SRP-6: Improvements and Refinements to the Secure Remote Password Protocol

Thomas Wu
Arcot Systems
tom@arcot.com

October 29, 2002

Abstract

This document addresses two specific security and operational issues with the Secure Remote Password Protocol, the first being the "two-for-one" active password guessing attack by an attacker posing as a server, and the second being the message ordering property which requires that the server wait for the client's first exponential residue before sending its own. The effect that these improvements have on real-world implementations of SRP is also explored.

1 Introduction

The *Secure Remote Password* protocol, first published in 1998 [3], is an authenticated key-exchange protocol designed to resist both passive and active network adversaries even when used with relatively short, human-memorizable passwords. The original protocol, sometimes referred to as "SRP-3" for historical reasons and specified in [4], operates in a group defined by a large safe prime N and a primitive root g. Reviewing briefly, the server computes its *verifier* value v for a user identity I as follows:

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

$$v = g^x$$

All values are computed modulo N. The value s is a random salt, which is stored along with v. The authentication protocol itself proceeds as described in Table 1.

	Client		Server
1.		$\xrightarrow{I}$	(lookup s, v)
2.	$x = H(s, I, P)$	$\xleftarrow{s}$	
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.	$M_1 = H(A, B, S)$	$\xrightarrow{M_1}$	(verify M_1)
7.	(verify M_2)	$\xleftarrow{M_2}$	$M_2 = H(A, M_1, S)$
8.	$K = H(S)$		$K = H(S)$

Table 1: The Secure Remote Password Protocol (SRP-3)

H is a secure hash function, and the values a, b, and u are generated randomly. At the end of a successful protocol run, both sides will share a secret session key K.

As a *strong password protocol*, SRP attempts to prevent passive adversaries from obtaining any useful information about the password by observing successful protocol runs, and it seeks to limit active adversaries to a single on-line password guess for every impersonation attempt. It is easy to show that, for each session that he interferes with, an active adversary can always get one password guess by noting that both sides of the protocol have only the password as a shared secret, and that an active attack can simply guess at a password and "pretend" to be one of the authenticating parties, using that password guess as the shared secret. Ideally, a strong password protocol seeks to limit an attacker to that theoretical minimum if such an active attack is attempted.

SRP in its current form allows an active attacker to make and verify *two* password guesses per impersonation attempt[2]. This property does not pose a significant practical security threat to existing implementations, since each guessing attempt results in a failure detectable to both sides and would still require an unrealistic number of on-line attempts, even with the number required cut in half. Nevertheless, this falls short of the theoretical limit, and a simple change to the protocol that eliminates this "two-for-one" attack will be presented.

SRP, as originally proposed, also imposes a limitation on the ordering of its protocol messages. In Steps 3 and 4 of Table 1, the server must wait for the client's value of A before revealing its value of u. In some cases, this restriction prevents certain forms of optimization when SRP is integrated into existing security and authentication protocols. It is sometimes beneficial to have the flexibility to send both key exchange messages asynchronously to save time on high-latency links, or to send all of the server's key exchange messages first to reduce the number of network round trips. A relatively minor modification to SRP will be presented which eliminates this "message-ordering" limitation.

2 Two-for-one guessing

As noted previously, an active attacker can validate two password guesses per impersonation attempt. Note that the server is supposed to send the client the value $v + g^b$, which is just $g^x + g^b$. An attacker who does not know either x or v can make a single guess at v while attempting to impersonate the server by using that guessed value of v in the server's calculation. However, because of the symmetry of the equation for the server's value, it is also possible for the attacker to insert an additional password guess by sending the client the value $g^x + g^y$, where x and y are guessed passwords; the second password guess y simply takes the place of the random exponent b. If the actual password is x, the client will subtract out g^x and use g^y as the base in its session key calculation. In this case, the attacker can use y as its b value in its own session key calculation, since it is the discrete log of g^y, the client's base. The same holds if x and y are swapped.

In fact, this attack applies to any variant of the protocol in which an attacking server can send the client a key exchange value that results in the client using a base value whose discrete log is known to the server if the client's password is either of two values chosen by the server.

One simple way to remove the symmetry in the server's key exchange value is to multiply v by some value k agreed to by both sides:

$$B = kv + g^b$$

Incidentally, this variant can also be implemented by multiplying the g^b value by k with no difference in security; however doing it this way requires the client to implement modular division or modular inversion to compute his session key, whereas the variant as currently proposed does not.

It is easy to see that this variant does not protect against the server guessing two passwords at once if the attacker knows the discrete log of k in the SRP group. If $k = g^j$, where the adversary knows j, he can send the client:

$$B = kg^x + kg^y$$

If the actual password is x, the client will subtract out kg^x from this value, leaving it with kg^y as its base.

But $kg^y = g^j g^y = g^{j+y}$, so the attacker knows the discrete log of the client's base and can determine if x is the correct password. The same holds if x and y are swapped.

The value of k can change from one run of the protocol to the next, it can be associated with the SRP group parameters N and g, or it can be a constant. Although the last option is the simplest, care must be taken to ensure that k is never a known exponential of g, in light of the previous analysis.

Note that as a safe prime, N can be expressed as $2q+1$, where q is an odd prime. Also note that N must be 2 (mod 3) because a value of N that was 1 (mod 3) would make q divisible by 3. The Legendre symbol $(\frac{3}{N})$ can thus be computed:

$$\left(\frac{3}{N}\right) = \left(\frac{N}{3}\right)(-1)^{(N-1)(3-1)/4} = \left(\frac{N}{3}\right)(-1)^q = -\left(\frac{N}{3}\right) = -\left(\frac{2}{3}\right) = -(-1)^{(3^2-1)/8} = 1$$

This means that the value $k = 3$ is always a quadratic residue modulo N, which in turn means that g could never be chosen to be equal to k. Since 3 is also not an integral power of any other integer, it is also nearly impossible for g to be a known root of k by accident. Using $k = 3$ ensures that any set of parameters N and g that are safe to use with SRP are also safe to use with k, from the standpoint of eliminating the "two-for-one" attack.

3 Message ordering

The implications of the "message-ordering" property of SRP can be better understood by studying the various proposals to optimize SRP and integrate it into existing protocols. Table 2 shows a version of SRP suggested in [3] that required only three messages, with an optional fourth message for the server to authenticate itself to the client. In this version of the protocol, the parameter u is no longer an explicit protocol message, but is instead calculated as a function of the server's B value.

$C \Longrightarrow S$	I, A
$C \Longleftarrow S$	s, B
$C \Longrightarrow S$	M_1
$C \Longleftarrow S$	M_2 (optional)

Table 2: Original optimized SRP

This version of the protocol requires that both sides agree on the group parameters N and g in advance, because the client needs to know them in order to calculate A in the first message. In practice, however, this is difficult to arrange. Since the password verifier value (i.e. $v = g^x$) for a particular user on a particular server is tied to a particular set of parameters, the client cannot unilaterally select the parameters the way it might in, say, an unauthenticated Diffie-Hellman key exchange. Even if SRP group parameters were standardized by bit length, the client would still need to know the user's group size before starting the negotiation, which would pose problems for both usability and implementation. Forcing all users of a particular protocol to use a single standard group with a fixed bit length would resolve this issue, but the loss of flexibility resulting from mandating a single fixed group would be a serious problem if the protocol were intended to be general-purpose in nature.

In practice, it is more natural for the server to send the client the values of the group parameters for that user after receiving the username in the first protocol flow from the client. This way, the client does not need to anticipate or otherwise keep track of which parameters are used for which users or servers; it only needs to verify their validity, which can be done mathematically or by simple table lookup. Table 3 shows the resulting sequence of message flows, as documented in [4].

Unfortunately, adding the transmission of the group parameters has lengthened the protocol by a full round-trip. An astute observer might wonder why the server couldn't send his value B as part of his first reply to the client and then have the client reply with both A and M_1 folded into one message flow? The

$C \Longrightarrow S$	I
$C \Longleftarrow S$	N, g, s
$C \Longrightarrow S$	A
$C \Longleftarrow S$	B
$C \Longrightarrow S$	M_1
$C \Longleftarrow S$	M_2 (optional)

Table 3: SRP with parameters

problem with this rearrangement is the security requirement that the server's value u must never be sent before receiving the client's value A. In protocols that use the message B to derive the shared value u, the same restriction applies to B, since revealing B reveals u to the client. As described in Section 3.2.4 of [3], revealing u to the client before he sends his value A allows him to carry out an attack. More specifically, an attacker who knows the server's verifier value v (but not x) can pose as a client and send the value $A = g^a v^{-u}$ instead of $A = g^a$, and this will "cancel out" the v^u term in the server's session key calculation, allowing the attacker to impersonate the user whose verifier he has stolen without performing even a dictionary attack. Even if u is calculated and sent separately from B, it would still be necessary to send u itself after A, and the client would not be able to send M_1 along with A because he could not compute it without knowing u.

The key to resolving this apparent impasse is the realization that it is not absolutely necessary to have the server withhold knowledge of u from the client until he has received A. The real constraint is that the client be unable to *manipulate* A with knowledge of u so that it has the special form described earlier. It is also useful to observe that the attack described previously against a server that reveals u too soon was made simple by the fact that the value of u did not depend on A, so the client could freely manipulate A once he knew the "final" value of u.

If, however, the value of u is taken as the output of a one-way hash function whose input includes the client's message A, for example:

$$u = H(A, B)$$

this attack becomes considerably harder to mount, even if the server's value B is sent first. The attacker must find a value u for which

$$u = H(g^a v^{-u}, B)$$

Even though the attacker can choose the values of a and u arbitrarily, a hash function H with preimage resistance makes it difficult for him to pick a value of u and work backwards to find a suitable value of a. The output length of H must be long enough to resist an exhaustive search attack based on varying the value of a. Using the full output of a secure hash function like SHA-1 to compute u will make such an attack infeasible.

This optimization shortens the protocol by a full round-trip, even with group parameters sent by the server. Table 4 shows a sequence of protocol flows that takes advantage of the removal of the "message-ordering" property and sends the server's key exchange message along with the parameters, saving a round trip and restoring the protocol to the minimal number of rounds in the original "optimized" SRP.

$C \Longrightarrow S$	I
$C \Longleftarrow S$	N, g, s, B
$C \Longrightarrow S$	A, M_1
$C \Longleftarrow S$	M_2 (optional)

Table 4: SRP with optimized message ordering

4 Conclusion

Table 5 shows the SRP protocol incorporating the two refinements introduced in this paper.

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

Table 5: SRP with refinements (SRP-6)

The end result of these relatively small refinements is a strong password protocol that holds active adversaries to the desired limit of one password guess per impersonation attack, as well as a protocol that offers more flexibility for implementors by allowing the ordering of the client and server key exchange messages to vary. This is particularly useful for protocols that send the user's group parameters N and g in the server's first message and wish to save a round trip by including B in this message. One example of such a protocol is SSL/TLS [1], which has a fairly fixed set of message flows; integration with SRP is considerably easier when the server can send all of its key exchange messages in its `ServerKeyExchange` message, which is defined by the protocol to precede the `ClientKeyExchange` message from the client.

The improved protocol will be called "SRP-6" to distinguish it from previous variants of SRP, both official and unofficial.

References

[1] T. Dierks and C. Allen. *The TLS protocol version 1.0.* Certicom, January 1999. Request For Comments (RFC) 2246.

[2] P. MacKenzie. On the security of the SPEKE password-authenticated key exchange protocol. Technical Report 2001/057, Lucent Technologies, 2001.

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

[4] T. Wu. *The SRP Authentication and Key Exchange System.* Stanford University, September 2000. Request For Comments (RFC) 2945.