Information Leakage in Encrypted Key Exchange

Sarvar Patel. Bellcore, sarvar@bellcore.com.

Abstract: *Encrypted Key Exchange(EKE) protocol promises to protect against off-line dictionary attacks on passwords, as long as we are careful about avoiding information leakage. EKE presents a way to avoid leakage. We show this method to be ineffective and recommend another method. Furthermore we present a general framework to transfer under encryption difficult numbers without leaking information about the password. As an example, we show how to transfer a prime number without leaking information.*

Introduction: A random key along with a secure protocol provides the best method of authentication over an insecure network. Unfortunately, humans loathe random keys. One of the greatest network threats is also one of the oldest: Bad passwords allow an eavesdropper to mount an off-line dictionary attack. For example, when an eavesdropper listens for a challenge R and the password encrypted response $\{R\}_{PW}$, then, off-line, the eavesdropper encrypts R with all passwords in the dictionary and compares the result with $\{R\}_{PW}$. If one of the passwords in the dictionary matches, then the secret password has been discovered.

Bellovin and Merrit[1] have presented interesting and counter-intuitive protocols which are secure not only against active attacks, but also against off-line dictionary attacks. Information leakage, which allows one to eliminate password choices, is tricky to avoid in these protocols. The authors suggest a way of random padding to block information leakage. We show this method to be ineffective and the problem of leakage to be more subtle. Unless the leakage problem is addressed, these protocols have limited use as protection against off-line dictionary attacks. We recommend a way not of eliminating leakage, but of keeping it in check. In fact, we can reduce leakage to an extent that off-line attacks are no better than on-line ones. We also present ways to transfer under encryption difficult numbers, like prime numbers, without leaking any information about the password used to encrypt the prime. We also present a general framework where numbers of particular characteristics can be transferred without leakage.

EKE uses public-key and secret-key cryptography allowing a user and a server, sharing a password, to authenticate and exchange information over an insecure network. EKE has variants using RSA, Diffie-Hellman Exchange, and ElGamal. The former two are sketched in Figure 1. In step 1 of RSA scheme, A generates a random public key E_A and the corresponding private key D_A. A transfers the public key to B by encrypting it in a secret-key cryptosystem using the password PW. B responds by generating a random session key R and encrypts it using the public key, which in turn is further encrypted using the password PW. The rest of the steps are standard exchanges of challenges C_A and C_B to authenticate session key R. An eavesdropper listening to the exchanges has no easy way to validate password guesses PW'. The only way to validate the guesses is to break the public key system.

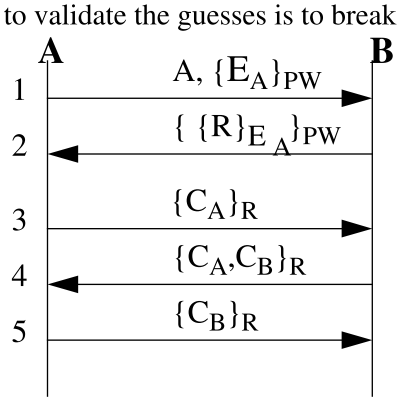

Figure 1A: EKE using RSA

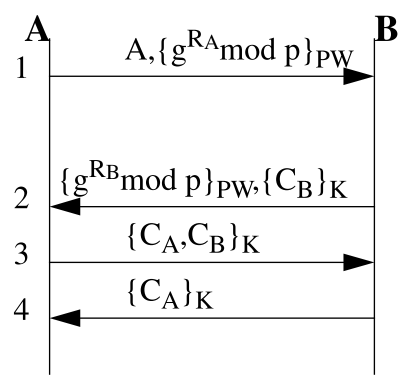

Figure 1B: EKE using Diffie-Hellman

Similarly, key exchange in Diffie-Hellman is encrypted using password PW to protect against a 'man in he middle' attack. Also an eavesdropper cannot validate password guesses because R_A and R_B are random hence g^{R_A}(mod p) and g^{R_B}(mod p) will also appear random. Furthermore, even if they are correctly guessed, there is no way to calculate the session key K=$g^{R_A R_B}$(mod p). However, user A can calculate session key K by raising the received g^{R_B}(mod p) to known R_A. In the rest of the steps, challenges are sent to validate session key K.

Information Leakage: Suppose we are using password PW to encrypt numbers mod a prime p, but the block size of the system is larger than p. Now an intruder can use password guesses PW' from a dictionary and try to decrypt for example $\{g^{R_A}(\text{mod p})\}_{PW}$. If the decryption results in a number greater than p then that password guess PW' can be rejected. Thus some information is leaking, similarly all passwords in the dictionary can be tried and those resulting in decryption greater than p will be rejected. Then more encryptions from other sessions can be used to narrow the viable passwords in the dictionary down to one.

Even if the block size does not have more bits than that require to code a number of length p, information leakage is still possible. Lets say n bits are the minimum required to code p, that leaves us with 2^n-p numbers greater than p, and all password guesses whose decryptions yield these numbers can be rejected.

Random Padding: The authors in [1] suggest a way to deal with both problems by randomly adding multiples of p to the number, so that all numbers are possible, and the receiver can always remove the random adding by performing mod p. In Diffie-Hellman key exchange the random adding should make no difference in calculating the session key. If m is the size of the block then 2^m numbers are possible. Let $x = \left\lfloor \frac{2^m}{p} \right\rfloor$. If input is less than 2^m mod p then pick a random $0 \le j \le x$ and add jp to input. If input is between 2^m mod p and p then pick random j $0 \le j \le x - 1$ and add jp to input.

Leakage in Random Padding: Unfortunately, there is a weakness in this method of random padding which can under certain conditions allow significant leakage. Not only must the random padding method make all numbers possible under decryption, but it must also make the probability of all numbers the same. Unmodified, the above method does not guarantee equal distribution. It is susceptible to a modified off-line attack. First we collect encryptions from many sessions, then for each password guess from the dictionary, we keep a coarse histogram of the occurrence of ranges of numbers. So instead of rejecting a subset of passwords after each session encryption, we now count occurrences of numbers in some ranges. Then when we see that some distribution are not uniformly distributed, then we try an on-line attack for that password guess. Eventually we will be successful, but much sooner than a brute force on-line attack.

We will present an example to motivate, but because of space constraints the general case with details will be in the full paper. Assume that only one interval of p completely fits in 2^m. At the two end intervals a) 0 to (2^mmod p) and b) p to 2^m, the probability distribution due to random padding is not uniform. Assuming uniformity we would expect 2(2^mmod p)/2^m numbers to fall in those intervals. However, due to random padding, we now expect (2^mmod p)/p numbers to fall in those intervals. Assuming 2^mmod p is about p/2, then we expect randomly about 2/3 of the numbers to fall in the two ends. With random padding we expect only 1/2 of the numbers. Modified off-line attack can easily exploit such odds.

Another Method: Picking prime number p arbitrarily close to 2^m is another method of reducing information leakage. In this method, while picking prime p, we set certain number of high bits to one. Also if we have a larger block, we set the high bits to one and then pick a prime by randomly trying lower bits till one passes a primality test. We will pick k such bits, with higher k giving us less information leakage. The only way to invalidate a password guess is if the decryption of a session encryption results in 1s at all high k bits and the number formed by the remaining m-k bits is greater than the number formed by the lower m-k bits in prime p. The chance of that happening is less than $1/2^k$. We would like to pick k such that to mount an off-line attack an intruder would need to hear or carryout as many sessions as in mounting a direct on-line attack. We will call this the off-line/on-line threshold bits, T. In calculating T, we assume the best case for the intruder that $1/2^k N_{PW}$ are rejected then at next iteration $(1-1/2^k)N_{PW}$ are left where N_{PW} are the number of passwords in the dictionary. Since we want to terminate after i iterations when only one password is left we have the equation $1 = \left(1 - \frac{1}{2^k}\right)^i N_{PW}$, Since we want to require N_{PW} iterations, the same as in off-line attack, we substitute N_{PW} for i and solve for k or T bits. $T = \log\left(\frac{1}{1 - 2^{-\frac{(\log N_{PW})}{N_{PW}}}}\right)$, For example N_{PW} =1000,000, The high T=16 bits have to be 1, to guarantee the same number of iterations from an off-line attack as required by an on-line attack.

Transferring special numbers under encryption. The authors of EKE[1] assume there is no way to encrypt special numbers like prime without leaking some information. We present a method which encrypts primes without any leakage, furthermore we show how in general to tackle special numbers. In order to encrypt a prime p, user A should first find the next higher prime, p', by primality testing, and then A should pick a random number j, $p \le j \le p' - 1$ and encrypt and transmit j. User B upon receiving encrypted j will decrypt it with the password PW and then will descend starting with j until it finds a prime. Now when a intruder tries to decrypt with a guessed password, he is no wiser because all numbers have equal probability of occurring. Since primes are distributed by 1/logN, so it is possible to find them in reasonable time.

In general, many special numbers can be encrypted by this method. First we need a test to verify whether a number satisfies the special predicate. Then we have to find the next number which also satisfies the predicate. Then we pick a random number from the range starting with the current predicate and ending right before the next predicate. For example, even when we try to hide odd numbers we basically follow the same scheme. First we find the next odd number, then we randomly pick the current number or the next even number because after that we have the next odd number. However, there may be some special numbers, like RSA number n=pq, for which there are no good test like for odd and prime numbers.

Conclusion We have shown how information leakage is possible in the scheme outlined by authors of EKE. Instead we propose another way of avoiding leakage to the point that off-line attacks become as burdensome as on-line attacks. Finally we show how apparently untransferable numbers, like primes, can be encrypted without leaking information.

[1] S. Bellovin, and M.Merritt, “Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks” *Proc. IEEE Computer Society Symposium on Research in Security and Privacy*, 5/92.