A New Two-Server Approach for Authentication with Short Secrets (To appear in USENIX Security '03)

John Brainard, Ari Juels, Burt Kaliski, and Michael Szydlo
RSA Laboratories
Bedford, MA 01730, USA
E-mail: {jbrainard,ajuels,bkaliski,mszydlo}@rsasecurity.com

April 9, 2003

Abstract

Passwords and PINs continue to remain the most widespread forms of user authentication, despite growing awareness of their security limitations. This is because short secrets are convenient, particularly for an increasingly mobile user population. Many users are interested in employing a variety of computing devices with different forms of connectivity and different software platforms. Such users often find it convenient to authenticate by means of passwords and short secrets, to recover lost passwords by answering personal or "life" questions, and to make similar use of relatively weak secrets.

In typical authentication methods based on short secrets, the secrets (or related values) are stored in a central database. Often overlooked is the vulnerability of the secrets to theft *en bloc* in the event of server compromise. With this in mind, Ford and Kaliski and others have proposed various password "hardening" schemes involving multiple servers, with password privacy assured provided that some servers remain uncompromised.

In this paper, we describe a new, two-server secure roaming system that benefits from an especially lightweight new set of protocols. In contrast to previous ideas, ours can be implemented so as to require essentially *no intensive cryptographic computation* by clients. This and other design features render the system, in our view, the most practical proposal to date in this area. We describe in this paper the protocol and implementation challenges and the design choices underlying the system.

1 Introduction

In this paper, we consider a basic, pandemic security problem: How is it possible to provide secure services to users who can authenticate using only short secrets or weak passwords?

This problem is of growing importance as Internet-enabled computing devices become ever more prevalent and versatile. These devices now include among their ranks an abundant variety of mobile phones, personal digital assistants (PDAs), and game consoles, as well as laptop and desktop PCs. The availability of networks of computers to highly mobile user populations, as in corporate environments, means that a single user may regularly employ many different points of remote access. The roaming user may additionally employ any of a number of different devices, not all of which necessarily possess the same software or configuration.

While smartcards and similar key-storage devices offer a secured, harmonized approach to authentication for the roaming user, they lack an adequately developed supporting infrastructure in many computing environments. At present, for example, very few computing devices contain smartcard readers – particularly in the United States. Furthermore, many users find physical authentication tokens inconvenient. Another point militating against a critical reliance on hardware tokens is the common need to authenticate roaming users who have lost or forgotten their tokens, or whose tokens have malfunctioned. Today, this is usually achieved by asking users to provide answers to a set of "life" questions, i.e., questions regarding personal and private information. These observations stress that roaming users must be able to employ passwords or other short pieces of memorable information as a form of authentication. Indeed, short secrets like passwords and answers to life questions are the predominant form of authentication for most users today. They are the focus of our work here.

To ensure usability by a large user population, it is important that passwords be memorable. As a result, those used in practice are often highly vulnerable to brute-force guessing attacks [18]. Good credential-server designs must therefore permit secure authentication assuming a weak key (password) on the part of the user.

1.1 SPAKA protocols

A basic tool for mutual authentication via passwords, and one well developed in the literature, is *secure password-authenticated key agreement* (SPAKA). Most SPAKA protocols are descendants of Bellovin and Merrit's EKE protocol [3, 4], and are predicated on either Diffie-Hellman key agreement or key agreement using RSA. The client and server share a password, which is used to achieve mutual assurance that a cryptographically strong session key is established privately by the two parties. To address the problem of weak passwords, SPAKA protocols are constructed so as to leak no password information, even in the presence of an active attacker. When used as a means of authentication to obtain credentials from a trusted server, a SPAKA protocol is typically supplemented with a throttling or lockout mechanism to prevent on-line guessing attacks. Many roaming-credentials proposals involve use of a SPAKA protocol as a leverage point for obtaining credentials, or as a freestanding authentication protocol. A comprehensive, current bibliography of research papers on the topic of SPAKA protocols (of which there are dozens) is maintained by David Jablon, and may be found at [15].

The design of most SPAKA protocols overlooks a fundamental problem: The server itself represents a serious vulnerability. As SPAKA protocols require the verifying server to have cleartext access to user passwords (or to derivative material), compromise of the server leads potentially to exposure of the full database of passwords. While many SPAKA protocols store passwords in combination with salt or in some exponentiated form, an attacker who compromises the server still has the possibility of mounting off-line dictionary attacks. Additionally, these systems offer no resistance to server corruption. An attacker that gains control of the authenticating server can spoof successful login attempts.

To address this problem, Ford and Kaliski [11] introduced a system in which passwords are effectively protected through distribution of trust across multiple servers. Mackenzie, Shrimpton, and Jakobsson [21] extended this system, leading to more complex protocols, but with rigorous security reductions in a broadly inclusive attack model. Our work in this paper may be regarded as a complement, rather than a successor to the work of these authors. We propose a rather different technical approach, and also achieve some special benefits in our constructions, such as a substantially reduced computational load on the client. At the same time, we consider a different, and in our view more pragmatic security model than that of other distributed SPAKA protocols.

1.2 Previous work

The scheme of Ford and Kaliski reduces server vulnerability to password leakage by means of a mechanism called *password hardening*. In their system, a client parlays a weak password into a strong one through interaction with one or multiple hardening servers, each one of which blindly transforms the password using a server secret. Ford and Kaliski describe several ways of doing this. Roughly speaking, the client in their protocol obtains what may be regarded as a blind function evaluation σ_i of its password P from each hardening server S_i. (The function in question is based on a secret unique to each server and user account.) The client combines the set of shares $\{\sigma_i\}$ into a single secret σ, a strong key that the user may then use to decrypt credentials, authenticate herself, etc. Given an appropriate choice of blind function evaluation scheme, servers in this protocol may learn no information, in an information-theoretic sense, about the password P. An additional element of the protocol involves the user authenticating by means of σ (or a key derived from it) to each of the servers, thereby proving successful hardening. The harderened password σ is then employed to decrypt downloaded credentials or authenticate to other servers.

Mackenzie *et al.* extend the system of Ford and Kaliski to a threshold setting. In particular, they demonstrate a protocol such that a client communicating with any k out of n servers can establish session keys with each by means of password-based authentication; even if $k-1$ servers conspire, the password of the client remains private. Their system can be straightforwardly leveraged to achieve secure downloadable credentials. The Mackenzie *et al.* system, however, imposes considerable overhead of several types. First, servers must possess a shared global key and local keys as well (for a total of $4n+1$ public keys). The client, additionally, must store $n+1$ (certified) public keys. The client must perform several modular exponentiations per server for each session, while the computational load on the servers is high as well. Finally, the Mackenzie *et al.* protocol is somewhat complex, both conceptually and in terms of implementation. On the other hand, the protocol is the first such provided with a rigorous proof of security under the Decision Diffie-Hellman assumption [7] in the random oracle model [2].

Frykholm and Juels [13] adopt a rather different approach, in which encrypted user credentials are stored on a single server. In this system, *no* trust in the server is required to assure user privacy under appropriate cryptographic assumptions. Roughly stated, user credentials are encrypted under a collection of short passwords or keys. Typically, these are answers to life questions. While the Frykholm-Juels system provides error tolerance, allowing the user to answer some questions incorrectly, it is somewhat impractical for a general population of users, as it requires use of a large number of questions. Indeed, the authors recommend a suite of as many as fifteen such questions to achieve strong security. The work of Frykholm and Juels is an improvement on that of Ellison *et al.* [10], which was found to have a serious security vulnerability [5]. This approach may be thought of as an extension to that of protecting credentials with password-based encryption. The most common basis for this in practice is the PKCS #5 standard [1].

1.3 Our work: a new, lightweight system

It is our view that most SPAKA protocols are over-engineered for real-world security environments. In particular, we take the position that that mutual authentication is often not a requirement for roaming security protocols *per se*. Internet security is already heavily dependent upon a trust model involving existing forms of server-side authentication, particularly the well studied Secure Sockets Layer protocol (SSL) [12]. SSL is present in nearly all existing Web browsers. Provided that a browser verifies correct binding between URLs and server-side certificates, as most browsers do, the user achieves a high degree of assurance of the identity of the server with which she has initiated a given session. In other words, server authentication is certainly important, but need not be provided by the same secret as user authentication. Thus many SPAKA protocols may be viewed as replicating functionality already provided in an adequately strong form by SSL, rather than building on such functionality.

Moreover, it may be argued that SPAKA protocols carry a hidden assumption of trust in SSL or similar mechanisms to begin with. SPAKA protocols require the availability of special-purpose software on the client side. Given that a mobile user cannot be certain of the (correct) installation of such software on her device, and that out-of-band distribution of special-purpose software is rare, it is likely that a user will need to download the SPAKA software itself from a trusted source. This argues an *a priori* requirement for user trust in the identity of a security server via SSL or a related mechanism. In this paper, we assume that the client has a pre-existing mechanism for establishing private channels with server-side authentication, such as SSL.

Our system represents an alternative to SPAKAs in addressing "hardening" problem; it is a two-server solution that is especially simple and practical. The idea is roughly as follows. The client splits a user's password (or other short key) P into shares for the two servers. On presenting a password P' for authentication, the client provides the two servers with a new, random sharing of P'. The servers then compare the two sharings of P and P' in such a way that they learn whether $P = P'$, but no additional information. The client machine of the user need have no involvement in this comparison process.

As we explain, it is beneficial to configure our system such that users interact with only one server on the front-end, and pass messages to a second, back-end server via a protected tunnel. This permits the second server to reference accounts by way of pseudonyms, and thereby furnishes users with an extra level of privacy. Such privacy is particularly valuable in the case where the back-end server is externally administered, as by a security-services organization. Much of our protocol design centers on the management of pseudonyms and on protection against the attacks that naïve use of pseudonyms might give rise to.

1.4 Organization

In section 2, we describe the core cryptographic protocol our system for two-server comparison of secret-shared values. We provide an overview of our architecture in section 3, discussing the security motivations behind our choices. In section 4, we describe two specialized protocols in our system; these are aimed at preventing false-identifier and replay attacks. We provide some implementation details for our system in section 5. We conclude in section 6 with a brief discussion of some future directions.

2 An Equality-Testing Protocol

Let us first reiterate and expand on the intuition behind the core cryptographic algorithm in our system, which we refer to as *equality testing*. The basic idea is for the user to register her password P by providing random shares to the two servers. On presenting her password during login, she splits her password into shares in a different, random way. The two servers compare the two sharings using a protocol that determines whether the new sharing specifies the same password as the original sharing, without leaking any additional information (even if one server tries to cheat). For convenience, we label the two servers "Blue" and "Red". Where appropriate in subscripts, we use the lower-case labels "blue" and "red".

Registration: Let $\mathcal{H}$ be a large group (of, say, 160-bit order), and $+$ be the group operator. Let f be a collision-free hash function $f : \{0,1\}^* \rightarrow \mathcal{H}$. To share her password at registration, the user selects a random group element $R \in_U \mathcal{H}$. She computes the share P_{blue} for Blue as $P_{blue} = f(P) + R$, while the share P_{red} of Red is simply R. Observe that the share of either server individually provides no information about P.

Authentication: When the user furnishes password P' to authenticate herself, she computes a sharing based on a new random group element $R' \in_U \mathcal{H}$. In this sharing, the values $P'_{blue} = f(P') + R'$ and $P'_{red} = R'$ are sent to Blue and Red respectively.

The servers combine the shares provided during registration with those for authentication very simply as follows. Blue computes $Q_{blue} = P_{blue} - P'_{blue} = (f(P) - f(P')) + (R - R')$, while Red similarly computes $Q_{red} = P_{red} - P'_{red} = R - R'$. Observe that if $P = P'$, i.e., if the user has provided the correct password, then $f(P)$ and $f(P')$ cancel, so that $Q_{blue} = Q_{red}$. Otherwise, if the user provides $P \neq P'$, the result is that $Q_{blue} \neq Q_{red}$ (barring a collision in f). Thus, to test the user password submitted for authentication, the two servers need merely test whether $Q_{blue} = Q_{red}$, preferably without revealing any additional information.

For this task of equality testing, we require a second, large group $\mathcal{G}$ of order q, for which we let multiplication denote the group operation. The group $\mathcal{G}$ should be one over which the discrete logarithm problem is hard. We assume that the two servers have agreed upon this group in advance, and also have agreed upon (and verified) a generator g for $\mathcal{G}$. We also require a collision-free mapping $w : \mathcal{H} \rightarrow \mathcal{G}$. For equality testing of the values Q_{red} and Q_{blue}, the idea is for the two servers to perform a variant of Diffie-Hellman key exchange. In this variant, however, the values Q_{red} and Q_{blue} are "masked" by the Diffie-Hellman keys. The resulting protocol is inspired by and may be thought of as a technical simplification of the PET protocol in [16]. Our protocol uses only one component of an El Gamal ciphertext [14], instead of the usual pair of components as in PET. Our protocol also shares similarities with SPAKA protocols such as EKE. Indeed, one may think of the equality $Q_{red} = Q_{blue}$ as resulting in a shared secret key, and inequality as yielding different keys for the two servers.

There are two basic differences, however, between the goal of a SPAKA protocol and the equality-testing protocol in our system. A SPAKA protocol, as already noted, is designed for security over a potentially unauthenticated channel. In contrast, our intention is to operate over a private, mutually authenticated channel between the two servers. Moreover, we do not seek to derive a shared key from the protocol execution, but merely to test equality of two secret values with a minimum of information leakage. Our desired task of equality testing in our system is known to cryptographers as the *socialist millionaires' problem*. (The name derives from the idea that two millionaires wish to know whether they enjoy equal financial standing, but do not wish to reveal additional information to one another.) Several approaches to the socialist millionaires' problem are described in the literature. Often, researchers are also concerned in addressing the problem to ensure the property of *fairness*, namely that both parties should learn the answer or neither. We do not treat this issue here, as it does not have a major impact on the overall system design. (A protocol unfairly terminated by one server in our system is no worse than a password guess initiated by an adversary, and will be immediately detected by the other server.)

Note that in this protocol, the client need perform no cryptographic computation, but just a single (addition) operation in $\mathcal{H}$. (The client performs some cryptographic computation to establish secure connections with Blue and Red in our system, but this may occur via low-exponent RSA encryption – as in SSL – and thus involves just a small number of modular multiplications.) Moreover, once the client has submitted a sharing, it need have no further involvement in the authentication process. Red and Blue together decide on the correctness of the password submitted for authentication. Given a successful authentication, they can then perform any of a range of functions providing privileges for the user: Each server can send a share of a key for decrypting the user's downloadable credentials, or two servers can jointly issue a signed assertion that the user has authenticated, etc.

2.1 Protocol details

As we have already described the simple sharing protocols employed by the client in our system for registration and authentication, we present in detail only the protocol used by the servers to test the equality $Q_{red} = Q_{blue}$. We assume a private, mutually authenticated channel between the two servers. Should the initiating server (Blue) try to establish multiple, concurrent authentication sessions for a given user account, the other server (Red) will refuse. (In particular, in Figure 1, Red will reject the initiation of a session in which the first flow specifies the same user account as for a previously established, active authentication session.) Alternative approaches permitting concurrent login requests for a single account are possible, but more complicated. If Blue initiates an authentication request with Red for a user U for which Red has received no corresponding authentication request from the user, then Red, after some appropriate delay, will reject the authentication.

Let $Q_{blue,U}$ denote the current share combination that Blue wishes to test for user U, and $Q_{red,U}$ the analogous Red-server share combination for user U. In this and any subsequently described protocols in this paper, if a server fails to validate any mathematical relation denoted by $\overset{?}{=}$, $\overset{?}{\neq}$, $\overset{?}{>}$, or $\overset{?}{\in}$, it determines that a protocol failure has taken place; in this case, the authentication session is terminated and the corresponding authentication request rejected.

We let $\in_R$ denote uniform random selection from a set. We indicate by square brackets those computations that Red may perform prior to protocol

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | sales@docketalarm.com | 1-866-77-FASTCASE

fastcase® Smarter legal research.