Zack Rossman

Ethan Lewis

Professor Buttyan

4/17/19

Chat Application Design Document

**Overview:**

Our application will rely on a chat server, rather than peer to peer networking, to send messages between parties. As for the major components, there will exist a single network instance which facilitates the sending of messages between multiple (two or more) participants in the chat session. The sending of such messages will take place within a secure channel that exists only for the lifetime of the chat session, such that a new secure channel will be defined by a fresh shared secret K (from which a new MAC key Km is derived) along with newly initialized message sequence numbers.

There are four stages in a chat session lifetime: instantiation of participants/parameters, establishment of a secure channel, operation, and termination. Below, we have provided a high-level walkthrough of each stage. A more in-depth description of the components appears in the Detailed Description section of this document.

Instantiation of participants/parameters:

* Instantiate a server and participants
* Create and store public key/private key pair for each participant
* Create a certificate (which contains a public key) for each participant

Establishment of a Secure Channel

* Establish a shared secret K, distribute amongst the participants
* All participants derive a MAC key Km from K
* Create sndstate.txt and rcvstate.txt files for each participant containing
  + Shared secret K
  + MAC key Km
  + A single sndsqn number, initialized to 0
  + A rcvsqn number for every participant (except itself), initialized to 0

Operation:

* Participants proceed with sending and receiving of messages until they decide to stop

Termination:

* All participant data is deleted, thus protecting parameters and messages of the recent secure channel

**Attacker models (assumptions made about our attackers)**

* Capabilities:
  + Chosen ciphertext and plaintext with oracle
  + Cannot be a legitimate protocol participant, but malicious
  + Full control over communication of honest parties
  + Cannot break cryptographic primitives
* Objectives:
  + Discover shared secret K
  + Discover MAC key Km derived from K
  + Discover the private key of any participant
  + Systematically decrypt encrypted messages that were sent within a party which did not include the attacker
  + Systematically forge messages and MAC keys
* Attacks:
* Replay attack
* Forgery attack
* Padding oracle attack
* Decrypting old messages with an old K (since ISO 11770 doesn’t provide perfect forward secrecy)

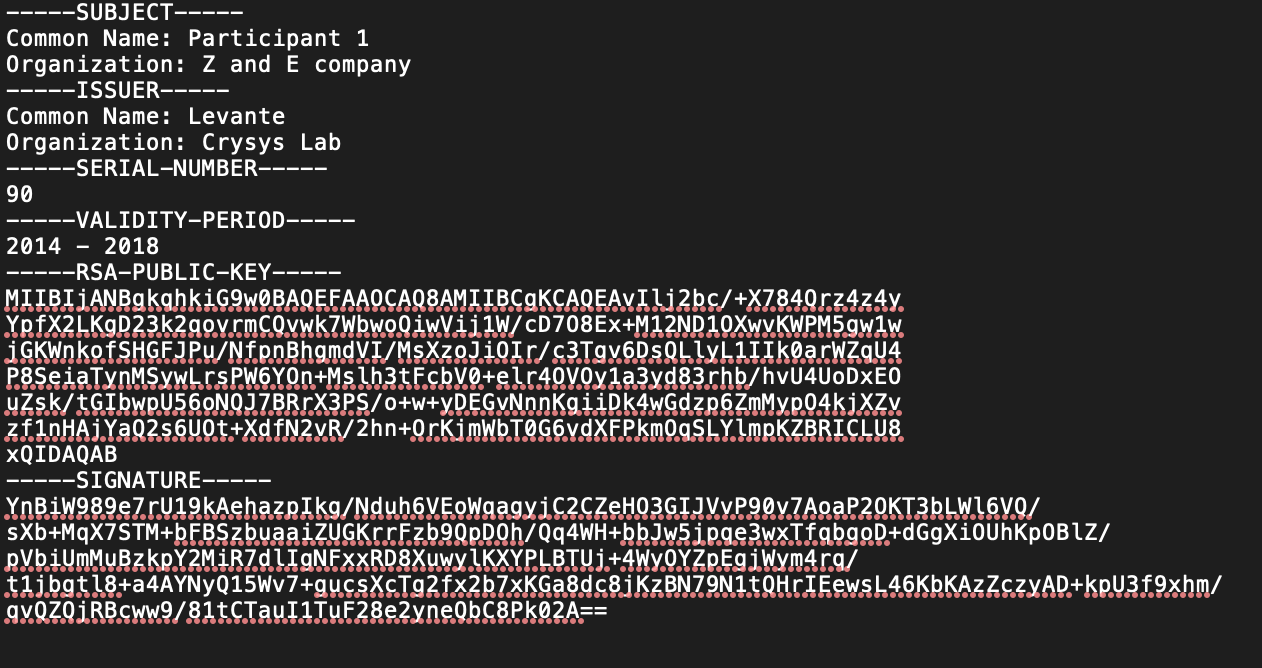
**Security requirements identified based on attacker model**

* Key exchange protocol:
  + Using public key crypto to securely establish keys
    - Ensures integrity, freshness, and authentication due to each member participating in key derivation.
    - We are assuming that each participant knows each other’s public key
* Message sending and receiving:
  + Our combination of encryption function and MAC function ensures no chosen ciphertext / padding the oracle attacks will work
  + This is because MAC verification comes before any decryption of padding
  + We are using a cryptographic pseudo random number generator to generate our IV to ensure no predictability.

**Detailed Description:**

Instantiation of participants/parameters:

* Store each pubkey/privatekey pair in a .pem file within each participant’s private, local directory, “netsim/network/\*participant\_address\*/keypairs/”
* Store each pubkey in a public directory of the network, “netsim/network/pubkeys/”
* Store each certificate in a public directory of the network, “netsim/network/certs/”
  + Certificate format:
    - Subject common name = “Participant #”
    - Subject organization = “Z and E company”
    - Issuer common name = “Levente”
    - Issuer organization = “Crysys Lab”
    - Serial number = \*integer between 1 and 100\*
    - Validity Period = \*start year\* - \*end year\*, inclusive
  + Example



Establishing Secure Channel:

• Establish a shared secret K amongst the participants:

* We will not be using a 3rd party (server) to establish shared secret K between chat room participants because 3rd party will be able to decrypt the messages it is relaying.
* We will not be using Diffie-Hellman key exchange protocol to establish shared secret K between chat room participants because DH is a key agreement protocol, and it’s difficult for more than two parties to agree on a K
* Instead we will use the ISO 11770 key exchange protocol, a key transport protocol, to establish shared secret key K
  + Elect one participant Pk to decide K
  + Pk sends message PubEnckpi+(A|K|T\_Pk|Sigkpk-(B|K|T\_Pk)) to all participants (use hybrid encryption, HW6, for this pubkey encryption)
  + Each recipient, Pi, of this message will decrypt Pk’s message with their private key, Pi-, to obtain the fresh shared secret K

• Each participant derives a unique MAC key, Km, and encryption key, Ke, from K

* Km = HMAC(“mac key”, K)
* Ke = HMAC(“encryption key”, K)
* No need to include a fresh random for input to HMAC since MAC key will be computed with a new K for each session

• Each participant creates rcvstate.txt and sndstate.txt files consisting of:

* Private MAC key Km
* The derived key Ke
* A single sndsqn number, initialized to 0
* A rcvsqn number for every participant (except itself), initialized to 0
  + E.g the rcvsqn number for participant B is labeled “rcvsqnB: #” and the rcvsqn number for participant C is labeled “rcvsqnC: #”, etc.

Composing & sending a message

**•** Encryption scheme:

* For the general encryption of messages sent, we will be using a combination of encryption and MAC functions
  + Our scheme is: ENCKe(m) | MACKm(ENCKe(m)). This ensures that:
    - No decryption and verification of padding before MAC verification
    - Chosen ciphertext attacks will fail

• Encrypting a message:

* Message format (same as challenge 1 from HW 6):
* Fill out appropriate header fields (Ver, T, Len, SQN), where SQN comes from sndsqn.txt field
* Compute a random IV (using PyCryptodome Random library)
* Encrypted payload = [ Payload | padding ] is encrypted with AES in CBC mode using the random IV
* Compute MAC value with HMAC using SHA-256 on the header fields (Ver, T, Len, SQN), the IV, and the encrypted Payload (and padding)
* Send message = [header | IV | encrypted payload | MAC value] to the server, tell server to forward onto selected recipients
* Update the sender’s sndstate.txt file such that sndsqn = sndsqn + 1

Receiving a message

* All recipients receive a message = [header | IV | encrypted payload | MAC value] from server
* Verify that the SQN value in the header is greater than rcvsqn from rcvsqn.txt (explicit sequence numbering) that corresponds to the sender (value of sender ID found in the first character of the message’s .txt file in the recipient’s /IN directory (e.g. “A0000.txt was a message sent from A, whereas “B0001.txt” was a message sent from B)
* Verify computed MAC value which is computed from the header, IV, and encrypted payload
* Decrypt the encrypted payload using AES CBC mode
* Verify that the padding is valid, strip padding from plaintext message
* Update the rcvstate.txt file such that rcvsqn = SQN from the header (explicit sequence numbering)

Ending a chat session:

* Each participant deletes the shared secret K and derived MAC key Km
* Each participant deletes send/received messages from their local repositories