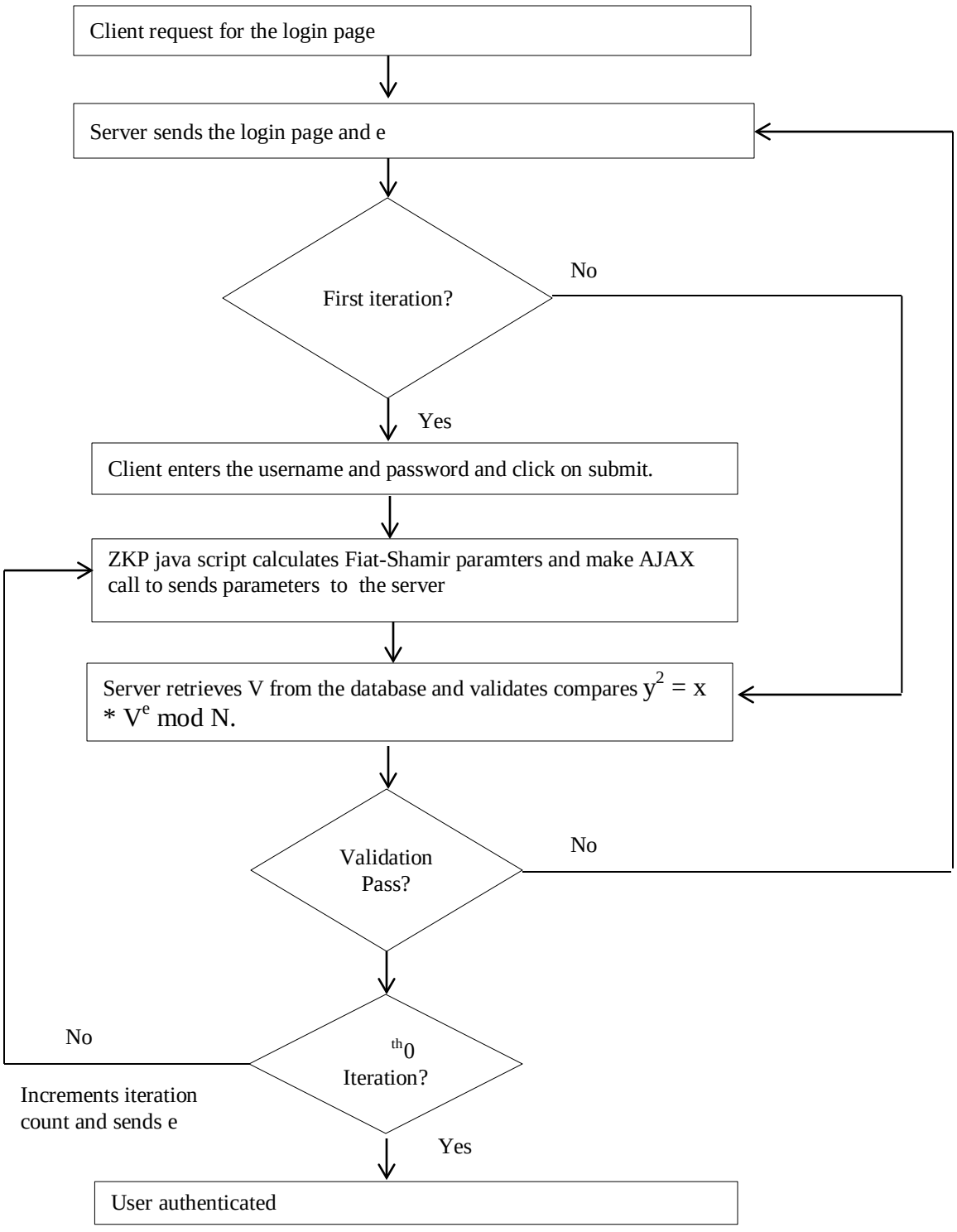


Figure 26: Login module in Yioop

This restriction makes this feature counterintuitive. To overcome this restriction, we have implemented a JavaScript function which will make an AJAX call to the server and sends Fiat-Shamir parameters. The server validates the parameters and sends random

value in response. The script will calculate new parameters and sends it to the server. This process continues for the given number of iterations. For the last iteration, the script sends form data to the server and server sends the appropriate view page. Thus the user's password is never shared over the network. The flow diagram of the login module is as shown in Figure 26.

6.4 Conclusion

The zero knowledge authentication system is a unique way to authenticate the user. There are few advantages of this system. First, it does not send a password over the network so even if an attacker breaks into the network; he will not be able to obtain any useful details. Second, it does not require any additional hardware such as a token generator or biometric scanning devices to verify the user. Third, there is no change in the system from the user's point of view.

CHAPTER 7
CONCLUSION

In this project, I have got a chance to learn about various security protocols. The first protocol I studied was the Diffie-Hellman key generation protocol. The cipher-suite which uses this protocol provides Perfect Forward Secrecy (PFS). PFS prevents retrospective decryption of previously intercepted traffic. We have also written a PFS check script, which can be used to check whether a website supports PFS or not. Yioop supports PFS; thus, it provides long-term security to the user data.

Tor network provides anonymous Internet surfing to users. It also allows publishers to publish the service without revealing their identity (IP address). We have incorporated code so Yioop can crawl the Tor hidden services and display them in the search result.

We have used hashcash algorithm to design the hash CAPTCHA. Hash CAPTCHA prevents spam submissions, thus improves the quality of the website. It can be used to replace text CAPTCHA, which is not very user-friendly. We have implemented hash CAPTCHA in Yioop and also provided an option to toggle between existing text CAPTCHA and hash CAPTCHA.

The zero knowledge authentication system is based on Fiat-Shamir protocol. Users can log in to the account without sharing their actual password. This system provides security to the user's password as it is not sent over the network and also not stored on the server. On the login page, the user needs to give the password and JavaScript attached to the page will generate Fiat-Shamir parameters from the user's password. We have incorporated zero knowledge authentication in Yioop's login module.

CHAPTER 8
REFERENCES

[1] Introduction and Feature list of Yioop. Retrieved on September 09, 2013 from https://www.seekquarry.com/?c=main&p=documentation.
[2] Tilborg, Henk C. A. Van, and Sushil Jajodia. "Perfect Forward Secrecy." Encyclopedia of Cryptography and Security. New York: Springer, 2011. 921-22.
[3] Stamp, Mark. "Simple Authentication Protocols." Information Security: Principles and Practice. Hoboken: John Wiley & Sons, 2011. 21820.
[4] TLS Handshake protocol. Retrieved on September 09, 2013 from http://msdn.microsoft.com/en-us/library/windows/desktop/aa380513(v=vs.85).aspx
[5] Browser support for PFS. Retrieved on September 30, 2013 from http://news.netcraft.com/archives/2013/06/25/ssl-intercepted-today-decrypted-tomorrow.html.
[6] SSL lab test. Retrieved on September 30, 2013 from https://www.ssllabs.com/
[7] What happened in heartbleed. Retrieved on April 10, 2014 from http://www.forbes.com/sites/josephsteinberg/2014/04/10/massive-internet-security-vulnerability-you-are-at-risk-what-you-need-to-do/
[8] Inception of Tor. Retrieved on October 01, 2013 from https://www.torproject.org/about/overview.html.en
[9] "Tor: The Second-generation Onion Router." Proceeding SSYM'04 Proceedings of the 13th Conference on USENIX Security Symposium 13 (2004): 21. ACM. Web. 01 Oct. 2013.
[10] Why we need Tor. Retrieved on October 01, 2013 from https://www.torproject.org/about/overview.html.en
[11] Lasse Overlier , Paul Syverson, Locating Hidden Servers, Proceedings of the 2006 IEEE Symposium on Security and Privacy, p.100-114, May 21-24, 2006
[12] Duck duck go Tor hidden service. Retrieved on April 14, 2014 from http://www.reddit.com/r/onions/comments/1a4rmf/duck_duck_go_hidden_service/
[13] CAPTCHA. Retrieved on April 14, 2014 from http://en.wikipedia.org/wiki/CAPTCHA
[14] Different types of the CAPTCHAs. Retrieved on April 14, 2014 from http://www.findexamples.com/5-examples-of-different-types-of-captchas/
[15] Gursev Singh Kalra."Attacking CAPTCHAs for Fun and Profit." McAfee® Foundstone® Professional Services. Retrieved on 14 April from http://www.mcafee.com/us/resources/white-papers/foundstone/wp-attacking-captchas-for-fun-profit.pdf
[16] Users response on CAPTCHA. Retrieved on April 14, 2014 from http://www.experienceu.com/news/guidelines-for-website-creators-and-designers
[17] Cryptographic hash function. Retrieved on April 14, 2014 from http://en.wikipedia.org/wiki/Cryptographic_hash_function
[18] Example of a cryptographic hash function. Retrieved on April 14, 2014 from http://en.wikipedia.org/wiki/Cryptographic_hash_function
[19] Secure Hash Algorithm. Retrieved on April 14, 2104 from http://en.wikipedia.org/wiki/Secure_Hash_Algorithm

[20] Lum Jia Jun, Brandon."Implementing Zero-Knowledge Authentication with Zero Knowledge." The Python Papers Monograph. Retrieved on April 14, 2014 from http://ojs.pythonpapers.org/index.php/tppm/article/download/155/142
[21] Applicability of attacks in different authentication mechanisms. Retrieved on April 14,214 from http://www.isaca.org/Journal/Past-Issues/2007/Volume-3/Pages/Analyzing-the-Security-of-Internet-Banking-Authentication-Mechanisms1.aspx
[22] Stamp, Mark. "Simple Authentication Protocols." Information security: Principles and practice. Hoboken: John Wiley & Sons, 2011. 226-30