The Modelling and Analysis of Security Protocols
Notes for Lecture 2

Tom Chothia
T.Chothia@cwi.nl

I. THE USES OF ENCRYPTION

Encryption is not just used to keep a message secret. Understanding the different uses of encryption is key to understanding and designing security protocol. The uses of encryption are:

1. Keeping data secret: Only the holders of the key can read the encrypted data.
2. Authentication: The encrypted message must have come from someone who had the key.

The Kerberos protocol illustrates all of these uses:

1. A \rightarrow S : \ A, B, N_a
2. S \rightarrow A : \ \{K_{ab}, B, L, N_a\}_{K_{sa}}, \ \{K_{ab}, A, L\}_{K_{sa}}
3. A \rightarrow B : \ \{A, T_a\}_{K_{ab}}, \ \{K_{ab}, A, L\}_{K_{sa}}
4. B \rightarrow A : \ \{T_a + 1\}_{K_{ab}}

In messages 2 and 3 encryption is used to keep the key \( K_{ab} \) secret.

In message 3 the encryption with \( K_{ab} \) in the \( \{K_{ab}, A, L\}_{K_{sa}} \) part of the message tells B that this part of the message came from S and therefore, as B trusts S to generate new keys, it authenticates the key inside.

\( A \) should only accept a fresh key from S. In message 2 \( A \) knows that the key \( K_{ab} \) must be fresh because it is bound to the nonce, which \( A \) sent in message 1, using encryption with the key \( K_{sa} \). If the key was not bound to this nonce then it could be a replay of an old message.

II. TYPES OF ENCRYPTION

We consider cryptographic functions to be “perfect”. We are looking for faults in the protocol design, not in a particular implementation. We define the various types of cryptographic operation using “constructor” and “destructor” function. The constructor functions build up data values (e.g., encrypt) and the destructor functions break them down (e.g. decrypt).

A. Symmetric Key Encryption

Symmetric key encryption uses the same key to encrypt and decrypt. We describe it using two functions:

constructor function: \( encrypt(\_\_\_\_) \)
destructor function: \( decrypt(\_\_\_\_) \)

Using these functions \( \{M\}_K \) would mean \( encrypt(M, K) \) and:

\( decrypt(encrypt(M, K), K) = M \)

This defines “deterministic” encryption, i.e., encrypting the same values with the same key results in the same cipher text:

\[
\begin{align*}
encrypt(M, K) &= encrypt(M', K') \\
&\iff M = M' \text{ and } K = K'
\end{align*}
\]

This might leak some data to the attacker, so encryption schemes can be padded to make them probabilistic. We defined this using an extended encryption function. For probabilistic encryption \( \{M\}_K \) would be \( encrypt(M, R, K) \) for a random value \( R \) and:

\[
\begin{align*}
decrypt(encrypt(M, R, K), K) &= M
\end{align*}
\]

As a new value \( R \) will be used for each encryption it no longer holds that \( \{M\}_K = \{M\}_K \) so the attacker cannot compare encrypted values.

B. Public Key Cryptography

Public key cryptography schemes, such as RSA, use a public key and a private key. The public key is handed out and can only be used for encryption. The private key is kept secret and is used for decryption. We will often define a principal as someone that has a particular private key. In this case the private key should never be sent in messages.

For private key encryption we use the functions:

constructor functions: \( encrypt(\_\_\_\_), pub(\_\_) \)
destructor function: \( decrypt(\_\_\_\_) \)

\( pub \) constructs the public key from a private key. There is no way to get the value \( key \) from the value \( pub(key) \).

Public key encryption \( E_K(M) \) means \( encrypt(M, K) \) and:

\[
\begin{align*}
decrypt(\text{encrypt}(M, pub(K)), K) &= M
\end{align*}
\]

Public key encryption is much less efficient than private key encryption so when two principals with public keys want to exchange data they often use a protocol to set up a symmetric “session” key and then use that symmetric key to send the data.

C. Signing

In schemes such as RSA messages encrypted with a private key can be decrypted with the public key. We could model this with the equation: \( decrypt(encrypt(M, K), pub(K')) = M \). This allows the private key owner to “sign” a message so that anyone with their public can verify that they signed it.
In practice we use a special signing operation for signing that leaves the text readable. To model this we use:

constructor functions: \( \text{sign}(\_), \text{pub}(\_) \)

destructor functions: \( \text{verify}(\_), \text{message}(\_) \)

with the equations:

\[
\text{verify}( \text{sign}(M, K) , \text{pub}(K) ) = M \\
\text{message}( \text{sign}(M, K) ) = M
\]

Signed messages provide proof that the message originated from the principal that signed the message. However, it does not prove that the message is fresh or tell us what the signer intended the message to mean. Many attacks on protocols are based on replaying sign messages out of context.

D. Hashing

Hashes, such as SHA, generate a unique short code from a larger message. It does not matter to us that the hash code is shorter than the original message so we model it with just one function:

constructor function: \( \text{hash}(\_) \)

Note that \( \text{hash}(M) = \text{hash}(N) \) if and only if \( M = N \). We can check a message against a hash, but there is no way to reconstruct a message from its hash.

E. Real Encryption

The equations presented so far tell us everything we need to know in order to analyse protocols. However, it is no bad thing to understand how the cryptographic processes really work and it is very important if you are ever going to implement a protocol.

- AES (the Advanced Encryption Standard) is one of the most popular symmetric key encryption algorithms: http://en.wikipedia.org/wiki/Advanced_Encryption_Standard
- RSA is one of the most popular public key encryption and signing algorithms: http://en.wikipedia.org/wiki/RSA
- The SHA family of algorithms are the most popular way of hashing a message: http://en.wikipedia.org/wiki/SHA