Cryptographic Protocol Building Blocks

Sources:

- *Applied Cryptography* by B. Schneier
- *Making, Breaking Codes: An Introduction to Cryptology* by P. Garrett
- *Cryptography Decrypted* by H.X. Mel and D. Baker
- *Security in Computing* by C.P. Pfleeger
- *Cryptography and Network Security* by W. Stallings
- *Modern Cryptography* by W. Mao

Cryptographic Protocols

A *protocol* is a set of steps followed by two or more parties in carrying out a task.

A *cryptographic protocol* is a protocol that involves some cryptographic algorithm.

Cryptography is used to prevent or detect cheating or eavesdropping.

Types of protocols:

- Arbitrated
- Adjudicated
- Self-enforcing

An *arbitrated protocol* involves a disinterested third party that is trusted to carry out certain steps.

Example: Using certified check to buy car

1. Buyer writes check and gives it to bank

2. Bank sets aside enough of buyer's funds to cover check, certifies it, and gives it back to buyer

3. Seller gives title to buyer and buyer gives certified check to seller

4. Seller deposits check

Problems with computer arbitrators:

- Parties *may not trust* an arbitrator they don't know.

- There is *delay* inherent in an arbitrated protocol.

- The arbitrator is a performance *bottleneck* and a point of *vulnerability*.

An *adjudicated protocol* consists of two subprotocols.

- One subprotocol is *not arbitrated* and is used *every time* parties want to complete the overall protocol.

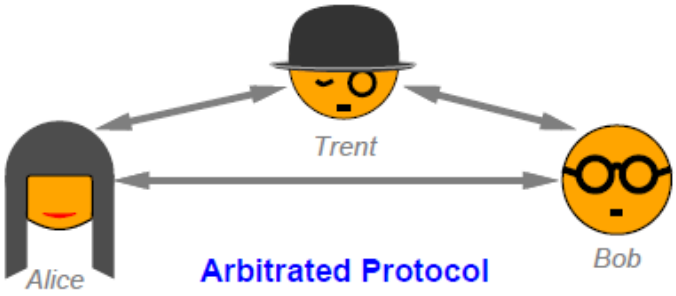
- The other subprotocol is *arbitrated* and is used only when there is a *dispute*.

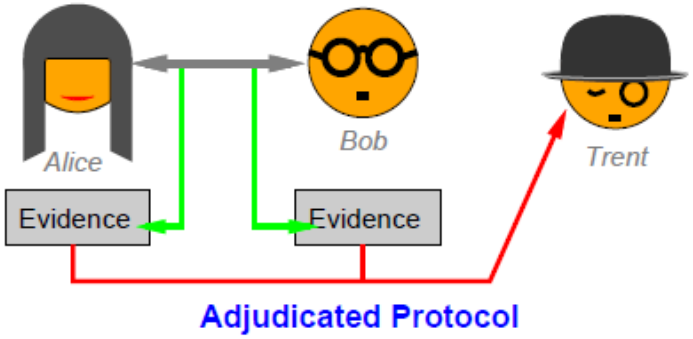
The arbitrator in the second subprotocol is called an *adjudicator*.

Example: judge in contract dispute

A *self-enforcing protocol* ensures fairness itself.

If one party tries to cheat, the other party immediately detects this and the protocol stops.





[wwwhome.cs.utwente.nl/~sape/sse/schneier02.pdf]

Attacks against Protocols

Attacks against cryptographic protocols may focus one any of the following:

- Cryptographic algorithms used in the protocol
- Cryptographic techniques used to implement the algorithms or protocols
- The protocols themselves

Attacks may be active or passive.

In a *passive attack*, the attacker gathers information but does not affect the protocol.

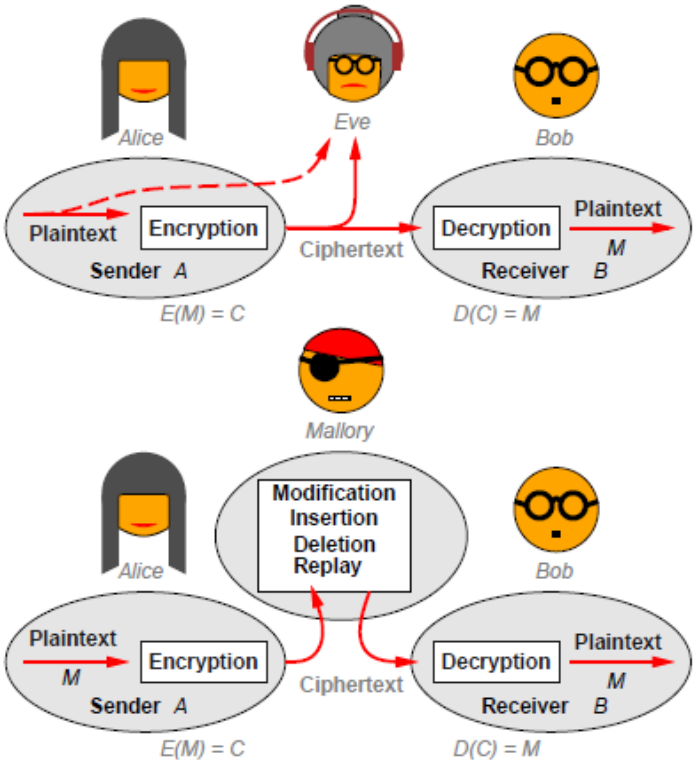
In an *active attack*, the attacker tries to alter the protocol, e.g., by:

- Inserting, deleting, or substituting messages
- Interrupting the communication channel
- Altering information stored on a computer

An attacker can be one of the parties involved in the protocol, in which case they are called a *cheater*.

A *passive cheater* tries to obtain more information than the protocol allows.

An *active cheater* disrupts the protocol in progress.



Passive vs. Active Attacks

[wwwhome.cs.utwente.nl/~sape/sse/schneier02.pdf]

Communications Using Symmetric Cryptography

Protocol:

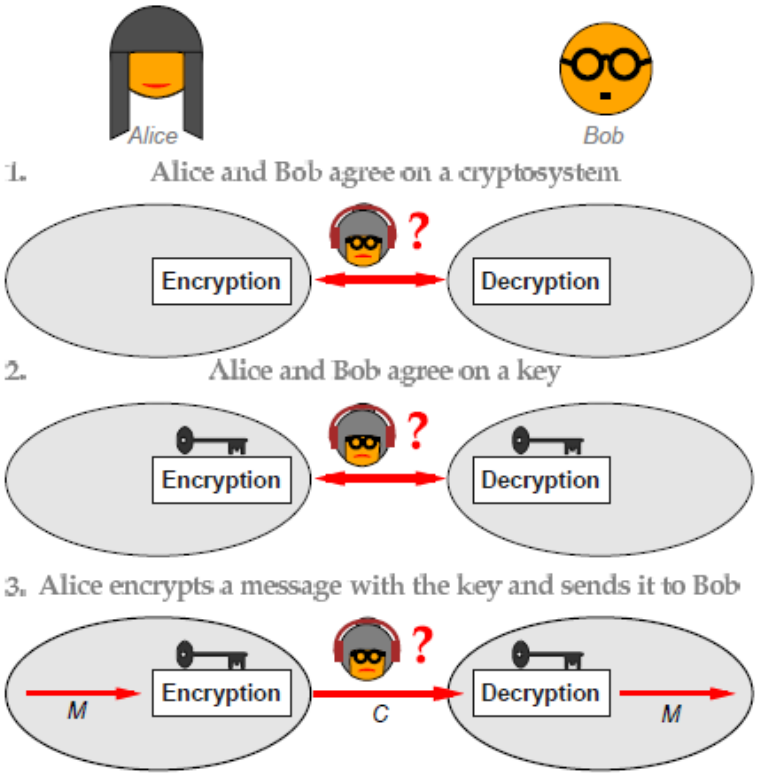
1. Message sender and receiver agree on a cryptosystem

2. Sender and receiver agree on a key

3. Sender encrypts message using encryption algorithm and key

4. Sender transmits ciphertext to receiver

5. Receiver decrypts ciphertext with same algorithm and key and reads the plaintext



Symmetric Cryptography

[wwwhome.cs.utwente.nl/~sape/sse/schneier02.pdf]

Problems with symmetric cryptosystems:

- Keys must be distributed in secret.
- If a key is compromised, the attacker can decrypt any message encrypted with it or impersonate one of the parties.
- The total number of keys required for n users is $n(n - 1)/2$.

One-Way Functions

"One-way functions" are a fundamental building block of many cryptographic protocols.

They are central to public-key cryptography.

A *one-way function* is a function f that is relatively easy to compute but whose inverse f^{-1} is hard to compute.

More formally, f should be computable in polynomial time, but f^{-1} should not be.

Example: Computing x^2 in a finite field is easy; computing $x^{1/2}$ is thought to be intractable.

An additional requirement is that a one-way function should be hard to invert on "almost all" points in its range.

A *trapdoor one-way function* is a one-way function that is easy to invert given a secret *inverse key*.

One-Way Hash Functions

"One-way hash functions" are another building block for many cryptographic protocols.

A *hash function* takes a variable-length input and produces a fixed-length *hash value* (typically 128-512 bits).

Hash functions are generally many-to-one, but good ones distribute hash values randomly.

The hash value for an input is a sort of "fingerprint" for it.

A *one-way hash function* (*cryptographic hash function, message digest*) is a one-way function that is also a hash function.

Thus, it is hard to generate a pre-image that hashes to a particular value.

A good one-way hash function is also *collision-free*: it is computationally hard to generate two pre-images that hash to the same value.

A one-bit change in the pre-image changes half of the bits in the hash value on average.

A one-way hash function can be used to "fingerprint" files.

A *message authentication function* (*MAC*) is a one-way hash function with a secret key.

The hash value is a function of both the pre-image and the key.

This means the key is required to verify the hash value.

Communications Using Public-Key Cryptography

In 1976, Whitfield Diffie and Martin Hellman described *public-key cryptography*.

Recall that this involves two keys – one public and one private.

It is computationally hard to deduce the private key from the public one.

Anyone with the public key can encrypt a message, but only the person with the private key can decrypt it.

Public key cryptography is based on trap-door one-way functions.

Encryption is the easy direction; decryption is the hard direction.

The secret or trapdoor is the private key.

Protocol for sending a message with public-key cryptography:

1. Message sender and receiver agree on a public-key cryptosystem

2. Message receiver transmits his public key to message sender

3. Sender encrypts message using public key and transmits ciphertext to receiver

4. Receiver uses private key to decrypt ciphertext

Public-key cryptography *solves* the *key management problem*.

A secure message can be sent with no prior arrangements.

More typically, however, a *network of users* agrees on a public key cryptosystem.

Every user has their own public key and private key.

All public keys are published somewhere.

Modified protocol:

1. Sender gets receiver's public key from database

2. Sender encrypts message using receiver's public key and transmits ciphertext to receiver

3. Receiver decrypts the ciphertext using his private key

These preceding two protocols assume that the sender is certain they know the intended receiver's actual public key.

They can be compromised by an attacker who

1. *Substitutes* their own public key for the intended receiver's
2. *Intercepts* the ciphertext encrypted with it

Hybrid Cryptosystems

Public-key algorithms are not a substitute for symmetric algorithms:

- Symmetric algorithms are generally 1000 times faster than public key algorithms.

- Public-key cryptosystems are vulnerable to chosen-plaintext attacks, because the cryptanalyst can encrypt trial messages.

For these reasons, public-key algorithms are used to encrypt *keys* instead of messages.

Typically, public-key cryptography is used to *secure and distribute session keys* for use with a symmetric algorithm.

Hybrid cryptographic protocol with session key [Schneier]:

1. Bob transmits his public key to Alice

2. Alice generates *random session key* *K*, encrypts it using the Bob's public key, and transmits it to him

3. Bob decrypts Alice's message using his private key to recover the session key

4. Both parties encrypt their communication using the *same session key*

The session key is *destroyed* when it is no longer needed.

This reduces the risk of it becoming compromised.

The private key is at risk of being compromised, but it is used only *once* per communication.

Digital Signatures

Handwritten signatures have long been used to prove authorship of or agreement to the contents of a document.

Ideally, signatures have the following properties [Schneier]:

1. The signature convinces the document's recipient that the signer deliberately signed the document.

2. The signature is *unforgeable*.

3. The signature *cannot be transferred* to other documents.

4. The signed document is *unalterable*.

5. The signature *cannot be repudiated*.

(Of course, handwritten signatures don't completely satisfy these properties in practice.)

There are obstacles to doing the same sort of thing with computers:

- Computer files are easy to copy.
- Even if a person's signature is hard to forge, it'd be easy to cut and paste an image of it.
- Computer files are easy to modify after they are signed.

Public-key algorithms like RSA can be used for *digital signatures*.

Encrypting a document with your private key amounts to a secure digital signature:

1. The sender *encrypts* the document with his *private key*, thereby signing the document.
2. The sender transmits the signed document to the receiver.
3. The receiver *decrypts* the document using the sender's *public key*, thereby *verifying* the signature.

The digital signature comes close to satisfying the desired properties (1-5 above) of handwritten signatures.

The sender can cheat in some circumstances by reusing the document and signature together.

Example: retransmitting digital check

Hence, a *timestamp* is often attached to the message and signed.

The timestamp is recorded by the receiver along with the message.

Public key algorithms may be *too slow* for signing large documents.

To save time, digital signature protocols often employ *one-way hash functions*.

The *hash* of a document is signed instead of the document itself.

Both the one-way hash function and the digital signature algorithm are agreed upon in advance.

Digital signature protocol with one-way hash function [Schneier]:

1. Alice produces a one-way hash of a document.

2. Alice encrypts the hash with her private key, thereby signing the document.

3. Alice sends the document and her signed hash to Bob.

4. Bob produces a one-way hash of the document Alice sent. Using the digital signature algorithm, he decrypts the signed hash with Alice's public key. If the signed hash matches the hash he generated, the signature is valid.

The storage requirements for the document and signature are much lower than with a symmetric algorithm.

The signature can also be kept separate from the document.

There are many digital signature algorithms.

All of them are public key algorithms.

Secret information is used to sign documents and public information is used to verify signatures.

We will denote signing a message with a private key K by $S_K(M)$.

Verifying a signature with the corresponding public key is denoted $V_K(M)$.

Multiple Keys

Using one-way hash functions, it is easy to sign a document *multiple times* [Schneier]:

1. Alice signs the hash of the document.

2. Bob signs the hash of the document.

3. Bob transmits his signature to Alice.

4. Alice transmits the document, her signature, and Bob's signature to Carol.

5. Carol verifies both Alice's signature and Bob's signature.

Nonrepudiation and Digital Signatures

A *direct* digital signature involves only the sender and receiver of a message.

Someone who signs a document with a direct digital signature can later *disavow* the signature and claim their private key was lost or stolen.

This problem can be addressed by using an *arbitrated* digital signature.

Arbitrated digital signature protocol [Schneier]:

1. Alice signs a message.

2. Alice creates a header containing some identifying information. She concatenates the header with the signed message, signs that, and transmits it to the arbitrator.

3. The arbitrator verifies the outside signature and confirms the identifying information. Then he signs it all and sends it to both Alice and Bob.

4. Bob verifies the arbitrator's signature, the identifying information, and Alice's signature.

5. Alice verifies the message the arbitrator sent to Bob. If she did not originate the message, she announces this promptly.

Digital Signatures with Encryption

Digital signatures can be used together with cryptography to both ensure *privacy* and provide *proof of authorship* [Schneier]:

1. Alice signs the message with her private key.

   $S_A(M)$

2. Alice encrypts the signed message with Bob's public key and sends it to Bob.

   $E_B(S_A(M))$

3. Bob decrypts the message with his private key.

   $D_B(E_B(S_A(M))) = S_A(M)$

4. Bob verifies with Alice's public key and recovers the message.

   $V_A(S_A(M)) = M$

Random and Pseudo-Random Sequence Generation

Algorithmic random number generators produce *pseudo-random sequences* not true random sequences.

Ordinary random number generators are not secure enough for cryptographic applications.

Informally, a random number generator is *cryptographically secure* if, starting with a *seed s* of *n true random bits*, the generator uses it to produce, in time *polynomial* in *n*, a *sequence of bits*

$$X_1, X_2, \ldots, X_{n^k}$$

with the property that, given the generator and the first *t* of the X_i, but *not* the seed *s*, it is *computationally infeasible to predict* X_{t+1} with better than 0.5 probability of success.

Such a generator is said to pass the *next-bit test*.