Security Engineering

Lesson 4
Protocols: Basic and Intermediate

Spring 2010
Dr. Marenglen Biba

Outline

3.1 Key Exchange

3.2 Authentication

3.3 Authentication and Key Exchange

3.4 Formal Analysis of Authentication and Key-Exchange Protocols

3.5 Multiple-Key Public-Key Cryptography

3.6 Secret Splitting

3.7 Secret Sharing

3.8 Cryptographic Protection of Databases

Key Exchange

- A common cryptographic technique is to encrypt each individual conversation with a **separate key**.
- This is called a **session key**, because it is used for only one particular communications session.
- Session keys are useful because they only exist for the duration of the communication.
- How this common session key gets into the hands of the conversants can be a **complicated matter**.

Key Exchange with Symmetric Cryptography

- This protocol assumes that Alice and Bob, users on a network, each share a secret key with the Key Distribution Center (KDC) - Trent in our protocols.
  - **(1)** Alice calls Trent and requests a session key to communicate with Bob.
  - **(2)** Trent generates a **random session key**. He encrypts two copies of it: one in Alice's key and the other in Bob's key. Trent sends both copies to Alice.
  - **(3)** Alice decrypts her copy of the session key.
  - **(4)** Alice sends Bob his copy of the session key.
  - **(5)** Bob decrypts his copy of the session key.
  - **(6)** Both Alice and Bob use this session key to communicate securely.

Problems of the protocol

- This protocol relies on the absolute security of Trent, who is more likely to be a **trusted computer program** than a trusted individual.
- If Mallory **corrupts** Trent, the whole network is compromised.
  - He has all of the secret keys that Trent shares with each of the users; he can read **all past communications traffic** that he has saved, and all future communications traffic. All he has to do is to tap the communications lines and listen to the encrypted message traffic.
- The other problem with this system is that Trent is a **potential bottleneck**. He has to be involved in every key exchange. If Trent fails, that disrupts the entire system.

Key Exchange with Public-Key Cryptography

- Alice and Bob use public-key cryptography to agree on a session key, and use that session key to encrypt data.

  (1) Alice gets Bob's public key from the KDC.

  (2) Alice generates a **random session key**, encrypts it using Bob's public key, and sends it to Bob.

  (3) Bob then decrypts Alice's message using his private key.

  (4) Both of them encrypt their communications using the same session key.

Man-in-the-Middle Attack

(1) Alice sends Bob her public key. Mallory intercepts this key and sends Bob his own public key.

(2) Bob sends Alice his public key. Mallory intercepts this key and sends Alice his own public key.

(3) When Alice sends a message to Bob, encrypted in "Bob's" public key, Mallory intercepts it. Since the message is really encrypted with **his own public key**, he decrypts it with his private key, re-encrypts it with Bob's public key, and sends it on to Bob.

(4) When Bob sends a message to Alice, encrypted in "Alice's" public key, Mallory intercepts it. Since the message is really encrypted with his own public key, he decrypts it with his private key, re-encrypts it with Alice's public key, and sends it on to Alice.

Man-in-the-Middle Attack

- Even if Alice's and Bob's public keys are stored on a **database**, this attack will work. Mallory can **intercept** Alice's database inquiry and **substitute** his own public key for Bob's.
- He can do the same to Bob and substitute his own public key for Alice's. Or better yet, he can break into the database and substitute his key for both Alice's and Bob's.
- This **man-in-the-middle attack** works because Alice and Bob **have no way to verify** that they are talking to each other.
- Assuming Mallory doesn't cause any noticeable network delays, the two of them have no idea that someone sitting between them is reading all of their supposedly secret communications.

Interlock Protocol

- The **interlock protocol**, invented by Ron Rivest and Adi Shamir, has a good chance of defeating the man-in-the-middle attack. Here's how it works:

  (1) Alice sends Bob her public key.

  (2) Bob sends Alice his public key.

  (3) Alice encrypts her message using Bob's public key. She sends **half of the encrypted message** to Bob.

  (4) Bob encrypts his message using Alice's public key. He sends **half of the encrypted message** to Alice.

  (5) Alice sends the other half of her encrypted message to Bob.

  (6) Bob puts the two halves of Alice's message together and decrypts it with his private key. Bob sends the other half of his encrypted message to Alice.

  (7) Alice puts the two halves of Bob's message together and decrypts it with her private key.

Interlock Protocol

- The important point is that half of the message is useless without the other half; **it can't be decrypted**. Bob cannot read any part of Alice's message until step (6); Alice cannot read any part of Bob's message until step (7).
- There are a number of ways to do this:
  - If the encryption algorithm is a block algorithm, half of each block (e.g., every other bit) could be sent in each half message.
  - Decryption of the message could be dependent on an **initialization vector** which could be sent with the second half of the message.
  - The first half of the message could be a **one-way hash function** of the encrypted message and the encrypted message itself could be the second half.

Interlock Protocol

- Mallory can still substitute his own public keys for Alice's and Bob's in steps (1) and (2).
- But now, when he intercepts half of Alice's message in step (3), he **cannot decrypt it** with his private key and re-encrypt it with Bob's public key.
  - He must invent a **totally new message** and send half of it to Bob.
- When he intercepts half of Bob's message to Alice in step (4), he has the same problem. He **cannot decrypt it** with his private key and re-encrypt it with Alice's public key.
  - He has to invent a **totally new message** and send half of it to Alice.
- By the time he intercepts the second halves of the real messages in steps (5) and (6), it is too late for him to change the new messages he invented since Alice and Bob know half of the message..
  - **The conversation between Alice and Bob will necessarily be completely different**.

Key Exchange with Digital Signatures

- Implementing digital signatures during a session-key exchange protocol circumvents the man-in-the-middle attack as well.

- Trent **signs** both Alice's and Bob's public keys. The signed keys include a **signed certification** of ownership.

- When Alice and Bob receive the keys, they each **verify Trent's signature**. Now they know that the public key belongs to that other person.

- The key exchange protocol can then proceed.

Key Exchange with Digital Signatures

- Mallory cannot impersonate either Alice or Bob because he doesn't know **either of their private keys**.

- He cannot substitute his public key for either of theirs because, while he has one **signed by Trent**, it is signed as being Mallory's.

Key and Message Transmission

- Alice and Bob **need not complete** the key-exchange protocol before exchanging messages. In this protocol, Alice sends Bob the message, *M*, without any previous key exchange protocol:

  (1) Alice generates a **random session key**, *K*, and encrypts *M* using *K*.

  $E_K(M)$

  (2) Alice gets Bob's public key from the database.

  (3) Alice encrypts *K* with Bob's public key.

  $E_B(K)$

  (4) Alice sends both the encrypted message and encrypted session key to Bob.

  $E_K(M), E_B(K)$

  For added security against man-in-the-middle attacks, Alice can sign the transmission.

  (5) Bob decrypts Alice's session key, *K*, using his private key.

  (6) Bob decrypts Alice's message using the session key.

Key and Message Broadcast

Alice will send the encrypted message to Bob, Carol, and Dave:

(1) Alice generates a random session key, *K*, and encrypts *M* using *K*.

$E_K(M)$

(2) Alice gets Bob's, Carol's, and Dave's public keys from the database.

(3) Alice encrypts *K* with Bob's public key, encrypts *K* with Carol's public key, and then encrypts *K* with Dave's public key.

$E_B(K)$, $E_C(K)$, $E_D(K)$

(4) Alice broadcasts the encrypted message and all the encrypted keys to anybody who cares to receive it.

$E_B(K)$, $E_C(K)$, $E_D(K)$, $E_K(M)$

(5) Only Bob, Carol, and Dave can decrypt the key, *K*, each using his or her private key.

(6) Only Bob, Carol, and Dave can decrypt Alice's message using *K*.

Outline

3.1 Key Exchange

3.2 Authentication

3.3 Authentication and Key Exchange

3.4 Formal Analysis of Authentication and Key-Exchange Protocols

3.5 Multiple-Key Public-Key Cryptography

3.6 Secret Splitting

3.7 Secret Sharing

3.8 Cryptographic Protection of Databases

Authentication

- When Alice logs into a host computer (or an automatic teller, or a telephone banking system, or any other type of terminal), how does the host know who she is?
- How does the host know she is not Eve trying to falsify Alice's identity?
- Traditionally, **passwords** solve this problem. Alice enters her password, and the host confirms that it is correct.
- Both Alice and the host know this secret piece of knowledge and the host requests it from Alice every time she tries to log in.

Authentication Using One-Way Functions

- What Roger Needham and Mike Guy realized is that the host does not need to know the passwords; the host just has to be able to **differentiate** valid passwords from invalid passwords.
- This is easy with one-way functions. Instead of storing passwords, the host **stores one-way functions** of the passwords.
  - **(1)** Alice sends the host her password.
  - **(2)** The host performs a one-way function on the password.
  - **(3)** The host compares the result of the one-way function to the value it previously stored.
- Since the host **no longer stores a table** of everybody's valid password, the threat of someone breaking into the host and stealing the password list is mitigated.
- The list of passwords operated on by the one-way function is **useless**, because the one-way function **cannot be reversed** to recover the passwords.

Dictionary Attacks and Salt

- A file of passwords encrypted with a one-way function is still vulnerable.
- In his spare time, Mallory compiles a list of the 1,000,000 most common passwords.
- He operates on all 1,000,000 of them with the one-way function and stores the results.
- If each password is about 8 bytes, the resulting file will be no more than 8 megabytes; it will fit on a few floppy disks.
- Now, Mallory steals an encrypted password file. He **compares** that file with his file of encrypted possible passwords and sees what matches.
- This is a **dictionary attack**, and it's surprisingly successful!

Dictionary Attacks and Salt

- **Salt** is a way to make DA more difficult.
- Salt is a **random string** that is concatenated with passwords before being operated on by the one-way function.
- Then, both the salt value and the result of the one-way function are stored in a database on the host.
- If the number of possible salt values is large enough, this practically eliminates a dictionary attack against commonly used passwords:
  - Mallory has to **generate the one-way hash for each possible salt value**.

SKEY

- SKEY is an authentication program that relies on a one-way function for its security.
- To set up the system, Alice enters a random number, *R*. The computer computes *f*(*R*), *f*(*f*(*R*)), *f*(*f*(*f*(*R*))), and so on, about a hundred times. Call these numbers *x*1, *x*2, *x*3,..., *x*100. Alice takes the list for safekeeping. The computer also **stores *x*101**, in the clear, in a login database next to Alice's name.
- The first time Alice wants to log in, she types her name and *x*100. The computer calculates *f*(*x*100) and **compares** it with *x*101; if they match, Alice is authenticated. Then, the computer **replaces** *x*101 with *x*100 in the database. Alice crosses *x*100 off her list.
- **Every time Alice logs in, she enters the last uncrossed number on her list: *x*i**. The computer calculates *f*(*x*i) and compares it with *x*i+1 stored in its database.
- Eve can't get any useful information because each number is only used once, and the function is one-way. Similarly, the database is not useful to an attacker. Of course, when Alice runs out of numbers on her list, she has to reinitialize the system.

Authentication Using Public-Key Cryptography

- Even with salt, the protocol has serious security problems.
- When Alice sends her password to her host, **anyone** who has access to her data path can read it.
- She might be accessing her host through a convoluted transmission path that passes through four industrial competitors, three foreign countries, and two forward-thinking universities.
- Eve can be at any one of those points, listening to Alice's **login sequence**.
- If Eve has access to the processor memory of the host, she can see the password **before the host hashes it**.

Authentication Using Public-Key Cryptography

- Public-key cryptography can solve this problem. The host keeps a file of every user's public key; all users keep their own private keys.
- When logging in, the protocol proceeds as follows:

  (1) The host sends Alice a random string.

  (2) Alice encrypts the string with her **private key** and sends it back to the host, along with her name.

  (3) The host looks up Alice's **public key** in its database and decrypts the message using that public key.

  (4) If the decrypted string matches what the host sent Alice in the first place, the host allows Alice access to the system.

Secure proof-of-identity protocols

(1) Alice performs a computation based on some **random numbers** and her **private key** and sends the result to the host.

(2) The host sends Alice a **different** random number.

(3) Alice makes some computation based on the random numbers (both the ones she generated and the one she received from the host) and her private key, and sends the result to the host.

(4) The host does some computation on the various numbers received from Alice and her public key to verify that she knows her private key.

(5) If she does, her identity is verified.

Message Authentication

- When Bob receives a message from Alice, how does he know it is authentic?
- If Alice signed her message, this is easy. Alice's digital signature is enough to convince anyone that the message is authentic.
- **Symmetric cryptography** provides some authentication.
- When Bob receives a message from Alice encrypted in their shared key, he knows it is from Alice. No one else knows their key.
  - However, Bob has no way of **convincing a third party** of this fact.
  - Bob can't show the message to Trent and convince him that it came from Alice. Trent can be convinced that the message came from either Alice or Bob (since no one else shared their secret key), but he has no way of knowing which one

Outline

3.1 Key Exchange

3.2 Authentication

3.3 Authentication and Key Exchange

3.4 Formal Analysis of Authentication and Key-Exchange Protocols

3.5 Multiple-Key Public-Key Cryptography

3.6 Secret Splitting

3.7 Secret Sharing

3.8 Cryptographic Protection of Databases

Authentication and Key Exchange

- These protocols **combine authentication with key exchange** to solve a general computer problem: Alice and Bob are on opposite ends of a network and want to talk securely.
- How can Alice and Bob exchange a secret key and at the same time each be sure that he or she is talking to the other and not to Mallory?
- Most of the protocols assume that Trent shares a **different secret key** with each participant, and that all of these keys are in place before the protocol begins.

Symbols

TABLE 3.1
Symbols used in authentication and key exchange protocols

A	Alice's name
B	Bob's name
E_A	Encryption with a key Trent shares with Alice
E_B	Encryption with a key Trent shares with Bob
I	Index number
K	A random session key
L	Lifetime
T_A, T_B	A timestamp
R_A, R_B	A random number, sometimes called a **nonce**, chosen by Alice and Bob respectively

Wide-Mouth Frog

- The Wide-Mouth Frog protocol is probably the simplest symmetric key-management protocol that uses a trusted server.
- **Both Alice and Bob share a secret key with Trent.**
- The keys are just used for key distribution and not to encrypt any actual messages between users. Just by using two messages, Alice transfers a session key to Bob:
  - **(1)** Alice concatenates a **timestamp**, Bob's name, and a random session key and encrypts the whole message with the key she shares with Trent. She sends this to Trent, along with her name.

    $A, E_A(T_A, B, K)$
  - **(2)** Trent decrypts the message from Alice. Then he concatenates a new timestamp, Alice's name, and the random session key; he encrypts the whole message with the key he shares with Bob. Trent sends to Bob:

    $E_B(T_B, A, K)$

Yahalom

Alice and Bob share a secret key with Trent

(1) Alice concatenates her name and a random number, and sends it to Bob.

A,R_A

(2) Bob concatenates Alice's name, Alice's random number, his own random number, and encrypts it with the key he shares with Trent. He sends this to Trent, along with his name.

$B,E_B(A,R_A,R_B)$

(3) Trent **generates two messages**. The first consists of Bob's name, a **random session key**, Alice's random number, and Bob's random number, all encrypted with the key he shares with Alice. The second consists of Alice's name and the random session key, encrypted with the key he shares with Bob. He sends both messages to Alice.

$E_A(B,K,R_A,R_B),E_B(A,K)$

Yahalom

(4) Alice decrypts the first message, extracts K, and **confirms that R_A has the same value** as it did in step (1). Alice sends Bob two messages. The first is the message received from Trent, encrypted with Bob's key. The second is R_B, encrypted with the session key.

$E_B(A,K),E_K(R_B)$

(5) Bob decrypts the message encrypted with his key, extracts K, and **confirms that R_B has the same value** as it did in step (2).

At the end, Alice and Bob are each **convinced** that they are talking to the other and not to a third party.

The novelty here is that Bob is the **first one to contact Trent**, who only sends one message to Alice.

Needham-Schroeder

This protocol, invented by Roger Needham and Michael Schroeder, also uses **symmetric cryptography and Trent**.

(1) Alice sends a message to Trent consisting of her name, Bob's name, and a random number.

A,B,R_A

(2) Trent generates a **random session key.** He encrypts a message consisting of a **random session key** and Alice's name with the secret key he shares with Bob. Then he encrypts Alice's random value, Bob's name, the key, and the encrypted message with the **secret key he shares with Alice**. Finally, he sends her the encrypted message:

$E_A(R_A,B,K,E_B(K,A))$

(3) Alice decrypts the message and extracts K. She confirms that R_A is the same value that she sent Trent in step (1). Then she sends Bob the message that Trent encrypted in his key.

$E_B(K,A)$

Needham-Schroeder

(4) Bob decrypts the message and extracts K. He then generates another random value, R_B. He encrypts the message with K and sends it to Alice.

$E_K(R_B)$

(5) Alice decrypts the message with K. She generates R_B - 1 and encrypts it with K. Then she sends the message back to Bob.

$E_K(R_B - 1)$

(6) Bob decrypts the message with K and verifies that it is R_B - 1.

Replay attacks

- In this attack, Mallory can record old messages and then use them later in an attempt to subvert the protocol.
- The presence of R_A in step (2) assures Alice that Trent's message is legitimate and not a replay of a response from a previous execution of the protocol.
- When Alice successfully decrypts R_B and sends Bob R_B - 1 in step (5), Bob is ensured that Alice's messages are not replays from an earlier execution of the protocol.

Needham-Schroeder

- The major security hole in this protocol is that old session keys are valuable. **If Mallory gets access to an old *K*, he can launch a successful attack**.
  - All he has to do is record Alice's messages to Bob in step (3). Then, once he has *K*, he can pretend to be Alice:

(1) Mallory sends Bob the following message:

$E_B(K,A)$

(2) Bob extracts *K*, generates R_B, and sends Alice:

$E_K(R_B)$

(3) Mallory intercepts the message, decrypts it with *K*, and sends Bob:

$E_K(R_B - 1)$

(4) Bob verifies that **"Alice's"** message is $R_B - 1$.

Now, Mallory has Bob convinced that he is Alice.

Otway-Rees

(1) Alice generates a message consisting of an **index number**, her name, Bob's name, and a **random number**, all encrypted in the key she shares with Trent. She sends this message to Bob along with the index number, her name, and his name:

$I,A,B,E_A(R_A,\mathrm{I},\mathrm{A},\mathrm{B})$

(2) Bob generates a message consisting of a **new random number, the index number**, Alice's name, and Bob's name, all encrypted in the key he shares with Trent. He sends it to Trent, along with Alice's encrypted message, the index number, her name, and his name:

$I,A,B,E_A(R_A,\mathrm{I},\mathrm{A},\mathrm{B}),E_B(R_B,\mathrm{I},\mathrm{A},\mathrm{B})$

(3) Trent generates a **random session key**. Then he creates two messages. One is Alice's random number and the session key, encrypted in the key he shares with Alice. The other is Bob's random number and the session key, encrypted in the key he shares with Bob. He sends these two messages, along with the index number, to Bob:

$I,E_A(R_A,K),E_B(R_B,\mathrm{K})$

Otway-Rees

(4) Bob sends Alice the message encrypted in her key, along with the index number:

$I, E_A(R_A, K)$

(5) Alice decrypts the message to recover her key and random number. She then confirms that both have not changed in the protocol.

- Assuming that all the random numbers match, and the index number hasn't changed along the way, Alice and Bob are now convinced of each other's identity, and they have a secret key with which to communicate.

Kerberos

Kerberos is a variant of Needham-Schroeder. In the basic Kerberos Version 5 protocol, Alice and Bob each share keys with Trent. Alice wants to generate a session key for a conversation with Bob.

(1) Alice sends a message to Trent with her identity and Bob's identity.

A,B

(2) Trent generates a message with a timestamp, a lifetime L, a random session key, and Alice's identity. He encrypts this in the key he shares with Bob. Then he takes the timestamp, the lifetime, the session key, and Bob's identity, and encrypts these in the key he shares with Alice. He sends both encrypted messages to Alice.

$E_A(T,L,K,B),E_B(T,L,K,A)$

Kerberos

(3) Alice generates a message with her identity and the timestamp, encrypts it in *K*, and sends it to Bob. Alice also sends Bob the message encrypted in Bob's key from Trent.

$E_K(A,T),E_B(T,L,K,A)$

(4) Bob creates a message consisting of the timestamp plus one, encrypts it in *K*, and sends it to Alice.

$E_K(T + 1)$

- This protocol works, but it assumes that everyone's clocks are synchronized with Trent's clock.
  - In practice, the effect is obtained by synchronizing clocks to within a few minutes of a secure time server and detecting replays within the time interval.

Protocols: Lessons Learned

- There are some important lessons in the previous protocols, both those which have been broken and those which have not:
  - Many protocols failed because the designers tried to be too clever. They optimized their protocols by **leaving out important pieces**: names, random numbers, and so on. The remedy is to make everything explicit.
  - Trying to optimize depends a whole lot on the **assumptions you make**. For example: If you have authenticated time, you can do a whole lot of things you can't do if you don't.
  - The protocol of choice depends on the underlying **communications architecture.** Do you want to minimize the size of messages or the number of messages? Can all parties talk with each other or can only a few of them?

Outline

3.1 Key Exchange

3.2 Authentication

3.3 Authentication and Key Exchange

3.4 Formal Analysis of Authentication and Key-Exchange Protocols

3.5 Multiple-Key Public-Key Cryptography

3.6 Secret Splitting

3.7 Secret Sharing

3.8 Cryptographic Protection of Databases

The need for formal analysis

- The problem of establishing secure session keys between pairs of computers (and people) on a network is so fundamental that it has led to a great deal of research.
- Some of the research focused on the development of protocols like the ones discussed until now.
- This, in turn, has led to a greater and more interesting problem: the **formal analysis** of authentication and key-exchange protocols.
- People have found flaws in seemingly secure protocols years after they were proposed, and researchers wanted tools that could prove a protocol's security from the start. Although much of this work can apply to general cryptographic protocols, the emphasis in research is almost exclusively on authentication and key exchange.

Approaches

There are four basic approaches to the analysis of cryptographic protocols:

1. Model and verify the protocol using **specification languages** and **verification tools** not specifically designed for the analysis of cryptographic protocols.
2. Develop **expert systems** that a protocol designer can use to develop and investigate different scenarios.
3. Model the requirements of a protocol family using **logics** for the analysis of knowledge and belief.
4. Develop a formal method based on the **algebraic term-rewriting properties** of cryptographic systems.

BAN logic

- BAN logic is the most widely used logic for analyzing authentication protocols.
- It assumes that authentication is a **function of integrity and freshness**, and uses **logical rules** to trace both of those attributes through the protocol.
- Although many variants and extensions have been proposed, most protocol designers still refer back to the original work.
- BAN logic doesn't provide a proof of security; it can only reason about authentication. It has a simple, straightforward logic that is easy to apply and still useful for detecting flaws.

BAN logic

Some of the statements in BAN logic include:

- Alice believes X. (Alice acts as though X is true.)
- Alice sees X. (Someone has sent a message containing X to Alice, who can read and repeat X - possibly after decrypting it.)
- Alice said X. (At some time, Alice sent a message that includes the statement X. It is not known how long ago the message was sent or even that it was sent during the current run of the protocol. It is known that Alice believed X when she said it.)
- X is fresh. (X has not been sent in a message at any time before the current run of the protocol.)

Rules in BAN Logic

- BAN logic also provides rules for reasoning about belief in a protocol. These rules can then be applied to the logical statements about the protocol to prove things or answer questions about the protocol.
- For example, one rule is the message-meaning rule:
  - IF Alice believes that Alice and Bob share a secret key, *K*, and Alice sees *X*, encrypted under *K*, and Alice did not encrypt *X* under *K*, THEN Alice believes that Bob once said *X*.
- Another rule is the nonce-verification rule:
  - IF Alice believes that *X* could have been sent only recently and that Bob once said *X*, THEN Alice believes that Bob believes *X*.

BAN Logic

There are four steps in BAN analysis:

(1) Convert the protocol into idealized form, using the statements previously described.

(2) Add all assumptions about the initial state of the protocol.

(3) Attach logical formulas to the statements: assertions about the state of the system after each statement.

(4) Apply the logical postulates to the assertions and assumptions to discover the beliefs held by the parties in the protocol.

Outline

Multiple-Key Public-Key Cryptography

- Public-key cryptography can be extended to *n* keys.
- **If a given subset of the keys is used to encrypt the message, then the other keys are required to decrypt the message.**
- Example:
  - Alice can encrypt a message with K_A so that Ellen, with K_B and K_C, can decrypt it.

Three-Key Key Distribution

Alice	K_A
Bob	K^B
Carol	K_C
Dave	K_A and K_B
Ellen	K_B and K_C
Frank	K_C and K_A

Multiple-Key Public-Key Cryptography

All the possible combinations are summarized below; there are no other ones.

Three-Key Message Encryption	
Encrypted with Keys:	**Must be Decrypted with Keys:**
K_A	K_B and K_C
K_B	K_A and K_C
K_C	K_A and K_B
K_A and $_K B$	K_C
K_A and K_C	K_B
K_B and K_C	K_A

Outline

3.1 Key Exchange

3.2 Authentication

3.3 Authentication and Key Exchange

3.4 Formal Analysis of Authentication and Key-Exchange Protocols

3.5 Multiple-Key Public-Key Cryptography

3.6 Secret Splitting

3.7 Secret Sharing

3.8 Cryptographic Protection of Databases

Secret Splitting

- There are ways to take a message and **divide it up into pieces**.
- **Each piece by itself means nothing**, but put them together and the message appears.
- The simplest sharing scheme splits a message between two people. Here's a protocol in which Trent can split a message between Alice and Bob:

  (1) Trent generates a random-bit string, *R*, the same length as the message, *M*.

  (2) Trent XORs *M* with *R* to generate *S*.

  $M \,(+)\, R = S$

  (3) Trent gives *R* to Alice and *S* to Bob.

To reconstruct the message, Alice and Bob have only one step to do:

(4) Alice and Bob XOR their pieces together to reconstruct the message:

$R \,(+)\, S = M$

Secret Splitting

- This technique, if done properly, is absolutely secure. Each piece, by itself, is absolutely worthless.

- **No amount of computing power can determine the message from one of the pieces.**

- However, this protocol has a problem: If any of the pieces gets lost and Trent isn't around, so does the message!!!

Outline

3.1 Key Exchange

3.2 Authentication

3.3 Authentication and Key Exchange

3.4 Formal Analysis of Authentication and Key-Exchange Protocols

3.5 Multiple-Key Public-Key Cryptography

3.6 Secret Splitting

3.7 Secret Sharing

3.8 Cryptographic Protection of Databases

Secret Sharing

- Launch program for a nuclear missile
  - This is easy to solve. Make a **mechanical** launch controller.
  - Give each of the **five officers** a key and require that at least three officers stick their keys in the proper slots before you'll allow them to blow up whomever we're blowing up this week. (If you're really worried, make the slots far apart and require the officers to insert the keys simultaneously—you wouldn't want an officer who steals two keys to be able to vaporize Toledo.)
  - We can get even more complicated. Maybe the general and two colonels are authorized to launch the missile, but if the general is busy playing golf then five colonels are required to initiate a launch. Make the launch controller so that it requires five keys. Give the general three keys and the colonels one each. The general together with any two colonels can launch the missile; so can the five colonels. However, a general and one colonel cannot; neither can four colonels.

Threshold scheme,

- A more complicated sharing scheme, called a **threshold scheme**, can do all of this and more—mathematically.
- At its simplest level, you can take any message and divide it into *n* pieces, called **shadows or shares**, such that any *m* of them can be used to reconstruct the message. More precisely, this is called an **(*m*,*n*)-threshold scheme**.
- With a (3,4)-threshold scheme, Trent can divide his secret sauce recipe among Alice, Bob, Carol, and Dave, such that **any three of them** can put their shadows together and reconstruct the message. If Carol is on vacation, Alice, Bob, and Dave can do it. If Bob gets run over by a bus, Alice, Carol, and Dave can do it.
  - However, if Bob gets run over by a bus while Carol is on vacation, Alice and Dave can't reconstruct the message by themselves.

Outline

3.1 Key Exchange

3.2 Authentication

3.3 Authentication and Key Exchange

3.4 Formal Analysis of Authentication and Key-Exchange Protocols

3.5 Multiple-Key Public-Key Cryptography

3.6 Secret Splitting

3.7 Secret Sharing

3.8 Cryptographic Protection of Databases

Cryptography for Databases

- The membership database of an organization is a valuable commodity.
  - On the one hand, you want to **distribute the database** to all members. You want them to communicate with one another, exchange ideas.
  - On the other hand, if you **distribute the membership** database to everyone, copies are bound to fall into the hands of insurance salesmen and other annoying senders of junk mail.
- Cryptography can solve this problem.
- We can **encrypt the database** so that it is easy to extract the address of a single person but hard to extract a mailing list of all the members.

Cryptography for Databases

- Choose a one-way hash function and a symmetric encryption algorithm.
- Each record of the database has two fields.
  - The **index field** is the **last name of the member**, operated on by the one-way hash function.
  - The **data field** is the full name and address of the member, **encrypted using the last name as the key**.
- Unless you know the last name, you can't decrypt the data field.
- Searching a specific last name is easy.
  - First, hash the last name and look for the hashed value in the index field of the database.
  - If there is a match, then that last name is in the database.
- Finally, decrypt the full name and address using the last name as the key.

Break

0011

Intermediate Protocols

12

45

Intermediate Protocols

4.1 Timestamping Services

4.3 Undeniable Digital Signatures

4.4 Designated Confirmer Signatures

4.5 Proxy Signatures

4.6 Group Signatures

4.7 Fail-Stop Digital Signatures

4.9 Bit Commitment

Time in documents

- In the digital world there is no way to examine a digital document for signs of tampering.
- It can be copied and modified endlessly without anyone being the wiser. It's trivial to change the date stamp on a computer file. No one can look at a digital document and say: "Yes, this document was created before November 4, 1952."
- Haber and Stornetta at Bellcore thought about the problem They wanted a digital timestamping protocol with the following properties:
  - The data itself must be **timestamped**, without any regard to the physical medium on which it resides.
  - It must be **impossible to change a single bit** of the document without that change being apparent.
  - It must be impossible to timestamp a document with a date and time **different** from the present one.

Arbitrated Solution

- This protocol uses Trent, who has a trusted timestamping service, and Alice, who wishes to timestamp a document.
  - **(1)** Alice transmits a copy of the document to Trent.
  - **(2)** Trent records the date and time he received the document and retains a copy of the document for safekeeping.
- Now, if anyone calls into question Alice's claim of when the document was created, she just has to call up Trent. He will produce his copy of the document and verify that he received the document on the date and time stamped.

Arbitrated Solution

- This protocol works, but has some obvious problems.
- First, there is **no privacy**. Alice has to give a copy of the document to Trent. Anyone listening in on the communications channel could read it. She could encrypt it, but still the document has to sit in Trent's database. Who knows how secure that database is?
- Second, the database itself would have to be **huge**. And the bandwidth requirements to send large documents to Trent would be hard to manage.

Improved Arbitrated Solution

- One-way hash functions and digital signatures can clear up most of these problems easily:
  - **(1)** Alice produces a one-way hash of the document.
  - **(2)** Alice transmits the hash to Trent.
  - **(3)** Trent appends the date and time he received onto the hash and then digitally signs the result.
  - **(4)** Trent sends the signed hash with timestamp back to Alice.
  - Alice no longer has to worry about revealing the contents of her document; the hash is sufficient. Trent no longer has to store copies of the document (or even of the hash), so the massive storage requirements and security problems are solved (remember, one-way hash functions don't have a key). Alice can immediately examine the signed timestamped hash she receives in step (4), so she will immediately catch any transmission errors.
- The only problem remaining is that Alice and Trent can still collude to produce any timestamp they want.

Intermediate Protocols

4.1 Timestamping Services

4.3 Undeniable Digital Signatures

4.4 Designated Confirmer Signatures

4.5 Proxy Signatures

4.6 Group Signatures

4.7 Fail-Stop Digital Signatures

4.9 Bit Commitment

Undeniable Digital Signatures

- Normal digital signatures can be **copied exactly**.
- Sometimes this property is useful, as in the dissemination of public announcements.
- Other times it could be a problem. Imagine a digitally signed personal or business letter.
- If many copies of that document were floating around, each of which could be verified by anyone, this could lead to embarrassment or blackmail. The best solution is a digital signature that can be proven valid, but that the recipient cannot show to a third party without the signer's consent.

Undeniable Digital Signatures

- Like a normal digital signature, an undeniable signature **depends on the signed document and the signer's private key**.
- But unlike normal digital signatures, an undeniable signature cannot be verified without the **signer's consent**.
  - **(1)** Alice presents Bob with a signature.
  - **(2)** Bob generates a random number and sends it to Alice.
  - **(3)** Alice does a calculation using the random number and her private key and sends Bob the result. **Alice could only do this calculation if the signature is valid**.
  - **(4)** Bob confirms this.

Intermediate Protocols

- 4.1 Timestamping Services
- 4.3 Undeniable Digital Signatures
- **4.4 Designated Confirmer Signatures**
- 4.5 Proxy Signatures
- 4.6 Group Signatures
- 4.7 Fail-Stop Digital Signatures
- 4.9 Bit Commitment

Designated Confirmer Signatures

- Alice would like a way to designate one particular person to be in charge of **signature verification** for the whole company.
  - Alice, or any other programmer, would be able to sign documents with an undeniable protocol. But the verifications would all be handled by Carol.
- As it turns out, this is possible with **designated confirmer signatures.**
- Alice can sign a document such that Bob is convinced the signature is valid, but he cannot convince a third party;
- At the same time Alice can designate Carol as the future confirmer of her signature. Alice doesn't even need to ask Carol's permission beforehand; **she just has to use Carol's public key**. And Carol can still verify Alice's signature if Alice is out of town or has left the company.

Intermediate Protocols

4.1 Timestamping Services

4.3 Undeniable Digital Signatures

4.4 Designated Confirmer Signatures

4.5 Proxy Signatures

4.6 Group Signatures

4.7 Fail-Stop Digital Signatures

4.9 Bit Commitment

Proxy Signatures

- Designated confirmer signatures allows a signer to designate someone else to verify his signature.
- Alice, for instance, needs to go on a business trip to someplace which doesn't have very good computer network access—to the jungles of Africa, for example. Or maybe she is incapacitated after major surgery.
- She expects to receive some important e-mail, and has instructed her secretary Bob to respond accordingly.
- **How can Alice give Bob the power to sign messages for her, without giving him her private key**?

Proxy Signatures

- **Proxy signatures** is a solution. Alice can give Bob a **proxy**, such that the following properties hold:
  - ***Distinguishability***. Proxy signatures are distinguishable from normal signatures by anyone.
  - ***Unforgeability***. Only the original signer and the designated proxy signer can create a valid proxy signature.
  - ***Proxy signer's deviation***. A proxy signer cannot create a valid proxy signature not detected as a proxy signature.
  - ***Verifiability***. From a proxy signature, a verifier can be convinced of the original signer's agreement on the signed message.
  - ***Identifiability***. An original signer can determine the proxy signer's identity from a proxy signature.
  - ***Undeniability***. A proxy signer cannot disavow an accepted proxy signature he created.

Intermediate Protocols

- 4.1 Timestamping Services
- 4.3 Undeniable Digital Signatures
- 4.4 Designated Confirmer Signatures
- 4.5 Proxy Signatures
- **4.6 Group Signatures**
- 4.7 Fail-Stop Digital Signatures
- 4.9 Bit Commitment

Group Signatures

- A company has **several computers**, each connected to the local network.
- Each department of that company has its own printer (also connected to the network) and **only persons of that department** are allowed to use their department's printer.
- Before printing, therefore, the printer **must be convinced** that the user is working in that department.
- At the same time, the company wants privacy; the user's name may not be revealed.
- If, however, someone discovers at the end of the day that a printer has been used too often, the director must be able to discover who misused that printer, and send him a bill.

Group Signatures

- The solution to this problem is called a **group signature**. Group signatures have the following properties:
  - Only members of the group can sign messages.
  - The receiver of the signature can verify that it is a valid signature from the group.
  - The receiver of the signature cannot determine which member of the group is the signer.
  - In the case of a dispute, the signature can be "opened" to reveal the identity of the signer.

Group Signatures with a Trusted Arbitrator

This protocol uses a trusted arbitrator:

(1) Trent generates a large pile of public-key/private-key key pairs and gives every member of the group a **different list of unique private keys**. No keys on any list are identical. (If there are *n* members of the group, and each member gets *m* key pairs, then there are *n***m* total key pairs.)

(2) Trent publishes the master list of all public keys for the group, in **random order**. **Trent keeps a secret record of which keys belong to whom**.

(3) When group members want to sign a document, he chooses a key at random from his personal list.

(4) When someone wants to verify that a signature belongs to the group, he looks on the master list for the corresponding public key and verifies the signature.

(5) In the event of a dispute, Trent knows which public key corresponds to which group member.

Intermediate Protocols

4.1 Timestamping Services

4.3 Undeniable Digital Signatures

4.4 Designated Confirmer Signatures

4.5 Proxy Signatures

4.6 Group Signatures

4.7 Fail-Stop Digital Signatures

4.9 Bit Commitment

Fail-stop digital signatures,

- If Eve forges Alice's signatures after a brute-force attack, then Alice can prove they are forgeries.
- If Alice signs a document and then disavows the signature, claiming forgery, a court can verify that it is not a forgery.
- The basic idea behind fail-stop signatures is that for every possible public key, **many possible private keys work with it**.
  - **Each of these private keys yields many different possible signatures.**
- However, Alice has only one private key and can compute just one signature. Alice doesn't know any of the other private keys.
- Eve wants to break Alice's private key.
- She collects signed messages and, using her array of Cray computers, tries to recover Alice's private key. Even if she manages to recover a valid private key, there are so **many possible private keys** that it is far more likely that she has a different one.
- The **probability** of Eve's recovering the proper private key can be made so small as to be **negligible**.

Intermediate Protocols

4.1 Timestamping Services

4.3 Undeniable Digital Signatures

4.4 Designated Confirmer Signatures

4.5 Proxy Signatures

4.6 Group Signatures

4.7 Fail-Stop Digital Signatures

4.9 Bit Commitment

Bit Commitment

- Consider this situation:
  - Alice wants to commit to a prediction (i.e., a bit or series of bits) but **does not want to reveal** her prediction until sometime later.
  - Bob, on the other hand, wants to make sure that Alice cannot change her mind after she has committed to her prediction.

Bit Commitment Using Symmetric Cryptography

(1) Bob generates a random-bit string, *R*, and sends it to Alice.

R

(2) Alice creates a message consisting of the bit she wishes to commit to, *b* (it can actually be several bits), and Bob's random string. She encrypts it with some **random key, *K*,** and sends the result back to Bob.

$E_K(R,b)$

That is the commitment portion of the protocol. Bob **cannot decrypt** the message, so he does not know what the bit is.

When it comes time for Alice to reveal her bit, the protocol continues:

(3) Alice sends Bob the key.

(4) Bob decrypts the message to reveal the bit. He checks his random string to **verify the bit's validity**.

Bit Commitment Using One-Way Functions

This protocol uses one-way functions:

(1) Alice generates two random-bit strings, $R1$ and $R2$.

$R1,R2$

(2) Alice creates a message consisting of her random strings and the bit she wishes to commit to (it can actually be several bits).

$(R1,R2,b)$

(3) Alice computes the one-way function on the message and sends the result, as well as one of the random strings, to Bob.

$H(R1,R2,b),R1$

- This transmission from Alice is evidence of commitment. Alice's one-way function in step (3) **prevents** Bob from inverting the function and determining the bit.
- When it comes time for Alice to reveal her bit, the protocol continues:

(4) Alice sends Bob the original message.

$(R1,R2,b)$

(5) Bob computes the one-way function on the message and compares it and $R1$, with the value and random string he received in step (3). If they match, the bit is valid.

End of Lesson

- Readings
  - App. Crypto: Chapter 3 and 4 (only sections and subsections explained)