Cryptography and Secure Channels

Roger M. Needham

Huygens Systems Research Laboratory
Universiteit Twente
Enschede
Secure Channel

A secure channel between two principals $A$ and $B$ is a mechanism whereby $A$ can send messages to $B$ over a public network in such a manner that one or more of the following are true:

1. The data $A$ sent cannot be altered without detection
2. The data can only be read by $B$
3. The data is attributed to $A$ and $A$ cannot deny this

Note that 3 implies 1, but that 3 is much stronger than 1.
Secure Channels in Centralized Systems

P1

P2

P3

P4

Kernel
Encryption functions $D$ and $E$

Text $T$ encrypted with key $K$ is denoted $E(K, T)$ or $\{T\}_K$

$K_1$ and $K_2$ are cognate if $D(K_2, E(K_1, T)) = T$

If $K_1 = K_2$ then the cryptosystem is called symmetric, otherwise it is asymmetric
Asymmetric Cryptosystems

Usually referred to as Public-Key cryptosystems, because one key is usually made public and the other kept secret.

When $K_1$ and $K_2$ are cognate, we write $K$ and $K^{-1}$.

Public keys will be denoted by $K$, secret ones by $K^{-1}$.
• Without knowing $K_2$, it must be \textit{computationally infeasible} to find $T$, given $E(K_1, T)$.

• Given $T$ and $E(K_1, T)$, it must be computationally infeasible to find $K_1$ (unless $K_1$ will be used once only).

• In an asymmetric cryptosystem, given $K_1$ (or $K_2$), it must be computationally infeasible to find $K_2$ (or $K_1$).
Assume $K_A$ is public knowledge and known to be associated with principal $A$ ("A’s public key $K_A$ is in the phone book") and $K_A^{-1}$ is kept secret by $A$.

In all asymmetric cryptosystems in use today, when $D(K, E(K^{-1}, T)) = T$, also $E(K^{-1}, D(K, T)) = T$.

A message encrypted with $K_A^{-1}$ is said to be signed by $A$. 
Believing Signatures

Assume $A$ signs $M$ by encrypting it with his secret key $K_A^{-1}$ and $B$ then decrypts the result with $A$’s public key $K_A$.

$B$ believes that $A$ encrypted $M$ because only $A$ knows $K_A^{-1}$; $B$ does not know this.
Believing Signatures

$B$ must be sure he has $M$ — he could compute $D(K_A, X)$ for any $X$ and produce false candidates for $M$

$B$ must know $M$ to be “good”:

- $B$ knows $M$ in advance
- $B$ knows parts of $M$ in advance
- $B$ knows that $M$ has structural redundancy which will not occur by chance

An adversary must not be able to construct a message with an acceptable structure
A can only deny having signed $M$ by stating that $K^{-1}_{A}$ is not secret any more.
Confidentiality

A signed message is not confidential. Anyone can decrypt A’s signed message by using public key $K_A$.

To make $M$ confidential — for B’s eyes only — it should be encrypted with B’s public key $K_B$; B can decrypt this with $K_B^{-1}$.
A must believe that $K_B$ is $B$’s public key and not somebody else’s

How $B$ can be convinced of such a thing is not trivial and will be discussed in the lecture on Chapter 21
Confidential messages are often signed as well. If this is done, the correct order is

\[ E(K_B, E(K_A^{-1}, M)) \]

Never sign something you cannot read!
Suppose $A$ and $B$ share a key $K_{AB}$ and $A$ and $B$ both believe that $K_{AB}$ is a secret of $A$ and $B$ only.

If a trusted third party $AS$ (an authentication server) provided $A$ and $B$ with $K_{AB}$, $A$ and $B$ must believe that $AS$ will not use $K_{AB}$ or disclose it to others.

Then, a message from $A$ to $B$ or vice versa, encrypted with $K_{AB}$ will be both confidential and signed.
There exist functions called *message digests* or *secure hashes* that take a message and produce a fixed-size number (say 128 bits) in such a manner that it is computationally infeasible to produce different $M$ and $M'$ that yield the same digest.

Denoting the digest function by $C$, a signed and confidential message can then be transmitted as

$$E(K_B, \{M, E(K_A^{-1}, C(M))\})$$
The RSA Algorithm

Proposed by Rivest, Shamir and Adleman

Let $n = p \cdot q$ be the product of two large primes of comparable size

Then $\phi(n) = (p - 1) \cdot (q - 1)$ (Euler’s function) and

$\forall x : x^{\phi(n)} \equiv 1 \pmod{n}$

Choose $d$ and $e$ such that $d \cdot e \equiv 1 \pmod{\phi(n)}$ and

let $K = \{e, n\}$ and $K^{-1} = \{d, n\}$

Define $E(K, M) \overset{\text{def}}{=} M^e \pmod{n}$

and $D(K^{-1}, X) \overset{\text{def}}{=} X^d \pmod{n}$
\[ D(K^{-1}, E(K, M)) \equiv (M^e \pmod{n})^d \pmod{n} \]
\[ \equiv (M^e)^d \pmod{n} \]
\[ \equiv M^{e \cdot d} \pmod{n} \]
\[ (\exists k :) \equiv M^{k \cdot \phi(n)+1} \pmod{n} \]
\[ \equiv M^{\phi(n)k} \cdot M^1 \pmod{n} \]
\[ \equiv 1^k \cdot M \pmod{n} \]
\[ \equiv M \pmod{n} \]

If \( M \in [0 \ldots n - 1] \) \( \Rightarrow \) \( D(K^{-1}, E(K, M)) = M \)
The security of RSA depends on the difficulty of factoring \( n \).
If you know \( p \) and \( q \), you know \( \phi(n) \).
If you know \( \phi(n) \) and \( e \), you can find \( d \).
If you choose \( n \) to be the product of two primes of 300 bits or more, factoring \( n \) is well beyond the current state of the art.
Choosing 500-bit primes is probably good enough until the year 2000.
DES is a symmetric encryption algorithm that takes blocks of 64 bits and encrypts them under a 56-bit key.

$G$ is a one-way function; the basic step below is iterated and decorated.
Symmetric encryption is orders of magnitude faster than asymmetric encryption (see Table 21.1, p. 554)
Computing message digests is faster than symmetric encryption
Symmetric encryption in hardware can run at network speeds
Asymmetric encryption is rarely use for bulk data; it is used a lot for channel set-up and exchanging a symmetric-encryption key
Attacks

- Attacks on the cryptographic algorithm
- Attacks on the messages
- Attacks on keys
- Attacks on the protocol
Kinds of attack

- Ciphertext-only
- Known plaintext
- Chosen plaintext
Encrypting as little as possible with long-term keys is best. This can be done by replacing $E(K, M)$ by $E(K, K')E(K', M)$ where $K$ is the long-term key and $K'$ is a key for the occasion. This is similar to transmitting a (symmetric) session key using a long-term (asymmetric) authentication key.
To get somebody to prove he has a certain key now, it is common to use a challenge-response protocol:

“If you are Joe, encrypt this number with $K^{-1}_{Joe}$”

Joe can avoid a chosen-plaintext attack by returning:

“I’ve encrypted your number XOR-ed with a number I chose, here is my number and the encryption”
X.509 is the CCITT’s recommendation for authentication and privacy.

It uses a message digest based on a multiplicative function.

Using this message digest combined with RSA — which is also based on multiplication — is highly dangerous.
When data is transmitted using a block cipher, it is possible to alter data by replacing or reordering blocks. This can be prevented by using a checksum that detects permutations; a message digest would, of course, do fine. Simple CRCs can be combined with Cipher Block Chaining mode encryption; here information from the encryption is of one block is carried forward into the next.
Binary keys are difficult to remember. Keys are therefore often derived from passwords (DES has a 56-bit key: 8 seven-bit ASCII characters)

The derivation usually reduces the size of the key space, sometimes to the extent that keys can easily be guessed

Passwords are often chosen from a guessable space

Kerberos derives keys from passwords and is vulnerable to this attack
Attacks on the Protocol

An enemy can squirrel away messages from many runs of protocol (without being able to decipher them)
Some protocols can be fooled by the insertion of old messages
For instance, $C$ could perhaps convince $A$ and $B$ to use a session key that was previously used by $A$ and $C$
A message is *fresh* if it belongs to the current run of a protocol. Freshness can be guaranteed by the inclusion of timestamps or *nonces*.
If authentication-protocol messages are untyped, it is sometimes possible to convince a party in the protocol to accept a replayed message $i$ as message $j$

A draft proposal for X.509 could be broken this way
A channel can be defined as *the series of messages encrypted with A’s secret key* $K_A^{-1}$

Everything that arrives on this channel can be assumed having been sent by $A$

We may name this channel by the key that decrypts its traffic, $K_A$

We say that

Channel $K_A$ *speaks for* $A$
A secure channel is established when, provided two communicating parties are indeed who they claim to be, these parties obtain a shared, secret key.
Example: Kerberos

A and B wish to communicate, S is a mutually trusted Authentication server

A and B share symmetric keys $K_{AS}$ and $K_{BS}$, respectively with S

$T_X$ is the time as perceived by X

$K_{AB}$ is the key that will be given by S to A and B for their secret communication

1. $A \rightarrow S \; A, \; B$
2. $S \rightarrow A \; E(K_{as}, \; (T_s, \; K_{ab}, \; B, \; E(K_{bs}, \; (T_s, \; K_{ab}, \; A))))$
3. $A \rightarrow B \; E(K_{bs}, \; (T_s, \; K_{ab}, \; A)), \; E(K_{ab}, \; (A, \; T_a))$
4. $B \rightarrow A \; E(K_{ab}, \; (T_a + 1))$
What is a party?

Getting security right requires distinguishing between a workstation, the person behind the workstation, the operating system of the workstation and the processes running on the workstation.

In the absence of a personal encryption device (smart card), a person must trust the workstation and its operating system not to divulge his secret (Password).

Processes running on a machine must always trust that machine and its operating system.

Encryption channels between machines can therefore be used to *multiplex* the channels of the processes on those machines.
Assume the parties $A$ and $B$ know each other’s public keys $K_A$ and $K_B$

$A$ nor $B$ trusts the other to invent a *good* key, but they trust themselves

$A$ and $B$ both generate a candidate key $L_A$ and $L_B$, respectively

$A$ sends $E(K_A^{-1}, L_A)$ to $B$ and $B$ sends $E(K_B^{-1}, L_B)$ to $A$

$A$ and $B$ decrypt these messages and compute $K_{AB} = L_A \text{ XOR } L_B$
If you believe $L$ is a *good* key, then you must believe that $L \oplus L'$ is a good key, provided $L'$ was chosen *independently* of $L$

In the previous protocol, $A$ can force a particular key on $B$. or vice versa

A better protocol has $A$ send a *digest* of $L_A$ to $B$

Then $B$ sends $L_B$

Finally, $A$ sends $L_A$

$B$ must verify that the digest of $L_A$ matches what he received
Cryptography is a tricky business fraught with subtlety
It is not for amateurs
Most authentication protocols that were published were wrong (including Needham-Schroeder)