Homework #4 Solutions

Brian A. LaMacchia
bal@cs.washington.edu
bal@microsoft.com

Question 1a

- **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 ≡ 1 mod (p-1)(q-1). Assume for this problem that half the bits in d are 1's and |d| = 1024.**

Question 1a

- Recall Josh's description of how to multiply fast
  - Follow the bits in the exponent (high to low, ignoring the high bit since it's always 1), a 0 is a square and a 1 is a square followed by a side multiply.
  - So the cost of exponentiating is determined by the size of the exponent and the number of 1 bits it has.
- The cost of encrypting (exponentiating to e) is thus going to involve 16 squares and 1 side multiply, since $|e| = 17$ and has only the high & low bit set.
- The cost of decrypting (exponentiating to d) is going to involve 1023 squares and 511 side multiplies (because $|d| = 1024$ and half the bits in d are 1s).

Question 1b - Rebalanced RSA

- n = pq, |p| = |q| = 512, p, q prime
- Server chooses the decryption exponent d first such that
  - $d \equiv r1 \bmod p-1$, $d \equiv r2 \bmod q-1$, |r1| = |r2| = 160
  - Assume half the bits in r1 & r2 are 1s
- The server then computes e such that $ed \equiv 1 \bmod (p-1)(q-1)$.
  - |e| = 1024, assume half the bits in e are 1s
- Encryption $E(X) = X^e \bmod n$
- Decryption D(X) is now done by computing $X^{r1} \bmod p$, $X^{r2} \bmod q$, and using CRT to construct $X^d \bmod n$.

Question 1b

- Question 1(b): Compute the relative cost of RSA encryption to RSA decryption in the Rebalanced RSA case for $|n| = 1024$, $|r1| = |r2| = 160$, again in terms of modular multiplications in the exponentiations.
- What's the speedup for a server compared to the "regular" RSA in Question 1(a)?

Question 1b

- We proceed as in 1(a), looking at the sizes and number of 1 bits in the various exponents.
- Encrypting: e is now "full size", so $|e| = 1024$ and half the bits in e are 1s
  - Computing X^e mod n will involve 1023 squares and 511 side multiplies
- Decrypting: we need to compute X^{r1} mod p and X^{r2} mod q.
  - $|r_1| = |r_2| = 160$ (by definition), and each has 80 1 bits.
  - Each exponentiation will involve 159 squares and 79 side multiplies. So together, the server has to perform 318 squares and 158 side multiplies.

Question 1b

- Just comparing the number of exponentiations, we've reduced the server's workload from 1023 squares & 511 side multiplies to 318 squares and 158 side multiplies, which is about a 3.2X speedup for the server.
  - Plus, the server gets to do these calculations mod p and mod q, which means it's dealing with smaller numbers.
- The client is a lot slower.
  - 1023 squares and 511 side multiplies vs. 16 squares and 1 side multiply in 1(a).

Question 2 – IPSEC cost

- **C sends data to S in packets of varying length; for each packet 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:**
  - **C can perform IKE key establishment with S in 25,000 μs to derive a symmetric encryption key for the session.**
  - **C can perform a single symmetric encryption operation on a 16-byte plaintext in 0.25 μs.**

Question 2a

- 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?
- Encrypting each packet requires one IKE key establishment and some number of symmetric encryption operations.
  - Bandwidth = bits/second

Question 2a - Solution

- For a 1K packet, we need one IKE + (1K/16) block encryptions
  - The IKE takes 25000 μs
  - The block encryptions take 64*.25 μs = 16 μs
  - The total time is 25016 μs, and we've processed 1K bytes in that time, or about 40933 bytes/sec.

Question 2a - Solution

- **But wait, what about the additional bytes ESP adds?**
  - No effect on bandwidth limits from encryption, but ESP does add overhead, so it's fair to look at that cost too.
- **We don't know the actual wire speed, but we know that ESP adds some bytes to each packet.**
  - Assume 56 bytes added to each packet
  - So we have to send 1024+56 = 1080 bytes to actually transmit 1024 data bytes.
    - 1080/1024 = 1024/X → X = 970.9 bytes
    - 970.9 bytes in 25016 μs = 38811 bytes/sec

Question 2b

- 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)?

Question 2b - Solution

- For a 100K packet, we need one IKE + (1K/16) block encryptions
  - The IKE takes 25000 μs, 100K = 102400 bytes
  - (100K/16) block encryptions take (100K/16)*.25 μs = 6400*.25 μs = 1600 μs
  - Total time: 26600 μs, or about 3,849,624 bytes/sec.
  - Adding in 56 header bytes, we have to send 102456 bytes instead of 102400 bytes.
    - 102456/102400 = 102400/X → X = 102344 bytes
    - 102344 bytes in 26600 μs = 3,847,518 bytes/sec
- In the limit, the limiting factor is that it takes .25μs to encrypt 16 bytes, so that's an upper limit of 64,000,000 bytes/sec.

Question 3 – KDC Eavesdropping

- KDC, S and C are members of the same Kerberos realm
- When C obtains a ticket for S, KDC assigns ephemeral key $K_{C,S}$
- KDC knows $K_{C,S}$ and can eavesdrop on all traffic protected with $K_{C,S}$
- Assume KDC is passive (i.e. KDC cannot modify messages it sees on the wire).
- Question 3(a): Show how C and S can establish key K′ known only to them

Question 3a

- Key establishment is easy, right? All we have to do is have C and S perform a Diffie-Hellman key exchange.

Question 3a

- Key establishment is easy, right? All we have to do is have C and S perform a Diffie-Hellman key exchange.
- But how do C and S know they're talking to each other while doing so?

Question 3a

- Key establishment is easy, right? All we have to do is have C and S perform a Diffie-Hellman key exchange.
- But how do C and S know they're talking to each other while doing so?
- One of the benefits of Kerberos is that it give you mutual authentication "for free"
  - C and S authenticate each other as a side-effect of running the Kerberos protocol.

Question 3a

- Key establishment is easy, right? All we have to do is have C and S perform a Diffie-Hellman key exchange.
- But how do C and S know they're talking to each other while doing so?
- One of the benefits of Kerberos is that it give you mutual authentication "for free"
  - C and S authenticate each other as a side-effect of running the Kerberos protocol.

Question 3a

- So one way to solve the problem is to add mutual authentication to Diffie-Hellman. Here's how:
- First, have C and S run the Kerberos protocol to obtain shared secret $K_{C,S}$ (known to C, S and the KDC).
- Now have C and S run ephemeral Diffie-Hellman (using some common set of parameters).
- Initialization: each party (C and S) chooses a secret exponent (a and b respectively)
  - C computes $A = g^a \bmod p$, S computes $B = g^b \bmod p$
- C → S: A, HMAC-SHA1(A, $K_{C,S}$).
  - S verifies the keyed MAC and knows that A really came from C.
- S → C: B, HMAC-SHA1(B, $K_{C,S}$).
  - C verifies the keyed MAC and knows that B really came from S.
- C computes $B^a \bmod p$ to get $K' = g^{ab} \bmod p$.
- S computes $A^b \bmod p$ to get $K' = g^{ab} \bmod p$.
- Now C and S hold a new shared secret, K', which is unknown to the KDC.
  - C and S used $K_{C,S}$ to authenticate a later, public key-based exchange.

Question 3b

- What happens if the KDC is active?

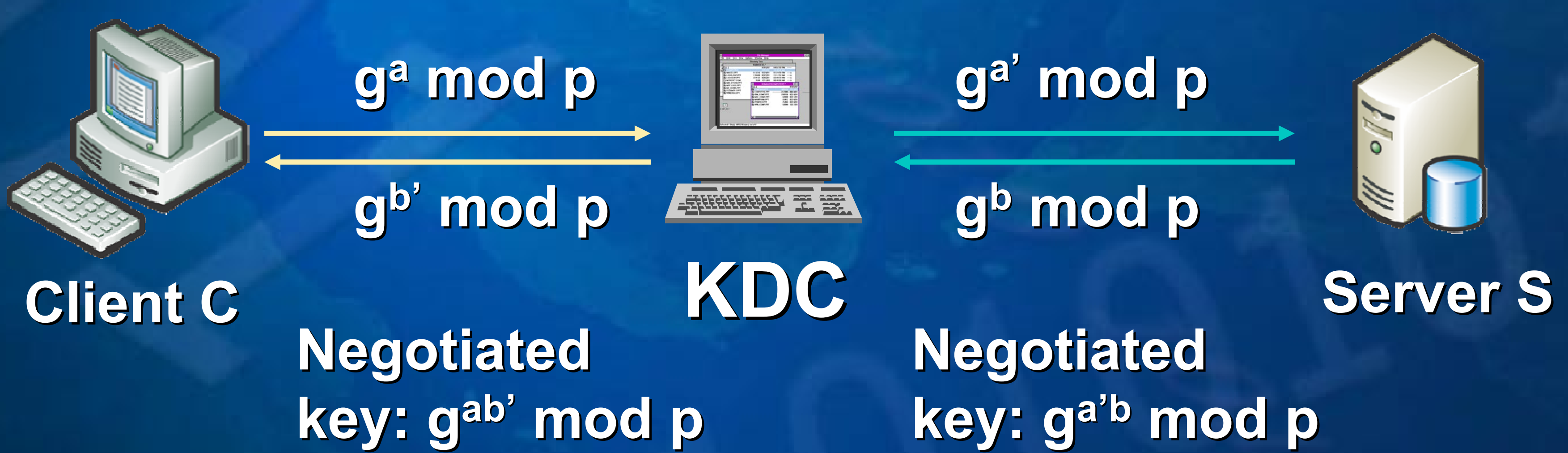
Question 3b

- What happens if the KDC is active?

- If the KDC is active then it can intercept messages, rewrite A & B to A' and B' and conduct a man-in-the-middle attack.

Man-in-the-Middle

- An active KDC can play a man-in-the-middle attack.
  - KDC can also defeat the modified D-H in 3(a) because it knows $K_{C,S}$



Question 3c (Extra Credit)

- Assume C and S are jointly members of two independent Kerberos realms:
  - Realm 1: KDC1, yielding key $K_{C,S,1}$
  - Realm 2: KDC2, yielding key $K_{C,S,2}$
- Assuming that KDC1 and KDC2 do not collude with each other, devise a protocol between C and S to create an encrypted channel that is secure against active eavesdropping from both KDC1 and KDC2.
- > 3c) With two independent keys, so long as the KDCs don't collude they can't play man-in-the-middle.

Question 3c (Extra Credit)

- With two independent keys shared between C and S, so long as the KDCs don't collude neither can perform a man-in-the-middle attack.
  - "Independent" here means "Neither key is known to both KDC1 and KDC2".
- Solution: as in 3(a) above, but use both keys in the HMAC authenticators
  - Neither KDC1 or KDC2 knows both, so even if they're active they can't switch messages on C or S

Question 3c

- C and S run Kerberos in both realms, obtaining $K_{C,S,1}$ and $K_{C,S,2}$
- Now have C and S run ephemeral Diffie-Hellman (using some common set of parameters).
- Initialization: C computes $A = g^a \bmod p$, S computes $B = g^b \bmod p$
- C → S: A, HMAC-SHA1(A, $K_{C,S,1,}K_{C,S,2}$).
  - S verifies the keyed MAC and knows that A really came from C.
- S → C: B, HMAC-SHA1(B, $K_{C,S,1,}K_{C,S,2}$).
  - C verifies the keyed MAC and knows that B really came from S.
- C computes $B^a \bmod p$ to get $K' = g^{ab} \bmod p$.
- S computes $A^b \bmod p$ to get $K' = g^{ab} \bmod p$.
- Now C and S hold a new shared secret, K', which is unknown to the KDC.