Analysis of the Efficient Short-Password Key Exchange Protocol

Scott Lulovics

March 21, 2006

1 ESP-KE Specification

1.1 Overview

The Efficient Short-Password Key Exchange (ESP-KE) Protocol was proposed by Michael Scott in 2001 for using a small, easily-remembered short password to facilitate secure key exchanges. Since both parties (and theoretically only those two parties) share the password, an attacker trying a Man-in-the-Middle cannot succeed without correctly guessing the password. Even though the password is small, typically around 16-bits, the attacker cannot perform a guessing attack without revealing his presence to the two parties.

1.2 Protocol Setup

The global domain parameters are α, β, p and q, where p is a prime (of around 1024 bits), q is a prime divisor of $p-1$ (of around 160 bits), and α and β are unrelated generators of the prime order sub-group of order q. The goal of the protocol is to establish a session key k based on nonces and the shared password. The core of the protocol is a basic Diffie-Hellman key exchange, but there are modifications made for the password. At the end of the protocol, both parties (assuming they're honest) are in possession of a session key $k = \alpha^{ab}$, just like in Diffie-Hellman.

1.3 Protocol Specification

Alice		**Bob**
Choose a randomly from $[1, p-1]$		Choose b randomly from $[1, p-1]$.
$A = \alpha^a / \beta^P$		$B = \alpha^b$
	$A \rightarrow \quad \leftarrow B$	
$k = B^a$		If $A = 0:$ Abort
		$k = (A \cdot \beta^P)^b$
		$M_B = H(0, k, P)$
	$\leftarrow M_B$	
If $H(0, k, P) \neq M_B:$ Abort		
$M_A = H(1, k, P)$		
	$M_A \rightarrow$	
		If $H(1, k, P) \neq M_A:$ Abort

1.4 Guessing Attacks

In addition to modeling the protocol, the major feature I looked at was guessing attacks, of which there are two varieties. Online guessing attacks are when an adversary simply guesses a value and an honest participant tells the intruder whether their guess is correct. These attacks, while valid, are easily defeated with such measures as allowing only three incorrect guesses.

Offline guessing attacks of are much more concern. These attacks are where an adversary obtains enough information to go ”offline” and perform a brute-force search for the correct value. Such attacks, if found, represent a serious security breach. Unlike an online guessing attack, neither participant will be aware that the password has been compromised, or even that an attempt is being made.

2 Modeling ESP-KE

2.1 Constraints

In order to check for security properties in ESP-KE I used a constraint solver written in Prolog. The model itself was written in CAPSL, which was then compiled down to Prolog bundles and a strand representation which the constraint solver program could understand. Since the constraint solver was not respectful of some CAPSL's constructs, in the end my use of it was somewhat limited.

2.2 CAPSL

My choice to model this protocol was the Common Authentication Protocol Specification Language (CAPSL), a high-level language which aims to be interoperable with many protocol analysis tools. Sadly, the tools which I wanted to use were not correctly integrated with CAPSL, which forced many redesigns of the model. In the end, the model-checking code I used produced no valuable output, other than to confirm the secrecy and authentication properties which existed in the underlying Diffie-Hellman exchange.

What I ended up modeling in CAPSL was a simplified version of ESP-KE. In order to get around the constraint solver's lack of knowledge of modular arithmetic, I most of the protocol was abstracted into nonces, encryption and decryption.

In the following CAPSL snippet, $\{m\}k$ is the encryption of m under the key k, $[a, b]$ is the concatenation of a and b, and lastly $sha(\cdot)$ is a secure hash function.

$$\begin{aligned}
A \rightarrow B: &\quad \{N_a\}P; \\
B \rightarrow A: &\quad [N_b, sha([0, [N_a, N_b], P])]; \\
A \rightarrow B: &\quad sha([1, [N_a, N_b], P]);
\end{aligned}$$

2.3 Modeling Results

Due the to the nature of Prolog, it was beyond my abilities to specify exactly what the constraint solver should find when looking for a guessing attack. Between this and the inability to model the mathematics involved, the model was of little use.

3 Analysis

3.1 Looking Guessing Attacks

Since I determined early on that the actual model would be of little use when looking for guessing attacks, I devoted a portion of my time towards looking at the mathematics behind the protocol and analyzing it by hand.

3.2 Trivial Online Guessing Attack

The only attack I found by hand was an online-guessing attack that exploits some of the mathematical properties. While the attack is trivial, and wouldn't ever be used in practice, it does show that addition checks should be done to ensure for valid input. The attack is as follows:

Alice		**Evil Bob**
Choose a randomly from $[1, p-1]$		
$A = \alpha^a / \beta^P$		$B = 1$
	$A \rightarrow$ $\quad \leftarrow B$	
$k = B^a = 1^a = 1$		
		$M_B = H(0, 1, Guess(P))$
	$\leftarrow M_B$	
If $H(0, k = 1, P) \neq M_B$: Abort		
$M_A = H(1, k = 1, P)$		
	$M_A \rightarrow$	
		If Alice responded, the guess was correct.

Alice should check that the B sent from Bob is not 1, because Bob can predict the result of $k = 1^a$. Usually, in order to make a guess of the password (and only the password), Bob would have to first correctly guess k, but not in this attack. If Alice wasn't counting the incorrect guesses, Bob could brute-force the password in $2^{|P|}$ interactions, where $|P|$ is around 16 bits.

3.3 Password Verifiers

Gavin Lowe formalized guessing attacks in a paper in 2001:

> A guessing attack consists of an intruder guessing a value g, and then verifying that guess in some way. The verification will be by the intruder using g to produce a value v which we call the verifier

Basically, then, looking for a guessing attack boils down to looking for verifiers which an intruder can obtain to validate his guesses offline. This makes sense - an offline attacker can generate every possible n-bit password, but it needs some way to tell whether each one is correct.

Since the value we want to protect is the password, I looked at all the messages transmitted during the protocol to find any possible verifiers. Really only four messages are sent:

- $A = \alpha^a / \beta^P$
- $B = \alpha^b$
- $M_A = H(1, k, P)$
- $M_B = H(0, k, P)$

B is useless, since it contains no information about P in any form. M_A and M_B can be seen as variants of $M = H(k, P)$. So there are only two possible verifiers for the password to worry about - A and M.

To use A as a verifier, we need to compute some value of A' from our guess to compare against A. Since A' would have to be of the form $\alpha^a / \beta^{Guess(P)}$, we need to find a to compute A'. However, since $a = log_\alpha(\beta^P / A)$ we need to know $log_\alpha(\beta)$ - and if we were able to find it we would break the assumption that discrete logs are hard.

A similar problem exists when using M as a verifier. To use it, we need to compute some value M' from our guess to compare against M. Since $M' = H(k, P)$ we need to find the correct k to plug into the hash function. Well, since $k = \alpha^{ab}$ finding k would be solving the Computational Diffie-Hellmen assumption (i.e. given g^a and g^b it has hard to find g^{ab}).

4 Conclusions

The ESP-KE Protocol is similar enough to the basic Diffie-Hellman Key Exchange so that the security properties Diffie-Hellman provide carry through. Furthermore, since there aren't any valid verifiers an attacker can gain, there are no offline dictionary attacks possible against this protocol. The only improvement that would benefit security would be to have Alice check the input from Bob, to ensure its not 1, or some other value which makes it easier to guess B^a.