Extending Web Applications with a Lightweight Zero Knowledge Proof Authentication

Sławomir Grzonkowski
DERI Galway, NUIG
IDA Business Park
Galway, Ireland
slawomir.grzonkowski@deri.org

Wojciech Zaremba
DERI Galway, NUIG
IDA Business Park
Galway, Ireland
wojciech.zaremba@deri.org

Maciej Zaremba
DERI Galway, NUIG
IDA Business Park
Galway, Ireland
maciej.zaremba@deri.org

Bill McDaniel
DERI Galway, NUIG
IDA Business Park
Galway, Ireland
bill.mcdaniel@deri.org

ABSTRACT

User authentication is a crucial requirement for secure transactions and access to the sensitive resources on the Web. We propose, implement and evaluate a Zero-Knowledge Proof Authentication (ZKP) algorithm based on isomorphic graphs. The proposed mechanism allows for authentication with varying confidence and security levels.

We suggest that most of the computations should be carried out by the user's web browser without revealing password or login at any point in time; instead generated random isomorphic graphs and permutation functions based on the user login/password can be exchanged.

Our experimental evaluation shows that by combining the asynchronous web with ZKP protocols, it is feasible to satisfy existing usability standards on the web.

Categories and Subject Descriptors

K.6.5 [**Management Of Computing And Information Systems**]: Security and Protection|Authentication; C.2.0 [**Computer-Communication Networks**]: General - Security and Protection

General Terms

Protocols, Security

Keywords

Web 2.0 authentication, Password, Web 2.0, Zero Knowledge Proof, Graph Isomorphism

1. INTRODUCTION

Each time a web-application provides authentication, both login and password details are sent. In most cases the credentials are transferred from a client to a server using HTTP[1]. Although for security reasons in most of the existing solutions, the credentials are not stored on the servers in plaintext form, they are given to the servers in a readable form during the authentication procedure. The other approach is to use HTTP Digest mode [2], but the drawback of this approach is that the server can impersonate the user to a third party. This is a serious privacy problem. Developers of non-commercial web applications often ignore this issue. For commercial software, including banking and on-line shopping, developers use asymmetric cryptography communication protocols, for instance, HTTPS[3]. This protocol sets up a secure connection but the credentials are still sent. Public key solves the problem partially because users are required to provide public and private key-pairs or digital certificates. Users, however, are accustomed to login and passwords that are easy to remember and convenient in use.

The motivation for this research is to develop a fast and lightweight userid/password login model using zero knowledge proof (ZKP) [10] to provide added security. Such an authentication approach is considered to be the most secure way of proving identity [14]. The proposed mechanism allows for authentication with varying confidence and security levels. Our work demonstrates that this has been infeasible until now due to its specific requirements: asynchronous communication and computational requirements for user's browsers. However, the advent of asynchronous Web technologies coupled with a novel method of implementing ZKP proves the feasibility of such an approach now. We also constructed a prototype that demonstrate and evaluate our work.

1.1 Paper Overview

Zero Knowledge Proof authentication is described in Section 2. In Section 3 we briefly describe authentication on the web. In Section 4, we present our protocol and discuss its security capabilities in Section 5. We further implement

[1]Hypertext Transfer Protocol – HTTP/1.1: http://www.ietf.org/rfc/rfc2616.txt
[2]RFC 2617: http://tools.ietf.org/html/rfc2617
[3]HTTP Over TLS: http://tools.ietf.org/html/rfc2818

and evaluate our approach in Section 6. We review related work in Section 7. We conclude the paper in Section 8.

2. ZERO KNOWLEDGE PROOF

The zero-knowledge proof term has been formalized by Goldwasser, Micali, and Rackoff [10]. It describes challenge-response authentication protocols, in which parties are required to provide the correctness of their secrets, without revealing these secrets. Such protocols exist for any NP-set, provided that one-way functions exist [10].

During the authentication procedure, a user, for instance *Alice*, must respond to a number of challenges issued by the server. If the protocol is repeated t times, all t rounds must be answered successfully to prove *Alice's* identity. The server is always convinced with probability 1-2^{-t}. In zero-knowledge proof protocols, the verifier cannot learn anything from the authentication procedure. Moreover, the verifier is unable to cheat the prover because of having always only one value from two possible challenge responses; this is not sufficient to calculate the prover's secret. Furthermore, the verifier cannot cheat the prover because the protocol is repeated as long as the verifier is not convinced; due to random challenge selection, the verifier cannot pretend to be the prover to a third party.

We discuss various approaches to zero-knowledge protocols. Each of them uses different data structures, hence has unique properties that we briefly describe.

2.1 Classical Problems and ECC

There are two main classical problems that enable zero-knowledge proofs: the discrete logarithm problem [14] and the square-root problem [14]. These problems strongly depend on generating prime numbers. Moreover, the security of a key that is 1024-bits-long is comparable with the security of a 160-bit key in Elliptic Curve Cryptography (ECC) [16, 13]. Therefore, the classical approach is considered to have been replaced with other solutions.

ECC is an efficient and attractive alternative to the classical public key cryptosystems. The main operation involved in ECC is point multiplication. The computations tend to be slow and inaccurate because of round-off error. In order to make the computations fast and accurate, they are performed using finite fields; this involves finding a multiplicative inverse of a number, which is the main problem for making an efficient implementation. To perform computations faster and avoid many multiplicative inverse operations, points can be represented in projective coordinates.It eliminates the need to compute many multiplicative inverses, but it requires more scalar multiplication than with affine coordinates.

2.2 Graph Isomorphism

Two 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) $\in E_1 \leftrightarrow$ (π(u), π(v)) $\in E_2$. An example is depicted on Figure 1. The problem is not likely to be NP-complete [19], but it is NP. There is no known polynomial-time algorithm that solves it (see Section 5).

If we apply this problem to ZKP, a public key is composed of two isomorphic graphs G_1 and G_2, whereby the permutation π_p

$$G_2 = \pi_p(G_1)$$

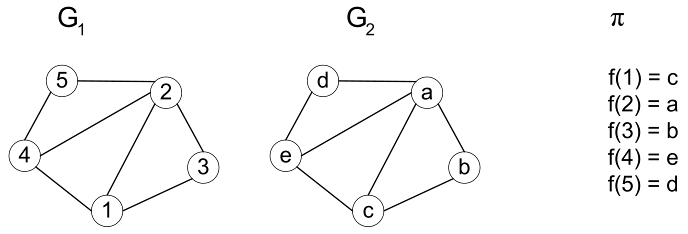

Figure 1: An example of graph isomorphism $G_2 = \pi(G_1)$.

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, 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.

3. AUTHENTICATION ON THE WEB

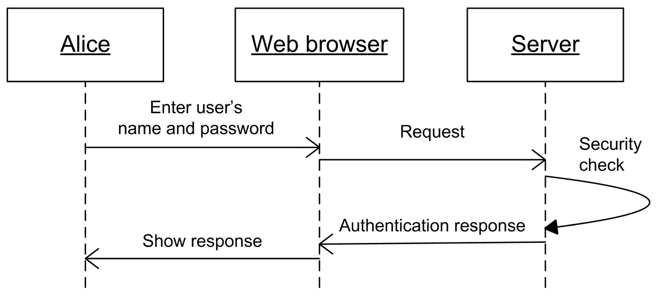

Figure 2: Classical way of authentication on the Web

The classical way of authenticating a user in a web application, is to ask the user for a login and password. Then, the login and password are sent (see Figure 2) directly to the server; and the server responds to the request. If security is needed, an HTTPS protocol is used. But the credentials are still sent through the Internet and servers are able to read them, even if the password is stored in a hashed and salted form. For example, very often a user visits a web site whose certificate cannot be verified by the user's browser. However, the browser usually enables the user to accept the certificate although its verification was unsuccessful. Furthermore, to take the full potential of HTTPS technology, a user should buy a certificate issued by a well-known certificate authority and then the certificate should be installed on the user's browser. Installing a certificate to a web browser happens rarely and is prone to various attacks [21].

4. DESIGN APPROACH

To authenticate a user, ZKP protocols require more than two steps: a user's request, a server's challenge, a user's response, and a server's response. Due to the ZKP nature, the server's challenges cannot be delivered to the user with the login form. Therefore, such protocols differ from the classical approaches, and require more flexible communication means. In addition, the authentication process is more computationally expensive and consumes more bandwidth.

To satisfy these requirements, we decided to combine Ajax (Asynchronous JavaScript and XML)[4] that is an asynchronous web technology with a graph isomorphism protocol. We chose this NP problem since it does not require user's browsers to find coprime numbers nor multiplicative inverses. Computations are performed using natural numbers and matrices; therefore, such arithmetic is easier to implement than, for example, elliptic curves. Our technology choice did not require plugins and was supported by default settings of the vast majority of web browsers. Hence, integration with existing applications was straightforward. Furthermore, their start-up time was negligible.

Due to the limited space of the paper, we describe only the two core elements of the idea: the Authentication procedure and the Private-key algorithm.

4.1 Authentication procedure

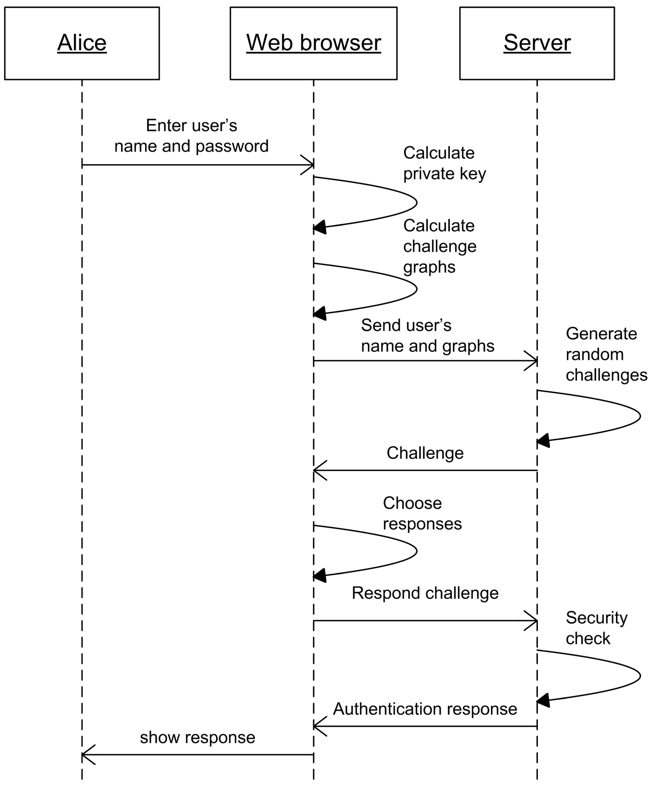

Figure 3: ZKP implementation on the Web

A sequence diagram presented on Figure 3 shows our approach in detail. *Alice* types her username and password, but the password never leaves her browser. In contrary to existing approaches like HTTP MD5 digest, the server does not have any information that would allow for impersonation *Alice* at another server. The browser uses the password to calculate her public-private key pairs (see Section 4.2) and then executes the ZKP protocol. The browser is responsible for a number of new tasks: calculating private keys from passwords, generating challenge graphs, and responding to the challenge. A server has only one more responsibility: generating random challenges. There is also one more interaction between a browser and a server in comparison with classical approaches (see Section 3). Thus, the main question is if the new approach is feasible and will not require long waiting times for users.

[4]Ajax: A New Approach to Web Applications: http://www.adaptivepath.com/ideas/essays/archives/000385.php

4.2 Private-key algorithm

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 function. Such a transformation must always generate the same-size permutations.

4.2.1 Transformation

To compute a hash from a password we use Secure Hash Algorithm - Version 1.0 (SHA1)[5]. Such a hash is composed of numbers 0-9, a-f; and therefore, we can interpret the hash as a hexadecimal number.

All the generated hashes have the same size, i.e. 160 bits and so their corresponding index numbers when sorted in lexicographical order in a table are from *38!+1* to *40!*. If the hash alone were used as an index into a table of permutations stored in lexicographical order, different hashes would be mapped to permutations of varying lengths. Thus, we extend all the hashes with a hexadecimal character b at the beginning of each obtained string. This modification causes that the new values of hashes point to index positions between *41!+1* and *42!*; therefore, all the generated permutations have the same length.

The key size can be changed, if another hash generation algorithm is used.

4.2.2 Calculating a permutation at a certain position

The web-application calculates the permutation indicated by the extended hash in two steps, the first being based on the observation that all the natural numbers can be represented as:

$$a_0 0! + a_1 1! + a_2 2! + ... + a_k k! \text{ for any natural number k}$$

Moreover, such a representation is unambiguous. There is a need to find the $a_1 1! + a_2 2! + ...a_k k!$ representation of the extended hash (see Section 4.2.1). This can be done using Algorithm 1, which converts a given decimal number to vector a[]. The GreatestFactorial() function returns the smallest factorial that is greater than the given number.

Algorithm 2 converts vector a[] to a table permutation[] that contains the result permutation.

5. SECURITY

The implementation of the algorithms can be done as a browser extension or as a script. In the case of a script, for example JavaScript, two parties of the protocol are controlled by the server. Hence, if the server cannot be trusted,

[5]RFC 3174, US Secure Hash Algorithm 1 (SHA-1): http://tools.ietf.org/html/rfc3174

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
```

such implementation gives no benefits. If the implementation is done as a browser extension, then it is browser dependent.

In our considerations we also assume that that the website is free from Cross-site scripting (XSS) vulnerabilities and that the communication between *Alice* and her browser is completely secure. Hence, the user's computer does not contain any viruses or malicious software. In addition, the human factor is also involved: she can write down her credentials; during authentication procedure, somebody can see her password looking at the keyboard and then re-use it. Moreover, our prototype requires JavaScript. A user who turns it off, will not be able to use ZKP approach. Another problem is phishing, we do not consider it in this paper since our solution can be easily combined with existing approaches, for example sign-in seal [1].Our algorithm is also dependent on SHA-1, which we use in the private-key algorithm (see Section 4.2). It was supposed that 2^{80} operations are needed to find a collision in SHA-1. However, in August 2005 was presented an attack[6] on the full version of SHA-1 that lowered the required computational complexity to 2^{63}. Security of our application depends also on the NP problem and a dictionary attack feasibility.

5.1 Attacks on Graph Isomorphism

Efficient algorithms for some very specific graphs were proposed; Babai [2] discussed probabilistic algorithms; Corneil et al. [6] described algorithms for determining if two graphs are isomorphic; however, in our approach the problem is to find the permutation, not to determine if two given graphs are isomorphic. Therefore, this work can be used only to increase speed of brute-force attacks. There are also informal publications [7] but such hypotheses were already questioned by counterexamples. Our protocol uses square matrices of size 41; therefore there are 41! possibilities of such matrices. The generated graphs are completely random, and thus none of the proposed algorithms are useful to find the secret.

[6]SHA1 attack: http://www.iacr.org/conferences/crypto2005/rumpSchedule.html

5.2 Dictionary Attacks

Florencio and Herley [8] performed research about password habits. They gathered a lot of statistical data and shown that most users use plenty of passwords every day, but the passwords are low quality and the users tend to re-use and forget them.

If a user's public key is available or if the server is compromised, an attacker may check password candidates offline. Such attacks can be very effective [17], especially if users tend to choose low entropy passwords. Hence, guessing attacks were topics of many studies [11, 3].

If the communication between the server and *Alice* is encrypted, the only feasible way to attack the protocol is to run a trivial online dictionary attack and attempt authentication with password candidates. This attack can be, however, detected and slowed down by using captchas [18].

6. IMPLEMENTATION AND EVALUATION

As a proof of the concept we implemented a prototype. Its server side was a Java servlet that was deployed on a Jakarta Tomcat server. The code was not platform specific to make it easily portable to other languages. The client side was implemented in Ajax. Such a choice, however, requires users to trust the server.

We performed our evaluation on 6 different configurations

- Conf 1 - laptop, 1 CPU 2 GHz, 2 GB of RAM, Firefox 2
- Conf 1b - the same laptop with Internet Explorer 7
- Conf 1c - the same laptop with Opera 9
- Conf 2 - laptop Centrino 1.4GHz (1 CPU), 256 RAM, Firefox 2
- Conf 3 - desktop computer with 1 CPU 2.8 GHz, 2 GB RAM, Firefox 2
- Conf 4 - laptop with 1 CPU Xenon 1.4 GHz, 768 MB of RAM, Firefox 2

6.1 Performance tests

An important challenge for our application was to satisfy the existing response-time standards. Such a standard was proposed in 1968 by Miller [15]. He defined three main time constraints: 0.1 second is for keeping the user attention attracted; 1 second is for keeping a user flow though uninterrupted; finally, 10 seconds for keeping user's attention on the dialog. Three decades after his findings, those rules are still applicable for web applications [5].

To indicate if our approach was feasible for web authentication, we evaluated our implementation in two ways: we measured authentication time for a given amount of challenges, and we counted the data that was exchanged to achieve certain security levels. Figure 4 shows the results that were obtained for *Conf 1*. For the Firefox 2 browser, 1 second, and therefore a user's impersonation risk level at $1/2^{30}$, was crossed for more than 30 challenges. Then,

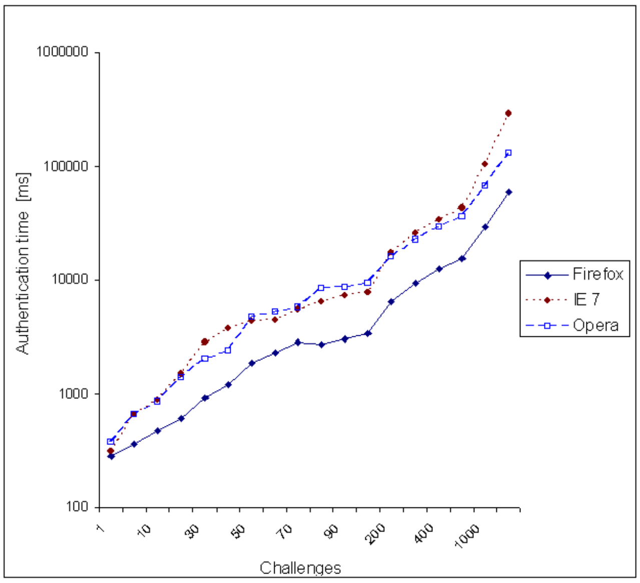

Figure 4: Authentication time for various browsers up to 2000 challenges

we also measured and compared our implementation in two other browsers: Internet Explorer 7 and Opera 9. Both browsers needed more than 1 seconds to achieve security risk level $1/2^{20}$. To process 100 challenges, Opera, which performed worse, needed 9.3 seconds The Opera browser tended to fail the authentication process for 300 and more challenges although the credentials were correct. When the computation lasted longer than 3.5 seconds, the user's thoughts flow was always interrupted by Internet Explorer which was displaying a message: 'A script on this page is causing Internet Explorer to run slowly. If it continues to run, your computer may become unresponsive. Do you want to abort the script? yes/no'. If the computation were not canceled, the browser returned correct results up to 1000 challenges. Therefore, the authentication process was transparent from the user's perspective up to 30 challenges. The additional calculations we performed on the obtained results showed that one challenge in Firefox 2 cost approximately 32 ms, 84 ms for Opera, and 82 ms for Internet Explorer.

6.2 Data measurements

Our other objective was to count data that was exchanged between a user's browser and a server. In the classical approaches, we cannot set the security level, and thus we always sent the same amount of data that contains a login and a password. In zero-knowledge proof approaches, we sent more data, the more confidence we need. This parameter is neither browser nor hardware dependent and a single challenge costs approximately 394 bytes. Moreover, the data transmitted by our server in step 6 was equal to the amount of challenges, whereas in step 10 the server always sends 4 or 5 bytes depending on the authentication result. Table 1 summarizes our evaluation. All of the tested browsers and configurations were able to execute 10 challenges below 1 second, and 100 challenges for less than 10 seconds. Finally, we note that in all ZKP protocols together with linear growth of amount of challenges (t), we have an exponential drop of cheating probability (2^{-t}).

7. RELATED WORK

7.1 Existing Web Solutions

Yahoo! [9] proposed a simple challenge-response solution. This idea is very similar to HTTP Digest access authentication that was proposed in 1999. These solutions use, however, MD5 which is considered to be insecure. The server is supposed to store passwords in a plain-text form or to have a hash. Even having only the hash, the server is able to impersonate the user.

7.2 PAKE

Password-Authenticated Key Exchange (PAKE) is a family of protocols that affords a reasonable level of security using short memorized passwords for protecting information over insecure channels. Such protocols are also a topic of IEEE P1363[7] standard working group.

Encrypted Key Exchange (EKE) [4] that was introduced in 1992, combines both asymmetric and symmetric cryptography findings. The protocol has several versions and was followed by other propositions. Simplified Password-authenticated Exponential Key Exchange (SPEKE) protocol was developed by Jablon [12] for commercial purposes. The main difference in comparison with EKE is that the password is used to influence the selection of the generator parameter in the session-key generation function.

Secure Remote Password (SRP)[8] [20] was developed in 1997. Security of this protocol is dependent 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 modulo exponentiations, whereas EKE requires only one. Moreover, the protocol is vulnerable to offline dictionary attacks.

8. CONCLUSIONS AND FUTURE WORK

We presented an approach for a zero-knowledge authentication on the Web. This solution allows the server to specify confidence and security levels. We demonstrated that in order to increase security, some of the computations should be performed by the users' browsers. We have adapted and extended a graph isomorphism algorithm for web applications, in which users still use logins and passwords pairs that they are accustomed to. They can take advantage of asymmetric cryptography findings: a prover is able to prove the knowledge of a secret without revealing it. The approach taken reduces computation and communication cost, which are crucial for any web applications.

The presented implementation requires the server to be trusted. Therefore, we also plan to deliver a browser plugin implementing the proposed protocol.

Finally, we thoroughly validated the feasibility of our idea by evaluating a prototype implementation. We plan to perform more tests involving SSL, SRP and OpenId[9] Protocols.

[7]IEEE P1363: http://grouper.ieee.org/groups/1363/
[8]RFC 2945: http://tools.ietf.org/html/rfc2945
[9]OpenId: http://openid.net/

Configuration	Challenges	Cheating probability [%]	Bytes exchanged	Time [ms]	Single challenge cost [ms]
conf1	10	0.09765625	4045	469	46.9
conf1b				844	84.4
conf1c				875	87.5
conf2				1302	130.2
conf3				484	48.4
conf4				1522	152.2
conf1	100	7.88861E-29	39595	3422	34.22
conf1b				9329	93.29
conf1c				7703	77.03
conf2				8070	80.70
conf3				3703	37.03
conf4				9824	98.24

Table 1: Sample data for various configurations

9. ACKNOWLEDGMENTS

The work presented in this paper was supported (in part) by the Lion project supported by Science Foundation Ireland under Grant No. SFI/02/CE1/I131 and Enterprise Ireland under Grant No. *ILP/05/203*. A method and apparatus for authenticating a user is a pending patent in the Irish patent office. The authors would like to thank Peter Corcoran, Stefan Decker and the eLITE project members for their help and fruitful discussions.

10. REFERENCES

[1] What is a sign-in seal? http://security.yahoo.com/article.html?aid=2006102507 last viewed on 10 april 2008.

[2] L. Babai, P. Erdos, and S. M. Selkow. Random graph isomorphism. *SIAM Journal on Computing*, 9(3):628–635, 1980.

[3] M. Baudet. Deciding security of protocols against off-line guessing attacks. In *CCS '05: Proceedings of the 12th ACM conference on Computer and communications security*, pages 16–25, New York, NY, USA, 2005. ACM.

[4] S. M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. pages 72–84.

[5] A. Bouch, A. Kuchinsky, and N. Bhatti. Quality is in the eye of the beholder: meeting users' requirements for internet quality of service. In *CHI '00: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 297–304, New York, NY, USA, 2000. ACM Press.

[6] D. G. Corneil and C. C. Gotlieb. An efficient algorithm for graph isomorphism. *J. ACM*, 17(1):51–64, 1970.

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

[8] 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*, pages 657–666, New York, NY, USA, 2007. ACM Press.

[9] S. Garfinkel. Fingerprinting your files. mit technology review. Technical report, August 2004.

[10] 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*, pages 291–304, New York, NY, USA, 1985. ACM Press.

[11] L. Gong, M. A. Lomas, R. M. Needham, and J. H. Saltzer. Protecting poorly chosen secrets from guessing attacks. *IEEE Journal on Selected Areas in Communications*, 11(5):648–656, 1993.

[12] D. P. Jablon. Strong password-only authenticated key exchange. *SIGCOMM Comput. Commun. Rev.*, 26(5):5–26, 1996.

[13] N. Koblitz. Elliptic curve cryptosystems. 48(177):203–209, Jan. 1987.

[14] A. J. Menezes, S. A. Vanstone, and P. C. V. Oorschot. *Handbook of Applied Cryptography.* CRC Press, Inc., Boca Raton, FL, USA, 1996.

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

[16] V. S. Miller. Use of elliptic curves in cryptography. In *Lecture notes in computer sciences; 218 on Advances in cryptology—CRYPTO 85*, pages 417–426, New York, NY, USA, 1986. Springer-Verlag New York, Inc.

[17] A. Narayanan and V. Shmatikov. Fast dictionary attacks on passwords using time-space tradeoff. In *CCS '05: Proceedings of the 12th ACM conference on Computer and communications security*, pages 364–372, New York, NY, USA, 2005. ACM.

[18] B. Pinkas and T. Sander. Securing passwords against dictionary attacks. In *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*, pages 161–170, New York, NY, USA, 2002. ACM.

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

[20] T. Wu. The secure remote password protocol. In *Proceedings of the 1998 Internet Society Network and Distributed System Security Symposium*, pages 97–111.

[21] H. Xia and J. C. Brustoloni. Hardening web browsers against man-in-the-middle and eavesdropping attacks. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 489–498, New York, NY, USA, 2005. ACM Press.