Single Password, Multiple Accounts

Mohamed G. Gouda Alex X. Liu[1] Lok M. Leung[2] Mohamed A. Alam[2]

Department of Computer Sciences,
The University of Texas at Austin,
Austin, Texas 78712-0233, U.S.A.
{gouda, alex, raymond1, malam}@cs.utexas.edu

Abstract. Most users have multiple accounts on the Internet where each account is protected by a password. To avoid the headache in remembering and managing a long list of different and unrelated passwords, most users simply use the same password for multiple accounts. Unfortunately, the predominant HTTP basic authentication protocol (even over SSL) makes this common practice remarkably dangerous: an attacker can effectively steal users' passwords for high-security servers (such as an online banking website) by setting up a malicious server or breaking into a low-security server (such as a high-school alumni website).
To solve this problem, we propose a new password protocol that is simple, secure, efficient and user-friendly. In terms of *simplicity*, the protocol only involves three messages, and the protocol is easy to understand and implement. In terms of *security*, the protocol is secure against the attacks that have been discovered so far including the ones that are difficult to defend, such as the malicious server attacks described above and the recent phishing attacks. In terms of *efficiency*, each run of our protocol only involves a total of four computations of a one-way hash function. In terms of *usability*, the protocol requires a user to remember only one password consisting of eight (or more) random characters, and this password can be used for all of his accounts.

Keywords: Passwords, Authentication, Malicious Server Attacks, Protocol Design, Network Security

1 Introduction

Authentication of an entity is usually done by verifying one or more of the followings:

1. Something the entity *is* (by biometric techniques, such as fingerprint or voiceprint identification)
2. Something the entity *has* (by PKI certificate, ID cards, smart cards),

[1] Alex X. Liu is the corresponding author of this paper.
[2] Lok M. Leung and Mohamed A. Alam participated in this work while they were undergraduate students in The University of Texas at Austin.

3. Something the entity *knows* (by passwords, PINs)

Of these three authentication methodologies, only the last two are suitable for remote authentication. On today's Internet, server authentication is usually done by SSL [5] using PKI certificates, which is something the server has, while client authentication is usually done using passwords, which is something the client knows.

1.1 The Problem

Many people today have multiple accounts on the Internet. For example, one may have an email account on *www.yahoo.com*, a travel account on *www.travelocity.com*, a credit card account on *www.discovercard.com*, a banking account on *www.chase.com*, an online stock trading account on *www.fidelity.com*, etc. Forrester Research reports that a typical web user manages an average of 15 passwords on a daily basis [10]. Most of these accounts are protected by passwords. As more services move to the Internet, the number of accounts a user needs to manage is expected only to grow. If one uses different and unrelated passwords for each account, then remembering all these unique passwords is a daunting task. It has been observed in [1] that a typical user can only remember 4 or 5 passwords effectively. Because of this, the common practice is to use the same password for multiple accounts. However, the predominant HTTP basic authentication protocol (even over SSL) [4] makes this practice remarkably dangerous because the protocol allows a server to know the password of each of its clients.

Let us first examine how the HTTP basic authentication protocol works. A client C first registers with a server (such as a web site) S using password P. The registration results that S stores in its password file the pair of user name C and password verification data $MD(P)$. Here MD denotes a message digest (one-way hash) function such as MD5 [16] and SHA-1 [18]. Later on when C wants to login on S, C sends his user name C and password P to S. Then S applies the message digest function MD to the received password and compares the result with the stored password verification data $MD(P)$. If they are equal, then the authentication of C is successful; otherwise it is unsuccessful. The HTTP basic authentication protocol usually runs on top of SSL [5], which allows the client to authenticate the server using certificate mechanisms, and provides an encrypted channel between the client and the server.

Allowing a server to know the passwords of its clients such as what the HTTP basic authentication protocol does, is dangerous for two reasons.

1. *A server may be untrustworthy.* An attacker can set up a malicious server (at cost as low as $100 U.S. dollars), and allures people to register using passwords by offering free goods or services. The attacker can reasonably estimate that some of his clients use the same password for their financial accounts. After gathering those passwords from the registered clients, the attacker can impersonate them to login on some financial servers such as an online banking or stock trading server, which could cause significant loss to the clients.

2. *A server may be compromised.* Instead of directly trying to break into a high-security server such as an online financial server to gain unauthorized access, which would be difficult, an attacker can first try to break a low-security server such as an art bulletin board set up by an amateur. Once a low-security server is broken, which is relatively easy, since some clients of the broken poorly-defended server may use the same password for their accounts on high-security servers, the attacker can then use the captured passwords to access those high-security servers.
   Some users classify servers into high-security servers and low-security servers, and use one password for high-security servers and another different password for low-security servers. This practice is remarkably insecure. First, this classification is highly subjective and a typical user may not do it right. This would cause the same password being used on both a high-security server and a low-security server. Second, the two different passwords are possibly related, such as by being formed according to the same pattern. Thus knowing one password might enable an attacker to discover the other one. Third, not every high-security server is trustworthy. For example, an employee of a high-security server may be bribed to breach the passwords of its clients to attackers. In addition, gathering a user's personal information from multiple low-security servers could be helpful in discovering the other password for high-security servers.

We call the attacks of stealing passwords by setting up a malicious server or by compromising a benign server "*malicious server attacks*". We use the term "malicious servers" to denote the servers that are either set up or compromised by an attacker. When using the HTTP basic authentication protocol, a user has to use a different and unrelated password for every different server to prevent malicious server attacks. Remembering many totally different and unrelated passwords is certainly not viable for most users. And writing down all the user names and passwords on a piece of paper is certainly not a good idea because compromising of this list could cause serious loss.

1.2 Our Solution

In this paper, we propose a new password protocol, named Single Password Protocol (short for SPP), which allows a user to use one single password (and one single user name) for all of his accounts while defeating malicious server attacks. SPP embodies two basic techniques: challenge/response and one-time server-specific tickets.

SPP works basically as follows. Let P be the single password that a client C remembers. When C registers with a server S, C generates a challenge and ticket verification data, then sends them to S, and S stores them in its password file. Later on, when C tries to login on S, S prompts C with the stored challenge. Then C uses the challenge, the server's name S, and his password P to mint a one-time server-specific ticket, together with a new challenge and new ticket verification data, and sends them to S. Then, S verifies the received ticket using

the stored ticket verification data. If the ticket is valid, then the authentication of C is successful, and S subsequently replaces the stored challenge and ticket verification data by the new challenge and new ticket verification data that S received along with the one-time server-specific ticket from C.

SPP allows a client to use the same user name and password for all of his accounts while defeating malicious server attacks because of the following two reasons.

1. *A client never sends out his password.* In SPP, a client uses the challenge received from a server, the server's name, and his password, to mint a one-time server-specific ticket, and sends the one-time server-specific ticket, instead of his password, to the server for authentication. The password of a client is used for minting a one-time server-specific ticket, and it is the ticket that is used for authenticating the client. Therefore, a malicious server has no chance of capturing the passwords of its clients.
2. *Each ticket can only be used once.* At any moment, for one client, ticket verification data in different servers are different. Therefore, a malicious server cannot replay a received ticket to other servers to gain unauthorized access. At any server, for one client, ticket verification data are changed unpredictably after each successful login.

1.3 Why SPP

SPP is designed for client authentication across the Internet. It has the following three exceptional properties:

1. *Simple.* SPP only involves three messages: first, client C sends to server S his user name C; second, S sends to C the stored challenge; third, C sends to S a one-time server-specific ticket, together with a new challenge and new ticket verification data. This simple structure makes SPP easy to understand and easy to implement.
2. *Secure.* The security of SPP is based on two reasonable assumptions. First, SPP is layered on top of SSL. This assumption is reasonable because SSL has become the industry standard for securing communication over the Internet and has been built into all major web browsers, web servers, email clients, email servers, etc. Second, the single password that a client uses to protect all of his accounts is a string of eight (or more) random characters, which is not guessable. This assumption is reasonable because using SPP, a user needs to remember only one password for all of his accounts. Under these two assumptions, SPP is secure against a long list of attacks: eavesdropping attacks, message replay attacks, message loss attacks, message modification attacks, message insertion attacks, man-in-the-middle attacks, brute force attacks, online dictionary attacks, off-line dictionary attacks, password file compromise attacks, message log file compromise attacks, Denning-Sacco attacks [3], malicious server attacks, server spoofing attacks, social engineering attacks and even phishing attacks.

3. *Efficient.* SPP only involves one type of computation, hashing (i.e., computing message digest), which requires a negligible amount of processing time. In addition, it only involves four executions of this hashing computation: one execution on the server side and three executions on the client side.
   SPP itself does not involve any sort of encryption/decryption operations, although the messages in SPP are encrypted by the underlying SSL protocol. This property benefits SPP in both performance and availability. In terms of performance, computing a message digest requires orders of magnitude less processing time than performing an encryption/decryption. In terms of availability, if a password protocol stores some encrypted data using some encryption algorithm on the server side, then a client cannot login on his accounts on a machine that the corresponding decryption algorithm is not available.
4. *User-friendly.* Using SPP, a user only needs to remember one password, and this password can be used for all of his accounts. In addition, this password can be changed easily at the user's will during each login without the notice of the server.

The rest of this paper proceeds as follows. In Section 2, we present the single password protocol. In Section 3, we review and examine existing password protocols and compare them with our protocol. Section 4 concludes.

2 Single Password Protocol

In this section, we present our Single Password Protocol (SPP for short). For ease of understanding, we first present four intermediate versions of it starting from the HTTP basic authentication protocol. We show that each intermediate version is vulnerable to a particular attack, and each attack is countered by the following versions.

The notations used in this section is listed in the following table.

C	Client
S	Server
P	Password remembered by client
n, n_i	Random number
$MD()$	Message digest (one-way hash) function
$MD^2()$	$MD(MD())$
$\|$	Concatenation

Table 1: Notations

Note that the message digest (one-way hash) function $MD()$ used in paper is assumed to have the property that a polynomial-bounded adversary should not be able to gain any information about the input by examining the output of such a function. Example viable candidates include MD5 [16] and SHA-1 [18].

2.1 Version I

The first version is the HTTP basic authentication protocol, which is shown in Figure 1. In this protocol, for each registered client C, whose password is P, server S stores $MD(P)$ as password verification data in a password file. We assume server S has some out-band means of authenticating the initial registration, discussion of which is out the scope of this paper. Each time client C tries to login on server S, C sends his user name C and password P to S. Then S uses its stored password verification data $MD(P)$ to authenticate P. More precisely, S first computes the message digest of the received password, and then compares the result with its stored password verification data $MD(P)$. If they are equal, then the authentication is successful.

C knows	P
S stores	$MD(P)$
$C \rightarrow S$:	$C\|P$

Fig. 1. Version I (vulnerable to malicious server attacks)

Despite the wide use of the HTTP basic authentication protocol, it suffers from *malicious server attacks*. To launch this type of attack, an attacker first sets up a malicious server and allures people to register with him using a user name and a password. Because people often use the same user name and password on multiple servers, the attacker is able to henceforth impersonate such clients to other servers such as an online banking website. Clearly, malicious server attacks can be extremely damaging.

2.2 Version II

To counter malicious server attacks, we use the challenge/response technique. When client C registers with server S, he first generates a random number n, computes $MD(n|P)$, and then sends them to S. Server S stores n as a challenge and $MD(n|P)$ as password verification data in its password file for client C. Note that the party who remembers n is server S, not client C, although n is generated by C. Client C still only remembers his password P. Since C generates a random number as a challenge independently for each registration, the possibility that the two challenges on two different servers are equal is negligible. Each time client C tries to login on server S, S sends n as the challenge back to C. Then C computes the message digest of $(n|P)$ and sends the result as the response to S. Server S compares this response with the stored password verification data $MD(n|P)$. If they are equal, then the authentication is successful. Note that client C does not send his password P to S, even in the initial registration. Knowing $MD(n|P)$ does not allow server S to gain any information about the

password P, and does not allow server S to impersonate client C to other servers. Therefore, this protocol is secure against malicious server attacks. This password protocol as version II is shown in Figure 2.

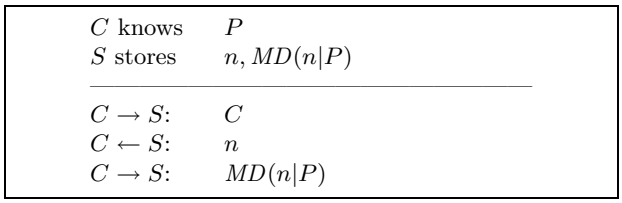

Fig. 2. Version II (Vulnerable to password file attacks)

Unfortunately, this password protocol is vulnerable to *password file compromise attacks*. To launch this type of attack, an attacker first steals the password file of a server by some means such as breaking into the server or colluding with insiders of the server; second, the attacker tries to discover either the password of the client or the valid response that the client would use to login on the server using the information in the appropriated password file. According to the above version II protocol, stealing the password file of server S enables the attacker to impersonate any of its clients because the response is the same as the password verification data.

2.3 Version III

To counter password file compromise attacks, the response should not be computable from the password verification data. Accordingly, we change the password verification data from $MD(n|P)$ to $MD^2(n|P)$. Note that an attacker cannot deduce the response $MD(n|P)$ from the password verification data $MD^2(n|P)$ in polynomial time. This password protocol of version III is shown in Figure 3.

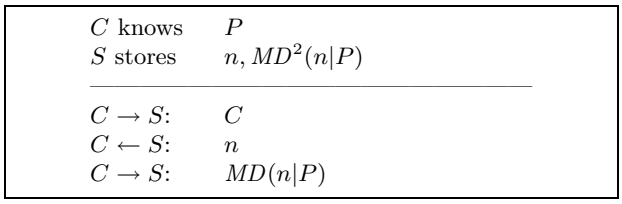

Fig. 3. Version III (Vulnerable to message log file attacks)

Unfortunately, this password protocol is vulnerable to *message log file compromise attacks*. Some servers with stringent security requirements record every

message that they send or receive into a message log file. To launch message log file compromise attacks, an attacker first steals the message log file of a server just like stealing the password file in the password file compromise attack; second, the attacker tries to gain unauthorized access using the information in the appropriated message log file. In the above protocol of version III, the response that client C uses to authenticate himself to server S, namely $MD(n|P)$, may be stored in the message log file of S. Stealing such a message log file of server S enables the attacker to impersonate any of its clients because the response that a client uses to authenticate himself to a server is always the same.

2.4 Version IV

To counter message log file compromise attacks, we use the technique of one-time tickets. Most tickets in real life, such as movie tickets, are one-time tickets in the sense that they can only be used once. Here, we want the response from a client to contain a one-time ticket, which can be used to authenticate the client only once. Therefore, the response from a client in each login needs to be unique every time. To achieve this uniqueness, we change the random number n involved in the protocol accordingly. Let n_i be the challenge and $MD^2(n_i|P)$ be the ticket verification data stored in server S for client C. Every time when C tries to login on server S, S sends the challenge n_i back to C. Then, C first computes the one-time ticket $MD(n_i|P)$; second, C chooses a new challenge n_{i+1} and computes new ticket verification data $MD^2(n_{i+1}|P)$; third, C sends all of them $(MD(n_i|P)|n_{i+1}|MD^2(n_{i+1}|P))$ to S. When S receives this message, S first verifies the received one-time ticket $MD(n_i|P)$ using the stored ticket verification data $MD^2(n_i|P)$. If the one-time ticket is valid, then the authentication is successful and subsequently S replaces the stored n_i and $MD^2(n_i|P)$ by n_{i+1} and $MD^2(n_{i+1}|P)$. This password protocol of version IV is shown in Figure 4.

C knows	P
S stores	$n_i, MD^2(n_i\|P)$
$C \rightarrow S$:	C
$C \leftarrow S$:	n_i
$C \rightarrow S$:	$MD(n_i\|P)\|n_{i+1}\|MD^2(n_{i+1}\|P)$

Fig. 4. Version IV (Vulnerable to server spoofing attacks)

Unfortunately, this password protocol is vulnerable to *server spoofing attacks*. To launch server spoofing attacks, a malicious server S first pretends to client C and tries to login on another benign server S', who has C as a client and stores n'_i as the challenge and $MD^2(n'_i|P)$ as the ticket verification data for client C. Responding to the login request from S, S' sends the challenge n'_i to the

malicious server S. After malicious server S obtains n'_i from server S', S stops communicating with S'. Later on, when C tries to login on S, S sends to C the stolen challenge n'_i, instead of the correct one n_i. Since C only remembers his password P, C assumes that this challenge is the one stored in S. Consequently, C sends to S the response $MD(n'_i|P)|n'_{i+1}|MD^2(n'_{i+1}|P)$, which can be used by the malicious server S to login on S' as client C.

2.5 Final Version

To counter server spoofing attacks, we make the one-time ticket from a client to be specific to the server that the ticket is intended for. More precisely, we change the one-time ticket from $MD(n_i|P)$ to $MD(n_i|S|P)$, and accordingly change the ticket verification data from $MD^2(n_i|P)$ to $MD^2(n_i|S|P)$. In this way, when malicious server S sends to client C the wrong challenge n'_i, which he steals from benign server S', C will send to S a wrong response $MD(n'_i|S|P)|n'_{i+1}|MD^2(n'_{i+1}|S|P)$, which cannot be replayed as a valid response to S'. The valid response to S' should be $MD(n'_i|S'|P)|n'_{i+1}|MD^2(n'_{i+1}|S'|P)$. These modifications complete our single password protocol, which is shown in figure 5.

C knows	P
S stores	$n_i, MD^2(n_i\|S\|P)$
$C \rightarrow S$:	C
$C \leftarrow S$:	n_i
$C \rightarrow S$:	$MD(n_i\|S\|P)\|n_{i+1}\|MD^2(n_{i+1}\|S\|P)$

Fig. 5. Single Password Protocol (SPP)

Note that in this paper, we assume that SPP is running on top of SSL. A client authenticates a server using SSL before SPP starts. Therefore, a malicious server cannot forge its identity.

3 Related Work

Many password protocols have been proposed, especially in the past decade. In this section, we review these password protocols and compare them with SPP.

3.1 One-Time Password Protocols

In one-time password protocols, a client uses a different password for every authentication with a server. There are mainly two such protocols: Lamport's

one-time password protocol [12] (which was implemented in S/KEY [7–9] and OPIE [14]) and Rubin's one-time password protocol [17]. Both one-time password protocols were designed before the wide deployment of SSL. The motivation for both one-time password protocols are preventing eavesdropping attacks, which are now easily defeated by the confidentiality provided by the widely deployed SSL. To make either of the above one-time password protocols secure against malicious server attacks, a client has to remember multiple "seed" passwords or multiple lists of one-time passwords for multiple accounts. In addition, both one-time password protocols need a client to re-register with its server when the client uses up his current list of one-time passwords. This re-registration requirement is extremely inconvenient for clients because each registration costs their significant effort, and it is extremely harmful for e-commerce servers because it can cause annoyed customers to leave. In contrast, SPP only requires a user to remember one single password for all of his accounts.

3.2 Strong Password Protocols

Strong password protocols often have strong security properties, but they usually require computationally intensive operations such as modular exponentiations, asymmetric encryptions/decryptions, etc. Many such protocols have been proposed, such as EKE [2] and SRP [19].

Most of these strong password protocols were proposed before the wide deployment of SSL. They are mostly designed to achieve the following two goals: (1) To establish a session key between a client and a server after the server authenticates the client using passwords. This is not a must-have functionality for a password protocol any more because a session key is established after the client authenticates the server using SSL. (2) To prevent dictionary attacks. This goal is always expensive to achieve. Using SPP, dictionary attacks are not a problem because a client is required to choose a single strong password to protect all of his accounts.

Those strong, and consequently heavy weight, password protocols are not well-suited for Internet authentication because of the heavy computational cost [6]. A password protocol that is desirable for the Internet must be computationally efficient in both the server side and the client side because of two reasons. First, on the server side, a server on the Internet (mostly a web server) often has heavy service load and consequently cannot afford significant computation for authenticating every client. Second, on the client side, the computing device of a client may have limited computational power, for example, a palm or a cell phone. In contrast, SPP is a light weight password protocol that involves negligible amount of processing time for both the server side and the client side.

3.3 Web-specific Password Protocols

Except for the HTTP basic authentication protocol mentioned in Section 2, the HTTP specification provides another authentication mechanism: digest authentication protocol [4]. The HTTP digest authentication protocol uses the

challenge/response technique, which basically works as follows. When client C registers with server S, S stores C's password P. When C wants to login on S, S generates a nonce n and sends it to C as a challenge. Then C computes $MD(n|P)$ and sends the result to S as a response. Server S verifies the received response using the stored password P and the generated nonce n. Because a server knows the passwords of its clients, the HTTP digest authentication protocol is vulnerable to malicious server attacks and password file compromise attacks. In addition, because the response that a client sends to a server is not specific to that server, the HTTP digest authentication protocol is vulnerable to server spoofing attacks and phishing attacks.

3.4 Single Sign-on Protocols

The basic idea of single sign-on protocols, such as the Microsoft Passport protocol [15], is to use one central server to authenticate clients for multiple servers, instead of each server authenticating clients by itself. Although single sign-on protocols provide clients the convenience of remembering only one password, which is registered in the single sign-on server, such protocols have the following main disadvantages. First, single sign-on protocols introduces a single point of failure. If the single sign-on server fails working, then all the servers that depend on it fail authenticating their clients, which is extremely destructive. Compromising the single sign-on server has high pay-offs for attackers and thereby makes attack attempts more likely. In fact, Microsoft Passport protocol has already been shown to have security flaws in [11]. Second, as pointed out in [13], single sign-on protocols have high cost of integration because servers need to register with the single sign-on server in order to get the service, and consequently lacks universal adoption. For the servers that do not use single sign-on protocols, a client has to register with them individually using passwords. If a client registers with a malicious server using the same password that he uses for the single sign-on server, then the malicious server can impersonate the client to login on multiple servers.

4 Conclusions

The predominant HTTP basic authentication protocol (over SSL) makes the common practice of using the same password for multiple accounts remarkably dangerous: an attacker can effectively steal users' passwords for high-security servers by setting up a malicious server or breaking into a low-security server. To defeat this type of malicious server attack, we propose the Single Password Protocol (SPP). SPP employs two basic techniques: challenge/response and one-time (server-specific) tickets.

SPP has four favorable properties that make it a promising password protocol for client authentication on the Internet. First, SPP is simple. It only has three messages, which makes SPP easy to understand and implement. Second, SPP is secure. SPP defeats all the existing attacks that an attacker can launch on

password protocols. Third, SPP is efficient. SPP only involves a total of four computations of a one-way hash function, which requires negligible amount of processing time. Last, SPP is user-friendly. SPP enables a user to safely use one password for all of his accounts.

References

1. A. Adams and M. A. Sasse. User are not the enemy. *Communications of the ACM*, 42(12):40–46, 1999.
2. S. M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *Proceedings of the 1992 IEEE Computer Society Conference on Research in Security and Privacy*, pages 72–84, 1992.
3. D. E. Denning and G. M. Sacco. Timestamps in key distribution systems. *Communications of the ACM*, 24(8):533–536, 1981.
4. 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*, 1999.
5. A. O. Freier, P. Karlton, and P. C. Kocher. The ssl protocol version 3.0 internet draft. http://wp.netscape.com/eng/ssl3/draft302.txt. March 1996.
6. K. Fu, E. Sit, K. Smith, and N. Feamster. Dos and don'ts of client authentication on the web. In *Proceedings of the 10th USENIX Security Symposium*, August 2001.
7. N. Haller. The s/key one-time password system. *RFC 1760*, 1995.
8. N. Haller and C. Metz. A one-time password system. *RFC 1938*, 1996.
9. N. M. Haller. The S/KEY one-time password system. In *Proceedings of the Symposium on Network and Distributed System Security*, pages 151–157, 1994.
10. R. Kanaley. Login error trouble keeping track of all your sign-ons? here's a place to keep your electronic keys, but you'd better remember the password. *San Jose Mercury News (Feb. 4, 2001)*.
11. D. P. Kormann and A. D. Rubin. Risks of the passport single signon protocol. *Computer Networks*, 33:51–58, 2000.
12. L. Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–771, 1981.
13. P. D. McDaniel. *Handbook of Information Security*, chapter Computer and Network Authentication. John Wiley and Sons Inc., September 2004.
14. D. L. McDonald, R. J. Atkinson, and C. Metz. One time passwords in everything (opie): Experience with building and using stronger authentication. In *Proceedings of the 5th USENIX UNIX Security Symposium*, 1995.
15. Microsoft Passport, http://www.passport.net/. Date of access: November 18, 2004.
16. R. Rivest. The md5 message-digest algorithm. *RFC 1321*, 1992.
17. A. D. Rubin. Independent one-time passwords. Proceedings of the 5th USENIX Security Symposium:167–175, 1995.
18. U.S. National Institute of Science and Technology. Secure hash standard. *Federal Information Processing Standard (FIPS) 180-1*, 1993.
19. T. Wu. The secure remote password protocol. In *Proceedings of the Internet Society Symposium on Network and Distributed System Security*, pages 97–111, March 1998.