Extended Password Key Exchange Protocols Immune to Dictionary Attack *

David P. Jablon
Integrity Sciences, Inc.
http://world.std.com/~dpj/

Abstract

Strong password methods verify even small passwords over a network without additional stored keys or certificates with the user, and without fear of network dictionary attack. We describe a new extension to further limit exposure to theft of a stored password-verifier, and apply it to several protocols including the Simple Password Exponential Key Exchange (SPEKE). Alice proves knowledge of a password C to Bob, who has a stored verifier S, where $S=g^C$ mod p. They perform a SPEKE exchange based on the shared secret S to derive ephemeral shared key K_1. Bob chooses a random X and sends g^X mod p. Alice computes $K_2=g^{XC}$ mod p, and proves knowledge of $\{K_1,K_2\}$. Bob verifies this result to confirm that Alice knows C. Implementation issues are summarized, showing the potential for improved performance over Bellovin & Merritt's comparably strong Augmented-Encrypted Key Exchange. These methods make the password a strong independent factor in authentication, and are suitable for both Internet and intranet use.

1. Introduction

In our enlightened age of public-key cryptography, passwords are still used everywhere, and people still can't remember passwords large enough to use as ordinary encryption keys. Dictionary attacks against many methods, including Kerberos [BM89], are often taken for granted, and the concept of "good" and "bad" passwords is part of computer security folklore. What is often overlooked is that the quality of a password is largely determined by strength of the verification method. A bank teller machine uses a 4-digit password in a strong way. Since the early 1990's, the discovery of strong methods permits easily-memorized passwords to be verified over an insecure network, without using additional keys, and without fear of network dictionary attack.

We describe new and modified protocols in this class, that verify a potentially low-entropy shared secret, and protect it as much as possible; The secret is not revealed to anyone who doesn't already have it. Our goal is also to gracefully handle passwords of large-entropy too. When considering theft of a host-stored hashed-password database, large passwords still provide more security than small, but strong methods don't fall to network attack when password entropy is less than optimal.

Known methods that presume both parties share the same secret include:

- EKE -- Encrypted Key Exchange [BM92]
- The "secret public key" methods [GLNS93]
- SPEKE -- Simple Password Exponential Key Exchange [Jab96], and
- OKE -- Open Key Exchange [Luc97].

Use of a one-way hashed password on *both* sides prevents the need for clear-text passwords on the host. But a thief who steals the hashed password from the host can use it to masquerade as the user in the protocol. One solution to this problem is the Augmented Encrypted Key Exchange (A-EKE) described in [BM93]. In this method the host's verifier is a one-way function of the password, which is used to verify a proof that the user knows the password, and a thief cannot use the verifier directly to masquerade as the user in the protocol. We describe a new alternative to this method.

The limitation of all such extended methods is that a stolen verifier permits brute-force attack. The severity of this depends on the quality of the password relative to the resources and ingenuity of the thief. We conservatively assume that a brute-force attack on the verifier will reveal the password, and assume that a "best effort" is made to keep the host's verifier database secret. We also note that the requirement for secret host data is generally required for all mutual authentication techniques. Our goal remains simple: Provide the strongest security possible for both small and large passwords.

In § 2 we review EKE and SPEKE, two previously published methods for shared-key authentication, and the A-EKE extended method. In § 3 we describe a new "B" extension that replaces the "A" in A-EKE, and show several new combinations for extended methods. Security analysis and constraints are discussed in § 4,

* Proceedings of the Sixth Workshops on Enabling Technologies: Infrastructure for Collaborative Engineering (WET-ICE '97 Enterprise Security), IEEE Computer Society, Cambridge, MA, June 18, 1997, pp. 248-255.
0-8186-7967-0/97 $10.00

implementation and performance issues in § 5, and further thoughts on benefits and limitations of these methods are in § 6.

2. Review of SPEKE, DH-EKE, and A-EKE

We review three shared-secret protocols. In SPEKE and DH-EKE, both parties share a common secret as the basis for authentication, with the distinctive feature that eavesdropper dictionary attack on a small secret is prevented. A-EKE extends DH-EKE to reduce the vulnerability to theft of a host's stored verifier.

In both SPEKE and DH-EKE, two parties, Alice and Bob share knowledge of a secret value S, where S represents a password, or an agreed-upon one-way function of the password. They perform a modified Diffie-Hellman [HDM77] exchange (DH) to agree on a large key K, and then (at least) one party sends P(K), a proof of knowledge of K, to the other. We use knowledge of K to imply knowledge of S. The difference between these protocols is in how the exchange is modified to incorporate S.

2.1. SPEKE

In SPEKE, prior to the protocol exchange, Alice and Bob agree to use the shared secret S to determine the parameters for the DH protocol. A simple example uses Z_p^*, with prime p, where p = 2q+1 for a prime q.

Alice→Bob: $h(S)^{2 R_A}$
Bob→Alice: $h(S)^{2 R_B}$

R_A and R_B are random numbers, and all exponentiation is performed modulo p. Both parties compute $K = h(S)^{(4 R_A R_B)}$ mod p, and exchange proofs of knowledge of K. The 2 in the exponent forces the exponential to be a generator of the subgroup of order q, and the result K is tested to insure that it's not 1. Further details can be found in [Jab96].

This paper describes an additional limitation for SPEKE in § 4.2, which is especially relevant for our extension.

2.2. DH-EKE

In contrast with SPEKE, DH-EKE [BM92] uses a fixed base g, and the DH "public keys" are kept secret from eavesdroppers by symmetrically encrypting them using the password as a (weak) key.

Alice→Bob: $E_S(g^{R_A})$
Bob→Alice: $E_S(g^{R_B})$

Both parties compute $K = g^{(R_A R_B)}$ mod p, and exchange proofs of knowledge of K. The value g is a generator of Z_p^*, and p and E are chosen carefully to insure that the exponentials do not contain verifiable plain-text for E_S. The symmetric encryption can be a surprisingly simple XOR function, such as:

$E_S(x) = x \oplus (h(S) \bmod p)$

The goals of SPEKE and DH-EKE are:

1. Without using S, one can't successfully perform the exchange.
2. Without using S, attempting an exchange reveals minimal information about S.[1]
3. Observing messages in a valid exchange reveals no information about S.[2]

These goals restrict the Diffie-Hellman parameters used in SPEKE, and more severely in DH-EKE. Analysis of how they are met (or not met) is in [BM92, STW95, Jas96, Pat97], and a summary of required constraints is listed in [Jab96]. The additional limitation in § 4.2 is also relevant to DH-EKE.

2.3. A-EKE

Augmented-EKE has been described in [BM93, BM95], and further discussed in [STW95]. In this extended method, DH-EKE is used to negotiate a key K, based on shared knowledge of a secret verifier S, where $S = h(C)$, a one-way function of the password C. In our discussion of extended methods, Alice is a user, and Bob a host who stores the verifier.

A-EKE defines an additional digital signature function where a private/public key pair $\{U_C, V_C\}$ is chosen as a function of C. Bob stores the public key V_C as a second verifier for C. The protocol begins with a DH-EKE exchange that derives K based on shared knowledge of S. Alice then computes her private key U_C, signs K with U_C, encrypts the signature using K as a symmetric key, and sends it to Bob. Bob decrypts using K, verifies the signature using V_C, and if the signature for K is correct, he knows that Alice knows C.

Alice: compute U_C,V_C from C
Both: derive K using DH-EKE based on S
Alice→Bob: $E_K(\{K\}_{U_C})$
Bob: decrypt with K to get $\{K\}_{U_C}$, and verify signature of K using V_C

If C is too-small, the "one-way" property doesn't preclude dictionary attack, so storage of both V_C and S must be protected. To reduce storage requirements, or perhaps to make a stolen-verifier dictionary attack more expensive, one can use $h(C)=V_C$.

A-EKE is essentially the addition of an "A" signature method to extend DH-EKE. "A" uses the password as a (weak) private key for a "secret public key" verifier stored by the host. Although someone with the verifier may crack a weak password, we still have the strong protection of EKE if the verifier is kept secret, or if the password is strong. The added insurance is that a stolen verifier can't be used directly to masquerade as the user in the protocol.

---

[1] Each attempt only reveals that $S \neq S'$, for one specific S'.

[2] Not in the information-theoretic sense, but in the usual sense, depending on a commonly accepted hard problem like discrete log.

As some attackers are more resourceful than others, this protects each password as much as possible.

Bellovin and Merritt [BM93] also discuss how "A" doesn't work with PK-EKE, a different variant of EKE that uses a public-key cryptosystem. In PK-EKE, one side chooses the session key for the other, which is discussed further in § 4.4.

3. New extended methods

We now describe a "B" extension, and three new combinations, naturally labeled A-SPEKE, B-SPEKE, and B-EKE, which are alternate ways of achieving our goals. These methods increase the size of the "extended family" of extended methods from one to four. We start by describing B-SPEKE.

3.1. B-SPEKE

"B" is similar to "A" in that it uses a public-key technique, but it uses a second Diffie-Hellman exchange instead of digital signature to prove Alice's knowledge of the password C. B-SPEKE assumes both parties have access to a verifier S=*h*(C), and that only Alice knows C. SPEKE derives a first ephemeral key based on the hashed-password S. The second exchange generates a second ephemeral shared key, using C as Alice's secret exponent. The two keys are combined to additionally prove knowledge of the clear-text password from Alice to Bob.

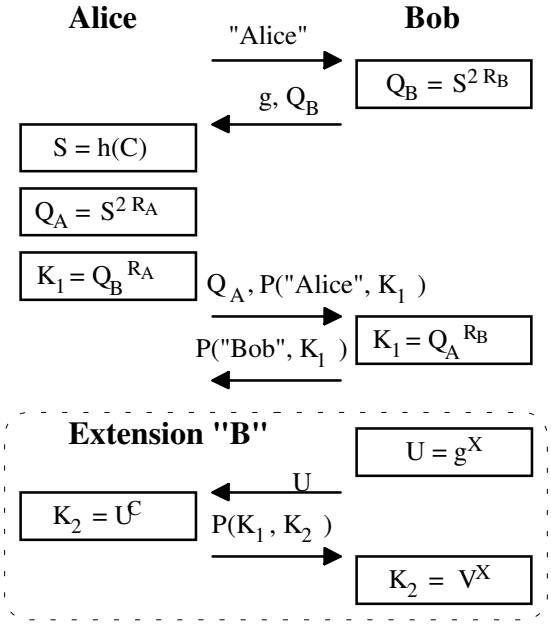

Figure 1. B-SPEKE, unoptimized

During password setup, a DH base g is chosen, and a DH exponential $V = g^C$ is computed where C is based on Alice's password. The shared secret for the primary SPEKE exchange is S = *h*(C), using a one-way function. The verifiers {g, V, S} are stored by Bob.

To mutually authenticate, as shown in figure 1, the primary SPEKE exchange uses S to create a shared key K_1. Alice computes S from C. In the secondary DH exchange, Bob chooses a random ephemeral exponent X, and sends $U = g^X$ as a challenge. Alice computes $K_2 = U^C$ and Bob computes $K_2 = V^X$, resulting in a second shared key. Alice then proves knowledge of the combined values $\{K_1, K_2\}$ to Bob, thereby proving knowledge of C. The low-entropy concerns are resolved by protecting the Bob's storage of V and S, and by the combined proof of K_2 with the partially-authenticated K_1. The subsequent session key is derived from K_1.

Interestingly, while the B extension adds two messages and A only one, optimizations in § 5 make this difference irrelevant.

3.2. B-EKE and A-SPEKE

Replacing SPEKE with DH-EKE in the above yields a B-EKE protocol. The differences between B-SPEKE and B-EKE largely affect the speed of an implementation. A-SPEKE is a straightforward application of Bellovin & Merritt's augmented technique of A-EKE to SPEKE, which has not been previously examined in the literature.

4. Analysis of B-SPEKE

The security of B-SPEKE clearly depends on the difficulty of the Diffie-Hellman and the discrete log problems. These two long-standing problems are closely related, and the reader is referred to [MW96] for qualified proof of their equivalence. Our informal analysis focuses on how the protocol protects against eavesdropper, middleman, and end-point attack, to either obtain information about the password, or to obtain a shared session key without using the password.

4.1. Preserving secrecy of K

It is noted in [STW95] that if the key (K) used in the primary stage of A-EKE is ever obtained by a third party, it allows a dictionary attack on the password. If K is later used to secure a session containing sufficient known plain-text, or remains in either system's memory for an extended time, the thread of a stolen K may be significant. This threat applies to any extended method. To prevent it, K should be used only for the clear-password-verification and derivation of the session key K' = *H*(K), and then K should be promptly destroyed.

4.2. The "password-in-exponent" problem

An added constraint for both SPEKE and DH-EKE is needed, which has not been well covered in earlier

papers. Discovery of the general problem of associativity between the symmetric and asymmetric cryptosystems in is attributed to Li Gong, but the [BM92] paper only describes how this affects the PK-EKE protocol. This same general problem also lurks in the DH methods, and we show here how to keep it in check.

SPEKE doesn't use a symmetric cryptosystem, but it does use a function to convert the password into a generator of a group. If the DH base is chosen as g^S for a well-known g, as regretfully suggested in [Jab96], an attacker can perform a dictionary attack after participating in one failed protocol exchange.[3]

Alice:	$Q = (g^S)^{R_A}$
Alice→Bob:	Q
Bob:	$K = Q^{R_B}$

The attack is accomplished by Bob, who sends g^X, using an arbitrary X, and for each candidate password T, he compares the received proof of K against his computed proof of $Q^{X/T}$. When a match is found, he knows T = S. This attack is only possible when S is exposed in the exponent corresponding to a known base. The preferred SPEKE method in [Jab96] and the method shown in § 2.1 do not have this problem.

DH-EKE is similarly broken if one uses Pohlig-Hellman-style symmetric encryption for the exponentials. In this case the broken SPEKE and broken DH-EKE protocols become essentially the same:

Alice:	$W = g^{R_A}$
Alice→Bob:	W^S
Bob:	$K = ((W^S)^{1/S})^{R_B}$

This problem appears confined to this special case. In general there must be no way to create a "dictionary of exponents" corresponding to a fixed base, that maps each exponent to a candidate password.

The problem surfaces again when the verifier for the secondary exchange in B-SPEKE is $V = g^C$, and we may be tempted to use this V as the base of the primary exchange, with the same modulus. In this case we avoid the problem by using $V = h(g^C)$ to randomly distribute the primary base.

The base in our primary SPEKE exchange can thus be formed with $g = S^2$, and $S = h(g^C)$. Common household hash functions such as SHA-1 seem sufficient for *h*, although further analysis of this use of a hash function may be desired.

4.3. Bilateral key negotiation

Either method "A" or "B" can be used with any suitable primary strong-password scheme, but only if both parties contribute to the derived key. As noted in [BM93], these extensions are not suitable for schemes where one party unilaterally chooses the key, such as PK-EKE or a "secret public key" method [GLNS93]. An attacker who has stolen a verifier, can become a middleman in a unilateral key negotiation, and force the same value of K obtained from Alice to Bob, or vice versa. He then can masquerade as Alice by passing her proof of the password onto Bob.

The use of DH in the primary key exchange precludes this attack, since even a middleman with knowledge of S cannot negotiate the same key in two distinct sessions with Alice and Bob.

The "optimal direct authentication protocol" in [Gon95] is also described in a form where both Alice and Bob contribute to the key. In this protocol, Alice chooses a random public key V_A, and sends it along with k', her contribution, in a message symmetrically encrypted using the password to Bob. Bob uses V_A to seal a message for Alice containing, among other things, his contribution k. The session key is formed with the one-way hash $h(k, k')$.

Alice→Bob:	$na, E_S(V_A, k')$
Bob→Alice:	$\{B, A, nb, cb, k, E_S(na)\}_{V_A}$
Alice→Bob:	$E_{h(k, k')}(nb)$

But how they contribute to the key is just as important. The problem here is that Mary can get in the middle, and using her knowledge of S she can decrypt Alice's first message, replace V_A with her own public key V_M, and send $E_S(V_M, k')$ to Bob. Bob returns his message sealed with V_M. Mary unseals it, learns k, and reseals it for Alice with V_A. Now Mary can get out of the middle and read all subsequent traffic encrypted with $h(k, k')$.

A correction for this protocol is to omit k', and compute the session key as $h(k, V_A)$. If Mary doesn't replace V_A, she can't learn k. If she does replace V_A, Mary has the intractable problem of trying to find an *x* such that $h(x, V_A) = h(k, V_M)$.

A similar change can be made to PK-EKE or similar protocols to further add to our extended family.

4.4. Dual proof of K_1 and K_2

The proof step in B-SPEKE can use any well-known method for proving knowledge of a high-entropy secret, just as in A-EKE. But it is important that the proof carefully combine K_1 and K_2.

The value of K_2 cannot be sent to Bob, either in the clear, or alone in a one-way hashed form. Since K_2 is presumed to have low entropy, the secondary DH exchange could permit dictionary attack by an eavesdropper.

It is also important that the value of K_2 not be sent to Bob in a reversibly encrypted form using K_1 as a key. If K_2 is encrypted by K_1,

Alice→Bob: $E_{K_1}(K_2)$

then someone who has stolen the verifier can perform a middle-man attack on the primary exchange, decrypt K_2 from Alice's proof, and re-encrypt K_2 in a proof for Bob.

Furthermore, if K_1 is encrypted by K_2,

[3] Independently, at least Li Gong, Susan Langford, and I discovered this problem in the paper.

Alice→Bob: $E_{K_2}(K_1)$

a dictionary attack is possible by a eavesdropper who could use knowledge of all possible values for K_2 to determine K_1, and decrypt the resulting session.

Thus the proof function P must combine both K_1 and K_2 to preserve the following information-hiding properties:

- knowledge of K_2 must not disclose K_1.
- knowledge of K_1 must not disclose K_2, with a computation faster than exhaustive attack on K_2.

$P(K_1, K_2)$ can use an HMAC construction.

4.5. Salt

[BM93] discusses how "salt" benefits the exchange, which we'll reiterate. A salt factor in the password-to-verifier function insures that two different users' stored verifiers appear different even when the passwords are the same, and prevents a single dictionary of verifiers from being applicable to all users. Here are two possibilities for B-SPEKE, where g is a unique salt for each user:

1. Bob sends g to Alice after she identifies herself, and before the exchange.
2. Use *self-salting*, where g is a function of Alice's name, known to both parties.

It might appear dangerous that Bob sends salt to anyone who asks for it. But in a network protocol, salt isn't particularly useful without the rest of the verifier.

4.6. Other analysis

Note that even if Boris steals S and V, and poses as Bob, we don't want him to learn C from this protocol any easier than by a direct dictionary attack. If S=*h*(C), then the difficulty of determining C from S depends (in part) on the irreversibility of *h*. If S=*h*(V), the difficulty of obtaining C from S depends on discrete log. Both these problems are presumed sufficiently hard.

If Boris uses S to perform the primary SPEKE exchange with Alice, he has a chance to test the returned proof of $\{K_1, K_2\}$ corresponding to his chosen value of g^X. But since he already knows S and V, his job to perform a dictionary attack is made no easier.

We also note that it is worthwhile to insure that V is of large order, and that C is large enough to make g^C irreversible. For Z_p^*, where p = 2q+1, C should be greater than 1 and less than p-1. The function C = 2 + (*h*(password) mod p-3), does the trick.

5. Performance and implementation issues

Given that large integer arithmetic is expensive, and that extended methods apparently double the amount of computation, a natural question to ask is: How slow *is* it? The answer depends mostly on the size of the number field, and the nature of the group.

There's enough discussion of bit-lengths in the literature that we won't dwell on it. Depending on the security goals, opinions vary on the size of a suitable field for DH security, in Z_p^*, from several hundred bits to a thousand or two. For elliptic curve groups, it seems reasonable to allow the field size to be much smaller, due to the apparently increased difficulty of computing discrete logs [P1363]. There is also the issue of using short DH exponents, which has been covered in [vOW96, Jab96].

The relative performance of different DH groups in optimized software implementations was described in [SOO95] as running about 6 to 7 times faster in elliptic curves than a traditional Z_p^* version of comparable security. For example, performing a DH exponentiation on a 25 MHz SPARC IPC (roughly equal to a 66 MHz Intel 486) was ~ 900 msec., while the equivalent elliptic curve operation ran in ~ 125 msec. Our initial experiments with unoptimized implementations also showed performance improvement, although with less dramatic results. There are a number of ongoing efforts to implement high performance methods in EC and Z_p^*, and our analysis of the relative speeds is currently incomplete.

We'll also leave detailed discussion of the technical differences when using EC groups for another paper, except for one important note: DH-EKE is not an ideal candidate for elliptic curve methods, due to its strict requirements for no verifiable plain-text in the encrypted exponentials. To make it work seems to require a practical one-to-one mapping function of the points on a curve onto a continuous range of integers, which appears to be impossible. SPEKE does not use symmetric encryption, which makes it easier to use elliptic groups. Mapping the password to an arbitrary prime-order point on an elliptic curve is a straightforward procedure.

So, assuming the most efficient possible implementation of group operations, what else might be done to make any of these extended protocols efficient?

The total computational effort required for both “A” and “B” extensions, whether using an optimized digital signature method, or a Diffie-Hellman exchange, appears to be roughly the same. The dominant cost in both choices is due to three exponential calculations during a run of the protocol. (One of four DH calculations is done during password setup.) But there remain several factors that might affect the *perceived response time* from the user's perspective:

- single- vs. multi-threaded client & server
- asynchronous vs. synchronous RPC messages
- for “A”, which digital signature scheme is used
- for “B”, different groups for each stage
- relative CPU speed of client & server
- speed of the communication channel

The simplistic form of B-SPEKE (figure 1) performs the “B” extension with two messages added after a

completed mutual SPEKE exchange. But the added proof of $\{K_1, K_2\}$ eliminates the need for the earlier proof of K_1 from Alice to Bob. By deleting the redundant proof and consolidating the messages, we get the protocol shown in figure 2.

B-SPEKE does not necessarily require more rounds of communication than SPEKE. If self-salting is used (§ 4.5), Alice can identify herself in the same message containing her SPEKE challenge Q_A. This allows the protocol to be further reduced to three messages, since Alice no longer has to obtain g from Bob.

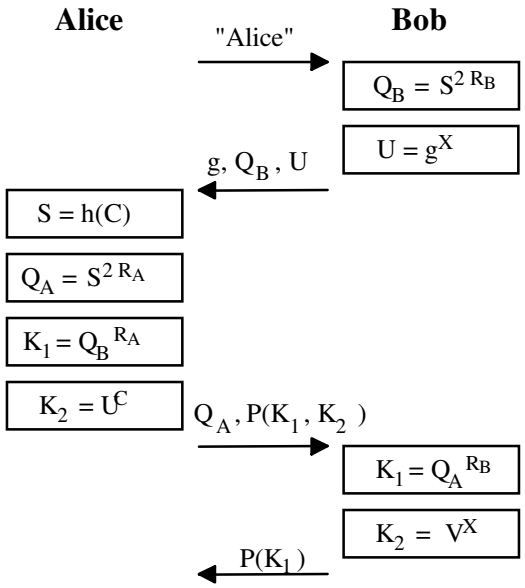

Figure 2. B-SPEKE, combined

Reducing messages is a concern for slow networks, but in fast networks a larger concern may be to reduce elapsed running time. Two ways to achieve this are parallel computation, and pre-computation of exponential values.

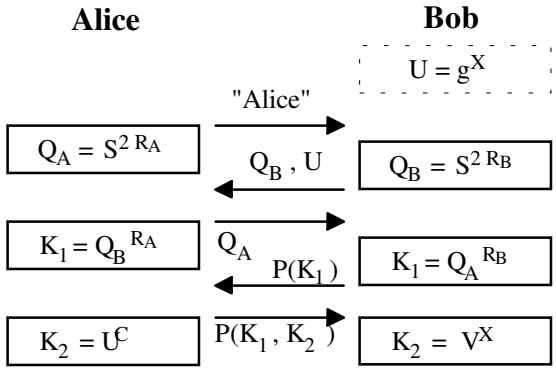

Figure 3. B-SPEKE — parallel computation

Figure 3 shows an arrangement that reduces the elapsed running time. Assuming that the time required for each exponentiation is z, and that communication time is negligible, the running time of the protocol is reduced from 7z to 3z. Part of the gain is due to the pre-computation of the pair (X, U) by Bob.

A cache of short-lived pre-computed values at the host seems reasonable to increase the efficiency of handling transient bursts of traffic. However, prolonged storage of pre-computed ephemeral data may increase the risk of disclosure. This risk must be evaluated with respect to the client and server's operating environment. The use of a fast hash function S = *h*(C) may be a concern for those truly fear that the verifier database may be stolen. In this case, one may want to deliberately insure that the function for S is slow (perhaps S=*h*(V)), to make dictionary attacks more expensive.

The increase in the number of messages in figure 3 from 4 to 5 is needed to keep both parties busy crunching numbers. But a protocol with an odd number of messages may also not fit with a standard request/reply communication model, which may force an empty 6th reply message. And the addition of two messages will result in a slower protocol if the average time to pass a message is greater than 2z. Also, note that the client keeps busy computing while the server is acting on the request, which either implies a multi-threaded client, or an asynchronous communication model.

When considering implementations of A-EKE or A-SPEKE, the most suitable signature operations seem to be ones where an arbitrary value can serve as the private key, which makes the exchange simpler, if not faster. In general, the large family of discrete log signature schemes seem suitable, and in these schemes at least one exponentiation is required for signing, and two for verifying. To eliminate an additional step of inversion, we could use a signature method such as the Nyberg and Rueppel variation [P1363].

Maximal use of pre-computation and parallel computation also affects the relative performance of "A" and "B". In DH-EKE, the calculation of the random exponentials can be done in advance on both the client and the server, and in the case of B-EKE, the challenge exponential can also be pre-computed on the server.

In figure 4 we show an optimized B-EKE, and in figure 5 an optimized A-EKE using a Nyberg-Rueppel method. Ignoring the pre-computation of Q_A, Q_B, and U_C, the A-EKE method runs in 4z, while B-EKE runs in 2z.

The ability to pre-compute both Q_A and Q_B might be reasonable for some DH-EKE systems, but it seems inappropriate for SPEKE, which determines these values based on each user's password.

Also, note that the protocols in figures 4 and 5 have Bob processing data after a reply is sent, and that there are two replies sent in sequence from Bob, with no

intervening request from Alice. This implies either a multi-threaded server implementation, or asynchronous message passing.

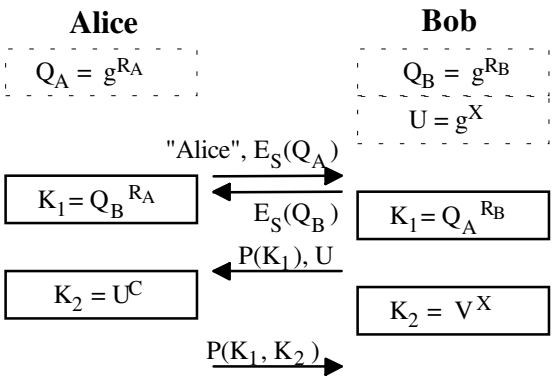

Figure 4. B-EKE with pre-computation

Even without pre-computation, if we assume ideal infinitely fast network, the time to complete a full A-EKE method is 5z, while B-EKE can run in 4z. This also applies to A-SPEKE vs. B-SPEKE. This advantage of "B" over "A" is due to the symmetry of the secondary DH exchange.

All considered, assuming use of optimized elliptic curve methods wherever possible, we estimate that a B-SPEKE implementation may be significantly faster than a comparable A-EKE method, in a fast network. In a very slow network, the cost of the two round-trips would make either computation negligible and tend to equalize perceived response time. In any case, either class of method requires a small fraction of a second to run on modern desktop computers.

In general, extended methods require no extra messages over their un-extended counterpart -- a two-message form can still negotiate an authenticated key, albeit without explicit verification of that key.

6. Benefits of "A" and "B" extensions

This section summarizes the benefits of extended strong methods. Assuming that the password C is sufficiently small, and that Alice authenticates Bob from his knowledge of the verifier S, [BM92] observes that two attacks are unavoidable for a thief who obtains S:

1. He can perform a dictionary attack to learn C.
2. He can masquerade as Bob to Alice.

It is reasonable to ask whether these threats are so overwhelming that the further protection afforded by the "A" and "B" extensions is irrelevant. With this in mind, we identify reasons why the further protection is useful.

The ability to succeed with a dictionary attack relates directly to the strength of C. When C is large enough, nobody can obtain C from S. Considering that a significant fraction of the user community might choose a "good" password, they deserve the extra protection.

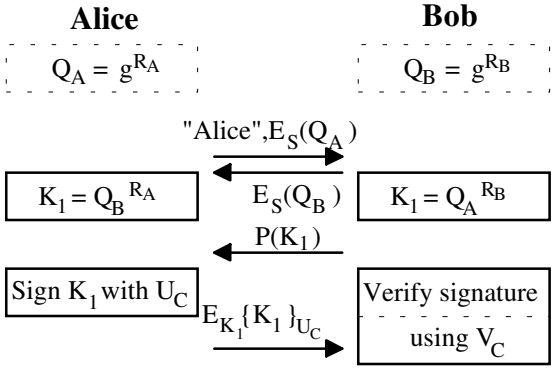

Figure 5. A-EKE with pre-computation

In special cases, either extension may make it appropriate for a large entropy C to serve as a common password for multiple hosts, for example when there is another separate mechanism in place for the authentication of the host to the user. Without using an "A" or "B" extension, this case allows one host to impersonate the user to another, which may be an unacceptable risk.

We also observe that knowledge of S, without knowledge of C, does not permit a thief to become a man-in-the-middle of a valid conversation between Alice and Bob. Without detailed knowledge of a valid conversation, he may be unable to successfully pose as Alice or Bob.

Extended strong password methods provide security that scales appropriately with the quality of the password. User-selected passwords have unpredictable entropy. They can be empirically qualified as "good" if they don't belong to a set of "bad passwords", but there's no guarantee that an attacker won't find a better dictionary. Even machine-generated passwords of known entropy typically have less-than-large entropy, to keep the password memorizable. These methods do the best that can be done for passwords of uncertain quality, and they handle both large and small passwords appropriately. The user is not unnecessarily exposed in cases of theft of the host database, on-line attacks can be detected, and arbitrarily powerful eavesdroppers learn nothing about even tiny passwords.

A typical use for these methods is in a single-sign-on system where user authentication incorporates a password. Passwords deserve special protection when there is a chance for their use in accessing multiple systems. Furthermore, since people tend to re-use passwords for multiple purposes, against all good advice to the contrary, strong password protection seems warranted even for "low-security" applications. Strong

methods are essential for this critical link between the user and the system.

Even with wide-spread deployment of public keys, memorized secrets retain a unique and important role in authentication. Systems based only on *stored* keys require secure long-term storage on both ends of the connection, or when relying on a certificate hierarchy, long-term indirect trust relationships between a multitude of parties. Strong password methods can reduce these dependencies, add an extra independent factor for authentication, and provide a simple and familiar way to initiate secure electronic conversations.

7. Summary

Several extended authentication protocols have been described for strong password authentication. These methods approach the limits for protecting small passwords without requiring extra keys. Discussion of implementation issues show that these methods are superior to traditional password methods, and can run faster than some previously described strong methods, by using elliptic curve groups, pre-computation, and by balancing the computational load.

8. References

Some of the references here are available on the web at: http://world.std.com/~dpj/

[BM92] S. Bellovin and M. Merritt, "Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks", *Proceedings of the I.E.E.E. Symposium on Research in Security and Privacy*, Oakland, May 1992.

[BM93] S. Bellovin and M. Merritt, "Augmented Encrypted Key Exchange: a Password-Based Protocol Secure Against Dictionary Attacks and Password File Compromise", *AT&T Bell Laboratories* (c. 1993).

[BM95] S. Bellovin and M. Merritt, "Cryptographic Protocol for Remote Authentication", U.S. Patent #5,440,635, August 8, 1995.

[GLNS93] L. Gong, M. Lomas, R. Needham, & J. Saltzer, "Protecting Poorly Chosen Secrets from Guessing Attacks", I.E.E.E. Journal on Selected Areas in Communications, Vol. 11, No. 5, June 1993, pp. 648-656.

[Gon95] L. Gong, "Optimal Authentication Protocols Resistant to Password Guessing Attacks", Proceedings of the 8th IEEE Computer Security Foundations Workshop, County Kerry, Ireland, June 1995, pp. 24-29.

[HDM77] M. Hellman, W. Diffie and R. Merkle, "Cryptographic Apparatus and Method", U.S. Patent #4,200,770, April 29, 1980.

[Jab96] D. Jablon, "Strong Password-Only Authenticated Key Exchange", *Computer Communication Review*, vol. 26, no. 5, pp. 5-26, October 1996.

[Jas96] B. Jaspan, "Dual-workfactor Encrypted Key Exchange: Efficiently Preventing Password Chaining and Dictionary Attacks", *USENIX Security Conference Proceedings*, July 1996.

[Luc97] S. Lucks, "Open Key Exchange: How to Defeat Dictionary Attacks Without Encrypting Public Keys", Proceedings of the Security Protocol Workshop '97, Springer-Verlag, April 7-9, 1997.

[MW96] U. Maurer and S. Wolf, "Diffie-Hellman Oracles", *CRYPTO 96*, LNCS 1109, Springer-Verlag, 1996, pp. 268-282.

[NIST94] National Institute of Standards and Technology, NIST FIPS PUB 186, "Digital Signature Standard", U.S. Department of Commerce, May 1994.

[Pat97] S. Patel, "Number Theoretic Attacks on Secure Password Schemes", Proceedings of the 1997 IEEE Symposium on Security and Privacy, May 5-7, 1997.

[P1363] IEEE P1363 working group, "IEEE P1363 Working Draft -- Standards for Public-key Cryptography", This document is currently available at: http://stdsbbs.ieee.org/1363

[SOO95] R. Schroeppel, H. Orman, S. O'Malley, "Fast Key Exchange with Elliptic Curve Systems", TR 95 03, Department of Computer Science, The University of Arizona, March 31, 1995.

[STW95] M. Steiner, G. Tsudik, and M. Waidner, "Refinement and Extension of Encrypted Key Exchange", *Operating Systems Review*, vol. 29, Iss. 3, pp. 22-30 (July 1995).

[vOW96] P. vanOorschot, M. Wiener, "On Diffie-Hellman Key Agreement with Short Exponents", *Proceedings of Eurocrypt '96,* Springer-Verlag LNCS, May 1996.

* * *