CSE 590TU Assignment #4 – Protocols I

Due at the beginning of class on January 31, 2006

1. **Speeding up SSL.** When SSL was first designed in the mid-1990s, one of the cited benefits, when used with RSA, was that the computational burden of the protocol was placed mostly on the server. This was due to the fact that the computationally-intensive parts of the protocol are the modular exponentiations for RSA encryption on the client and decryption on the server, and servers typically used public keys with a short encryption exponent of $e = 2^{16} + 1$.

   Question 1(a): Compute the relative cost of RSA encryption to RSA decryption in SSL in terms of modular multiplications mod n. The server's public key is (n, e), where n is a 1024-bit composite, n = p*q (p, q 512-bit primes), and $e = 2^{16}+1$. The server's private decryption exponent is d where $ed \equiv 1 \bmod (p-1)(q-1)$. Assume for this problem that half the bits in d are 1's.

   Over time, as clients have gotten faster relative to servers, this tradeoff has become less useful for SSL server operators. Variants of RSA have been proposed that alter the balance to be more favorable to servers. In one "Rebalanced RSA" variant, the server chooses the decryption exponent d first such that $d = r_1 \bmod p-1$ and $d = r_2 \bmod q-1$ where r_1 and r_2 are relatively small (say 160-bit) values. The server then computes e such that $ed \equiv 1 \bmod (p-1)(q-1)$ still holds; e will be a "full-size" value (of the same size as n) with roughly half its bits 1's. Encryption happens identically as before, but the server can decrypt the ciphertext C faster by computing $C^{r1} \bmod p$ and $C^{r2} \bmod q$ and then using the Chinese Remainder Theorem to compute C^d without doing any more exponentiations.

   Question 1(b): Compute the relative cost of RSA encryption to RSA decryption in the Rebalanced RSA case for $|n| = 1024$, $|r_1| = |r_2| = 160$, again in terms of modular multiplications. What's the speedup for a server compared to the "regular" RSA in Question 1(a)?

2. **Calculating the cost of IPSEC.** In this problem we're going to investigate the overhead of adding IPSEC encryption to network communications. Assume that we have a client computer C sending data to a server S over an IPSEC channel. C sends data to S in packets of varying length; for each session S and C have to perform an IKE key establishment once to agree on a symmetric encryption key, and then perform repeated symmetric encryptions until the entire packet is encrypted. Assume the following performance characteristics for C's encryption capabilities:
   a. C can perform IKE key establishment with S in 25,000 μs to derive a symmetric encryption key for the session.
   b. C can perform a single symmetric encryption operation on a 16-byte plaintext in 0.25 μs.

Question 2(a): If the average packet of data sent from C to S is 1KB (1024 bytes) in length, what's the maximum bandwidth that can be achieved between C and S?

Question 2(b): Now assume the average packet is 100KB; what's the maximum achievable bandwidth? What's the maximum bandwidth in the limiting case (i.e. one persistent session with an infinite-length packet to be sent)?

3. **Defending against KDC eavesdropping in Kerberos.** Let KDC, C and S be, respectively, a Key Distribution Center (including TGS functionality), Client and Server within a single Kerberos realm. Assume that C carries out the Kerberos protocol with KDC and S and that at the end of the protocol C and S share session key $K_{C,S}$. Because the KDC/TGS generated $K_{C,S}$ for C and S, the KDC could passively eavesdrop on the encrypted conversation between C and S.

   Question 3(a): Assuming the KDC is passive and can only watch the traffic between C and S, show how C and S can leverage $K_{C,S}$ to securely establish another shared secret K' known only to C and S.

   Question 3(b): Now assume the KDC is an "active" eavesdropper and can intercept and modify traffic between C and S. Does the protocol you designed in the previous question still provide secure communication between C and S? Why or why not?

   Extra Credit Question 3(c): Now assume C and S are jointly members of two independent Kerberos realms at the same time, centered on KDC1 and KDC2, yielding independent session keys $K_{C,S,1}$ and $K_{C,S,2}$ between C and S. Assuming the KDC1 and KDC2 do not collude with each other, devise a protocol between C and S leveraging $K_{C,S,1}$ and $K_{C,S,2}$ to create an encrypted channel that is secure against active eavesdropping from both KDC1 and KDC2.

   [Hint: you might find a keyed hash function useful in designing your protocols. A keyed hash function H(x,k) is a hash function that mixes a secret key k in with the process of hashing some public data x. Only a party in possession of both the secret k and the public data x can compute H(x,k). For example, HMAC-SHA1 is a very common keyed hash function in use today that uses the SHA1 hash internally to mix a secret key k (of any length) with data (of any length) to create a 160-bit keyed hash value.]