Lecture 1:
Course Introduction

Thomas Johansson

Chapter 9: Symmetric Key Distribution

- To understand the problems associated with managing and distributing secret keys.
- To learn about key distribution techniques based on symmetric key based protocols
- (To introduce the formal analysis of protocols.)

Key Management

To be able to use symmetric encryption algorithms such as DES or Rijndael we need a way for the two communicating parties to share the secret key.

- key distribution,
- key selection,
- key lifetime,

- **Static (or long-term) Keys:** to be in use for a long time period. The compromise of a static key is usually considered to be a major problem.
- **Ephemeral, or Session (or short-term) Keys:** have a short life-time, maybe a few seconds or a day. The compromise of a session key should only result in the compromise of that session's secrecy and it should not affect the long-term security of the system.

Key Distribution

- **Physical Distribution:**
- **Distribution Using Symmetric Key Protocols:** Once some secret keys have been distributed between a number of users and a trusted central authority, we can use the trusted authority to help generate keys for any pair of users as the need arises. They are usually very efficient but have some drawbacks. They assume that both the trusted authority and the two users who wish to agree on a key are both on-line.
- **Distribution Using Public Key Protocols:** two parties, who have never met or who do not trust any one single authority, can produce a shared secret key, using a key exchange protocol. The most common application of public key techniques for encryption: we agree a key by public key techniques and then use a symmetric cipher to actually do the encryption.

Key Selection

- All keys should be equally likely and really need to be generated using a true random number generator, however such a good source of entropy is hard to find.
- A truly random key will be very strong, it is hard for a human to remember.
-

Key size	Decimal digits	Printable characters
4	$10^4 \approx 2^{13}$	$10^7 \approx 2^{23}$
8	$10^8 \approx 2^{26}$	$10^{15} \approx 2^{50}$

Secret Key Distribution - Notation

- **Parties/Principals: A,B, S:** Two parties who wish to agree a secret are A and B, for Alice and Bob. We assume that they will use a trusted third party, or TTP, which we shall denote by S .
- **d Secret Keys:** K_{ab}, K_{bs}, K_{as}. K_{ab} denote a secret key known only to A and B.
- **Nonces:** N_a, N_b. Nonces are numbers used only once, they should be random. The quantity N_a denote a nonce originally produced by the principal A.
- **Timestamps:** T_a, T_b, T_s. The quantity T_a is a timestamp produced by A. When timestamps are used we assume that the parties try to keep their clocks in synchronization.

Secret Key Distribution - notation

The statement

$$A \rightarrow B : M, A, B, \{N_a, M, A, B\}_{K_{as}},$$

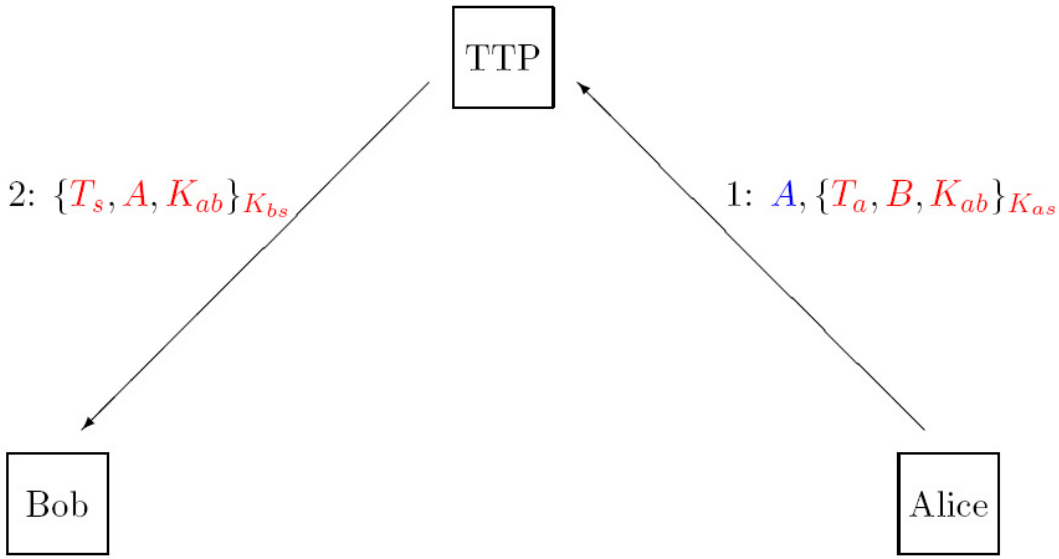
means A sends to B the message to the right of the colon. The message consists of

- a nonce M,
- A the name of party A,
- B the name of party B,
- a message $\{N_a, M, A, B\}$ encrypted under the key K_{as} which A shares with S. Hence, the recipient B is unable to read the encrypted part of this message.

Secret Key Distribution - assumptions

- A and B say, only share secret keys, K_{as} and K_{bs} with the trusted third party S. They want to agree/transport a session key K_{ab}. for a communication between themselves.
- **Attacker:** intercept any message flow over the network. Stop a message, alter it or change its destination. Also distribute her own messages over the network.
- New session key should be *fresh*, i.e. it has not been used before and has been recently created. The freshness property will stop attacks whereby the adversary replays messages so as to use an old key again.

Wide-Mouth Frog Protocol



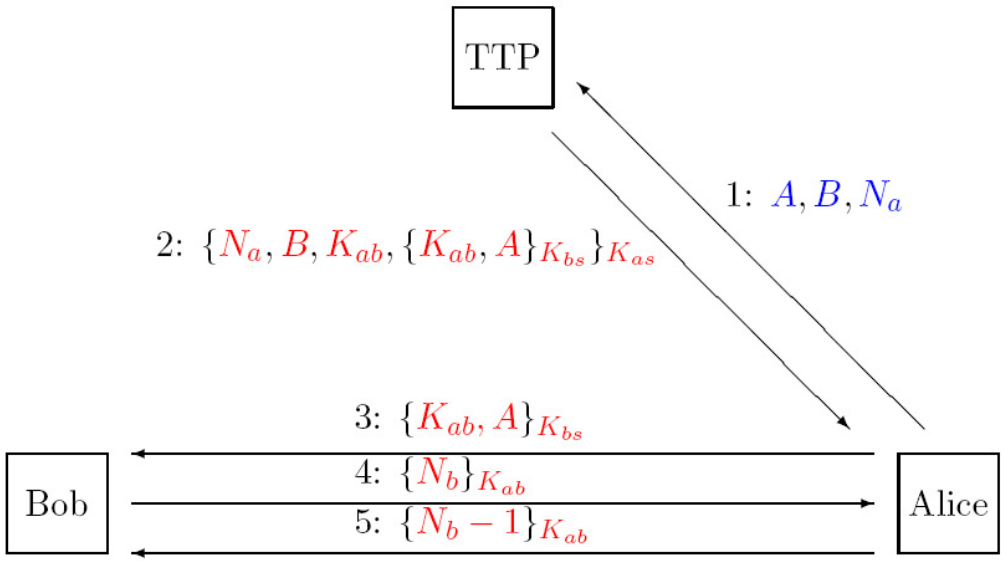
- S decrypts the last part of the message and checks that the timestamp is recent. This tells S he should forward the key to the party called B. S encrypts the key along with his timestamp and passes this encryption onto B.
- B decrypts the message received and checks the time stamp is recent. He can recover both K_{ab} and the name A of the person who wants to send data to him using this key.

Wide-Mouth Frog Protocol - Drawbacks

- synchronized clocks,
- assumes that A chooses the session key K_{ab} and then transports this key over to user B.

user A could have generated this key years ago and stored it on his hard disk, in which time Eve broke in and took a copy of this key.

Needham-Schroeder Protocol

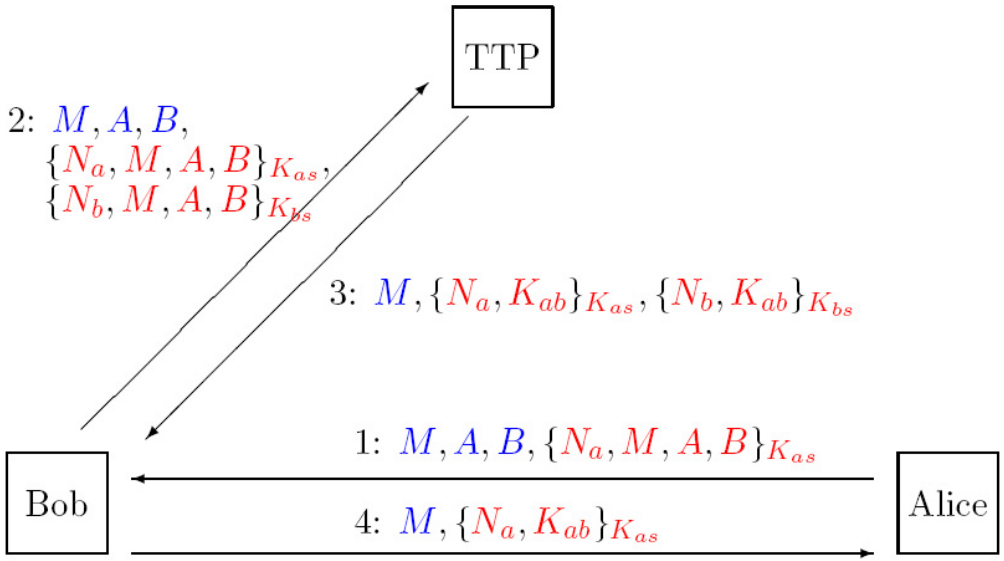


1. tells S that A wants a key to communicate with B.
2. S generates K_{ab} and sends it to A. The nonce N_a is included so that A knows this was sent after her first message. The session key is also encrypted under the key K_{bs} for sending to B.
3. The session key is encrypted under K_{bs} and sent to B.
4. B needs to check that this was not a replay. He encrypts a nonce back to A.
5. A encrypts a simple function of B's nonce back to B to prove to B

Needham-Schroeder Protocol - problem

- B does not know that the key he shares with A is fresh,
- An adversary who finds an old session transcript can, after finding the old session key by some other means, use the old session transcript in the last three messages involving B.

Otway-Rees Protocol



- Since the protocol does not make use of K_{ab} as an encryption key, neither party knows whether the key is known to each other. Does not offer *key confirmation*.

Kerberos

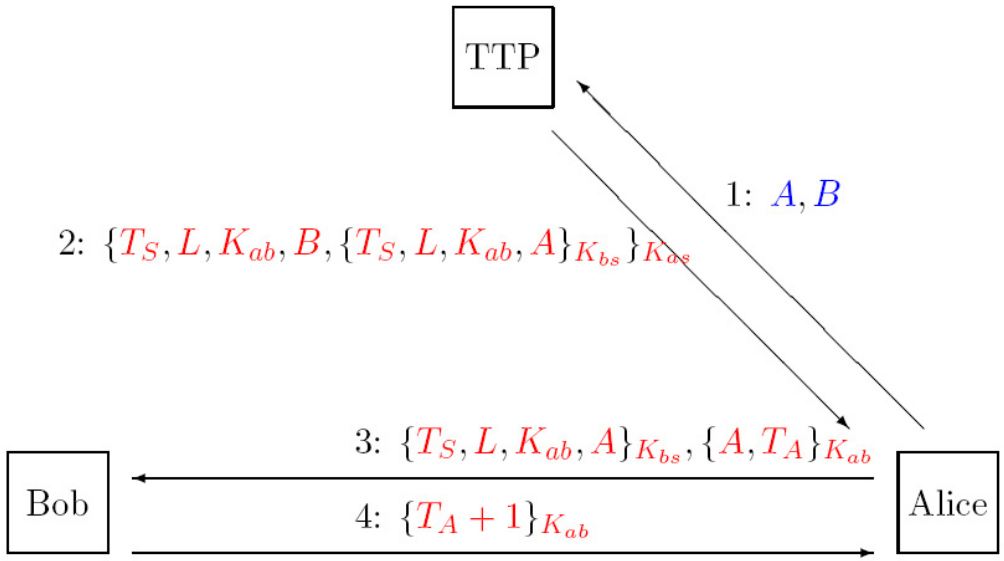
- authentication system based on symmetric encryption with keys shared with an authentication server.
- developed at MIT around 1987.
- The network consist of clients and a server. Clients may be users, programs or services.
- A central database of clients including a secret key for each client.
- Kerberos provide authentication of one entity to another and issue session keys to these entities.
- Kerberos run a ticket granting system to enable access control to services and resources.

Kerberos

Kerberos: an authentication server and a ticket generation server TGS.

Suppose A wishes to access a resource B.

- A logs onto the authentication server using a password.
- A is given a ticket from this server encrypted under her password. This ticket contains a session key K_{as}.
- uses K_{as} to obtain a ticket from the TGS S to access the resource B. The output is a key K_{ab} , a timestamp T_S and a lifetime L.



We have removed the problems associated with the Needham-Schroeder protocol by using timestamps, but this has created the requirement for synchronized clocks.

Formal Approaches to Protocol Checking

- To try and make the design of these protocols more scientific a number of formal approaches have been proposed.
- The most influential of these is the BAN logic.
- Read if you are interested.

Chapter 10: Hash Functions and Message Authentication Codes

- Guarantee integrity of information after the application of the function.
- A cryptographic hash function is keyless, a MAC has a key.
- A cryptographic hash function is usually used as a component of another scheme.

Hash Functions

- A cryptographic hash function h is a function which takes arbitrary length bit strings as input and produces a fixed length bit string as output, the *hash value*.
- A cryptographic hash function should be one-way: given any string y from the range of h, it should be computationally infeasible to find any value x in the domain of h such that

$$h(x) = y.$$

- Given a hash function with outputs of n bits, we would like a function for which finding preimages requires $O(2^n)$ time.

Hash Functions - collision resistant

- In practice we need something more than the one-way property.
- A hash function is called *collision resistant* if it is infeasible to find two distinct values x and x' such that

$$h(x) = h(x').$$

- *Birthday paradox:* To find a collision of a hash function f, we can keep computing

$$f(x_1), f(x_2), f(x_3), \ldots$$

until we get a collision. Output size n bits, then we expect to find a collision after $O(2^{n/2})$ tries.

Hash Functions - Second Preimage Resistant

- Second preimage resistant: given x it should be hard to find an $x' \neq x$ with $h(x') = h(x)$.
- a cryptographic hash function with n-bit outputs should require $O(2^n)$ operations before one can find a second preimage.

Hash Functions - requirements

- **Preimage Resistant:** It should be hard to find a message with a given hash value.
- **Second Preimage Resistant:** Given one message it should be hard to find another message with the same hash value.
- **Collision Resistant:** It should be hard to find two messages with the same hash value.

Lemma

Assuming a function is preimage resistant for almost every element of the range of h is a weaker assumption than assuming it either collision resistant or second preimage resistant.

Lemma

Assuming a function is second preimage resistant is a weaker assumption than assuming it is collision resistant.

Designing Hash Functions

- Designing functions of infinite domain is hard,
- one builds a so called *compression function*, which maps bits strings of length s into bit strings of length n, for $s > n$, and then chains this in some way to produce a function on an infinite domain.
- The most famous chaining method, *the Merkle-Damgård construction*.

Merkle-Damgård construction

- f is a compression function from s bits to n bits, $s > n$, believed to be collision resistant.
- use f to construct h which takes arbitrary length inputs.

1. $l = s - n$. Pad m with zeros so it is a multiple of l bits, write $m = m_1 m_2 \cdots m_t$. Set H to some fixed value.
2. **for** $i = 1$ **to** t **do** $H = f(H||m_i)$
3. Set $h(m) = H$.

Merkle-Damgård construction

- **Length strengthening:** input message is preprocessed by first padding with zero bits to obtain a message which has length a multiple of l bits. Then a final block of l bits is added which encodes the original length of the unpadded message in bits. The construction is limited to hashing messages with length less than 2^l bits.
- Theory: If f is collision resistant then so is h.

Constructions: The MD4 Family

Most widely deployed: MD5, RIPEMD-160 and SHA-1.

- MD4: 3 rounds of 16 steps and an output bitlength of 128 bits.
- MD5: 4 rounds of 16 steps and an output bitlength of 128 bits.
- SHA-1: 4 rounds of 20 steps and an output bitlength of 160 bits.
- RIPEMD-160: 5 rounds of 16 steps and an output bitlength of 160 bits.
- SHA-256: 64 rounds of single steps and an output bitlength of 256 bits.
- SHA-384: identical to SHA-512 except the output is truncated to 384 bits.
- SHA-512: 80 rounds of single steps and an output bitlength of 512 bits.

In recent years a number of weaknesses have been found in almost all of the early hash functions in the MD4 family, for example MD4, MD5 and SHA-1.

SHA-1

- the internal state of the algorithm is a set of five 32-bit values

$$(H1, H2, H3, H4, H5).$$

- define four round constants y_1, y_2, y_3, y_4.
- The length strengthening method used is to first append a one bit to the message, to signal its end and then to pad with zeros to a multiple of the block length = 512 bits. Finally the number of bits of the message is added as a seperate final block.

The data stream is loaded 16 words at a time into X_j for $0 \leq j < 16$.

Algorithm 10.4: SHA-1 Overview

$(A, B, C, D, E) = (H_1, H_2, H_3, H_4, H_5)$
/* Expansion */
for $j = 16$ **to** 79 **do**
    $X_j = ((X_{j-3} \oplus X_{j-8} \oplus X_{j-14} \oplus X_{j-16}) \lll 1)$
end
Execute Round 1
Execute Round 2
Execute Round 3
Execute Round 4
$(H_1, H_2, H_3, H_4, H_5) = (H_1 + A, H_2 + B, H_3 + C, H_4 + D, H_5 + E)$

The output is the concatenation of the final value of $H1, H2, H3, H4, H5$.

Algorithm 10.5: Description of the SHA-1 round functions

Round 1
for $j = 0$ **to** 19 **do**
$t = (A \lll 5) + f(B, C, D) + E + X_j + y_1$
$(A, B, C, D, E) = (t, A, B \lll 30, C, D)$
end
Round 2
for $j = 20$ **to** 39 **do**
$t = (A \lll 5) + h(B, C, D) + E + X_j + y_2$
$(A, B, C, D, E) = (t, A, B \lll 30, C, D)$
end
Round 3
for $j = 40$ **to** 59 **do**
$t = (A \lll 5) + g(B, C, D) + E + X_j + y_3$
$(A, B, C, D, E) = (t, A, B \lll 30, C, D)$
end
Round 4
for $j = 60$ **to** 79 **do**
$t = (A \lll 5) + h(B, C, D) + E + X_j + y_4$
$(A, B, C, D, E) = (t, A, B \lll 30, C, D)$
end

Three bit-wise functions of three 32-bit variables:

$$f(u, v, w) = (u \wedge v) \vee ((\neg u) \wedge w),$$

$$g(u, v, w) = (u \wedge v) \vee (u \wedge w) \vee (v \wedge w),$$

$$h(u, v, w) = u \oplus v \oplus w.$$

Hash Functions from Block Ciphers.

- Pad the message to be hashed and divide it into blocks

$$x_0, x_1, \ldots, xt,$$

- $H0 = IV$, and iterate

$$H_i = f(x_i, H_{i-1}).$$

- For example, a **Davies-Meyer hash**

$$f(x_i, H_{i-1}) = E_{x_i}(H_{i-1}) \oplus H_{i-1}.$$

Message Authentication Codes

- A hash function cannot protect message integrity because it is keyless.
- One can use a keyed hash function, called a *message authentication code*, or MAC.
- Two parties, who share a secret key, wish to ensure that a message transmitted between them has not been tampered with. They produce a check-value, or MAC, which is sent with the data, $code = MAC_k(m)$ where MAC is the check function, k is the secret key, m is the message.

Message Authentication Codes

- If we wish our message to remain confidential then we should encrypt it before applying the MAC. The user transmits

$$e_{k_1}(m), MAC_{k_2}(e_{k_1}(m)).$$

- data encapsulation mechanism, or DEM for short. Note, that different keys are used for the encryption and the MAC. Security: It should be hard given some MACs on some messages to produce a MAC on a new message.

MACs from hash functions.

- First idea: concaternate the key with the message and then apply the hash function. For example

$$MAC_k(M) = h(k||M).$$

- Problem (Merkle-Damgård construction.): Suppose one obtains the MAC c_1 on the t block message m_1

$$c_1 = MAC_k(m_1) = h(k||m_1).$$

Then, without knowledge of k, compute the MAC c_2 on the $t+1$ block message $m_1||m_2$ for any m_2 of one block in length, via

$$c_2 = MAC_k(m_1||m_2) = f(c1||m2).$$

- applies also to the length strengthened version.

HMAC

- Other unsuccessful attempts:

$$MAC_k(M) = h(M||k).$$

- The standard and secure way: **HMAC**, occurring in a number of standards documents, works as follows:

$$HMAC = h(k||p1||h(k||p2||M)),$$

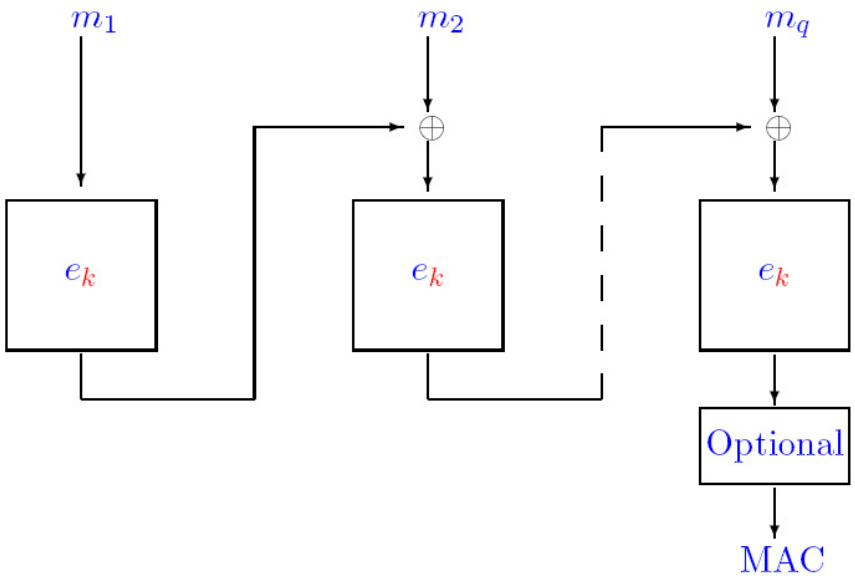
where p_1 and p_2 are strings used to pad out the input to the hash function to a full block.

Producing MACs from block ciphers.

- There are various types of MAC schemes based on block ciphers.
- Best known and most widely used by far are the CBC-MACs.
- various international standards dating back to the early 1980s. These early standards specify the use of DES in CBC mode to produce a MAC.

CBC MACs

- The data is padded to form a series of n-bit blocks.
- The blocks are encrypted using the block cipher in CBC Mode.
- Take the final block as the MAC, after an optional postprocessing stage and truncation.



Diskussion on padding, postprocessing and security implications, see the book.