Assignment #4 – Solutions

Problem 1

- Scenario: two companies, A & B, each running Kerberos-based systems. Key Distribution Centers KDC_A and KDC_B
- A and B want to link their Kerberos networks together
  - Have shared secret key K_{AB}
- What modifications do we need to make to the standard Kerberos protocol?
- The interesting case is when a client in one company wants to access a server in another company.
  - No change needed for intra-company communications.

Problem 1

- Let's assume a client C_A in A wants to communicate with a service S_B in B. So we want C_A to end up with a ticket to S_B.
- In order for C_A to get a ticket to S_B, C_A needs to talk to TGS_B (because TGS_B issues tickets to S_B).
- In order for C_A to get a TGT to talk to TGS_B, C_A needs to talk to KDC_B.
- But KDC_B can't authenticate C_A directly, so we need to modify the protocol so that C_A can
  1. Authenticate to KDC_A, and
  2. Ask KDC_A to request a TGT for TGS_B from KDC_B on C_A's behalf

Problem 1

Following the notation used in class:

- C_A authenticates to KDC_A:

$$C_A \dashrightarrow KDC_A: C_A, TGS_B, N_{C_A}$$

- KDC_A forwards the request for a TGT for TGS_B to KDC_B using their shared secret K_{AB}

$$KDC_A \dashrightarrow KDC_B: \{C_A, TGS_B, N_{KDC_A}\}K_{AB}$$

- KDC_B decrypts the request from KDC_A and returns a ticket for C_A to talk to TGS_B.

$$KDC_B \dashrightarrow KDC_A: T_{C_A,TGS_B}, \{K_{C_A,TGS_B}\}K_{AB}$$

where $T_{C_A,TGS_B} = TGS_B, \{C_A$, C-addr, lifetime, $K_{C_A,TGS_B}\}\ K_{TGS_B}$

Problem 1

- KDC_A can then decrypt and re-encrypt the session key:

$$KDC_A \;-\!\!\rightarrow C_A: T_{C_A,TGS_B}, \{K_{C_A,TGS_B}\}K_{C_A}$$

- C_A now knows K_{C_A,TGS_B} and can use this session key along with T_{C_A,TGS_B} to continue Phase 2 of Kerberos with TGS_B directly.

Problem 2

- Relative costs of RSA and AES, given
  - AES-128 encrypt/decrypt 1 block in time t
  - AES-256 encrypt/decrypt 1 block in time $1.4t$.
  - A single RSA encryption takes time an^2.
  - A single RSA decryption takes time bn^3.
  - A single RSA key generation step takes time cn^4.

Problem 2a

- How many AES-128 encryption operations can you perform in the time it takes to do a single RSA-1024 encryption?
- Time for a single RSA-1024 encryption: $a(1024^2) = a\,2^{20}$
- Time for a single AES-128 encryption: t

$$2^{20}\,\frac{a}{t}$$

Problem 2b

- How many AES-128 decryption operations can you perform in the time it takes to do a single RSA-1024 decryption?
- Time for a single RSA-1024 decryption: $b(1024^3) = b\ 2^{30}$
- Time for a single AES-128 decryption: t

$$2^{30}\frac{b}{t}$$

Problem 2c

- Moving from AES-128 to AES-256
- AES-256 encryptions per RSA-1024 encryption

$$2^{20}\frac{a}{1.4t} = 748982.8571428\ldots\frac{a}{t}$$

Problem 2d

- Moving from AES-128 to AES-256, RSA-1024 to RSA-2048
- AES-256 decryptions per RSA-2048 decryption
- One RSA-2048 decryption: $bn^3 = (2^{11})^3 b = 2^{33} b$
- One AES-256 decryption: $1.4t$

$$2^{33} \frac{b}{1.4t} = 6135667565.7142857\ldots \frac{b}{t}$$

Problem 2e (for AES-128/RSA-1024)

- 2^{20}AES-128 encryptions = 1 RSA-1024 decryption.
- Using AES-128 and RSA-1024, sending 16MB of data requires:
  - 1 RSA keygen = $c\ (1024)^4 = c\ 2^{40}$
  - 2 RSA encryptions = $2a(1024)^2 = 2a2^{20} = a2^{21}$
  - 2 RSA decryptions = $2b(1024)^3 = 2b2^{30} = b2^{31}$
  - Total time on RSA operations: $2^{21}(a + 2^{10}b + 2^{19}c)$
- 16MB of data = 1M (2^20) data blocks
  - Need two* AES operations per block (1 encrypt, 1 decrypt)
  - 2 2^{20}AES operations = 2 $*$ (one RSA decryption)

    = 2 $* b2^{30} = 2^{31}\ b$
- *NOTE: Some students may have interpreted "If you <u>send</u> 16MB…" as meaning "only count 1 AES encryption/block." We had intended for both the AES encrypt and decrypt to count, but we will accept answers that only count 1 AES encryption/block so long as they are internally consistent.

Problem 2e (AES-128/RSA-1024)

- Total time on RSA operations: $2^{21}(a + 2^{10}b + 2^{19}c)$
- Total time for AES operations: $2^{31}b$
- Total time for all operations: $2^{21}(a + 2^{10}b + 2^{19}c) + 2^{31}b$

  $= 2^{21}(a + 2^{10}b + 2^{19}c)$
- Fraction of overall time spent in RSA:

$$\frac{2^{21}(a+2^{10}b+2^{19}c)}{2^{21}(a+2^{11}b+2^{19}c)} = \frac{(a+2^{10}b+2^{19}c)}{(a+2^{11}b+2^{19}c)}$$

Problem 2e (for AES-256/RSA-2048)

- 2^{20}AES-128 encryptions = 1 RSA-1024 decryption.
- For RSA-2048:
  - 1 RSA keygen = $c\ (2048)^4 = c\ 2^{44}$
  - 2 RSA encryptions = $2a(2048)^2 = 2a2^{22} = a2^{23}$
  - 2 RSA decryptions = $2b(2048)^3 = 2b2^{33} = b2^{34}$
  - Total time on RSA operations: $2^{23}(a + 2^{11}b + 2^{19}c)$
- 16MB of data = 1M (2^20) data blocks
  - Need two AES-256 operations per block (1 encrypt, 1 decrypt)
  - 2 2^{20}AES-256 operations = $2\ *\ 1.4\ *$ (one RSA-1024 decryption)

    = $2\ *\ 1.4 * b2^{30}$

    = $1.4\ b\ 2^{31}$

Problem 2e (AES-256/RSA-2048)

- Total time on RSA operations: $2^{23}(a + 2^{11}b + 2^{19}c)$
- Total time for AES operations: $1.4\ b\ 2^{31}$
- Total time for all operations:

$$2^{23}(a + 2^{11}b + 2^{19}c) + 1.4\ b\ 2^{31}$$

$$= 2^{23}(a + 1.4b2^{8} + 2^{11}b + 2^{19}c)$$

- Fraction of overall time spent in RSA:

$$\frac{2^{23}(a+2^{11}b+2^{19}c)}{2^{23}(a+1.4b2^{8}+2^{11}b+2^{19}c)} = \frac{(a+2^{11}b+2^{19}c)}{(a+1.4b2^{8}+2^{11}b+2^{19}c)}$$

Problem 3

- First, let's look at MD5 vs. SHA-1
- MD5 has a 128-bit output, so with a birthday attack we would expect to find a collision in 2^{64} hash operations.
- SHA-1 has a 160-bit output, so 2^{80} hash operations for a collision via birthday attack.

$\frac{2^{80}}{2^{64}} = 2^{16}$, so we need 16 Moore's Law doublings

= 24 years

Problem 3

Now, RSA-768 vs RSA-1024. Let's compute the formula for $n = 768$ and $n = 1024$.

$n = 768$:

$$e^{2*768^{\frac{1}{3}}*\left((\log_2 768)\right)^{\frac{2}{3}}} = 7.794344\ldots \times 10^{35}$$

$n = 1024$:

$$e^{2*1024^{\frac{1}{3}}*\left((\log_2 1024)\right)^{\frac{2}{3}}} = 4.328252\ldots \times 10^{40}$$

Ratio: approx 55530.67904…

Problem 3

Ratio: approx 55530.67904…

Now, $\log_2$ 55530.67904… = 15.76

So we need 15.76 Moore's Law doublings

= 23.64 years

Bottom line: move from RSA-1024 to RSA-2048 first

Problem 4

- Alice and Bob live in different countries, exchange key K face-to-face, want to exchange a sequence of messages in the future.
  - At any point in time, Alice's computer can be seized, giving an attacker all the information stored on her computer at the time of seizure.
- Let $m_1, m_2, m_3, \ldots$ be the sequence of messages Alice and Bob exchange.
- How can we use K to secure each m_i, such that if Alice's computer is seized at time t, none of the $m_1, m_2, \ldots m_{t-1}$ are compromised?

Problem 4

- At any point in time, we want Alice's machine to contain only information necessary for encrypting future messages, and not anything that could be used to decrypt past messages.
- So, some things that don't work:
  - Encrypt each m_i with K directly (would have to keep K around, and when the computer is seized it exposes all prior m_i).
  - Encrypt each m_i with $K_i = H(i \parallel K)$ where H is a hash function (would still have to keep K around, and once seized would reveal past messages.

Problem 4

- One possible approach:
  - Let $K_0 = K$.
  - Let $K_i = H(K_{i-1})$
  - Store only the K_i for the next message to send.
  - Encrypt m_i with K_i.
  - Once m_i is sent, compute K_{i+1} and destroy K_i.

- Other solutions are possible…

Problem 5

- Modifying SSL/TLS to support session restart
  - Proposal: Whenever a session is established, the pre-master secret is used to derive a session identifier that can be retained by the client and server.
  - This session identifier is then cached along w/ the original pre-master secret, and the client can request restart by sending the identifier along with the rest of the session details (including the ciphersuite).
- How can an attacker exploit this protocol modification?

Problem 5

- An attacker can play man-in-the-middle between a client requesting restart and the server
- The attacker can't change the pre-master secret, but because the client sends the session details to the server, the adversary can change any of those details.
  - In particular, the adversary can change the ciphersuite, making it something easier
- This is called a *downgrade attack* – it causes the client and server to use a ciphersuite that neither would negotiate to absent interference from the adversary