

Internet Engineering Task Force
INTERNET DRAFT
[26](#) June 2000
Expires in six months

Radia Perlman
Sun Microsystems
Charlie Kaufman
Iris Associates
Eric Rescorla
RTFM, Inc.

Strong Password-Based Credentials Download Using Pseudorandom Moduli
[draft-perlman-strong-cred-00.txt](#)

Status of This Memo

This document is an individual contribution to the Internet Engineering Task Force (IETF). Comments should be sent to the authors Radia.Perlman@Sun.COM and CKaufman@Iris.COM.

Distribution of this memo is unlimited.

This document is an Internet-Draft and is in full conformance with all provisions of [Section 10 of RFC2026](#). Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at
<http://www.ietf.org/ietf/lid-abstracts.txt>

The list of Internet-Draft Shadow Directories can be accessed at
<http://www.ietf.org/shadow.html>.

Abstract

In a strong password-based protocol, the only thing the client needs to know is the user's password, and an eavesdropper, or someone impersonating either end, cannot do off-line password-guessing attacks. This sort of protocol can be used for credentials download, or for mutual authentication. Although password-based mutual authentication protocols can be used for credentials download, they are designed with some properties (such as not storing a password-equivalent at the server) that are not important in a credentials-download protocol. Therefore, a protocol designed specifically for credentials download can be fewer messages, higher performance at the server, and allow the server to operate in a stateless manner. Therefore it is possible, even likely, that different protocols might

be preferable in one case (credentials download) than another. This

document specifies a password-based protocol optimized for downloading of a user's credentials, such as a private key. We claim the desirable properties of such a protocol are that it be unencumbered, high performance at the server, stateless at the server, minimal number of messages, and acceptable performance at the client.

Table of Contents

Status of This Memo	1
Abstract	1
Table of Contents	1
1 Introduction	1
2 The Protocol	3
3 Generating p	4
4 Getting down to the bits	6
4.1 Computing p from username and password	6
4.2 Format of first message on the wire	8
4.3 Format of second message on the wire	8
5 . Performance Note	8
6 . Security Considerations	8
References	9
Author's Addresses	10
Full Copyright Statement	10

[1](#). Introduction

Protocols such as EKE [BM92], and SPEKE [Jab96] pioneered the notion of using a weak secret such as a password to obfuscate a Diffie-Hellman exchange, so that the exchange itself gives no information to eavesdroppers, and at most a single on-line password guess to someone impersonating either the client or the server. Later the protocols were adapted [BM94], [Jab97] to incorporate the feature of not storing a password-equivalent at the server, so that someone that

R. Perlman, et. al.

Expires: 17 May 2001

[Page 2]Intern

stole the server's database could not use the contents to directly impersonate the user (though they could do a dictionary attack on the contents to discover the user's password). SRP [Wu98] improved on those protocols in performance.

However, for credentials download, there is no advantage of avoiding a password-equivalent at the server because the purpose of the protocol is to obtain the user's credentials, which are stored at the server. So someone that steals the server's database already has the credentials. Also in [PK99] tricks were provided for making (nonextended) protocols such as EKE and SPEKE (but not the extended versions) stateless at the server. A lot of the work of the mutual authentication protocols such as extended EKE, extended SPEKE, and SRP, are not relevant for credentials download, and they require computation and state at the server that are not necessary for a protocol used solely for credentials download. Therefore, the preferred protocol for mutual authentication might be different from the preferred protocol for credentials download.

Note that the credentials must be encrypted with a different hash of the password than might be stored for use in EKE or SPEKE; otherwise the user's credentials can be directly derived from the information in the database.

Although a protocol based on (nonextended) EKE or SPEKE for credentials download is technically acceptable, patent issues might argue for a new approach, which we provide in this paper.

We claim the desirable properties of a credentials download protocol are that it be unencumbered, secure against dictionary attacks by an eavesdropper or someone impersonating either end, high performance at the server, stateless at the server, minimal number of messages, and acceptable performance at the client.

We claim the performance at the server is much more important than performance at the client, because the server must simultaneously serve many clients, perhaps many of whom are unauthorized and contacting the server solely to require it to consume resources. We

claim the performance at the client is not as important because it is serving only one user at a time, and the protocol is only executed once (per user, per session). So the only thing that matters at the client is whether performance is "good enough", whereas at the server, any performance gain is important.

Our protocol has at least as good performance at the server as any of the other schemes (and quite likely better, because the argument can be made that our scheme allows for use of smaller p 's).

In EKE g and p in the Diffie-Hellman exchange are fixed, and the public number $g^{**X} \bmod p$ is encrypted with a function of the user's password. In SPEKE, the base g is a function of the user's password.

[R. Perlman, et. al.](#)

Expires: 17 May 2001

[Page 3]Intern

In this paper we propose making the modulus p a function of the user's password.

This protocol can be used to download a user's private key to a workstation with appropriate software but no security information configured for that user, from a public place such as a directory. This can be thought of as bootstrapping the PKI. It is not possible to protect the user's private key with conventional access control since the user cannot authenticate until the user obtains the private key. And the private key cannot be world-readable, even if encrypted with the user's password, because the encrypted private key is subject to off-line password-guessing. Once the user's private key is recovered, any other information necessary to reconstruct the user's environment on that workstation, such as the CAs that user trusts as trust anchors for certificate chains, can also be retrieved from the directory. Information that needs to be integrity protected is signed with the user's private key. Information that needs to be secret (other than the user's private key, which must be protected in a different way) is encrypted with the user's public key.

The properties of this protocol are:

- . a user need only know her name and password
- . the protocol allows the server to operate in a stateless manner
- . an eavesdropper on the authentication exchange between Alice and Bob will gain no information from the exchange, no matter how guessable Alice's password.
- . Someone impersonating Bob or Alice will be able to verify only a

single password guess per interaction with the other side.

This protocol is not intended for devices such as PDAs which are owned by a specific user and carried by that user. For those devices, a high quality secret (such as the user's private key) would be configured. This protocol is intended for workstations with reasonable computational resources (say 400 MHz processors) that have reasonable software, but no user-specific information configured.

2. The Protocol

The basic idea is to use the user's password as a seed for generating a prime modulus, and always use 2 as the base. In order to provide for salt, the seed for the pseudorandom number generator which will select the prime modulus should be $h(\text{"Alice"}, \text{Alice's password})$. The server Bob will store p for user Alice. Alice's workstation must generate p based on Alice's name and her password when she logs in.

R. Perlman, et. al.

Expires: 17 May 2001

[Page 4]Intern

The protocol requires only 2 messages (request/response), so is obviously stateless for the server. The client calculates p from the user's password, chooses a random A , and sends $2^{**}A \bmod p$ to the server. The server chooses B , sends $2^{**}B \bmod p$ to the client, and sends back the credentials encrypted with $2^{**}AB$.

To save the server work, it loses no security to always use the same B for the same user (but not for different users). If the server stores, for Alice, B and $2^{**}B \bmod p$, then in this protocol there is only a single exponentiation required at the server (to raise $2^{**}A$ to the power B).

So Bob stores, for Alice:

- . name "Alice"
- . p (Sophie Germain prime pseudorandomly generated from seed $h(\text{"Alice"}, \text{password})$)
- . $Y = \text{Alice's credentials encrypted with her password}$
- . B
- . $2^{**}B \bmod p$

The protocol is as follows:

- 1) User types name ("Alice") and password to the workstation.
- 2) Workstation computes $h(\text{"Alice"}, \text{password})$, and uses that to seed a pseudorandom number generator to choose a Sophie Germain prime p that has 2 as a generator.
- 3) Workstation chooses a random A to serve in the Diffie-Hellman exchange.
- 4) Workstation sends "Alice", $2^A \bmod p$ to Bob.
- 5) Bob checks that Alice's name is exactly as stored in the database, and if not, sends that version of Alice's name back. If it is the same, Bob takes the B stored in the database, calculates $K = 2^{AB} \bmod p$ by raising what he received from Alice to the power B , and transmits $2^B \bmod p$, and Y encrypted with K to Alice.
- 6) Workstation calculates $2^{AB} \bmod p$ and uses it to decrypt Y .

[3. Generating \$p\$](#)

We want 2 to be a generator (with high probability). By the law of quadratic reciprocity, this calls for p being 3 mod 8. So our algorithm for generating p 's will only test numbers that are 3 mod 8.

We estimate 10 seconds as a reasonable maximum amount of time for the workstation to compute p . Diffie-Hellman moduli of 1000 bits are considered safe. But on today's desktops, it would take too much time (we estimate minutes) to generate a 1000-bit Sophie Germain prime. (A "Sophie Germain prime" p is one in which $(p-1)/2$ is also prime.)

What corners can be cut to make performance tolerable at the client machine? Suppose we used a smaller p , say 512 bits. It is estimated that it would take 8000 MIP-years to break 512-bit Diffie-Hellman. An eavesdropper on this exchange would have information sufficient to test passwords, but it would require breaking 512-bit Diffie-Hellman for every guessed password. Another threat from using breakable Diffie-Hellman is that if Bob's database were stolen, or someone learned the user's password, then we would lose perfect forward

secrecy for the attacker's price of breaking 512-bit Diffie-Hellman. If this were really considered a problem, then this entire exchange could take place inside an anonymous Diffie-Hellman exchange with fixed, large p . But for credentials download, (unlike mutual authentication) perfect forward secrecy is not a concern.

Another way to make performance better for the client is not to use a Sophie Germain prime, but instead use one where $p-1$ has at least one large (160 bit) prime factor. Selection becomes more complicated because determining whether 2 is a generator is probabilistic if the factorization of $p-1$ is not known. This is probably a reasonable approach; however, using a trick suggested by Jeff Schiller of having the user optionally supply a hint, we think performance with Sophie Germain primes can be tolerable, and the size of the Sophie Germain prime generated can be increased as desktops get faster, and it is a simpler solution. It is always best to start with a simple solution.

The way to look for a Sophie Germain prime is to start with a number n , and then look for the smallest number p of the form $3 \bmod 8$ larger than n where both p and $(p-1)/2$ are (probably) prime. The most efficient method is to first sieve to avoid numbers where either p or $(p-1)/2$ have a small (say less than 10000) prime factor, and then discard numbers unless both $2^{(p-1)/2} \bmod p = p-1$, and $2^{(p-3)/2} \bmod (p-1)/2 = 1$. If p passes this test, there is an overwhelming probability that p is a Sophie Germain prime and that 2 is a generator. As a performance enhancement, we propose that for purposes of interoperability that exactly those tests take place as opposed to some less efficient test that determines primality with higher certainty but which could introduce interoperability problems if not all implementations used the same tests.

The user-supplied hint trick involves having the workstation give the user a small value (say a single character, giving 6 bits of information) that will speed up the search. For instance, it can specify 6 of the bits of the p that will be chosen from the password. If the user remembers the hint, then it saves computation time. If the user does not furnish the hint, then authentication takes longer. If the user furnishes the wrong hint, it is possible for authentication to fail.

It is also important, in protocols of this sort, to avoid leaking information that would enable an eavesdropper to eliminate passwords. For instance, in the most straightforward implementation of EKE, one might encrypt $g^A \bmod p$ with a hash of the password. An eavesdropper that observed an encrypted $g^A \bmod p$ could do trial decryptions with

various passwords, and eliminate any passwords in which the result was larger than p .

To avoid leaking information, we specify that candidate p 's be chosen from a very narrow range. So we fix the top 64 bits to be a constant. For maximum performance given the size of the numbers, p should be close to a power of 2. Therefore we recommend fixing the top 64 bits to be 111...1. That means that the smallest possible p chosen would be 64 bits of 1 followed by 0's, and the larger would be all 1's. We recommend discarding A's and B's for which $2^{**}A \bmod p$ or $2^{**}B \bmod p$ wind up larger than the maximum possible p that could be generated. The probability of needing to discard an A is $1/2^{**}64$.

Another way in which information might be leaked is observing the amount of time required to generate p . But an eavesdropper will have no way of knowing how long it took to compute p , since the only thing the eavesdropper sees is the client's request (which occurs after computation of p).

Another way information could be leaked is if Alice chooses an A so small that $2^{**}A$ does not need to be reduced by p . In that case, she has not committed to a password, and can do a dictionary attack based on $\{Y\}2^{**}AB \bmod p$ (since she knows A), for any p . Using 2 as the base has the nice property that Bob can recognize this form of cheating, since $2^{**}A \bmod p$ will have only a single "1" in the binary representation. Bob must reject such values from Alice.

To support smooth migration to variations on this protocol in the future, to allow some negotiation of parameter sizes, we specify a minor version number and an alternate form of the second message in which the server can specify alternate parameter sizes and/or a "cookie".

[4.](#) Getting down to the bits

[4.1](#) Computing p from username and password

If the password is at least three characters long, the next to last character is ".", and the final character is from the set (0-9, a-z, A-Z, +, =), then the final two characters of the password are left

out of the calculation of P below and the final character is translated into an integer HINT between 0 and 63 depending on its position in that set of legal characters. This will be used as a performance enhancement (see below).

U = SHA1(Username)

P = SHA1>Password)

V = SHA1 ("Strong Password Authentication - Version 1.1 dated 16NOV2000")

Pseed = SHA1(U || P || V)

A = Randomly chosen 160 bit integer

Starting point for search for prime p is a 512 bit number with high order bits 0-63 all 1's, and the remaining bits from SHA1(Pseed || "1"), SHA1(Pseed || "2"), ... The chosen p will be the smallest number greater than or equal to the starting point for which $p \equiv 3 \pmod{8}$, neither p nor $(p-1)/2$ has prime factors less than 10,000, and for which $2^{((p-1)/2)} \pmod{p} = -1$ and $2^{((p-3)/2)} \pmod{((p-1)/2)} = 1$. The HINT is 6 bits of the chosen p. Since the bottom 3 bits must be 011 (to be $3 \pmod{8}$), the hint can be the representation of bits 503-508. (i.e., the low order bit of the penultimate byte and the high order 5 bits of the last byte).

If the user does not supply a hint, the user is told that future logins will run faster if the hint "x" is supplied in the future (where x is replaced with the character corresponding to the discovered HINT). The hint could be supplied with a separate user prompt, or by appending to the password a character (such as space) that would be otherwise be disallowed in the password, followed by x.

If there is a HINT, only candidate p's with bits 4-10 equal to the hint are considered. The effect of this is that if the user provides the HINT when entering the password, the computation of p will be much faster but ultimately will produce the same result as the calculation without the HINT. If the user specifies an incorrect HINT, a different p will be calculated and it will probably take a long time. The end result is that no passwords are disallowed (any value entered will result in a 512 bit p), but users willing to remember one more character of password will get in a lot faster.

Note that this is not equivalent to simply having a longer password, since the hint is not user-supplied, and has no logical (from a human

point of view) connection with the rest of the password.

It would also be reasonable to allow multiple characters for the hint, further speeding up processing at the client. In this case the first character supplied by the user would equal bits 4-9, the next 10-15, etc.

Above, we specified that the user interface for entering the hint would be that ".x" would be appended to the password. If "." is a legal character in a password, and the user happened to specify a password with "." as the penultimate character, then the final character would be interpreted as a hint, and a prime would be found, but it would be slow.

4.2 Format of first message on the wire: "Alice", $2^{*A} \bmod p$

The encapsulating protocol is assumed to provide the total length of the message. Message is $V \parallel '\0' \parallel 2^{*A} \bmod p \parallel \text{Username}$, where V has fixed length 160 bits, a single zero byte follows, $2^{*A} \bmod p$ has fixed length 512 bits, and username is variable length. $2^{*A} \bmod p$ is sent in "big endian" order. The recipient of this message MUST verify that V is the expected value or terminate the protocol. The zero byte is a "minor version" number. It MUST be zero for clients implementing this version of the protocol. It MAY be non-zero for future clients, indicating that those clients are willing to negotiate some variation on this protocol. The recipient of this message implementing this version of the protocol MUST ignore the value of that byte. Future versions of the protocol may specify optional alternate responses if the value is non-zero.

4.3 Format of second message on the wire: "Bob", $2^{*B} \bmod p$, and (Y) encrypted with 2^{*AB}

Bob parses the first message. He uses the user name to look up account information for Alice, and checks to make sure the case, accents, and punctuation of Alice's name as provided matches the version in his account database. If not, or if the version is not acceptable, or if $2^{*A} \bmod p$ has only a single "1" bit in its binary representation, or if Bob wants a stateless cookie before he does any exponentiation, then Bob replies, with the cookie, corrected form of Alice's name, required size of p (if the client sent the wrong size), complaint about the format of $2^{*A} \bmod p$, etc. If the version, Alice's name, and the prime size is acceptable, Bob then reads p , B , $2^{*B} \bmod p$, and Y from the database, raises the received $2^{*A} \bmod p$ to the power B , encrypts Y with $2^{*AB} \bmod p$ to get ENCY , and sends $V \parallel 2^{*B} \bmod p \parallel \text{ENCY}$.

5. Performance Note

By far the most time consuming part of this protocol is the client's generation of p from the username and password. The performance of this operation is especially important because it must be performed every time the client wishes to download his credentials.

Table 1 contains performance numbers for this operation. These numbers are the mean of 20 trials with different passwords. The passwords were chosen using words 20001-20020 in FreeBSD 3.4's /usr/share/dict/words. The username was "Alice". All numbers are in seconds. For reference, we have included numbers at a variety of key sizes.

Key Size	Normal	w/ Hint	
384	6.6	.13	(not recommended)
512	8.7	.15	
768	34	.57	
1024 *	120	2.0	

Figure 1: Performance of prime generation

* Only 5 trials were run for 1024 bits due to needing to submit this I-D.

These numbers were collected on a fairly unoptimized implementation of the algorithm of [Section 4.1](#) on a Pentium II/400 running FreeBSD 3.4. The bignum implementation was OpenSSL 0.9.6.

Note that although performance with a 1024 bit prime is quite bad, with a hint it's quite acceptable.

6. Security Considerations

This protocol is intended to be a method of downloading credentials which is somewhat less secure than having smart cards for each user. It can bootstrap deployment of a PKI, allowing deployment before users have smart cards. It is perfectly feasible for some users to obtain credentials through this protocol and others to have smart cards, and a user that subsequently obtains a smart card can simply have their credentials information removed from the directory. This protocol assumes trusted software at the client machine.

References

- [BM92] S. Bellovin and M. Merritt, "Encrypted Key Exchange: Password-based protocols secure against dictionary attacks", Proceedings of the IEEE Symposium on Research in Security and Privacy, May 1992.
- [BM94] S. Bellovin and M. Merritt, "Augmented Encrypted Key Exchange: a Password-Based Protocol Secure Against Dictionary Attacks and Password File Compromise, ATT Labs Technical Report, 1994.
- [DH76] W. Diffie and M. Hellman, "New Directions in Cryptography", IEEE Transactions on Information Theory, November 1976.
- [Jab96] D. Jablon, "Strong Password-Only Authenticated Key Exchange", Computer Communication Review, ACM SIGCOMM, vol. 26, no. 5, pp. 5-26, October 1996.
- [JAB97] D. Jablon, "'Extended Password Key Exchange Protocols Immune to Dictionary Attacks", Proceedings of the Sixth Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET-ICE '97), IEEE Computer Society, June 18-20, 1997, Cambridge, MA, pp. 248-255.
- [KPS95] C. Kaufman, R. Perlman, and M. Speciner, "Network Security: Private Communication in a Public World", Prentice Hall, 1995.
- [Pat97] S. Patel, "Number Theoretic Attacks On Secure Password Schemes", Proceedings of the IEEE Symposium on Security and Privacy, May 1997.
- [PK99] R. Perlman and C. Kaufman, "Secure Password-Based Protocol for Downloading a Private Key", ISOC NDSS Symposium, 1999.
- [WU98] T. WU, "The Secure Remote Password Protocol", ISOC NDSS Symposium, 1998.

Author's Address

Radia Perlman
Sun Microsystems

Phone:
Email: radia.perlman@sun.com

Charlie Kaufman
Iris Associates

Phone:
Email: ckaufman@iris.com

Eric Rescorla
RTFM, Inc.

Phone:
Email: ekr@rtfm.com

Full Copyright Statement

Copyright (C) The Internet Society (1999). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society

or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE."