A Practical Zero-Knowledge Proof Protocol for Web Applications

Article · December 2014

CITATIONS
3

READS
7,430

2 authors:

Slawomir Grzonkowski
University of Galway
41 PUBLICATIONS **538** CITATIONS
SEE PROFILE

Peter Corcoran
University of Galway
629 PUBLICATIONS **4,855** CITATIONS
SEE PROFILE

Journal of Information Assurance and Security.
ISSN 1554-1010 Volume 8 (2013) pp. XXX-XXX

A Practical Zero-Knowledge Proof Protocol for Web Applications

Sławomir Grzonkowski[1,2], **Peter Corcoran**[3]

[1]Symantec Ltd.
Dublin, Ireland
([2]At the time of writing this article: DERI, National University of Ireland, Galway)
slawomir_grzonkowski@symantec.com

[3]College of Engineering
National University of Ireland, Galway
pcor@wuzwuz.nuigalway.ie

Abstract: **A secure and scalable authentication protocol for client-server applications is presented. This is based on an efficient Interactive Zero-Knowledge Proof (ZKP) implementation where a local username/password pair is used to dynamically generate a unique key pair, the public key being transmitted and stored on a server as part of a service registration. When the user accesses this service authentication challenges are generated aperiodically. In order to generate responses to these challenges the client must regenerate the private key from the username/password pair. To allow broad applicability these are implemented using HTTP and JavaScript enabling the protocol to be supported by practically all standard web browsers. These algorithms are tested and evaluated across a range of web browsers and are shown to offer user response timings which are acceptable to users without compromising the strong authentication provided by the underlying ZKP algorithm.**
Keywords: Password authentication, Zero-knowledge proof, ZKP, Web-authentication, Concurrent ZKP

I. Introduction

Practically all password-based web-applications that are using the HTTPS protocol assume that a secure path to the authenticating server is provided. As a result users' logins and passwords are encrypted only with server's public key that is delivered while sessions are initiated. This approach has been exploited in many attacks. For example, SSLStrip intercepts plain-text credentials of users who forgot to type "https://" while supplying URLs in their address bars. Many experiments [1, 2] demonstrated that users inadequately interpret security warnings, especially those related to SSL/TSL. As a result a malicious user who set up a proxy with self-signed certificates has high chances that some user would confirm a security exception and accept a bogus request; and thus become prone to man-in-the-middle attacks. We note that the procedure of setting up such a proxy is only getting easier and easier. The required tools, e.g., burpsuite[1] or mitmproxy[2], are already recognized as an integral part of web-application debugging toolkits.

Such social engineering attacks are not the only way of defeating SSL. It was also demonstrated that many of its implementations did not strictly follow the rigid protocol guidelines and best practices; and thus validating SSL certificates in non-browser software is considered as the most dangerous code in the world [3].

Applying a complete Public Key Infrastructure (PKI) that would enable client-side private keys would solve this problem only partially as most users are accustomed to the use of conventional password based systems and will chose these over more complex security mechanisms. Using PKI would also require users to store their private-keys in a secure manner, which would also create additional risks and limitations that we would like to mitigate in our approach. Thus the motivation for this research was to develop an authentication (sub)protocol which would strengthen security of existing web applications against network-level attacks.

Our proposal employs a fast and lightweight zero knowledge proof (ZKP) algorithm [4] which provides security of the conventional PKI. It involves client-side hashing of the userid/password to transparently generate a key-pair and register the public key with the remote service provider. The server then generates aperiodic challenges and a lightweight JavaScript based client application computes the responses to each server-side challenge. A practical implementation is tested with multiple browsers for response timings and reliability. Our results show that this solution can also run on most of today's smartphone platforms and thus can be used to provide secure authentication

[1]http://portswigger.net/burp/
[2]http://mitmproxy.org/

for mobile as well as web applications. An additional property is that the computational security of our approach can be controlled by adjusting the number of challenges.

A. Contributions

We provide and evaluate a practical solution for an interactive password-based ZKP protocol. In comparison with our earlier work [5] we describe additional countermeasures to man-in-the-middle attacks and provide a solution for mutual authentication. A secondary goal is to provide a wider dissemination of the underlying techniques; thus portions of this work reiterate the underlying mechanisms and workflows of [5], but are summarized more concisely here and are hopefully more accessible to the reader. Note in particular that our approach does not require permanent storage for private keys or other secrets on the client-side, improving the underlying security model. A password-based protocol that retains zero knowledge properties in concurrent environments is provided to implement this. Other mechanisms such as repeatable biometrics [6, 7], or biometric key based authentication [8] could equally be adopted for use in consumer devices or services. A mitigation strategy is also provided to handle attacks that can be deployed between two parties, e.g., offline dictionary attacks by an eavesdropper or SSLStrip [9].

B. Paper Overview

In Section II we describe the concept of Zero-Knowledge Proof authentication. A reader familiar with our earlier work on ZKP [5], [10] may skip this section. In Section III we show how such proofs can be used to create authentication protocols. In Section IV we describe how a ZKP-based protocol that works in concurrent environments can be designed. We test this protocol in Section V and we analyze other threats in Section VI. In section VII we describe related work. Finally, in Section VIII we conclude the paper and we present some ideas for future work.

II. ZKP Authentication

The Zero-Knowledge Proof term has been formalized by Goldwasser, Micali, and Rackoff [4]. It describes a group of challenge-response authentication protocols, in which parties are required to prove the correctness of their secrets, without revealing these secrets. Such protocols exist for any NP problem, provided that one-way functions exist [4].

The use of ZKP as a mean of proving identity was first proposed by Fiege et al. [11]. The protocol depends on prime numbers and uses modular arithmetic and it requires digital certificates to work. Therefore, users are unable to use their login and passwords pairs and also the presence of a trusted third party is required to confirm the validity of digital certificates.

In this work we focus on interactive ZKP. However, there is also a variant of non-interactive zero-knowledge (NIZK) protocols. Some authors [12] have shown that it is enough for two parties to share a common random string to achieve zero-knowledge without interaction. NIZK protocols are also robust in a concurrent setting (see Section III-B). They preserve zero-knowledge even if the same string is shared globally [13]. However, NIZK require stronger cryptographic assumptions, i.e., trapdoor one-way permutations as opposed to arbitrary one-way functions, and in general they are much less efficient than interactive ZKP [14].

A. Zero-knowledge proof properties

Any valid zero-knowledge proof system must satisfy three requirements: 1) *completnes*: if a statement is true, an honest verifier will always convince of this fact the verifier; 2) *soundness*: if a statement is false, a cheating prover will not be able to convince the honest verifier, except some small probability; 3) *zero-knowledge*: if the statement is true, the verifier learns nothing about the secret.

These assumptions must hold if the honest party is following the pre-described protocol. Additionally, to decrease the *soundness error*, the verifier may increase the number of interactions to the satisfactory level. As shown in Section III, for efficiency reasons, those interactions can be executed in a parallel manner.

B. Approaches to ZKP

There are several approaches to ZKP [15], [11], [16]. To provide a web-applicable solution, we focus on graph isomorphism [16] since all the necessary calculations are performed on integers within predictable computations time. In case of an approach that would rely on large prime numbers, predictability would not guaranteed as they appear in a random manner [17].

C. Graph Isomorphism

Graphs G_1 = (V_1,E_1) and G_2 = (V_2,E_2) that have the same sets of vertices V_1=V_2={1,2,...,n} are isomorphic, if there exists a permutation π on vertices {1, 2, . . . , n} so that (u, v) $\in E_1 \leftrightarrow (\pi(u), \pi(v)) \in E_2$. Therefore, to produce an isomorphic graph G_2 of graph G_1, we have to rename its vertices. An example is depicted in Figure 1. The problem is not likely to be NP-complete [18], but it

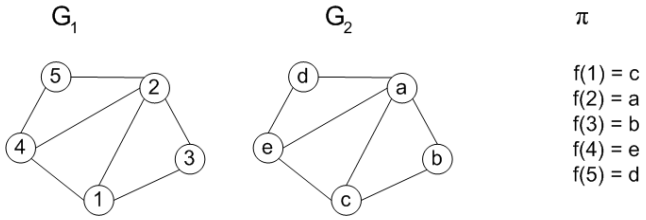


Figure. 1: An example graph isomorphism for $G_2 = \pi(G_1)$.

is NP. There are no known polynomial-time algorithms that can solve it (see Section VI-C).

In a zero knowledge proof protocol for graph isomorphism [16], a public key is composed of two isomorphic graphs G_1 and G_2, whereby the permutation π_p

$$G_2 = \pi_p(G_1)$$

is a private key. A prover generates a random permutation π_R, and sends a graph $G_R = \pi_R(G_1)$ to the verifier. Then, depending on the verifier's challenge, the prover sends back π_R or π_{R2} such that

$$\pi_{R2} = \pi_R \circ \pi_p^{-1}.$$

Thus, π_{R2} is a product of the permutation π_R and the inverse of the permutation π_p^{-1}. The verifier is able to check one of the conditions:

$$G_R = \pi_R(G_1) \text{ or } G_R = \pi_{R2}(G_2)$$

Knowledge about only one parameter π_{R1} or π_{R2} does not let the verifier compute the prover's private key. We note that it was demonstrated [16] that this protocol holds zkp properties: completeness, soundness and zero-knowledge.

III. Protocol Compositions

To build an interactive ZKP authentication protocol, there is a need to perform a number of prover-verifier interactions in a certain order. This order is called a "composition" [19]. In practical applications, invoking interactions one after another causes long waiting times for the prover due to network added latency for each interaction. Thus, there were many attempts to optimize ZKP protocols, especially by introducing parallelization and permitting concurrent executions. Many researchers [19], [20], [21], however, demonstrated that such improvements might cause protocols to lose their zero-knowledge properties. Thus, there is an important question regarding interactive zero-knowledge proofs: whether they preserve their zero-knowledge properties in a specific protocol composition. This determines if a given security protocol is applicable for practical realizations. We distinguish three main compositions of zero-knowledge proofs protocols: sequential, parallel, and concurrent. In all compositions we assume that both parties: the prover and verifier follow the prescribed protocol. We also assume that the actions of the honest parties in each executions of the protocol are independent from their actions in other executions of the protocol. An adversary may, however, run various executions of the same protocol and try to take some advantage from each execution. Therefore, it is crucial to create protocols that can preserve their zero-knowledge property in various networked environments. In a perfect situation, the users should keep track of the protocol executions. However, this is unrealistic in practice. There might be also a need for a coordinator, but trying to coordinate the protocol executions is problematic and requires additional infrastructure. Thus, in a practical solution the number of possible interactions between a prover and a verifier must be large enough to ensure each protocol cycle is different from previous cycles.

A. Sequential and Parallel Compositions

In a sequential compositions each prover-verifier interaction is invoked many times and each invocation follows the termination of the previous one.
In a parallel composition, many instances of the same protocol are invoked simultaneously. It means that *ith* messages of many instances of the protocol are sent at the same time.
A naive approach to parallelization is not sufficient to preserve zero-knowledge. To make it possible, the verifier must commit the challenge questions before receiving the commitment from the prover [20].

B. Concurrent Composition

A concurrent zero knowledge [21] is a generalization between sequential and parallel ZKP composition. It is based on the use of multiple instances of the protocol but invoked at arbitrary times and each proceeds at an arbitrary pace.
Concurrent zero-knowledge takes into account situations in which a malicious verifier can interact several times with the same prover in an interleaved way. To preserve zero knowledge, so-called timing assumptions were proposed [21]. Under these assumptions, there are a priori known bounds on the delays of messages with respect to some global clock. Each party has also a local clock whose rate is within a constant factor of rate of the global clock. Thus in some cases parties delay response messages or alternatively consider messages outdated (time-out). Dwork at al. [21] argue that using these timing assumptions of concurrent protocols preserve zero knowledge. Such a solution limits the possibility of nesting sessions by dishonest parties. Further, Dwork and Sahai [22] showed that the timing assumptions can be even pushed to the preprocessing phase. Richardson and Kilian [23] demonstrated how to construct a zero knowledge protocol under the concurrent assumption. However, their protocols are not constant round.
Goldreich [24] notes that proving that a protocol is parallel zero-knowledge suffices for concurrent composition: it is reasonable to assume that the parties' local clock have approximately the same rate, and that drifting is corrected by occasional clock synchronization; the global time can be divided into phases and each of them can consist of a constant number of rounds; thus each party wishing to execute the protocol just delays its invocation to the beginning of the next phase; subsequently, concurrent execution of constant-round protocol in this setting amounts to a sequence of time-disjoint almost-parallel executions of the protocol [24].
For practical implementations of zero-knowledge protocols, it is desirable that they preserve their security properties under the concurrent composition.

IV. ZKP for Web Applications

ZKP authentication protocols cause additional communication and computation overhead. Thus, in resource limited environments such as web-browsers or mobile

devices, there is a need for efficient protocols in terms of messages sent and steps performed.
We note that Web users are accustomed to simple username and password pairs for service login. It would be desirable to have a similar interface to ZKP protocols to simplify their presentation to end-users. However, ZKP was originally created to provide a reliable PKI infrastructure and did not envisage the use of plain-text usernames and passwords.
Introducing passwords creates additional risks, such as dictionary attacks on low-entropy passwords. There was no need to take them into consideration in the original approach. If we are to adapt ZKP techniques for standard Web usage it is desirable to minimize such new risks or the significant benefits of applying ZKP security will be lost. One of possible ways of protecting from weak passwords is to use proactive password checkers [25], which ensure that user provided secrets are cryptographically strong.
Before the advent of HTML5, web browsers did not have permissions to access the underlying file systems Thus, to keep compatibility with older standards, in a ZKP password-based protocol public and private keys have to be created each time the user requests authentication. A partial solution would be to use third-party plug-ins or to store confidential information in cookies; however, it would result either in lack of compatibility or in insufficient security [26].
The introduction of HTML5 enabled modern web browsers to use local sandboxed file system. We note that this improvement in our case introduces additional security risks. For example, 1) the user will need to create this storage for each computer used for authentication; thus the same confidential information would be stored in many places; and 2) the system could be copied by an adversary having access to the computer; as a result an offline dictionary attack could be mounted. As we show in this section, there is no need to create these extra threats.

A. Authentication Procedure

To start the authentication procedure, a user, say *Alice*, must first register. Then, in order to login, *Alice* types her username and password. Note that her password never leaves her browser. The browser uses the password to calculate her private key (see Section IV-B) and then to generate a set of challenge graphs (see Section IV-C.2). These graphs and her username are then sent to the server and are used to complete the challenge-response authentication protocol described in Section IV-C.
A sequence diagram is presented in Figure 2 and shows our top-level approach in detail. When compared with other authentication solutions, the browser is now responsible for some new tasks: calculating private keys from passwords; generating challenge graphs, and challenge responses. Furthermore, the server has the additional responsibility of generating random challenges for users (browsers). There is also an additional interaction between a browser and a server when

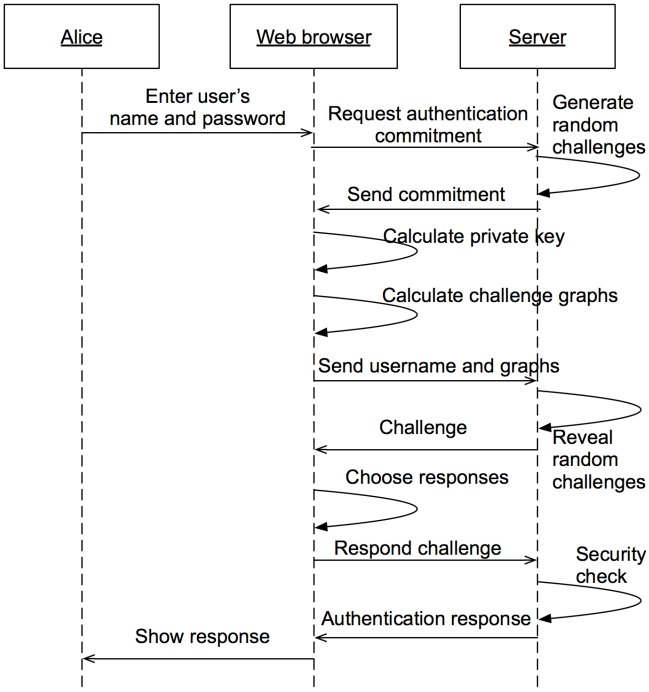


Figure. 2: ZKP implementation on the Web

compared to classical approaches such as HTTP Basic and Digest: the request-response interaction is replaced with request-challenge-response interaction. Thus, a browser must perform additional operations and communication tasks. The main question is whether this new approach is feasible and, in particular, will not cause extended waiting periods for users. This requires both an efficient implementation of the underlying algorithms and the use of distributed computing in the client-side browser.

B. Private-key algorithm

In a graph isomorphism-based protocol, a user's private key is a permutation. Since we want to keep users using login and password pairs, we transform passwords to corresponding permutations using a one-way hash function. Such a transformation must always generate same-size permutations to ensure they can be used together with users' public keys. The transformation we use has been depicted in Figure 3 and is explained in the following section.

1) Transformation

In our protocol, described in detail in Section IV-C, we used SHA1 [27]. The SHA1 algorithm computes a 160-bits-long hash for any string that is no longer than 2^{64}-bits. Such a hash is composed of characters 0-9, a-f; and therefore, we can interpret the hash as a hexadecimal number. We use this number as a seed for our algorithm. The algorithm generates n-long permutations are always in the range from *n!+1* to *(n+1)!* (see Section IV-B.2). Thus we add a constant prefix to ensure that the resulting permutations have the same size.
Hash functions are one of the most fundamental ele-

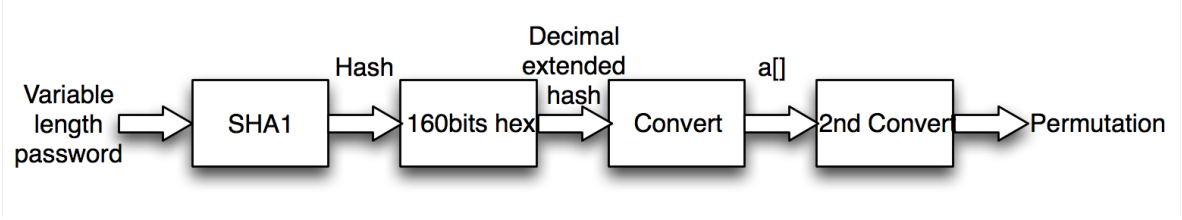


Figure. 3: The process of conversion a password to permutation

ments of cryptography. Thus, this field is constantly advancing both in terms of better software and hardware solutions [28]. Therefore, in our protocol the hash function can be easily changed to satisfy existing standards taking into account that our transformation relies on collision-resistance.

2) Calculating a permutation at a certain position

Algorithm 1 Convert($number$)

var $i := 0$
var $factor := 0$
var $a[] := newArray()$
while number > 0 **do**
  $factor := GreatestFactorial(number)$
  $a[i] := number/factor$
  $number := number - a[i] * factor$
  $i = i + 1$
end while
return a

Algorithm 2 2ndConvert($a[]$)

var $n := lengtha$
var $s := full(n)$ {returns a structure with integers *0,1,....(n-1)*}
var $a[] := newArray()$
for $i = 0$ to $n - 1$ **do**
  $s.insert(i)$ {Initializing our temporary structure}
end for
for $i = 0$ to $k - 1$ **do**
  $permutation[i] := s.elementAt(a[i])$ {getting an element at a certain position}
  $s.remove(a[i])$ {removing the taken element}
end for
return permutation

The web-application calculates permutations in two steps. The first being based on the observation that all the natural numbers can be represented as:
$a_0 0! + a_1 1! + a_2 2! + ... + a_k k!$ for natural number k
Such a representation is unambiguous and it is also known as factorial number system [29]. Thus, to represent a given number in it, there is a need to find the $a_1, a_2, ...a_k$ factors. In our implementation, this is done by using Algorithm 1, which converts a given decimal number (see Figure 3) to a table a[].
The algorithm uses the GreatestFactorial() function that returns the smallest factorial that is greater than the given number.
The second step of the conversion is performed by Algorithm 2. It takes the table a[] as an input and returns as the output a table permutation[] that contains the created permutation. The algorithm requires a temporary structure *s*, on which the following functions can be performed:

- insert (x) - adds a number *x* to the structure
- remove (x) - removes a number *x* from structure
- elementAt (j) - returns a number *i* that satisfies the condition: there are exactly *j-1* smaller numbers than *i* in the structure
- full(n) - return a structure with integers *0,1,....(n-1)*

For example, with s=0,1,2,3 by removing an element at third position we get s=0,1,3, and if we repeat the operation we have s=0,1.
The structure *s* could be a simple table or a tree. The complexity of operating the table would be *O(n)*. For a B-Tree, and in particular, an AVL tree, the computational complexity of removing elements is *O(log n)*.
An alternative approach to generate a private key from the password could rely on a variant of Fisher-Yates shuffle for generating a permutation [29]. However, to adjust the range of the resulted set of numbers we would have to apply the modulo operator. Since in our case the range would be moving, i.e., k=41, k=40,..., k=1, this solution would introduce bias in the selected set which could open a possibility of various statistical attacks [29].

C. Authentication Protocol

In this section, a prover gives a proof of knowledge of the password to the verifier. We describe both the registration and authentication procedures that we applied in our solution. We assume that *Alice* is the prover, whereas *Bob* is the verifier. *Alice* knows the permutation π_{priv} between two graphs G_1 and G_2. *Bob* also knows the graphs. *Alice* wants to convince *Bob* that she knows the permutation, but she is going to keep it secret.

1) Registration

The registration procedure requires *Alice* to generate and send her public key to *Bob*. She does not reveal to anybody nor store her private key. Her private key is generated by her browser every time after she types the

correct password. If she enters an incorrect password, a corresponding incorrect private key will be generated.

1. *Alice's* browser transforms her password to a permutation π_{priv} that is her private key
2. *Alice's* browser generates a random graph G_1
3. *Alice's* browser computes graph $G_2 = \pi_{priv}(G_1)$
4. *Alice's* browser sends graphs G_1, G_2, and her username to *Bob*. She keeps π_{priv} secret

Alice is able to convince anybody that she is the publisher of such credentials, while she keeps the password secret (see Section II-C).

2) Authentication

In our description we assumed that both parties validate each other's messages; and if the validation fails, the validating party interrupts the protocol.
Similarly to Dwork et al. [21], we also introduce the timing constraints of the following types:

- *Timeouts* (α). The prover requires the verifier to deliver certain messages before time specified by the *timeout* value. If that does not happen, the verifier interrupts the protocol.
- *Delays* (β). The prover delays its responses to the verifier according to the time specified by the *delay* value. This way the prover is protected from dishonest verifiers who could nest many sessions.

For readability and practical applications we extended the proposed notion with *Current time* (δ_i), which serves the purpose that each party must be able to measure how long the protocol is being executed to be able to calculate *timeouts* and *delays*. The parties measure this time independently and there is no need for synchronization of their local clocks.
It has been demonstrated that these timing constraints are necessary to preserve zero-knowledge in the concurrent composition [21] (see Section III-B). Values for α and β must be agreed before the protocol execution in a way that $\beta \geq \alpha$ and they do not cause long waiting times for the user. The protocol is composed of five constant rounds, in which the participating parties exchange messages:

1. *Alice's* browser is acquiring the number of challenges *t* from *Bob*
   - *Alice's* browser randomly selects γ_0 and γ_1; and then calculates $A_0 = \gamma_0(G_1)$ and $A_1 = \gamma_1(G_1)$
   - *Alice's* browser sends A_0 and A_1 to *Bob*
   - *Alice's* browser checks current time δ_1
   - *Alice's* browser sets β and α ($\beta \geq \alpha$)
2. *Bob* receives A_0 and A_1
   - *Bob* selects *t* random permutations $\mu_1, \mu_2, ..., \mu_t$
   - *Bob* randomly generates $v_1, v_2, ..., v_t$ such that for all $i \in \{1, 2, ..., t\}$,$v_i \in \{0, 1\}$
   - *Bob* calculates $Q_i = \mu_i(A_{v_i})$. He sends Q_i to *Alice*
3. *Alice's* browser receives and stores the *Bob's* message
   - *Alice's* browser generates *t* random permutations $\pi_{R1}, \pi_{R2}, ..., \pi_{Rt}$
   - *Alice's* browser calculates *t* graphs $G_{R1}, G_{R2}, ..., G_{Rt}$ such that $G_{Rn} = \pi_{Rn}(G_1)$ where $n \in \{1, 2, ..., t\}$
   - *Alice's* browser sends the generated graphs to the *Bob*
4. *Bob* stores the graphs
   - *Bob* sends $v_1, v_2, ..., v_t$ and $\mu_1, \mu_2, ..., \mu_t$ to *Alice*
   - *Alice's* browser checks current time δ_2 and checks if ($\delta_2 - \delta_1 \leq \alpha$). If this condition is not satisfied, *Alice's* browser considers the session as timeout and interrupts the authentication procedure
   - *Alice's* browser checks if $Q_i = \mu_i(A_{v_i})$ are correct. If all the equations are satisfied, *Alice's* browser constructs a response vector $R_1, R_2, ..., R_t$. If $v_1 = 0$, then $R_i = \pi_{Ri}$, otherwise $R_i = \pi_{Ri} \circ \pi_{priv}^{-1}$
   - *Alice's* browser stores current time δ_3 and waits until ($\delta_3 - \delta_1 \geq \beta$)
   - *Alice's* browser sends the response vector to *Bob*
5. *Bob* checks her response
   - *Bob* expects that if $v_i = 0$, then $Ri(G_1) = G_{Ri}$. If $v_i = 1$, then $Ri(G_2) = G_{Ri}$
   - *Bob* sends her a response. She is authenticated, only if all the challenge questions are responded correctly

The concept of this protocol is derived in part from [16] and is introduced in Section II-C. To adjust this protocol to the web environment, in which high latencies can be experienced, there is a need to execute multiple challenges in parallel. Goldreich and Krawczyk [19] argue that a simple parallelization results in a protocol that may not be zero knowledge. Thus, we have decided to apply extensions introduced by [20]. These extensions ensure that the verifier commits on random challenges before obtaining the prover's message; and this way the prover's graphs do not influence the selection of the challenge questions.
We combine our final ZKP protocol with the private-key algorithm described in Section IV-B, and in this way we obtain an interactive ZKP password-based protocol using isomorphic graphs.

D. Full Authentication Framework

The resulting protocol still has two problems. Firstly, the provided authentication is only unilateral: verifier does not give a proof of knowledge of the public key. Secondly, a ZKP protocol in which private key is derived from a password creates a possibility of mounting an offline guessing attack. This is a generic property of Interactive ZKP protocols that results from the fact that if the challenge question is known to an eavesdropper, the masked password verifier is sufficient to be a reference for the guessing attack [30].
To address those two problems we apply a generic framework that was proposed by Nyang [30]. This framework introduces a system set up in which a simplified EKE protocol is executed. As a result the the prover and the verifier share a session key and the prover is ensured if the authentication server has the corresponding password verifier. Therefore, by using this framework we are able to address both problems: the authentication procedure is mutual and offline dictionary attack is not possible due to the applied session key that is relaying on simplified EKE.

V. Implementation and Evaluation

We implemented a working proof of concept in Java and JavaScript to demonstrate the practicality of our ZKP protocol for use in real-world applications, including those running on mobile devices [31]. In our tests, all the generated permutations and graphs have the same size, which is *41* which corresponds to 41^2 =1681 bits long secrets. This value can be adjusted.
Our key consideration was to verify if our protocol could be successfully implemented across a range of web browsers; and thus, if such a system can be transparent for users in web applications. To achieve this performance and response times, the system must be in line with what users have come to expect from typical Web applications without compromising the strong security provided by ZKP. We also aimed to provide cross-browser functionality and thus we evaluated performance across multiple browsers with the base algorithm implemented in Java/JavaScript.
As a metric for response times we adopted the accepted response-time guidelines originally proposed in 1968 by Miller [32] as these appear to have stood the test of time. Three decades after his findings, these rules are still applicable for a majority of web applications [33].
Miller defines three main timing thresholds perceived by the users of a computing system:

- 0.1 second for keeping the user attention attracted - the user has the impression that the responses from the system are coming immediately;
- One second for keeping a user workflow uninterrupted - the users notice latencies when using the system, but they do not lose the feeling of interactivity while using it;
- 10 seconds for keeping the user's attention on the dialog - if the response takes longer than 10 seconds, users will want to perform other tasks.

We note that in many security solutions the users expect some latency because they know that some authentication process is ongoing. In fact they may feel that the authentication is inadequate, or that it has been bypassed if the system response is too immediate. Thus, in security applications 0.1 seconds authentication time may not be desirable. In fact Bouch and Sasse [34] argue that users may even judge a relatively fast service to be unacceptable unless it is also predictable, visually appealing and reliable. Thus, in our studies we aim to determine not only the authentication time, but also the level of security delivered within the given time constraints.

A. Evaluation Environment

For the the purpose of compatibility with legacy systems. We performed our tests on older web browsers, among them we included: Firefox 3.5.3, Safari 4.0.2, Internet Explorer 8 and Opera 9.64. The browsers were tested on the same machine - a Macbook Pro, with an Intel CPU and an installation of Mac OS X 10.5.8. In case of Internet Explorer, we tested it under Windows XP service pack 2 on the same Macbook. The tests did not involve human input since they were automated by a script.
To verify usability of our approach in a typical web authentication scenario, we evaluated our implementation in two ways: (i) we measured the authentication time for a given amount of challenges, and (ii) we counted the amount of data that was exchanged in order to achieve certain security levels. We note that Java/JavaScript performance is browser specific, but in the case of data, the results does not depend on the platform.

B. Testing Methodology

We developed an extension for our client-side script, which was responsible for executing the script for a given number of times for using a given amount of challenges. Time and other parameters were also measured by the script. Respective values were displayed by the script after each pass. This way we were able to collect many data samples and analyze our implementation efficiency, stability and also the number of bytes exchanged.
We performed tests for various protocol compositions introduced in Section III, in particular: sequential, parallel and concurrent.

C. Sequential Composition

The protocol proposed in Section IV-C can be implemented in a sequential ZKP workflow. In such a case, each browser's request contains only one challenge graph and also timing assumptions (α,β) are not taken into consideration. However, such a method of authentication causes longer authentication times. Our tests

showed that the time necessary for performing one challenge was between 115 (Opera) and 408 (Firefox) ms. Assuming that the authentication process should not be longer than 10 seconds, the Firefox browser is able to execute a maximum of 24 challenges. Opera which performed faster in such a setting is able to execute 24 challenges in 2.76 seconds. Therefore, the main research question is if such results are secure enough for a practical realization.

In interactive ZKP protocols, there is a possibility that a malicious user would try to guess the challenge questions. Thus the attacker could prepare a set of graphs knowing permutations to only one of the public key graphs G_1 or G_2; and then, attempt to authenticate, expecting a certain sequence of responses from the verifier. The attacker would interrupt and restart the authentication procedure after receiving a challenge question that is unable to respond.

If we define time that is necessary for one round as s, then the time that is necessary for an attacker to verify all the possibilities can be calculated as the number of possible zeros and ones in a string of a given size. Thus, the exact number can be calculated as:

$$s \sum_{k=0}^{n-1} 2^k$$

where n is the number of challenges. For example, in case of 24 challenges, there are 10^{24} possible authentication scenarios. According to the introduced formula, the attacker would interrupt 10^{23} sessions after the first verifier question, 10^{22} after the second question, etc. Taking into account the time required for a single challenge on Opera, the attacker would need approximately 22.3 days to issue 10^{24} authentication requests.

D. Parallel Composition

This protocol composition extends the sequential one in the way described in Section IV-C. The main difference between the sequential and parallel versions is that multiple challenges are sent in parallel and multiple responses are synchronously received in a single reply message. This enables faster authentication cycles, and more authentications within the same time window. In turn, these improvements make the parallel composition useful for a broader spectrum of applications.

1) Performance tests

Figure 4 presents our data samples using box plot-style charts. Each box plot represents 10 data samples obtained for a given number of challenges. Thus, each chart presents a group of 110 samples.

In the case of Firefox (Figure 4(a)), the median value of a one second long authentication time was missed for 100 challenges. With 110 data samples, the one second limit was missed 24 times. This was caused mainly by some random system events: only five data samples of 24 were inside the representative box plots range (outliers); the remaining 19 were classified as extreme outliers, being outside the representative range of value. Thus, assuming predictable and more stable system behavior, and also excluding extreme outliers from the data sample, we had 95 % authentication attempts validated within one second. If we consider all the samples, only one sample was greater than 3 seconds (3.126) and two other samples were above 2.5 seconds.

Performing tests on Safari, we obtained more stable data samples, but their median values were higher (see Figure 4(b)). Two samples were slightly above 2.5 seconds. The one second limit of the median value was missed with a setting of 50 challenges. Up to 40 challenges, 39 of 40 data samples were below one second.

In the case of Opera, we observed even more stable results than in the previous test. The data distribution of the obtained samples showed little variance (see Figure 4(c)). Similarly to Safari, the one second limit was exceeded when increasing the number of challenges to 50. All the obtained samples were below 2.5 seconds.

Finally, we tested Internet Explorer. Its results were the most stable and we also observed the smallest number of outliers (see Figure 4(d)). The one second response-time limit was reached when the number of challenges was set to 40, which was slightly worse than in the case of other browsers. Further analyzes of the data samples obtained shows that there are only three outliers and one extreme outlier. Therefore, 109 of 110 data samples of the experiment were within the representative range.

2) Authentication Cost

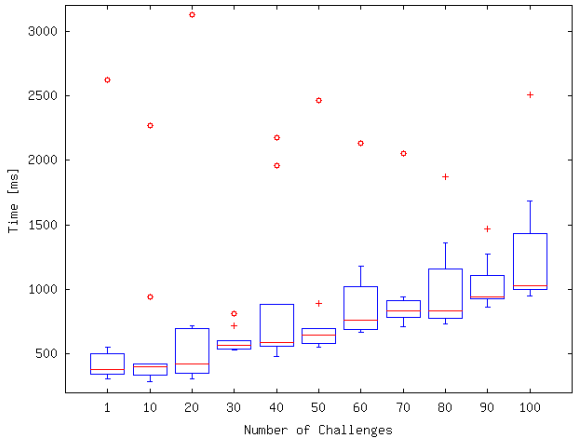
Our additional calculations on these data and regression models showed that one challenge in Firefox costs approximately 6.5 ms, start-up time 408 ms; for Opera these figures are 21 ms per challenge with start-up time 115; for Safari 19 ms per challenge and 228.5 ms start-up time; and finally 26 ms per challenge for Internet Explorer with start-up time of 188. Note that low per-challenge cost in Firefox was obtained with longer start-up time when compared to other browsers.

This means that depending on the number of challenges necessary to authenticate the user, a given browser might be unresponsive for one or two seconds after clicking the "login" button. According to Miller [32], the user workflow is not interrupted within this time constraint. We also note that in most applications authentication is performed once and then there is no need to repeat it because these details are stored in the connection session.

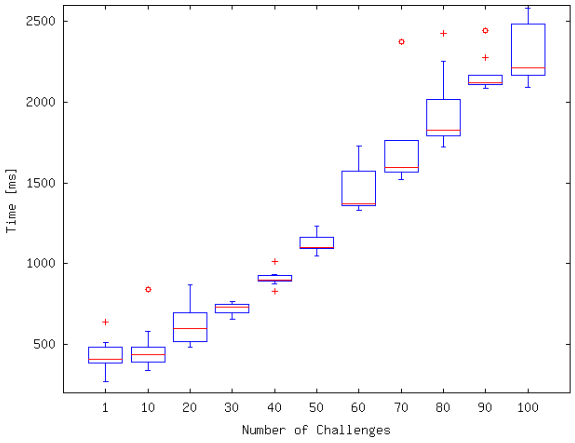
E. Concurrent Composition

A concurrent composition is particularly interesting for distributed environments such as the Internet. In comparison with the parallel composition, it ensures the validity of the protocol under stronger security assumptions [21].

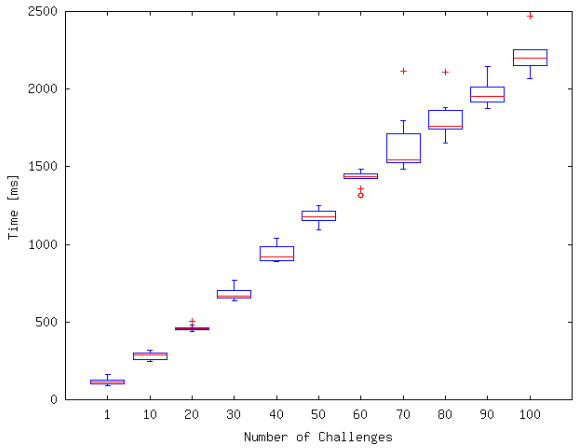
According to Goldreich [24], proving that a given protocol can preserve zero-knowledge in a parallel composition is sufficient for creating its valid concurrent composition (see Section III-B). There is, however, the necessity to introduce the notion of delays (β) and timeouts (α) to the protocol. Both α and β must be agreed



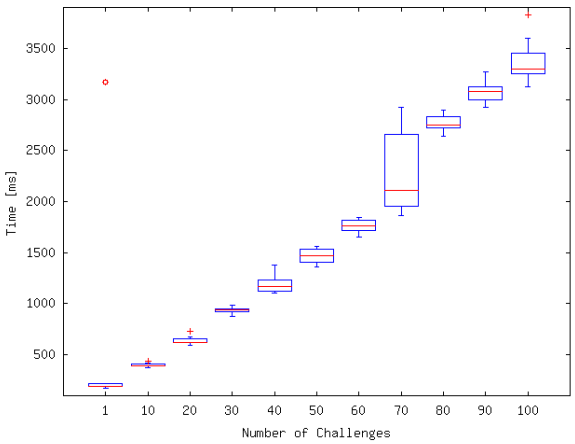
(a) Data distribution for the Firefox browser up to 100 challenges



(b) Data distribution for the Safari browser up to 100 challenges



(c) Data distribution for the Opera browser up to 100 challenges



(d) Data distribution for the Internet Explorer browser up to 100 challenges

Figure. 4: Tests for Firefox

by the prover and the verifier before executing the authentication procedure. There is also a requirement that $\beta \geq \alpha$.

Figure 5 depicts the steps of the protocol that we tested (see Section IV-C.2). The message sent by the verifier in Step 3 must reach the prover before α units of time are counted, otherwise the prover will interrupt the protocol. If the response is delivered on time, the prover waits until β units of time are counted before sending the last message to the verifier. This strategy limits the possibility of nesting sessions by a group of cooperating dishonest verifiers. We note that the timing assumptions are already present in many existing solutions [35], [36], [21].

On the one hand, in practical ZKP implementations, α and β should be high enough to ensure authentication even for slow browsers. On the other hand, the procedure should be fast enough to satisfy the response-time standards. Thus, we attempted to verify if authentication can be performed within one second.

We took into account results obtained for the parallel version and we set the number of challenges to 40. Since the authentication time is short, we also found it reasonable that timeouts (α) should be as high as possible. Therefore, we set $\beta = \alpha$. In the case of 40 challenges, Internet Explorer (IE) performed worst. Thus, we started our evaluation with this browser. In our tests, we sampled the authentication time 10 times. Initially we set β to 800 ms. In IE, all authentication times for this β value generated a browser timeout. Thus, we increased β to 900 ms. In this case 9 out of 10 samples generated timeouts. We increased β again to 1000 ms, and we noticed an improvement: only 2 out of 10 samples were considered as timeout. Finally, for β equal 1100 ms all the samples were approved within the specified time constraints.

With this setting we continued our evaluation and tested other browsers.We did not notice any timeouts for these values of α and β for any browser. The longest authentication time was observed for Firefox and it was 1304 ms. The quickest authentication was observed for Opera and it was 1134 ms. The median values for Firefox, IE, Opera, and Safari were respectively 1177 ms, 1203 ms, 1165.5 ms, 1163 ms. Thus, the difference between the highest and the lowest median values is 40 ms. Such a difference is impossible to notice by the end

users.
Therefore, the timing assumptions did not only provide the possibility of the concurrent composition, but they also made the users' perception of each browser almost exactly the same.

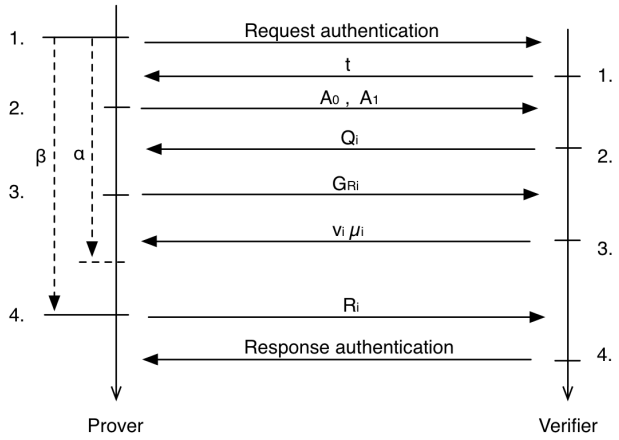


Figure. 5: The proposed protocol

F. Safe Range of Challenges

Similarly to sequential protocol, interrupted sessions are the main threat for interactive ZKP protocols in which all challenge questions are sent in parallel. In this scenario, an attacker prepares a set of graphs and corresponding permutations to G1 or G2; consequently, the attacker is able to respond to only one a priori chosen challenge for each graph. As long as the expected response does not come, the attacker interrupts and restarts the authentication procedure with the same graphs, expecting to obtain the different responses. In this scenario, the attacker's chances drop exponentially, with the linear growth of the number of challenges requested by the verifier. We can calculate these chances using the Bernoulli scheme: the probability of at least one success in N trials of nested session impersonation attacks can be expressed as

$$P(N,p) = \sum_{k=1}^{N} \binom{N}{k} (p)^k (1-p)^{N-k}$$

where p is equal to $1/2^t$, where t is the number of challenges. Taking advantage of the fact that this probability is supplemented by the probability of zero successes in N trials (k = 0), we are able to simplify the formula to

$$P(N,p) = 1 - (1-p)^N$$

To authenticate securely, the number of necessary attempts should be high enough to make such an attack impractical and infeasible. In the case of 30 challenges the attacker would need at least half-a-year to perform an interruption attack with only 10 % success probability. For 40 challenges, a successful attack is completely infeasible even assuming a smaller success probability.

G. Data measurements

In the classical authentication approaches [37], we cannot set the authentication confidence level; and consequently, the browsers always send logins and passwords. In zero-knowledge proof approaches, the browser sends more data, if higher security is required. In our tests, the total amount of data sent from the browser to the server is mainly determined by the first stage of the protocol (see Section IV-C) performed by the user's browser. The ratio between data sent in the authentication request and the challenge response is 71% to 29%. This parameter is neither browser nor hardware dependent. A single data challenge costs approximately 400 bytes. Moreover, the data transmitted by our server in round four (see Section IV-C.2) was equal to the amount of challenges, whereas in round five (see Section IV-C.2) the server always sent a short response code: "200 OK" or "403 Forbidden" that depends on the authentication result.

VI. Analysis of Protocol Threats

There are many attacks that can be executed against authentication protocols. They range from social engineering attacks, through key loggers to mounted dictionary attacks on password verifier. We discuss how they can be approached by the proposed protocol.

A. Offline Dictionary Attacks

In this scenario we assume that an attacker has a pair of the public graphs G_1 and G_2 and attempts to calculate if a given password candidate satisfies the relation $G_1 = \pi_{cand}(G_2)$, where π_{cand} is computed from the password candidate. Password dictionaries are different from language dictionaries. Therefore, we ran tests using dictionaries specially created for this purpose[3][4]. For the configuration we had, we were able to verify approximately 220 passwords per second.
The efficiency of dictionary attacks is, however, questionable. On the one hand, according to Florencio and Herley [38], users tend to pick low quality passwords that are easy to guess. These passwords are also often reused among many web-sites of different security measures [39]. On the other hand, Schneier [40] claims that only 3.8 % of users' passwords can be found in passwords dictionaries. Therefore, even if a single password can be verified with a dictionary in a reasonable time, the probability of success is really low. Moreover, a commercially deployed application can make this process even more difficult by taking advantage of a tool that imposes strong users passwords [25].
Nyang [30] proposed a generic framework for the whole family of interactive zero-knowledge proof authentication protocols. It is based on secret coin tossing [30] and it introduces additional complexity as it requires additional communication and computations to perform simplified EKE [41] (see Section VII-C) to proof that the server has the knowledge of the client's verifier. It results also in a shared key that can be further used to prevent man-in-the-middle-attacks.

[3] http://www.openwall.com/passwords/wordlists/password.lst

[4] http://www.maxalbums.com/password_list.php

B. Brute-force Attacks

To perform a naive brute force attack, we created a program that enumerates and tests passwords of a given length. We limited the number of characters to the keyboard accessible characters only. The speed of such an attack was comparable to the one obtained for dictionary attacks; and on average we were able to generate and test 226 passwords per second. For the computer used for evaluation, testing a password of size 6 characters or more was infeasible due to long computation time, which was estimated to longer than one year.

C. Algorithms for Determining Graph Isomorphism

Efficient algorithms for graphs of restricted types were proposed; For example, probabilistic algorithms were discussed by Babai et al. [42], [43]; Gil and Zibin [44] proposed efficient algorithms, but only for isomorphism of simple types; Spielman [45] proposed a solution only for strongly regular graphs. There is also a polynomial-time algorithm for determining the isomorphism of graphs of fixed genus [46] and for trees [47]; another group of approaches are brute-force algorithms for sub-graph isomorphism [48]. Furthermore, Corneil and Gotlieb [49] described algorithms for determining if two graphs are isomorphic; however, very often the problem is to find the permutation, not to determine if two given graphs are isomorphic. There are also informal publications claiming that there are polynomial algorithms for graph isomorphism [50], [51], but they were already questioned by counterexamples.

There are tools such as Nauty[5] that analyzes patterns of the graphs to find the isomorphism. Therefore, one of the challenges for the protocol is the selection of the optimal graphs. In our case we select graphs randomly; thus, an attacker cannot assume any of those restricted types. Hence, the only one feasible approach for the attacker would be to use a probabilistic method, for example [52]. However, it would not guarantee the success and also the complexity of the solution would be higher.

Another open question is the optimal number of vertices for the graphs used in the protocol. Ayeh and Namuduri [53] suggested that the number should be in the range of hundreds. However, they do not justify their choice and propose a slightly different protocol that is tested with Nauty.

Some other authors [54] [55] claims that for a similar problem of subgraph isomorphism, computers require 100000 years to find a solution in case of 81 vertices. Then, in case of using a specially designed hardware solution (FPGA-based), isomorphism of graphs of 256 vertices would provide security for 70000 years. This confirms the exponential growth of the time required for solving these type of graph problems. We note that if these results would be be considered a realistic threat to the *soundness error* of the proposed protocol, the size of the graphs can be customized with some computational impact on the prover-side. However, it was shown by us [56] that nowadays mobile devices are already capable of processing graphs of 256 vertices or bigger.

[5] http://cs.anu.edu.au/~bdm/nauty/

D. Message collision

A malicious verifier could collect prover's graphs until the same graph is sent twice. Then the verifier could ask the opposite challenge question in comparison with the one already asked. That would be sufficient to find the secret graph isomorphism. Since in our tests we used graphs of 41 vertices, we store our data on 1681 bits. According to Ferguson et al. [17], a rough estimation of the data that has to be collected before such an attack is possible can be calculated by using the birthday paradox; and in our case it is equal to $\sqrt{2^{1681}}$ samples. In terms of data storage and time requirements, such an attack is infeasible for shorter keys, e.g., 256 bit symmetric encryption in the AES encryption algorithm is believed to be secure for the next 30 years [17].

E. Practical attacks on users' passwords

As our experimental results show, the users simply will not be able to conduct offline dictionary or brute-force attacks. Adversaries will rather try to hijack other accounts by 1) attempting to install a keylogger to monitor user's input; 2) by using social engineering attacks such as phishing; 3) by compromising the server database; 4) by applying some well-known web specific attack such as request forge or cross-site scripting; or 5) in case if the adversary is on the same LAN network, by compromising the network connection for the user by applying tools such as SSL Strip.

A more sophisticated attack could rely on influencing the random numbers generator in a way that the graphs would be predictable or that they would belong to a class that it is known to be breakable, e.g., graphs with fixed genus or trees.

VII. Related Work

A. SSL/TSL

The Secure Socket Layer (SSL) protocol was created by Netscape to stimulate the sales of the company's cryptographically enabled web servers by distributing a free client that implemented the same cryptographic protocols [57]. Such a solution was necessary since solutions offered by HTTP [58], [37] were prone to sniffing and man-in-the-middle attacks.

SSL provides a mutual authentication for password-web applications. Its main difference is that its mutual authentication capabilities relies on digital certificates provided to the user browser. By default this approach is risky. However, techniques such as SSL Pinning [59] has been introduced to address this drawback. Additionally, there are other attacks on the protocol that replies on human-computer interaction such as SSLStrip.

B. Master password

There were several efforts to introduce a master password for a user to encrypt many unique passwords for websites that the user is visiting. For example, Halderman, Waters, and Felten [60] proposed a client side solution as a Firefox[6] plug-in for generating many passwords from a master user's password. This solution proposed the usage of separate secrets for each server the browser wants to communicate with.
In contrast to our proposal, password managers do not specify underlying authentication protocols; thus, they inherit the underlying protocol's drawbacks. They are also browser and computer dependent. A user would have to copy the passwords from one machine to another, if there would be a need in using more than one computers. Such a plug-in may provide additional resistance to various dictionary attacks since a plain-text user password is never stored on the server. Thus, it also maintains a greater level of security for the user, even if an end-server should become compromised. However, due to fact that the authentication protocol is not specified when using a Master Password approach alone, the problem of man-in-the-middle attacks or eavesdropping is not addressed.

C. EKE and similar propositions

Bellovin and Merritt proposed Encrypted Key Exchange (EKE) [41]. This protocol combines both asymmetric and symmetric cryptography. It has several versions that differs in security properties. In the most secure, the first communication stage uses the Diffie-Hellman (DH-EKE) protocol to establish a shared key, which aims at providing security against man-in-the-middle and other impersonation attempts. Then, the actual authentication stage is executed. It results in a symmetric key that is computed independently by communicating parties.
The EKE protocol was also followed by other propositions. The Simplified Password-authenticated Exponential Key Exchange (SPEKE) protocol [61, 62] was developed for commercial purposes. The main difference in comparison to EKE is that the password is used to influence the selection of the generator parameter in the session-key generation function. Thus, the protocol introduced so-called *forward secrecy*. It means that an attacker who successfully reveals a password, is unable to access session keys of past sessions. Zhang [63] showed, however, that in SPEKE an adversary is able to test multiple password candidates using a single impersonation attempt. In the EKE protocol, an adversary can gain information about at most one possible password in each impersonation attempt.
Password-Authenticated Key Exchange (PAKE) is a family of protocols based on DH-EKE that provides a reasonable level of security using short memorized passwords for protecting information over insecure channels. Such protocols are also a topic of the IEEE P1363[7] standard working group.
Secure Remote Password (SRP) [64] was developed in 1997. The security of this protocol depends on the strength of the applied one-way hash function. The protocol was revised several times, and is currently at revision six. SRP is often applied to telnet and ftp. The protocol is more computationally intensive than EKE. It requires two modular exponentiation, whereas EKE requires only one.

VIII. Conclusions and Future Work

A password-based interactive Zero Knowledge Protocol (ZKP) is proposed and evaluated. The stability of this approach is investigated, showing that it behaves in a reliable and robust manner across a wide range of contemporary web browsers. It is further shown that it satisfies a set of empirical response-time standards without compromising authentication security. The results obtained were used to determine the minimum and maximum possible number of challenges that would provide secure authentication while still satisfying usability standards. In addition the size of data required to perform authentication with a given confidence level is determined.
These considerations are helpful in deriving a formula for a successful attack on parallel ZKP protocols. A model is next presented of attacks on interactive ZKP protocols taking into account the number of challenges and the attacker's confidence level. Three versions (compositions) of the proposed protocol are compared: (i) sequential ZKP, (ii) parallel ZKP, and (iii) concurrent ZKP. It is shown that the concurrent composition not only increases security, but also makes the protocol behave similarly and in a predictable manner on a number of browsers. Next an analysis is presented of various threats and attacks for the proposed protocol and its resilience to each of these is evaluated.
Finally the requirements for a fully secure version are discussed and it is shown that this requires an adequate level of conventional password security to eliminate the possibility of matching through a password dictionary. It is shown that approx. 40 challenge-response cycles are more than adequate to satisfy both security and acceptable response-time constraints. For information-only transactions < 20 challenges will still provide quite robust security with improved user-response times.
Proposed future work includes a micropayment solution using the described protocol [31] and also work on a protocol for sharing social data using mobile devices [65]. Other interesting research topics include investigation of the relationship between the public and private key sizes of (sub)graph isomorphism-based protocols and their discrete logarithm or elliptic curves equivalents. These may offer further opportunities to enhance performance both in terms of user-response and improved cryptographic security.

[6]Firefox: `http://www.mozilla.com/firefox/`

[7]IEEE P1363: `http://grouper.ieee.org/groups/1363/`

Acknowledgments

The work presented in this paper was supported (in part) by the Lion project supported by Science Foundation Ireland under Grant No. RSF0840 and Enterprise Ireland under Grant No. REI 1005. A method and apparatus for authenticating a user is a patent granted [66].

References

[1] N. Vratonjic, J. Freudiger, V. Bindschaedler, and J.-P. Hubaux, "The inconvenient truth about web certificates," in *Economics of Information Security and Privacy III*, B. Schneier, Ed. Springer New York, 2013, pp. 79–117. [Online]. Available: http://dx.doi.org/10.1007/978-1-4614-1981-5_5

[2] D. Akhawe, B. Amann, M. Vallentin, and R. Sommer, "Here's my cert, so trust me, maybe?: Understanding tls errors on the web," in *Proceedings of the 22Nd International Conference on World Wide Web*, ser. WWW '13. Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee, 2013, pp. 59–70. [Online]. Available: http://dl.acm.org/citation.cfm?id=2488388.2488395

[3] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov, "The most dangerous code in the world: validating ssl certificates in non-browser software," in *ACM Conference on Computer and Communications Security*, 2012, pp. 38–49.

[4] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof-systems," in *STOC '85: Proceedings of the seventeenth annual ACM symposium on Theory of computing*, 1985, pp. 291–304.

[5] S. Grzonkowski, W. Zaremba, M. Zaremba, and B. McDaniel, "Extending web applications with a lightweight zero knowledge proof authentication," in *IEEE/ACM CSTST '08: Proceedings of the 5th international conference on Soft computing as transdisciplinary science and technology*, 2008, pp. 65–70.

[6] P. Corcoran and A. Cucos, "Techniques for securing multimedia content in consumer electronic appliances using biometric signatures," *Consumer Electronics, IEEE Transactions on*, vol. 51, no. 2, pp. 545–551, May 2005.

[7] P. M. Corcoran, C. Iancu, F. Callaly, and A. Cucos, "Biometric access control for digital media streams in home networks." *IEEE Trans. Consumer Electronics*, vol. 53, no. 3, pp. 917–925, 2007.

[8] P. M. Corcoran and A. Cucos, "Secure digital content reproduction using biometrically derived hybrid encryption techniques," US Patent granted, November 2005, patent Application 11/090,974 published as 2005/024763-A1.

[9] M. Marlinspike, "New tricks for defeating ssl in practice," 2009, "BlackHat, Washington DC".

[10] S. Grzonkowski, "A privacy-enhanced usage control model," Ph.D. dissertation, Digital Enterprise Research Institute, National University of Ireland, Galway, 2010.

[11] U. Feige, A. Fiat, and A. Shamir, "Zero-knowledge proofs of identity," *J. Cryptology*, vol. 1, no. 2, pp. 77–94, 1988.

[12] M. Blum, P. Feldman, and S. Micali, "Non-interactive zero-knowledge and its applications (extended abstract)," in *STOC*, 1988, pp. 103–112.

[13] U. Feige, D. Lapidot, and A. Shamir, "Multiple non-interactive zero knowledge proofs based on a single random string (extended abstract)," in *FOCS*, 1990, pp. 308–317.

[14] I. Damgard, "Concurrent zero-knowledge is easy in practice," 1999, appeared in the THEORY OF CRYPTOGRAPHY LIBRARY and has been included in the ePrint Archive. ivan@daimi.aau.dk 10500 received June 16th, 1999. Revised July 28th, 1999.

[15] L. C. Guillou and J.-J. Quisquater, "A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory," in *Lecture Notes in Computer Science on Advances in Cryptology-EUROCRYPT'88*, 1988, pp. 123–128.

[16] O. Goldreich, S. Micali, and A. Wigderson, "Proofs that Yield Nothing but their Validity," Technion, Tech. Rep., 1988, technical Report #498, preliminary version in FOCS 86.

[17] N. Ferguson, B. Schneier, and T. Kohno, *Cryptography Engineering: Design Principles and Practical Applications*. Wiley Publishing, 2010.

[18] U. Schöning, "Graph isomorphism is in the low hierarchy," in *STACS '87: Proceedings of the 4th Annual Symposium on Theoretical Aspects of Computer Science*. London, UK: Springer-Verlag, 1987, pp. 114–124.

[19] O. Goldreich and H. Krawczyk, "On the composition of zero-knowledge proof systems," *SIAM J. Comput.*, vol. 25, no. 1, pp. 169–192, 1996.

[20] M. Bellare, S. Micali, and R. M. Ostrovsky, "Perfect zero-knowledge in constant rounds," in *STOC '90: Proceedings of the twenty-second annual ACM symposium on Theory of computing*. New York, NY, USA: ACM, 1990, pp. 482–493.

[21] C. Dwork, M. Naor, and A. Sahai, "Concurrent zero-knowledge," *J. ACM*, vol. 51, no. 6, pp. 851–898, 2004.

[22] C. Dwork and A. Sahai, "Concurrent zero-knowledge: Reducing the need for timing constraints," in *CRYPTO '98: Proceedings of the 18th Annual International Cryptology Conference on Advances in Cryptology*, 1998, pp. 442–457.

[23] R. Richardson and J. Kilian, "On the concurrent composition of zero-knowledge proofs," in *EUROCRYPT*, 1999, pp. 415–431.

[24] O. Goldreich, "Concurrent zero-knowledge with timing, revisited," in *STOC*, 2002, pp. 332–340.

[25] A. Ciaramella, P. D'Arco, A. D. Santis, C. Galdi, and R. Tagliaferri, "Neural network techniques for proactive password checking," *Dependable and Secure Computing, IEEE Transactions on*, vol. 3, no. 4, pp. 327 –339, 2006.

[26] S. Grzonkowski, A. Gzella, H. Krawczyk, S. R. Kruk, F. J. M.-R. Moyano, and T. Woroniecki, "D-FOAF - Security Aspects in Distributed User Management System," in *Proceedings of IEEE International Conference of Technologies for Homeland Security and Safety. (TEHOSS 2005)*, 2005.

[27] D. Eastlake, 3rd and P. Jones, "'"us secure hash algorithm 1 (sha1) (rfc 3174)",," United States, 2001.

[28] H. Michail, A. Kakarountas, A. Milidonis, and C. Goutis, "A top-down design methodology for ultrahigh-performance hashing cores," *Dependable and Secure Computing, IEEE Transactions on*, vol. 6, no. 4, pp. 255 –268, 2009.

[29] D. E. Knuth, *Seminumerical Algorithms*, 3rd ed., ser. The Art of Computer Programming. Addison-Wesley, 1997, vol. 2.

[30] D. Nyang, "Armoring password based protocol using zero-knowledge with secret coin tossing," in *Information Theory, 2001. Proceedings. 2001 IEEE International Symposium on*, 2001, pp. 139–.

[31] S. Grzonkowski and P. M. Corcoran, "A secure and efficient micropayment solution for online gaming," in *Proceedings of the International IEEE Consumer Electronics Society's Games Innovations Conference 2009 (ICE-GIC 09)*, 2009.

[32] R. B. Miller, "Response time in man-computer conversational transactions," in *Proc. AFIPS Fall Joint Computer Conference Vol. 33*, San Francisco, Calif, 1968, pp. 267–277.

[33] J. Nielsen, *Designing Web Usability: The Practice of Simplicity*, Thousand Oaks, CA, USA, 1999.

[34] A. Bouch and M. Sasse, "It ain't what you charge, it's the way that you do it: a user perspective of network qos and pricing," 1999, pp. 639 –654.

[35] T. Beth and Y. Desmedt, "Identification tokens - or: Solving the chess grandmaster problem," in *CRYPTO '90: Proceedings of the 10th Annual International Cryptology Conference on Advances in Cryptology*, London, UK, 1991, pp. 169–177.

[36] B. C. Neuman and T. Ts'o, "Kerberos: An authentication service for computer networks," in *IEEE Communications, 32(9):33-38*, September 1994.

[37] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart, "Http authentication: Basic and digest access authentication (rfc 2617)," United States, 1999.

[38] D. Florencio and C. Herley, "A large-scale study of web password habits," in *WWW '07: Proceedings of the 16th international conference on World Wide Web*, 2007, pp. 657–666.

[39] S. T. Haque, M. Wright, and S. Scielzo, "A study of user password strategy for multiple accounts," in *Proceedings of the Third ACM Conference on Data and Application Security and Privacy*, ser. CODASPY '13. New York, NY, USA: ACM, 2013, pp. 173–176. [Online]. Available: http://doi.acm.org/10.1145/2435349.2435373

[40] B. Schneier, "Myspace passwords aren't so dumb," Tech. Rep., 2006.

[41] S. M. Bellovin and M. Merritt, "Encrypted key exchange: Password-based protocols secureagainst dictionary attacks," in *SP '92: Proceedings of the 1992 IEEE Symposium on Security and Privacy*. Washington, DC, USA: IEEE Computer Society, 1992, p. 72.

[42] L. Babai, P. Erdos, and S. M. Selkow, "Random graph isomorphism," *SIAM Journal on Computing*, vol. 9, no. 3, pp. 628–635, 1980.

[43] L. Babai and L. Kucera, "Canonical labelling of graphs in linear average time," in *Proc. of the 20-th Annual Symp. Foundations of Computer Science*. IEEE Computer Society, 1979.

[44] J. Gil and Y. Zibin, "Efficient algorithms for isomorphisms of simple types," *Mathematical. Structures in Comp. Sci.*, vol. 15, no. 5, pp. 917–957, 2005.

[45] D. A. Spielman, "Faster isomorphism testing of strongly regular graphs," in *STOC '96: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, 1996, pp. 576–584.

[46] I. S. Filotti and J. N. Mayer, "A polynomial-time algorithm for determining the isomorphism of graphs of fixed genus," in *STOC '80: Proceedings of the twelfth annual ACM symposium on Theory of computing*, 1980, pp. 236–243.

[47] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[48] J. R. Ullmann, "An algorithm for subgraph isomorphism," *J. ACM*, vol. 23, no. 1, pp. 31–42, 1976.

[49] D. G. Corneil and C. C. Gotlieb, "An efficient algorithm for graph isomorphism," *J. ACM*, vol. 17, no. 1, pp. 51–64, 1970.

[50] R. Czerwinski, "A polynomial time algorithm for graph isomorphism," *CoRR*, vol. abs/0711.2010, 2007.

[51] J. Duda, "Polynomial algorithm for graph isomorphism problem," *CoRR*, vol. abs/0804.3615, 2008.

[52] T. Czajka and G. Pandurangan, "Improved random graph isomorphism," *J. Discrete Algorithms*, vol. 6, no. 1, pp. 85–92, 2008.

[53] E. Ayeh and K. Namuduri, "Zero-knowledge proof based node authentication," University of North Texas, Tech. Rep., May 2009.

[54] L. Szöllösi, T. Marosits, G. Fehér, and A. Recski, "Fast digital signature algorithm based on subgraph isomorphism," in *Proceedings of the 6th international conference on Cryptology and network security*, ser. CANS'07, 2007, pp. 34–46.

[55] S. Ichikawa and S. Yamamoto, "Data dependent circuit for subgraph isomorphism problem," in *Field-Programmable Logic and Applications: Reconfigurable Computing Is Going Mainstream*, 2002, vol. 2438, pp. 203–210.

[56] S. Grzonkowski and P. M. Corcoran, "Sharing cloud services: user authentication for social enhancement of home networking," *IEEE Trans. Consumer Electronics*, vol. 57, no. 3, pp. 1424–1432, 2011.

[57] S. Garfinkel and G. Spafford, *Web Security, Privacy and Commerce*. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 2001.

[58] T. Berners-Lee, R. Fielding, and H. Frystyk, "Hypertext transfer protocol – http/1.0 (rfc 1945)," 1996.

[59] S. Fahl, M. Harbach, T. Muders, M. Smith, L. Baumgärtner, and B. Freisleben, "Why eve and mallory love android: an analysis of android ssl (in)security," in *ACM Conference on Computer and Communications Security*, 2012, pp. 50–61.

[60] J. A. Halderman, B. Waters, and E. W. Felten, "A convenient method for securely managing passwords," in *WWW '05: Proceedings of the 14th international conference on World Wide Web*, 2005.

[61] D. P. Jablon, "Strong password-only authenticated key exchange," *SIGCOMM Comput. Commun. Rev.*, vol. 26, no. 5, pp. 5–26, 1996.

[62] N. A. Safa, R. Safavi-Naini, and S. F. Shahandashti, "Privacy-preserving implicit authentication," *IACR Cryptology ePrint Archive*, vol. 2014, p. 203, 2014. [Online]. Available: http://eprint.iacr.org/2014/203

[63] M. Zhang, "Analysis of the speke password-authenticated key exchange protocol," *Communications Letters, IEEE*, vol. 8, no. 1, pp. 63–65, Jan. 2004.

[64] T. Wu, "The srp authentication and key exchange system (rfc 2945)," United States, 2000.

[65] S. Grzonkowski and P. Corcoran, "A Privacy-Enabled solution for sharing social data in Ad-Hoc mobile networks," in *29th International Conference on Consumer Electronics (ICCE 2011)*, Jan 2011.

[66] S. Grzonkowski and W. Zaremba, "A method and apparatus for authenticating a user," Patent granted, January 2008, patent App. US 12/811,317, Issued on April 2, 2013.

Author Biographies

Sławomir Grzonkowski is a Principal Research Engineer at Symantec, Security Technology and Response (STAR), malware operations, Ireland. In the past he was a researcher at the Digital Enterprise Research Institute (DERI), at NUI Galway, where he was also the leader of the Security Privacy & Trust Unit (USPT). His PhD research was conducted in the area of security and privacy aspects of the Semantic Web technologies, covering many distinct domains such as cryptography, trust, access control, and system usability. His post-doctoral studies were funded by Cisco Inc. and let him to continue his work on the same topics in the context of industry. He published more than 30 articles in journals, books, and refereed conferences. Slawomir is a frequent reviewer and a PC member of security-related conferences and journals. He also serves as Associate Editor of the IEEE Transactions on CE.

Peter Corcoran received his BAI and PhD degrees from Trinity College Dublin in 1984 & 1987 respectively. He has been a faculty member at NUI Galway for 25 years, completed more than 20 funded research projects, supervised more than 20 Engineering research Masters & PhD students. He is an IEEE Fellow with more than 250 technical publications, 65 journal papers and 100 International conference papers. He is also a prolific inventor with more than 300 granted US patents. He recently completed a 7 year spell in vice-Dean roles and is searching for new leadership challenges. His research interests include (i) smart-imaging/advanced digital imaging solutions; (ii) biometrics & authentication methods for handheld devices; (iii) smart grid & associated networking/security issues; (iv) cloud computing & CE devices. He is editor-in-chief of IEEE Consumer Electronics Magazine and is on the editorial board of several IEEE Transactions and Journals. He is co-Founder of several start-up companies including FotoNation (www.fotonation.com) and has served in a number of companies as consultant & expert witness. Peter's Google Scholar profile.