The Secure Remote Password Protocol*

Thomas Wu
Computer Science Department
Stanford University
tjw@cs.Stanford.EDU

November 11, 1997

Abstract

This paper presents a new password authentication and key-exchange protocol suitable for authenticating users and exchanging keys over an untrusted network. The new protocol resists dictionary attacks mounted by either passive or active network intruders, allowing, in principle, even weak passphrases to be used safely. It also offers perfect *forward secrecy*, which protects past sessions and passwords against future compromises. Finally, user passwords are stored in a form that is not *plaintext-equivalent* to the password itself, so an attacker who captures the password database cannot use it directly to compromise security and gain immediate access to the host. This new protocol combines techniques of zero-knowledge proofs with asymmetric key exchange protocols and offers significantly improved performance over comparably strong extended methods that resist stolen-verifier attacks such as Augmented EKE or B-SPEKE.

1 Introduction

Password authentication protocols come in many flavors, but they all solve the same problem: One party must somehow prove to another party that it knows some password P, usually set in advance. Such protocols range from the trivial to the incredibly complex, and many of them offer some form of protection from various attacks mounted by malicious or excessively curious third parties.

All methods of human authentication fall into three broad categories:

- Something the user *is* (voiceprint identification, retinal scanners)
- Something the user *has* (ID cards, smartcards)
- Something the user *knows* (passwords, PINs)

This paper deals with a particularly important subset of the last category known as *direct* password authentication. Mechanisms that fall into this rather exclusive category cannot rely on persistent stored information on the client side. The user's password, which is a memorized quantity, is the only secret available to client software. It is also assumed that the network between the client and server is vulnerable to both eavesdropping and deliberate tampering by the enemy. In addition, no trusted third party such as a key server or arbitrator can be used; only the original two parties can engage in the authentication protocol.

Such protocols have a surprisingly wide range of practical applications because they do not require anything more than a memorized password, making them much easier to use and less expensive to deploy than either biometric or token-based methods. One obvious application is handling remote, password-protected

*1998 Internet Society Symposium on Network and Distributed System Security

computer access. Most Internet protocols currently in use employ plaintext passwords for authentication, and it has been recommended that they be replaced with more secure alternatives if it can be done transparently [8]; a secure direct authentication protocol fits perfectly into such an architecture without introducing significant user-visible overhead. Even in situations where some form of security infrastructure already exists, a strong password system adds a strong independent factor to the authentication mechanism that adds to the overall strength of the system. This includes *multi-factor* systems that employ a password plus either a hand-carried or biometric device. Reference [9] contains an excellent treatment of these issues, and [1] also lists additional applications for secure direct authentication protocols.

Section 2 briefly reviews existing authentication protocols and explains both their strengths and their shortcomings.

Section 3 discusses the new authentication protocol in mathematical terms, suggests possible implementations, and explains the rationale behind its design.

Section 4 analyzes the security of the new protocol, proves its security against eavesdroppers by reducing it to existing hard mathematical problems, and discusses necessary conditions and restrictions.

Section 5 addresses performance and implementation issues.

2 Terminology and background

Throughout this paper, the terms *client* and *server* will be used to denote the *user* and *host* parties in a direct authentication protocol. Unless stated otherwise, the client is assumed to be a human user who, like typical computer users, can only remember relatively short passwords [7, 13]. Although a user may employ a piece of software to negotiate the authentication protocol in practice, this does not affect our definition of the client, since we have already stipulated that said client software cannot remember long-term keys on behalf of the user.

The terms *password* and *verifier* correspond to conventional private and public keys, differing in only two aspects: Unlike typical private keys, the password has limited entropy, constrained by the memory of the user. A verifier has similar mathematical properties to a public key, since it is easily computed from the password, yet deriving the password from the verifier is computationally infeasible. Instead of being a publicly-known quantity, however, the verifier is kept secret by the server. An authentication mechanism that requires the server to store a copy of the user's password or private key is known as a *plaintext-equivalent* mechanism, while one that only requires a verifier to be stored will be called a *verifier-based* mechanism.

Verifier-based protocols have a significant advantage over ones that are plaintext-equivalent. A system that uses plaintext-equivalent authentication becomes instantly compromised once the password database is revealed, since every user's password is stored there. A database of verifiers, on the other hand, can be protected just as easily and effectively as a database of plaintext-equivalent passwords, except that failure of said protection is not as catastrophic if only verifiers are compromised.

While any reasonably secure authentication protocol is expected not to leak any information about the password to eavesdroppers, protocols classified as *zero-knowledge* do not even leak any information about the password to the legitimate host (except the fact that the party at the other end really does know it). This subset of verifier-based protocols is strong indeed, since the host never stores plaintext-equivalent information and is never given any such information during the course of authentication. This reduces the damage that Trojan horses[1] can inflict, and it enables the authentication system to retain some degree of security even in the case of complete host compromise.

2.1 Older authentication techniques

In the simplest of all password authentication protocols, Carol (the user or client) sends Steve (the host or server) her username and her plaintext password, and Steve verifies the password, either by comparing it

[1] One huge benefit here is that passwords shared between different systems are not compromised if an attacker installs a Trojan horse on one of the systems. The Secure Remote Password protocol is one of the first authentication mechanisms that solves this problem.

directly to his version of Carol's password or applying a one-way hash function first and checking against a database of stored hashes. Since Carol's password is immediately exposed to any eavesdropping attack, this method is unacceptable in networks where such attacks are possible.

To counter this, Carol and Steve can employ a *challenge-response* protocol. In general terms, such a protocol would take the following form:

1. Carol sends her identity to Steve, along with some random message.
2. Steve sends Carol a random message, called a *challenge*.
3. Carol performs some computation based on the challenge, the first random message, and her password. She sends this *response* to Steve, who performs the same computation and verifies the correctness of Carol's response.

Since Steve's challenge is different for each authentication attempt, a captured response is useless for future sessions, defeating a simple replay attack. However, challenge-response protocols can be attacked in other ways. Eve can capture the random number, challenge, and response from a successful authentication attempt and start guessing passwords until she finds one that generates a response that matches the captured one. This attack is called a *dictionary attack* and has been used to exploit systems in the past, often quite successfully [7].

Challenge-response protocols are also plaintext-equivalent, so they can be easily defeated by an intruder who captures the password file, as well as one who can eavesdrop.

To work around the limitations of inherently weak authentication mechanisms, protocol designers have traditionally used one of three approaches:

- Increase the length of the key with an external device like a smartcard.
- Take advantages of physical phenomena to construct a channel that is more difficult to compromise (spread spectrum, quantum cryptography).
- Ignore the problem and hope "nobody notices."

The first method changes the authentication system so that it is no longer based on "something you know," losing the convenience benefits of password-only methods in the process. The second method is only applicable to a limited range of applications. The third method is the most common, and exposes a very dangerous attitude among protocol designers. This combination of weak authentication technology and lax attitudes towards password security have given password authentication a negative reputation in the security community [13].

2.2 Stronger solutions

In 1992, Bellovin and Merritt [1] presented a new protocol known as *Encrypted Key Exchange*, or EKE. By using a combination of symmetric and public-key cryptography, EKE resists dictionary attacks by giving a passive attacker insufficient information to verify a guessed password. EKE performs a key exchange as well, so both parties can encrypt their transmissions once authentication is established. In the most general form of EKE, the two communicating parties encrypt ephemeral public keys with a symmetric cipher, using their shared secret password as a key. Since it was invented, EKE has been developed into a family of protocols, many of which are stronger than the original or add new desirable properties. For example, DH-EKE [17] and SPEKE [9] add what is known as *forward secrecy*, which means that revealing the password to an attacker does not help him obtain the session keys of past sessions. It is also usually taken to mean that a stolen session key does not help an attacker carry out a brute-force guessing attack on the password. Reference [6] describes a set of "secret public-key" protocols that accomplish the same objectives as EKE.

To date, the family of EKE protocols represents the strongest level of password-based authentication protocols available. EKE's greatest failing is that it still suffers from plaintext-equivalence, requiring that both the client and host have access to the same secret password or hash thereof. There is one variant of EKE, known as *Augmented EKE* or A-EKE [2], which makes EKE a verifier-based protocol, but the modification also destroys forward secrecy [17].

Recently, additional work has been done to extend the EKE family of protocols to address the issue of holding plaintext-equivalent data in password files [10]. B-SPEKE is an example of such an extended method. These protocols add another key exchange round to verify the client's possession of the actual password as opposed to a stolen verifier from the password file. This fixes a major issue with EKE, at the expense of substantially increasing the running time and computational complexity of the resulting protocol.

The issue of avoiding plaintext-equivalence has been a glaring omission in secure protocol designs for quite some time, yet it must be addressed if it is to be considered a viable replacement for authentication systems like the `/etc/passwd` file in Unix systems [13]. In addition, poor performance has often been an obstacle to the adoption of stronger protocols; the protocols described in [2] and [10] are just slow enough to be uncomfortable for frequent, lightweight authentication purposes. An improvement in performance from, say, a 3 second delay to a 1.5 second delay at login time can often make the difference between an unbearable solution and a workable one.

3 A new framework

Designing a verifier-based protocol is considerably more difficult than designing a conventional shared-secret authentication protocol, because the verifier and password are by definition not equivalent (though the former may be derived from the latter), forcing the computational structure of the protocol to be inherently asymmetric. As is the case with public-key cryptography, only a handful of methods lend themselves to the mathematical manipulation necessary to construct secure verifier-based protocols. This is one of the reasons why such protocols are relatively rare in practice.

We have already seen protocols that use digital signatures (A-EKE) and protocols that use a secondary one-sided key exchange (B-SPEKE); this section introduces a new construction called *Asymmetric Key Exchange*, or AKE for short, which is a generalized form for a third class of verifier-based protocols. Later, we will introduce the Secure Remote Password protocol itself, which will refer to the more well-defined and specified instance of AKE that is of interest to modern password authentication systems.

3.1 Asymmetric key exchange

Like EKE, the primary function of AKE is to exchange keys between two parties, the client and server, and to use this key to verify that both parties actually know their passwords. Unlike EKE, AKE does not encrypt any of the protocol flows. Instead, it uses predefined mathematical relationships to combine exchanged ephemeral values with established password parameters. Avoiding encryption is advantageous for a number of reasons:

- It simplifies the protocol by eliminating the need to negotiate a common encryption algorithm. The alternative, specifying the algorithm with the protocol, makes the protocol dependent on one particular encryption algorithm.
- Any weakness in the encryption will usually result in a weakness in the resulting authentication protocol. In addition, when passwords are used as key material, issues of padding and verifiable plaintext can open the protocol to a variety of attacks [6]. Not using encryption in the protocol itself removes this potential hole.
- In some jurisdictions, software and hardware implementations of encryption algorithms are subject to legal restrictions or export regulations. A protocol that does not use encryption is not affected by such concerns.

AKE also differs from its predecessors in another way. Protocols like EKE use prearranged shared secrets as the basis for authentication. This means that both parties keep exactly the same secret string and use it indirectly to authenticate each other. Since possession of the secret is enough to impersonate either party, and since there are now two places from which the secret can potentially be stolen, both parties are responsible for exchanging the initial secret securely and guarding the secret carefully.

AKE, however, describes a "swapped-secret" approach, in which each party computes a secret and then applies a one-way function to that secret to generate a verifier, which is handed to the other party. Although it is still important to guard the verifier to prevent a dictionary attack, a stolen verifier is no longer enough to impersonate the user; the corresponding secret password is still needed.

A special case of this technique, in which only one party generates a secret and computes a verifier, appears to be quite useful if the other party is a multiuser system that stores many verifiers. In such an application, the user's secret (i.e. the password) never has to leave the local host during the initial password setup and password change procedures; only the verifier needs to be sent, greatly improving the overall security of the system.

w, x, y, z	Arbitrary parameters
$P(x)$	A "one-way" verifier-generating function
$Q(x,y)$, $R(x,y)$	"Mixing" functions for private and public parameters
$S(x,y)$	The session key generation function
K	Session key

Table 1: Mathematical Notation for AKE

Table 1 summarizes the notation used in this section. We make no assumptions at this point about the domain, range, or input/output types of the functions save for the following:

$$(\forall w,x,y,z)\; S(R(P(w),P(x)),Q(y,z)) = S(R(P(y),P(z)),Q(w,x)) \tag{1}$$

Equation 1 must be satisfied for AKE to work properly. By itself, it guarantees nothing about the security of the resulting protocol. That is entirely dependent on the choices of the functions $P()$, $Q()$, $R()$, and $S()$. For example, the function $P()$ should be one-way; it should be difficult to find x given $P(x)$.

To set things up for the AKE protocol, Carol and Steve select parameters x and z, respectively. These serve as the passwords in the protocol. Carol computes $P(x)$ and gives it to Steve, and Steve computes $P(z)$ and gives it to Carol. Carol and Steve are now ready to use AKE to exchange keys using the following steps:

Carol		Steve
(generate random w)	$\xrightarrow{P(w)}$	$K = S(R(P(w),P(x)),Q(y,z))$
$K = S(R(P(y),P(z)),Q(w,x))$	$\xleftarrow{P(y)}$	(generate random y)

Table 2: Generic AKE

At this point, Carol and Steve have performed the basic AKE protocol and have their respective session keys. If the values of x and z used to compute the session key correspond to the previously agreed-to values of $P(x)$ and $P(z)$, then by Equation 1, the two values of K will match. To complete the authentication process, Carol and Steve can use any mutually agreeable method to verify that their keys match; the security of the resulting protocol is obviously dependent on the choice of this method.

From the basic AKE protocol, one can see the role of each of the four parameters: x and z are the long-term secrets held by the two parties, while w and y are ephemeral parameters generated by each side to ensure that the session key varies from session to session. As stated earlier, the security of AKE depends on the four functions it uses. Obviously, $P()$ should be difficult to invert, and its output should also reveal little or no information about its input. The same can be said for $S()$; it should be chosen especially to protect its second argument from leakage. Additionally, it should be infeasible to reconstruct either value of K using only $P(w)$, $P(x)$, $P(y)$, and $P(z)$. No closed-form expression that does this should exist, and ideally we would like this to be as difficult as inverting $P()$. Further restrictions will depend on the exact implementation of AKE.

3.2 SRP: An AKE construction

AKE by itself is merely an interesting mathematical exercise. It describes the broad outline of a family of key-exchange protocols, but it is necessary to fill in some of the blanks to make the protocol applicable and enable further detailed security analysis. This section presents the *Secure Remote Password* (SRP) protocol, one possible interpretation of AKE and one that is believed to be simple, fast, and highly secure.

3.2.1 SRP specifications

In SRP, all computations are performed in a finite field $\mathrm{GF}(n)$. In other words, a large prime number n is chosen ahead of time, and all additions, multiplications, and exponentiations are performed modulo n. All input parameters and outputs of $P()$, $Q()$, $R()$, and $S()$ are thus integers between 0 and $n-1$ inclusive.

The "one-way" verifier-generator $P()$ becomes a modular exponentiation in $\mathrm{GF}(n)$:

$$P(x) = g^x \tag{2}$$

g is a *generator* in $\mathrm{GF}(n)$. Remember that there is an implicit modulo n in each computation.

The functions $Q()$, $R()$, and $S()$ are the following:

$$\begin{aligned} Q(w,x) &= w + ux && (3) \\ R(w,x) &= wx^u && (4) \\ S(w,x) &= w^x && (5) \end{aligned}$$

The role of the variable u will be explained in Section 3.2.4. In these equations, u is defined as a function of w and x: $u = f(w,x)$. By inspection, Equations 2–5 satisfy Equation 1. More information about the number theory used here can be found in [16].

As stated in Section 3.1, the two parties still need to verify that their session keys match and do so in a secure manner. SRP accomplishes this with a simplified MAC (Message Authentication Code) based on one-way hash functions.

3.2.2 The SRP protocol

What follows is a complete description of the entire SRP authentication process from beginning to end, starting with the password setup steps.

Table 3 shows the notation used in this section. The values n and g are well-known values, agreed to beforehand.

To establish a password P with Steve, Carol picks a random salt s, and computes

$$x = H(s, P)$$

$$v = g^x$$

Steve stores v and s as Carol's password verifier and salt. Remember that the computation of v is implicitly reduced modulo n. x is discarded because it is equivalent to the plaintext password P.

The AKE protocol also allows Steve to have a password z with a corresponding public key g^z held by Carol; in SRP, we set $z = 0$ so that it drops out of the equations. Since this private key is 0, the corresponding

n	A large prime number. All computations are performed modulo n.
g	A primitive root modulo n (often called a *generator*)
s	A random string used as the user's *salt*
P	The user's password
x	A private key derived from the password and salt
v	The host's password verifier
u	Random scrambling parameter, publicly revealed
a, b	Ephemeral private keys, generated randomly and not publicly revealed
A, B	Corresponding public keys
$H()$	One-way hash function
m, n	The two quantities (strings) m and n concatenated
K	Session key

Table 3: Mathematical Notation for SRP

public key is 1. Consequently, instead of safeguarding its own password z, Steve needs only to keep Carol's verifier v secret to assure mutual authentication. This frees Carol from having to remember Steve's public key and simplifies the protocol.

To authenticate, Carol and Steve engage in the protocol described in Table 4. A description of each step follows:

	Carol		Steve
1.		$\xrightarrow{C}$	(lookup s, v)
2.	$x = H(s, P)$	$\xleftarrow{s}$	
3.	$A = g^a$	$\xrightarrow{A}$	
4.		$\xleftarrow{B,u}$	$B = v + g^b$
5.	$S = (B - g^x)^{a+ux}$		$S = (Av^u)^b$
6.	$K = H(S)$		$K = H(S)$
7.	$M_1 = H(A, B, K)$	$\xrightarrow{M_1}$	(verify M_1)
8.	(verify M_2)	$\xleftarrow{M_2}$	$M_2 = H(A, M_1, K)$

Table 4: The Secure Remote Password Protocol

1. Carol sends Steve her username, (e.g. `carol`).
2. Steve looks up Carol's password entry and fetches her password verifier v and her salt s. He sends s to Carol. Carol computes her long-term private key x using s and her real password P.
3. Carol generates a random number a, $1 < a < n$, computes her ephemeral public key $A = g^a$, and sends it to Steve.
4. Steve generates his own random number b, $1 < b < n$, computes his ephemeral public key $B = v + g^b$, and sends it back to Carol, along with the randomly generated parameter u.
5. Carol and Steve compute the common exponential value $S = g^{ab+bux}$ using the values available to each of them. If Carol's password P entered in Step 2 matches the one she originally used to generate v, then both values of S will match.

6. Both sides hash the exponential S into a cryptographically strong session key.

7. Carol sends Steve M_1 as evidence that she has the correct session key. Steve computes M_1 himself and verifies that it matches what Carol sent him.

8. Steve sends Carol M_2 as evidence that he also has the correct session key. Carol also verifies M_2 herself, accepting only if it matches Steve's value.

This protocol is mostly the result of substituting the equations of Section 3.2.1 into the generic AKE protocol, adding explicit flows to exchange information like the user's identity and the salt s. Both sides will agree on the session key $S = g^{ab+bux}$ if all steps are executed correctly. SRP also adds the two flows at the end to verify session key agreement using a one-way hash function. Once the protocol run completes successfully, both parties may use K to encrypt subsequent session traffic.

3.2.3 Computation of B

Observant readers will notice that Steve's ephemeral public key in Step 4 is the sum of two exponential residues. Why not just make $B = g^b$ and simplify the protocol?

Unfortunately, that simplification opens the protocol to the an active dictionary attack, carried out by an attacker who masquerades as a legitimate host and convinces Carol to make an authentication attempt. The attacker, Sue, captures s from a legitimate session and proceeds as follows:

1. Carol sends Sue her username.

2. Sue sends Carol the salt s she snooped earlier.

3. Carol sends Sue her public exponential residue A.

4. Sue picks her own random b and u, computes her own residue B and sends B and u to Carol.

5. Carol computes her session key $S = B^{a+ux}$, computes K from S, and happily sends Sue a proof of that K.

6. Sue simulates network failure or simply notifies Carol that the password was incorrect.

Now, Sue has A and her own b, along with a proof of K from Carol. She can guess at a password p', compute x' from it and then v' from that, construct S' as $S' = (Av'^u)^b$, and finally $K' = H(S')$, and check it against Carol's proof of the real K. If they match, the guessed password is correct.

Since this attack comes from an impostor who does not know v (anyone who *does* know v can already perform a dictionary attack), one way to thwart it is to force the host to commit to its value of v in Step 4. However, the way in which the residues g^b and v are combined must be selected carefully. If we denote the "combining function" used to compute B as $B = f(v, g^b)$, then we wish to avoid using functions f that have the property that $f(g^x, g^y) = g^{\hat{f}(x,y)}$ for some easily-derived $\hat{f}()$. The attack described above can be extended to situations where $f()$ has this undesirable property. This rules out, for example, $f(x, y) = xy$.

In addition, we also wish the value of B to leak as little information about v as possible, which rules out $f(x, y) = x \oplus y$ (i.e. "exclusive-or") or $f(x, y) = E_y(x)$, where $E_k()$ is a symmetric encryption algorithm. In either of these cases, an attacker can carry out a *partition attack,* which facilitates an off-line password search by eliminating impossible passwords. For example, if we used $B = v \oplus g^b$, an attacker could capture B and compute a guessed verifier v' for each password guess. If $B \oplus v' > n$, then that particular password guess can be ruled out as impossible. If this is done over a number of sessions, an attacker may be able to reduce the number of possible passwords to a number small enough to permit brute-force guessing.

Modular addition appears to be the simplest operation that leaks no information about v while at the same time enabling SRP to resist a dictionary attack by a fake host. Additionally, g must be a primitive root of GF(n) in order to make all values of B equiprobable for any v. If this requirement is not met, a partition attack again becomes possible.

3.2.4 The role of u

Why is the parameter u used at all in the SRP protocol when it is broadcast in the clear in Step 4? Let us assume, for the moment, that an intruder, Chris, who has captured v poses as a fake client attempting to gain access to the host. Let us also assume that Chris has somehow discovered the value of u in Step 3, perhaps through psychic ability or (more likely) as a result of a flaw in Steve's random number generator. Under these circumstances, Chris can gain access to the host using the following steps:

1. Chris sends Carol's username to Steve.

2. Steve sends Carol's salt s to Chris.

3. Chris computes
$$A = g^a v^{-u}$$
and sends it to Steve instead of using the regular formula for A.

4. Steve sends $B = v + g^b$ back to Chris as expected.

5. Chris computes the session key K as:
$$K = H((B - v)^a \bmod n)$$

6. Chris sends Steve a proof of this K and logs in as Carol.

This attack works because Steve computes his session key as:

$$S = (Av^u)^b = (g^a v^{-u} v^u)^b = g^{ab}$$

Note that this value is independent of the long-term keys, and can easily be computed by Chris. Since he has the same session key that Steve has, he can fool Steve into believing that he is Carol. Obviously, this attack can also be carried out if u is fixed to a publicly-known value.

To prevent Chris from being able to cancel out the v term in this manner, Steve must not reveal the value of u until after he receives A from the user. Since u is communicated publicly, it is possible to "piggyback" it on top of another public value, thus transmitting it implicitly. For example, both sides can compute u as a simple function of B, in which case Steve must wait for Carol to send out A before he sends back B and reveals u. In either case, $u = 0$ must be avoided for obvious reasons.

4 Security analysis

It is easy to prove that both AKE and SRP are "correct" in the sense that both parties are guaranteed to agree on a session key if the correct passwords are supplied and the software on both sides functions properly. It is more difficult to show, as this section attempts to do, that these protocols are in fact secure. This means many things in the context of authentication protocols. In general terms, an intruder, who is defined here as a malicious third party interested in subverting communications between Carol and Steve, must not be able to gain access to the host merely by observing the messages exchanged during a successful run of the protocol. In the case of SRP, we would like to strengthen this definition further in the following ways:

1. No useful information about the password P or its associated private key x is revealed during a successful run. Specifically, we wish to prevent an attacker from being able to guess and verify passwords based on exchanged messages.

2. No useful information about the session key K is revealed to an eavesdropper during a successful run. Since K is a cryptographically strong key instead of a limited-entropy password, we are not concerned about guessing attacks on K, as long as K cannot be computed directly by an intruder.

3. Even if an intruder has the ability to alter or create his own messages and make them appear to originate from Carol or Steve, the protocol should prevent the intruder from gaining access to the host or learning any information about passwords or session keys. At worst, an intruder should only be able to cause authentication to fail between the two parties (often termed a *denial-of-service* attack).
4. If the host's password file is captured and the intruder learns the value of v, it should still not allow the intruder to impersonate the user without an expensive dictionary search.
5. If the session key of any past session is compromised, it should not help the intruder guess at or otherwise deduce the user's password.
6. If the user's password itself is compromised, it should not allow the intruder to determine the session key K for past sessions and decrypt them. Even present sessions should at least be protected from passive eavesdropping.

A protocol with these properties is *robust*; in other words, it resists being compromised even if the participants in the protocol are not completely reliable or secure. Informally, such a protocol tolerates a wide range of attacks, preventing an attack on any part or parts of the system from leading to further security compromises. If an attacker manages to obtain a user's password, for example, the potential for damage should stop as soon as the user changes that password. This ties in closely with the concept of forward secrecy, which protects past information from future compromises.

4.1 Reduction to Diffie-Hellman

Fortunately, the mathematical structure of the SRP protocol is sufficiently similar to the Diffie-Hellman (DH) problem [4], a problem that is believed to be computationally infeasible with sufficiently large parameters, that it is possible to construct a proof linking the intractability of DH to that of compromising SRP. This proof establishes the security of SRP against passive eavesdropper attack.

We begin by presuming the existence of an algorithm or method that yields the SRP session key in polynomial time given all the information that is publicly known or transmitted during a legitimate and successful run of the SRP protocol, *as well as the user's password*. The reason for giving away this piece of information will be evident shortly. Such an algorithm can be modeled as an oracle Q that accepts the values A, B, u, g, n, and x from Table 4 and computes the session key $S = g^{ab+bux}$ from this input.

$$Q(g^a, g^b + g^x, u, g, n, x) = g^{ab+bux}$$

The DH conjecture claims that it is difficult to compute g^{ab} in $\mathrm{GF}(n)$ given g^a and g^b. By fixing $u = 2$ and $x = (n-1)/2$, we can define the DH oracle $\hat{Q}$ in terms of the SRP oracle Q as follows:

$$\hat{Q}(A, B, g, n) = Q(A, B + g^{(n-1)/2}, 2, g, n, (n-1)/2)$$

Substituting $A = g^a$ and $B = g^b$, we have:

$$\hat{Q}(g^a, g^b, g, n) = g^{ab}$$

Thus, if there existed a method to compromise the session key used in SRP through a passive attack, that same method could be used to break a DH key exchange in polynomially-equivalent time. This proof establishes that SRP resists passive attack at least as well as the Diffie-Hellman protocol.

In terms of our security requirements, this directly satisfies Requirement 6 in our security analysis, since revealing the password does not permit the computation of any previously-used session key. This also satisfies Requirement 2, since an intruder who does not know the password x has even less information about the session key.

The preceding proof establishes that it is computationally infeasible to construct a session key even with the user's password x and all public information. This applies to all possible values of x, not just the correct one. In other words, an intruder who eavesdrops on a successful SRP run cannot construct a guess at the session key using only publicly-visible information and a guessed value of x. Without the ability to construct guesses at K, the messages M_1 and M_2 leak no information to the passive off-line attacker[2]. Since A and B do not leak any information either (see Section 3.2.3), a passive attacker cannot verify guesses at the user's password. Thus, SRP resists passive dictionary attacks and satisfies Requirement 1.

4.2 Resistance to the Denning-Sacco attack

The *Denning-Sacco Attack* [3] occurs when an intruder captures the session key K from an eavesdropped session and uses it either to gain the ability to impersonate the user directly or to conduct a brute-force search against the user's password.

If K is revealed to a passive eavesdropper Eve, she does not learn any new information from combining K with M_1 or M_2. This is true because both M_1 and M_2 can be computed directly from publicly-visible data and K. We have already established that Eve cannot construct meaningful guesses at the session key K from guessed passwords, and there does not appear to be any easier way for her to carry out a brute-force dictionary attack. It is thus conjectured that Requirement 5 is satisfied.

Note that this differs from the Augmented-EKE protocol in [2] because A-EKE requires the user to send a message that is dependent on both the long-term private key and the session key. It is this message that enables the Denning-Sacco attack against that protocol.

4.3 Resistance to active attacks

SRP has been carefully designed to thwart the active attacks illustrated in Sections 3.2.3 and 3.2.4. Although it is difficult to determine conclusively whether or not these precautions bulletproof the protocol completely from all possible active attacks, SRP resists all the well-known attacks that have plagued existing authentication mechanisms, such as the Denning-Sacco attack mentioned previously. While no successful attacks have been discovered against SRP, a more formal analysis of active attack scenarios would be welcome.

Active attacks can take many different forms, depending on what information is available to the attacker. An attacker who knows Carol's private key x can obviously pretend to be Carol when accessing the host[3]. Likewise, an attacker with v can masquerade as Steve when Carol tries to contact him. Although the amount of damage that can be caused by a leaked verifier is limited compared to plaintext-equivalent systems, the verifier should not be treated as a public quantity.

A man-in-the-middle attack, which requires an attacker to fool both sides of a legitimate conversation, cannot be carried out by an attacker who does not know Carol's password. An attacker who does not know x cannot fool Steve into thinking he is talking to Carol, so at least one half of the deception fails. If the attacker doesn't know v either, he is in worse shape, because he also can't fool Carol into believing that she is communicating with Steve.

4.4 Security assumptions and constraints

The validity of the preceding security analysis depends on a number of conditions, most of which concern the proper generation and screening of various parameters in the SRP protocol. This section will discuss these conditions and put forth a set of constraints that will satisfy them.

4.4.1 Discrete logarithms

In Section 3.1, we mentioned that the function $P()$ selected for use in AKE must be difficult to invert. It is obvious why this is important: The security of SRP and any other construction of AKE depends on keeping

[2]This assumes that the hash function used to generate M_1 and M_2 is cryptographically secure, a concept that is beyond the scope of this paper.

[3]In a typical client/server environment, many do not consider this an active attack, since a user can initiate contact with the host from any location. In any event, this is much easier to carry out compared to a more conventional active attack.

the private values w and y secret while publicly revealing $P(w)$ and $P(y)$.

Recall that in the case of SRP, we have

$$P(x) = g^x$$

where the base g and the implied modulus n are publicly known and agreed-upon values. Computing $P(x)$ is known as *discrete exponentiation*, and its inverse is known as a *discrete logarithm*. Finding discrete logarithms is a problem long believed to be computationally difficult for large values of n (512 bits or longer)[4] and has been the subject of a great deal of research [11]. The apparent security of discrete exponentiation as a one-way trapdoor is used by other key-exchange protocols, most notably Diffie-Hellman [4].

Note that the proof-by-reduction of Section 4.1 actually relied on the intractability of the Diffie-Hellman problem itself, not the intractability of computing discrete logarithms in $\mathrm{GF}(n)$. While the ability to solve discrete logarithms implies the ability to break DH, the implication in the other direction has yet to be proven. Without loss of generality, the most accurate assessment of SRP at this time is that its security is linked to that of the underlying Diffie-Hellman problem.

4.4.2 Group parameter agreement

Both [4] and [1] discuss the safe generation of n and g. For SRP, we wish to maximize the difficulty of calculating discrete logarithms in $\mathrm{GF}(n)$. For this reason, n must be a non-*smooth* prime, which means that $n-1$ must not consist entirely of small factors [15].

Some authentication protocols based on discrete logarithms are potentially susceptible to a *subgroup confinement attack* [9], where an attacker forces the session key used by either party to be confined to a small subgroup of $\mathrm{GF}(n)$. Because of the way it computes session keys, SRP resists this attack. Since the probability of generating a smooth prime at random is quite small [11], n can, in practice, be safely generated by selecting a random, large prime.

Nevertheless, for maximal security, the author recommends that n be a *safe prime*, which is a number of the form

$$n = 2p + 1$$

where p is also prime. These numbers resist discrete logarithm computation and contain the smallest possible number of subgroups, since $n-1$ contains the fewest possible number of factors, 2. If n is a safe prime, an attempted subgroup confinement attack can be easily detected and avoided in all cases.

The protocol descriptions until now have assumed that the parameters n and g have been established in advance of the authentication attempt. One way to accomplish this is to have the server send n and g to the client as part of the protocol. Alternatively, as suggested in [9], parameters can be embedded into the software at both ends. The former approach has the advantage of being more flexible by allowing different parameters to be used for each host and even each user according to individual security and performance requirements. The latter approach can be used to avoid issues of testing ephemeral parameters for safety, issues that will be discussed in the next section.

4.4.3 Constraint checks

The following is a list of constraint checks that must be performed by both sides to ensure the security of the SRP protocol. Client testing of n and g is only necessary if these values are transmitted and not embedded or prearranged.

n is a large safe prime (client) The client must ensure that n is large enough to resist attack; see Section 4.4.1 for recommendations. Using a probabilistic primality tester, the client should also ensure that both n and $(n-1)/2$ are prime.

[4] As computational speeds creep upwards, the lower size bound for n will gradually increase as well. For that reason, many are recommending 1024 bits for long-term security.

g is a primitive root of GF(n) (client) Assuming the factorization of $n-1$ is known, the algorithm described in [16] for testing generators can be used to verify g. If n is a safe prime, this test is particularly easy and fast.

$A \neq 0$ (server) This prevents the server's session key from being forced to a known value, namely zero.

$B \neq 0$ (client) This check prevents a dictionary attack on the password from a masquerading host.

$a, b > \log_g n$ The computations of g^a and g^b in GF(n) must "wrap around" to prevent an attacker from taking the algebraic logarithm of g^a to recover a. The probability of this happening is infinitesimal (less than 2^{-1014} for 1024-bit n), but the check is trivial.

5 Optimizing SRP

Implementing a protocol such as SRP for use in real systems brings practical issues like performance into play. The number of message rounds, the size of the exchanged messages, and the expected execution time of a successful authentication attempt are all important factors in designing concrete protocol specifications. Eliminating even one network message or computational round can significantly improve the utility of an authentication system [5].

5.1 Message rounds

$C \Longrightarrow S$	C
$C \Longleftarrow S$	s
$C \Longrightarrow S$	A
$C \Longleftarrow S$	B
$C \Longrightarrow S$	M_1
$C \Longleftarrow S$	M_2

Table 5: Original SRP

Recall from Section 3.2.2 that the full SRP protocol involved a total of three round trips between client and server, as shown in Table 5. This section will assume that u is transmitted implicitly along with B, as discussed in Section 3.2.4. It is possible to reduce the total number of messages exchanged by consolidating some of the individual transactions, grouping together pieces of information that do not depend on earlier messages. For example, since the salt s and the client's exponent are independent of each other, they can be sent in either order. By rearranging additional messages, it is possible to reduce SRP to two round trips, as shown in Table 6.

$C \Longrightarrow S$	C, A
$C \Longleftarrow S$	s, B
$C \Longrightarrow S$	M_1
$C \Longleftarrow S$	M_2

Table 6: Optimized SRP

It is possible to reduce the number of messages even further if one is willing to settle for one-way authentication instead of the mutual authentication that is provided by both Original SRP and Optimized

$C \Longrightarrow S$	C, A
$C \Longleftarrow S$	s, B
$C \Longrightarrow S$	M_1

Table 7: One-Way Optimized SRP

SRP. Table 7 shows a three-message, one-and-a-half round trip implementation of SRP that authenticates the client to the server, but not the other way.

Three messages appears to be the theoretical lower bound for a secure authentication protocol, which also matches the minimal protocol presented in [17]. This is based on the observation that a two-message protocol (one in which the client sends a message to the server and the server sends a message back) is trivially vulnerable to a replay attack, assuming that no out-of-band communication is used, like a biometric input device or synchronized clocks.

5.2 Execution speed

Of all the operations executed in the course of negotiating secure protocols like SRP, the slowest one by far is the iterated group operation, modular exponentiation in this case. By comparison, other functions such as hashing, addition, and multiplication require a negligible amount of processor time. Any discussion of performance issues necessarily centers around the speed of the group operation.

Instead of lumping all modular exponentiation operations into the same category and counting them, we can arrive at more accurate performance estimates by subdividing them into three categories. Our notation will be the following:

t_g The amount of time required to execute a modular exponentiation with a tiny base (e.g. $g = 2$).

t_e The amount of time required if the exponent is tiny.

t_b The amount of time required if neither base or exponent is tiny.

For the purposes of our benchmark data, we will use a 1024-bit safe prime modulus and 256-bit exponents. Tiny exponents are 32 bits long. Although performance figures will vary with differing parameter sizes, host architectures, and software implementations, their relative values should remain consistent.

Protocol	Client	Server
Diffie-Hellman/DH-EKE	$t_g + t_b$	$t_g + t_b$
SPEKE	$2t_b$	$2t_b$
A-EKE	$2t_g + t_e + t_b$	$2t_g + 2t_b$
B-SPEKE	$3t_b$	$t_g + 3t_b$
SRP	$2t_g + t_b$	$t_g + t_e + t_b$

Table 8: Reference Running Times

Table 8 gives the amount of time required to negotiate some well-known protocols, including the three verifier-based protocols currently in existence. In this table, Augmented-EKE is evaluated with the **p-NEW** digital signature algorithm discovered by Nyberg and Rueppel [14], one of the fastest such algorithms usable with A-EKE.

To save time, one could easily implement the client and server so that they do some of the computations in parallel. A reasonable lower bound on execution time can be calculated simply by taking the greater of the two times, since that will determine the critical path of the protocol.

Table 9 shows performance figures gathered from a 167 MHz single-processor Sun ULTRASparc-1 running Solaris 2.5. The GNU MP library, built with the GNU C compiler, was used to perform the multiple-precision arithmetic. For this platform, $t_g = 247$ ms, $t_e = 45$ ms, and $t_b = 379$ ms.

Protocol	Execution Time	Normalized
Diffie-Hellman/DH-EKE	626 ms	1.000
SPEKE	758 ms	1.211
SRP	873 ms	1.395
A-EKE	1252 ms	2.000
B-SPEKE	1384 ms	2.211

Table 9: Benchmarks on a 167 MHz ULTRASparc-1

SRP ends up being the fastest verifier-based protocol in the table. Compared to SRP, A-EKE requires 41% more running time, while B-SPEKE is nearly 60% slower. Tests with other implementations yield results that differ by a constant factor, so the "normalized" column remains accurate across platforms.

SRP has other performance advantages that these tables do not necessarily show. For example, to reduce running times even further, the values g^a and g^b can be precomputed before either party begins authentication. This is practical when the group parameters n and g are known to both parties ahead of time instead of being exchanged during the course of the protocol. Not all protocols can employ this strategy; in particular, the SPEKE family, because its exponentials are functions of the shared password, cannot do this [10].

To improve the running time any further, we would need to switch to another AKE construction that used something other than discrete exponentiation. One promising candidate is the elliptic curve cryptosystem [12], which can potentially offer the same level of security as cryptosystems based on the difficulty of discrete logarithms but with much shorter keys. Jablon [10] claims that elliptic curve methods improve the speed of group operations by a factor of 6 to 7 while maintaining an equivalent level of security. It is still too early to make any firm pronouncements on the security of elliptic curves, however, since they have not been analyzed as extensively as discrete exponents. In addition, elliptic curves are encumbered by royalty and patent restrictions, which is not the case for simple discrete exponentiation. In the author's opinion, SRP is efficient enough, even on today's hardware, that performance is not a significant issue. In most cases, the time required to negotiate the authentication protocol is not even noticeable (under 1 second) to the user, and this will only improve as hardware becomes faster. The well-established security afforded by discrete exponentiation should satisfy even the most conservative security requirements.

6 Conclusion

Password authentication protocols have traditionally used symmetric encryption, public-key encryption, or a combination of the two approaches to resist common attacks. Only recently, however, has there been significant attention paid to designing strong direct authentication protocols that could be deployed without depending on expensive external infrastructure. In this paper, we showed that existing protocols have started to address this problem with some success, but that there was still room for improvement. While current direct authentication technology offers a variety of tradeoffs between security and performance, the compromise has been somewhat unsatisfying to implementors.

Next, we presented the groundwork for a new family of authentication protocols called AKE, which employed a swapped-secret instead of a traditional shared-secret arrangement and which did not use any form of symmetric encryption to achieve its security. Section 3.1 outlined some of the benefits of avoiding encryption in an authentication protocol.

We then presented a construction of AKE, known as SRP, based on discrete exponentiation and outlined a proof (Section 4.1) that established a lower bound on the security of SRP. In our analysis, we put forth a series of requirements for a secure password protocol, emphasizing those that existing protocols failed to meet. We demonstrated that SRP, as a verifier-based, zero-knowledge protocol resistant to dictionary attacks, offered a number of new benefits for password system implementors:

- An attacker with neither the user's password nor the host's password file cannot mount a dictionary attack on the password. Mutual authentication is achieved in this scenario.
- An attacker who captures the host's password file cannot directly compromise user-to-host authentication and gain access to the host without an expensive dictionary search.
- An attacker who compromises the host does not obtain the the password from a legitimate authentication attempt.
- An attacker who captures the session key cannot use it to mount a dictionary attack on the password.
- An attacker who captures the user's password cannot use it to compromise the session keys of past sessions.

It is believed that this set of properties is at or near the theoretical limit of security that can be offered by a purely password-based protocol. SRP, which bases its security on the difficulty of solving the Diffie-Hellman problem in the multiplicative field modulo a large safe prime, meets these requirements and does so using only one exponential key exchange round, making it useful for applications in which good performance is an issue. It solves some outstanding issues with protocols like EKE and SPEKE without sacrificing either performance or security. SRP's security, simplicity, and speed make it ideal for a wide range of real-world applications in which secure password authentication is required.

Acknowledgements

The author would like to thank Dan Boneh (Stanford), John Gill (Stanford), Doug Tygar (CMU), Li Gong (JavaSoft), David Jablon (Integrity Sciences), and the many readers of `sci.crypt` for their comments and feedback regarding this paper. The author gratefully acknowledges the support of the Defense Advanced Research Projects Agency under Contract DABT63-94-C-0055. The author would also like to thank Eugene Jhong for his help with software coding and development and Paul Losleben for helping to make this research possible.

References

[1] 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.

[2] S.M. Bellovin and M. Merritt. Augmented encrypted key exchange: A password-based protocol secure against dictionary attacks and password file compromise. Technical report, AT&T Bell Laboratories, 1994.

[3] D. Denning and G. Sacco. Timestamps in key distribution systems. *Communications of the ACM*, August 1981.

[4] W. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, November 1976.

[5] L. Gong. Efficient network authentication protocols: Lower bounds and optimal implementations. *Distributed Computing*, 9(3):131–145, 1995.

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

[7] F.T. Grampp and R.H. Morris. Unix operating system security. *AT&T Bell Laboratories Technical Journal*, 63(8):1649–1672, October 1984.

[8] N. Haller and R. Atkinson. *On Internet Authentication*. Naval Research Laboratory, October 1994. Request For Comments (RFC) 1704.

[9] D. Jablon. Strong password-only authenticated key exchange. *Computer Communication Review*, 26(5):5–26, October 1996.

[10] D. Jablon. Extended password methods immune to dictionary attack. In *WETICE '97 Enterprise Security Workshop*, Cambridge, MA, June 1997.

[11] B.A. LaMacchia and A.M. Odlyzko. Computation of discrete logarithms in prime fields. *Designs, Codes, and Cryptography*, 1:46–62, 1991.

[12] A. Menezes and S.A. Vanstone. Elliptic curve cryptosystems and their implementations. *Journal of Cryptology*, 6(4):209–224, 1993.

[13] R.H. Morris and K. Thompson. Unix password security. *Communications of the ACM*, 22(11):594, November 1979.

[14] K. Nyberg and R.A. Rueppel. Message recovery for signature schemes based on the discrete logarithm problem. In *Advances in Cryptology—EUROCRYPT '94 Proceedings*. Springer-Verlag, 1995.

[15] S.C. Pohling and M.E. Hellman. An improved algorithm for computing logarithms in gf(p) and its cryptographic significance. *IEEE Transactions on Information Theory*, 24(1):106–111, January 1978.

[16] Bruce Schneier. *Applied Cryptography*. John Wiley & Sons, Inc., New York, 1996.

[17] M. Steiner, G. Tsudik, and M. Waidner. Refinement and extension of encrypted key exchange. *ACM Operating Systems Review*, 29(3), July 1995.