CS 5430 A3 - Rationale Document

Metadata

- Reuben Rappaport (rbr76)
- Alex Clark (ayc55)

Summary

This project allows for sending messages in four different configurations, allowing for varying degrees of secure messaging from one principal to another, even in the presence of a Dolev-Yao attacker on the network.

In every configuration aka mode of operation, we rely upon the same protocol for distributing a shared session key and MAC key, the ISO/IEC 11770-3 Key Transport Mechanism 2 key transport protocol. This protocol can be represented as follows:

```
1. A: kAB, iv = KeyGen(), Gen(AES Block Size)
2. A: e = Enc(A, kAB, iv; K_B)
3. A: s = Sign(B, tA, e; k2_A)
4. A -> M: B, tA, e, s
5. M -> B: B, tA, e, s
6. B: Verify(B, tA, e; s; K2_A)
7. B: check difference(tA, tB) < 2 minutes
8. B: B, kAB, iv = Dec(e; k_B)
```

In this key transport protocol, Alice sends a message to Bob consisting of several parts. `B` is an identifier for Bob, and in our implementation is just a string stating "Bob". `tA` is a timestamp taken from the local system time on Alice's machine in millisecond representation. `Enc` is a function signifying an asymmetric encryption algorithm, and in our implementation this algorithm is RSA with OAEP padding and a key size of 2048 bits: `K_B` and `k_B` are Bob's public and private RSA keys, respectively. `Sign` denotes a digital signature algorithm, which in our implementation is DSA with a key size of 1024 bits: `K2_A` and `k2_A` are Alice's public and private DSA keys. `kAB` is the session key or MAC key to be distributed, and `iv` is an initialization vector, with `KeyGen` being the function for generating appropriate AES keys (or for MAC keys, HmacSHA256 keys) and `Gen` being a generating the initialization vector, in our implementation using a Java SecureRandom object to ensure sufficient randomness and entropy.

Alice first generates the session key and initialization vector, then creates the ciphertext `e` by encrypting it with Bob's public key, using RSA with OAEP padding. She then creates the signature `s` by signing the resulting ciphertext, along with Bob's identifier and a timestamp, using her private signing key and the DSA algorithm. Alice then sends a message to Mallory consisting of Bob's identifier, the timestamp, the ciphertext and the signature. Mallory forwards this message on to Bob. Bob then verifies that the message's identifier, timestamp, and ciphertext match those in the signed message, using Alice's public DSA key for the verification. He also checks that the timestamp is sufficiently recent, being within 2 minutes of his own system time. Lastly, he decrypts the transported key and initialization vector, having the key available to use.

It should be noted that in steps 4 and 5, Mallory sends the mssage on without modification. A textbook Dolev-Yao attacker would be able to delete this message or attempt to modify it, albeit not be successful in meaningfully modifying it thanks to the non-malleable encryption used. However, for the purposes of this project and in particular demos, we do not give Mallory the ability to affect the key distribution. This is because allowing Mallory to do so could lead to any meaningful communication becoming impossible (except in plaintext mode), if Mallory prevented a session key from ever being distributed. Therefore, we have relaxed the Dolev-Yao restrictions in this case.

Algorithm and Key size Justifications

The algorithms we use in this project are as follows:

- AES with 128-bit keys, for session keys. Used to encrypt messages in Configurations 2 and 4.
- HMAC, using the cryptographic hash function SHA-256, with 256 bit keys. Used for message authentication in Configurations 3 and 4.
- RSA with OAEP padding, using 2048-bit keys. Used for asymmetric encryption in the key transport protocol.
- DSA, using 1024-bit keys. Used for signing and verification in the key transport protocol.

AES is a justified choice because it currently has no known practical attacks, even with the smallest standard key size of 128 bits.

HMAC and SHA-256 are valid choices because, although SHA-1 has recently been broken (albeit not in a way feasible to most users), a key size of 256 bits should prevent attacks for the foreseeable future.

RSA with OAEP padding is a justified choice because it is a non-malleable encryption scheme, which prevents an attacker from modifying a ciphertext into another valid ciphertext by tampering. The key size of 2048 bits is somewhat overkill but is of sufficiently large size that it should be secure, barring perhaps the NSA having access to efficient prime factorization. RSA with OAEP is not a valid choice for signing, however.

This leads us to DSA, used with SHA-1 and a key size of 1024 bits. This is reasonably justified because the SHAttered attack used for SHA-1 collisions is not feasible for use as an attack against our implementation.

Configuration 1: No Cryptography

In this mode of operation, we do not use any cryptographic protocols apart from the ISO/IEC 11770-3 Key Transport Mechanism 2 key transport protocol, for distributing a session key at the beginning of a session. This is not used in no-cryptography mode, but is distributed in every configuration due to it otherwise being necessary. Messages are simply sent in plaintext and due to their lack of protection, can be freely read, deleted, or modified by malicious attacker Mallory.

The way in which messages are distributed is as follows:

```
1. A -> M: m, n
2. M -> B: m', n'
3. B: check n' = (last seen n) + 1
```

Alice sends to Mallory a message `m` along with a sequence number `n`, which is intended to be sent on to Bob. Mallory then sends `m’` and `n’` to Bob, which can be the original message and sequence number, modifications of either or both of these, or nothing (i.e. deleting the message instead of sending it). Bob does not have any way of knowing if the message has been modified, but does check whether the sequence number is 1 greater than that of the last seen message with a valid sequence number. If the current sequence number is less than this, it is assumed that the number has been tampered with by an attacker, who may be attempting a replay attack. If the sequence number exceeds that of the last valid-sequence number message by more than 1, it is an indication that a message has been dropped or its sequence number modified, presumably by an attacker.

Configuration 2: Symmetric Encryption Only

In this mode of operation, in addition to the aforementioned key transport protocol, we use a protocol relying upon symmetric-key encryption with the session key `kAB` distributed at the start of each session. The cryptographic algorithm used is AES in CBC mode, with a 128-bit key and a random initialization vector. Messages and sequence numbers are encrypted with this session key and sent from Alice to Mallory to Bob, with tampering by Mallory resulting in either a failed decryption or a nonsense decrypted plaintext message (depending on whether the tampered-with message can be decrypted). This protects confidentiality but makes no guarantees about integrity.

```
1. A: e = Enc(m,n; kAB)
2. A -> M: e
3. M -> B: e'
4. B: m',n' = Dec(e'; kAB)
5. B: check n' = (last seen n) + 1
```

Alice first encrypts a message `m` and sequence number `n` with the shared session key `kAB`, outputting the ciphertext `e`. Alice then sends `e` on to Mallory, with the assumption that Mallory will send it unmodified to Bob. Mallory is able to modify the ciphertext, but does not have a feasible way of decrypting it assuming the session key has not been somehow obtained (which the key distribution protocol should prevent, through conventional eavesdropping or man-in-the-middle attacks). Mallory then sends `e’`, a potentially modified, deleted, or replaced (via replay of a previous message) ciphertext, on to Bob. Bob decrypts `e’` using `kAB`, and if the decryption is successful outputs a message `m’` and sequence number `n’`. The accuracy of `n’` is then checked in the manner described above in Configuration 1.

Configuration 3: MACs Only

In this mode of operation, the key transport protocol distributes a MAC key `k_MAC` at the beginning of the session. The cryptographic algorithm used is HmacSHA256 with a 256 bit key. Messages and sequence numbers are accompanied by a MAC tag which provides integrity verification. The inclusion of sequence numbers in the tagged message helps to defend against replay attacks. This protects integrity but does nothing for availability.

```
1. A: t = MAC(m,n; k_MAC)
2. A -> M: m,n,t
3. M -> B: m',n',t'
4. B: t'' = MAC(m',n'; k_MAC)
5. B: check t'' = t'
6. B: check n' = (last seen n) + 1
```

Alice first computes the tag `t` for the message `m` and sequence number `n` with the shared MAC key `k_MAC`. She then sends `t`, `m`, and `n` to Mallory. Mallory either deletes the message or forwards on a modified message `m’`, `n’`, and `t’` to Bob. When Bob receives the message he computes the new tag `t"` and checks that it is equal to the tag he received `t’`. If it is not then he knows that the message has been tampered with. He then performs the same sequence number check as in the plaintext configuration, attempting to resist replay attacks.

Configuration 4: Symmetric Encryption then MAC

In this mode of operation the key transport protocol distributes both a MAC key `k_MAC`and a session key `kAB` with initialization vector `iv` at the beginning of the session. The symmetric encryption algorithm is again AES with 128 bit keys and the MAC algorithm is HmacSHA256 with a 256 bit keys. The message and sequence number are first encrypted and then a MAC tag is generated for the ciphertext before both are sent to Bob. This protects both integrity and confidentiality but makes no guarantees about availability.

```
1. A: e = Enc(m, n; kAB)
2. A: t = MAC(e; k_MAC)
3. A -> M: e, t
4. M -> B: e',t'
5. B: t'' = MAC(e'; k_MAC)
6. B: check t'' = t'
7. B: m',n' = Dec(e'; kAB)
8. B: check n' = (last seen n) + 1
```

Alice first encrypts the message `m` and message number `n` with the session key `kAB`. She then computes the tag of the encrypted text with the MAC key `k_MAC` and sends both the ciphertext and the tag to Mallory. Mallory may delete the message or he may modify it or leave it unchanged before forwarding it to Bob. Bob computes the tag of the ciphertext he receives with `k_MAC` and checks that it is the same as the tag he received. If it is then he knows that integrity has been preserved. He then decrypts the message with his session key and checks that the message number is valid.