CS 5473 Lecture Notes

SELECTED TOPICS IN

COMPUTER SECURITY

Ruizhong Wei

Department of Computer Science
Lakehead University

Winter, 2011

Contents

Chapter 1

Introduction of Cryptography

1.1 Networks

- Network model (OSI)

  1. Physical layer: handles the transmission of raw bits over a communications channel. The protocols for the physical layer specify the medium used for the transmission (electronic, optical, wireless), offer choices for the signal format (serial, parallel, synchronous, asynchronous), and convert abstract raw bit stream into common codes understandable by all the connected parties.

     Data encoding, multiplexing schemes (FDM, TDM, CDMA, WDM)
  2. Data link layer: usually transmitted serially (asynchronous or synchronous communications), using some error correction methods. Some framing and flow control methods and protocols are considered in this layer. ISO adopted high-level data link control (HDLC).
  3. Network layer: subnets connected by bridges, switches and routers which need protocols (connection-oriented services or connectionless network services) Congestion control is one challenge in this layer.
  4. Transport layer: TCP/IP, UDP, DNS etc.
  5. Session layer: a small layer responsible for session management tasks and dialogue control.

6. Presentation layer: about syntax and semantics of the transmitted information. Data encoding (ASCII, EBCDIC, Unicode) compression (Huffman coding, JPEG,LZW, MPEG, etc). Encryption and decryption (or considered as at application layer).
7. Application layer: FTP, SMTP, SNMP, HTTP, etc.

- Network structure and topology and why communication over network is not secure.
- Wireshark `http://www.wireshark.org/download.html` is a network protocol analyzer for Unix and Windows.

  Tcpdump
-

1.2 Basics of Cryptography

1.2.1 Symmetric key encryption

Definition 1.2.1 *A cryptosystem is a five-tuple* $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$*, where the following conditions are satisfied:*

1. $\mathcal{P}$ *is a finite set of possible plaintexts.*
2. $\mathcal{C}$ *is a finite set of possible ciphertexts.*
3. $\mathcal{K}$*, the key space, is a finite set of possible keys.*
4. *For each key* $K \in \mathcal{K}$*, there is an encryption rule* $e_K \in \mathcal{E}$ *and a corresponding decryption rule* $d_K \in \mathcal{D}$*. Each* $e_K : \mathcal{P} \mapsto \mathcal{C}$ *and* $d_K : \mathcal{C} \mapsto \mathcal{P}$ *are functions such that* $d_K(e_K(x)) = x$ *for every plaintext* $x \in \mathcal{P}$*.*

In practice, a plaintext message is usually expressed as a string

$$\mathbf{x} = x_1 x_2 \cdots x_n$$

where $x_i \in \mathcal{P}, 1 \leq i \leq n$ and a ciphertext is also a string

$$\mathbf{y} = y_1 y_2 \cdots y_n,$$

where $y_i = e_K(x_i) \in \mathcal{C}, 1 \leq i \leq n.$

- Substitution Cipher

  In a Substitution Cipher, the key is a permutation on the set $\mathcal{P}$. The ciphertext is the substitution of the plaintext under that permutation. A simple example is that the set $\mathcal{P}$ is 26 English letters.

  In many modern encryption systems, pseudo random permutations are used for substitution.

- Permutation Cipher

  In a Permutation Cipher, a key is a permutation, which is used to permute the symbols in a block of plaintext.

  For example, suppose the key is the permutation

  $$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 4 & 3 & 1 & 6 & 2 & 5 \end{pmatrix}.$$

  The plaintext is:

  ```
  he walked up and down the passage two or three times.
  ```

  Then the ciphertext is (the space is omitted, when apply the permutation)

  ```
  WLEHKAUADENPONDDTWPSEHSAEWGAOTTRROEHIETESM.
  ```

- Product or combination of encryption systems

  Suppose we have two independent encryption functions (encryption rules) e_{K_1} and e_{K_2}. Then the product of combination of these two encryption system is $e_{K_1}(e_{K_2})$. Two cryptosystem can be product if and only if the cipertexts of the first system is contained in the plaintexts of the second system. However, sometimes a product of cryptosystems will not result a new crptosystem. Sometimes one crytosystem combines itself will create a new system.

- Security of symmetric key encryption

  In a perfect secure encryption, the distribution of cipher text should be uniformly random.

- Block cipher: DES (FIPS 46-3), AES(FIPS PUB 197), CAST-128/256, IDEA, RC5, Twofish
- Block cipher modes:
  - ECB
    $$y_i = e_K(x_i)$$
  - CBC
    $$\begin{aligned} y_0 &= IV \\ y_1 &= e_K(y_0 \oplus x_1), \\ y_2 &= e_K(y_1 \oplus x_2), \\ &\cdots\cdots \\ y_n &= e_K(y_{n-1} \oplus x_n). \end{aligned}$$
  - CFB
    $$\begin{aligned} y_0 &= IV \\ y_1 &= x_1 \oplus e_K(y_0), \\ y_2 &= x_2 \oplus e_K(y_1), \\ &\cdots\cdots \\ y_n &= x_n \oplus e_K(y_{n-1}). \end{aligned}$$
  - OFB
    $$\begin{aligned} z_0 &= IV \\ z_i &= e_K(z_{i-1}), i \geq 1 \\ y_i &= x_i \oplus z_i, i \geq 1 \end{aligned}$$
  - CTR
    $$\begin{aligned} y_1 &= x_1 \oplus e_K(c), \\ y_2 &= x_2 \oplus e_K(c+1), \\ &\cdots\cdots \\ y_n &= x_n \oplus e_K(c+n-1). \end{aligned}$$
- Stream cipher: synchronous, asynchronous, RC4, LFSR and sequences.

  In a Stream Cipher, we will use a key stream: $\mathbf{z} = z_1 z_2 \cdots$ to encrypt a plaintext. So the ciphertext will be
  $$\mathbf{y} = y_1 y_2 \cdots = e_{z_1}(x_1) e_{z_2}(x_2) \cdots.$$
  When the key stream is related to the plaintext, the cipher is called *non-synchronous* cipher. If the key stream is independent from the plaintext, then it is called *synchronous* cipher.

- Key size, block size and security (brute-force, chosen plain-text attacks).
- Security and efficiency.
- The limitation of secret key encryption for networks.

1.2.2 Public key encryption

In a public key system, the encryption function uses a public key (so anyone can encrypt), but the decryption uses a private key (only the person knows the key can decrypt).

- RSA

  Let $n = pq$, where p and q are large primes. Let $\mathcal{P} = \mathcal{C} = \mathbb{Z}_n$ and define

  $$\mathcal{K} = \{(n, p, q, a, b) : n = pq, ab \equiv 1 \bmod (p-1)(q-1), p, q \text{ primes}\}.$$

  For $K = (n, p, q, a, b)$, define

  $$e_K(x) = x^b \bmod n$$

  and

  $$d_K(y) = y^a \bmod n$$

  $(x, y) \in \mathbb{Z}_n$. The values n and b are public, and the values p, q, a are secret.

  For the security reason, the size of n should be at least 2048 bit long.

- ElGamal

  Let p be a prime such that the discrete logarithm problem in $\mathbb{Z}_p$ is intractable and let $\alpha \in \mathbb{Z}_p^*$ be a primitive element. Let $\mathcal{P} = \mathbb{Z}_p^*, \mathcal{C} = \mathbb{Z}_p^* \times \mathbb{Z}_p^*$, and

  $$\mathcal{K} = \{(p, \alpha, a, \beta) : \beta \equiv \alpha^a \bmod p\}.$$

  The values of p, α and β are public and a is secrete.

  For $K = (p, \alpha, a, \beta) \in \mathcal{K}$ and for a secret random number $k \in \mathbb{Z}_{p-1}$ define

  $$e_K(x, k) = (y_1, y_2),$$

where $y_1 = \alpha^k \pmod p, y_2 = x\beta^k \pmod p$.

For $y_1, y_2 \in \mathbb{Z}_p^*$, define

$$d_K(y_1, y_2) = y_2(y_1^a)^{-1} \pmod p.$$

For the security reason, $p-1$ should have a large prime factor.

- Elliptic curve (ECC)

  ECC uses the basic idea from ElGamal, but use points in some elliptic curve instead of numbers in $\mathbb{Z}_p$.

- Why need both secrete key and public key systems.

  RSA is 1500 times slower than DES. A 1024-bit key of RSA provide comparable strength to an 80-bit symmetric key system.

1.2.3 Message authentication

We want to make sure that the message has not been changed by any third party.

- Hash function collision-resistant property
- Birthday attack (the security of digest of length n is about $\sqrt{n}$ secure).
- MD5 (RFC 1321)
- SHA (FIPS 180-2, RFC 4634)
- SHA-2 (2010) and SHA-3 (2012, FIPS 202 was announced on August 5, 2015)
- MAC and HMAC

$$MAC(x)_t = HMAC(K, x)_t = h((K_0 \oplus opad||h((K_0 \oplus ipad)||x))_t$$

1.2.4 Digital signature

- RSA

- ElGamal (D-H)

- DSS

  Let p be a prime number such that $2^{L-1} < p < 2^L$ for $512 \leq L \leq 1024$ and L a multiple of 64, and let q be a 160-bit prime that divides $p-1$. Let $\alpha = h^{(p-1)/q} \pmod{p}$, where h is any integer with $1 < h < p-1$ such that $h^{(p-1)/q} \pmod{p} > 1$. Let $\mathcal{P} = \mathbb{Z}_p^*, \mathcal{A} = \mathbb{Z}_q \times \mathbb{Z}_q$, and define

  $$\mathcal{K} = \{(p, q, \alpha, a, \beta) : \beta \equiv \alpha^a \bmod p\}.$$

  The values of p, q, α and β are public and a is secret.

  For $K = (p, q, \alpha, a, \beta) \in \mathcal{K}$ and for a secret random number $k, 1 \leq k \leq q-1$, define

  $$sig_K(x, k) = (\gamma, \delta),$$

  where

  $$\gamma = (\alpha^k \bmod p) \bmod q$$

  and

  $$\delta = (x + a\gamma)k^{-1} \bmod q.$$

  For $x \in \mathbb{Z}_p^*$ and $\gamma, \delta \in \mathbb{Z}_q$, verification is done by performing the following computations:

  $$\begin{aligned} e_1 &= x\delta^{-1} \bmod q \\ e_2 &= \gamma\delta^{-1} \bmod q \end{aligned}$$

  and

  $$ver_K(x, \gamma, \delta) = \ true \ \Leftrightarrow (\alpha^{e_1}\beta^{e_2} \bmod p) \bmod q = \gamma.$$

- Combination of hash and signature.

1.2.5 Key distribution

- Diffie-Hellman

  Let p be a prime such that the logarithm problem in $\mathbb{Z}_p$ is infeasible. Let α be a primitive element of $\mathbb{Z}_p$. Then Alice and Bob do the following.

  1. Alice chooses $x_A \in \mathbb{Z}_p^*$. Then she computes and sends Bob $y_A = \alpha^{x_A} \bmod p$.
  2. Bob chooses $x_B \in \mathbb{Z}_p^*$. Then he computes and sends Alice $y_B = \alpha^{x_B} \bmod p$.
  3. The common key of Alice and Bob is

  $$K = y_B^{x_A} \bmod p = y_A^{x_B} \bmod p$$

- Man-in-the-middle attack and Oakley key exchange

  Outline of Oakley key exchange: Let I be the initiator and R be the receiver. The message exchanges are as follows.

  – I sends a cookie, the group to be used (value of $(\mathbb{Z}_p, \alpha)$), the value of $y_I = \alpha^{x_I} \bmod p$, I's nonce, I's identifier and R's identifier. I also indicates the public key encryption, hash and authentication algorithms to be used in this exchange. Then I appends a signature that signs the two identifiers, the nonce, (p, α), y_I, and the offered algorithms.

  – R verifies I's signature. Then R echoing back I's cookie, identifier, nonce and (p, α). R also includes in the message a cookie, $y_R = \alpha^{x_R} \bmod p$, the selected algorithms (which must be among the offered algorithms), R's identifier, and R's nonce. R appends a signature that signs the two identifiers, the two nonces, p, α, y_I, y_R, and the selected algorithms.

  – I verifies R's signature. The nonce values in the message assure that this is not a replay of an old message. To complete the exchange, I send a message back to R to verify that I has received R's message. This message contains two cookies, (p, α), y_I, two identifiers, two nonces the selected algorithms and a signature of the information.

- Other public key based key distribution: IKE (Internet Key Exchange, RFC 2409) and ISAKMP (Internet Security Association and Key Management Protocol, RFC 2408).

- Kerberos (RFC 1510)

  A simplified version of the Kerberos can be described as follows.

  1. U asked AS for a session key to communicate with V: U sends ID_U, TS_1 to AS, where ID_U is user U's identifier and TS_1 is time which is used to check the time synchronization.
  2. AS chooses a random key K, a time stamp T (also called a ticket), and a lifetime L (lifetime for the session key). So that the session key will be valid from time T to $T + L$. Then AS sends U: $m_1 = e_{K_U}(K||ID_V||T||L)$ and $m_2 = e_{K_V}(K||ID_U||T||L)$.
  3. U decrypts m_1 to obtain K, T, L and ID_V. Then U computes $m_3 = e_K(ID_U||T)$ and sends V the value of m_2 and m_3.
  4. V decrypts m_2 and obtains K, ID_U, T and L. Then V can computes T and ID_V from m_3. V checks whether the two values of T and the two values of ID_U are the same. If so, V computes $m_4 = e_K(T + 1)$ and sends m_4 to U.
  5. U decrypts m_4 and verifies the correctness of $T + 1$.

  If everything is correct, then U and V use K as a session key.

1.2.6 Public Key Infrastructure

- The purpose of PKI

- X.509: standard formats for public key certificates, certificate revocation lists, attribute certificates, and a certification path validation algorithm.

- PGP: use trust signatures

1.2.7 Case study: SSL

- Handshake Protocol

The protocol can be viewed as having four phases of exchanges.

Phase 1. Establish Security Capabilities

The client initiates the exchange by sending a client_hello message (in SSL record format) which contains: version, random number (nonce), session ID, CipherSuite which is a list of cryptographic algorithms supported by the client, compression method which is a list of compression methods the client supports.

After sending the client_hello message, the client waits for the server_hello message which contains the same parameters as the client_hello message but only one CipherSuite and one compression method are chosen.

The elements of Cipher Suite are key exchange method and CipherSpec which includes cipher algorithm, MAC algorithm, cipher type (stream or block), hash size, key material, IV size etc.

Phase 2. Server Authentication and Key Exchange

Server first sends its certificate message. Then a server_key_exchange message may be sent if it is required. The certificate_request message may be followed to request the client's certificate. Finally, server sends the server_done message which indicates the end of server hello.

Phase 3. Client Authentication and Key Exchange

Upon receipt of the server_done message, the client should verify the certificate and server_hello parameters. If everything is fine, then the client sends back messages to the server. First the client sends certificate message or no_certificate alert according whether the server requested a certificate. Next the client sends the client_key_exchange message which contains a 48-byte pre-master secret if RSA is used or public parameters of Diffie-Hellman scheme. Finally, the client may sends a certificate_verify message which provides verification of a client certificate.

Phase 4. Finish

This phase completes the setting up of a secure connection. The client sends a change_cipher_spec message and copies the pending CiperSpec into the current CipherSpec. Then the client sends the finished message under the new algorithms, keys, and secrets. The finished message is hash value of master_secret, handshake_message (which consists of all of the data from handshake messages up to but not including this message) and some other codes. The master_secret is computed from the pre_master secret. The master_secret is then used to generate the session keys which are used for encryption and authentication.

In response to these messages, the server sends its own change_cipher_spec message and its finished message. At this point, the handshake is complete and the client and the server may begin to exchange application layer data using the master_secret as session key.

- SSL record Protocal

  This protocol defines how to transmit an application message in SSL. The sender of a message does the following:

  - Fragmentation: The message is fragmented into blocks of 2^{14} bytes or less.
  - Compression: (optional).
  - Add MAC: Compute a MAC using a shared secret key K and add the resulting MAC to the fragment. The MAC function used in SSL is similar to HMAC. A hash function h and a shared secret key K is used. The MAC value is computed as follows.

    $$h(K||pad_2||h(K||pad_1||seqnum||type||length||M))$$

    where *seqnum* is the sequence number of the message, *type* is the high-level protocol used to process this fragment, *length* is the length of the fragment and M is the content of the fragment. pad_1 and pad_2 are fixed paddings which are repeating of fixed bytes.
  - Encrypt: Encrypt the compresses message plus the MAC using symmetric encryption (block cipher or stream cipher). The block cipher used in SSL are IDEA, DES, 3-DES, DES40, RC2, Fortezza. The stream cipher used are RC4s.
  - Append SSL record header: The header consists: Content Type (8 bits), Major Version (8 bits), minor version (8 bits) and Compressed Length (16 bits). The compressed length is the length

of the plaintext (or compressed plaintext) fragment in bytes. The maximum value is $2^{14}+2048$. The content types are change_cipher_spec, alert, handshake and application_data.

- TLS

1.2.8 Cryptanalysis

- Kerckhoff's principle
- Chosen plaintext attack and chosen ciphertext attack
- Linear analysis
- Differential analysis
- Attack public key systems

Chapter 2

Operating System Security

2.1 Operation systems

Provides the interface between the users (applications) and the computer hardware (CPU, main memory, disk drives, input/output devices, network interfaces, etc.)

- Multi-users and multi-tasking

  Different users may have different needs and rights, usually separated each other. Multi-tasks may share same computer resources.

- The kernel, non-essential OS application and userland applications

User applications	userland
Non-essential OS applications ——————————- The OS Kernel	Operating system
CPU, Memory, Input Output	Hardware

- Input/output devices drivers and system call (library)

  Device drivers encapsulate the details of how interact with the device while its API provides interact with application programs in a high level.

System calls or software interrupts let user applications request the kernel to perform actions on their behalf.

- Processes (running programs in RAM)
  - time slicing capability

    The operating system gives each running process a tiny slice of time to do the work, that enables multi-tasking.
  - process tree

    When a user creates a new process, the kernel sees this as an existing process (i.e., shell program) asking to create a new process. Usually, the new child process inherits the permissions of its parent, unless the parent deliberately creates a new child process with lower permissions. These processes form the process tree. In Linux, the root of the tree is the process `init`.

```
$ pstree | more
init-+-acpid
|-avahi-daemon---avahi-daemon
|-bonobo-activati---{bonobo-activati}
|-cron
|-cupsd
|-gdm---gdm-+-Xorg
|           `-x-session-manag-+-gnome-panel---{gnome-panel}
|                             |-gnome-settings--+-pulseaudio-+-gconf-helper
|                             |                 |            `-2*[{pulseaudio}]
|                             |                 `-{gnome-settings-}
|                             |-konsole---3*[bash]
|                             |-metacity
|                             |-ssh-agent
|                             `-{x-session-manag}
|-getty
|-konsole-+-2*[bash]
|         |-bash---vim
|         `-bash-+-pstree
|                `-vim
|-thinMS
```

  - process ID (PID) and user ID(uid)

    Usually each running process has a unique non-negative integer as its PID. In Linux, `ps` can be used to display processes with PIDs. A process is also associated with the user (`uid`) who executes the process and a group ID (`gid`). In Unix system, processes also have an effective user ID (`euid`) which may same as the uid or have higher privileges.
  - inter-process communication, pipes and sockets

Inter-process communications (IPC) can be done by reading and writing files, or sharing the same region of physical RAM. Pipes and sockets are used to create tunnels from one process to another.

- signals (UNIX) and remote procedure calls (windows)

  UNIX uses signals to send messages from process each other asynchronously. When a process received a signal from another processes, the operating system interrupts the current flow of execution of that process, and tries to handle that particular signal. For example, `Ctrl-c` is a signal of termination.

  Windows use remote procedure calls (RPC), which allow a process to call a subroutine from another process's program. To terminate a process, a kernel-level API named `TerminatProcess()` is used.

- daemons (UNIX) and services (windows)

  Many processes are running in computer without any user intervention which are known as daemons in UNIX and services in Windows.

• Filesystem (file access control, file permission matrix)

File systems are arranged in folders or directories. File permissions are checked by the operating system to determine if a file is readable, writable or executable by a user or group of users. This data is typically stored in the metadata of the file. UNIX uses permission matrix.

```
[wei~/css]$ls -l
total 6
drwxr-xr-x   3 wei    staff      512 Oct 29 20:24 custom-theme
-rw-r--r--   1 wei    staff     1292 Nov 10 22:13 testcss.css
```

• Memory management

- Address space (allocated a region of memory for a process executes)

  Contiguous address space for each process are used (or virtual contiguous addresses).

- Unix memory model: Text (machine code of the program), Data (initialized static program variables), BBS (block started by symbol contains uninitialized static variables), Heap (dynamic segment stores data generated during a process), Stack (keeps tack of the call structure of subroutines and their arguments).

Stack
......
Dynamic
BSS
Data
Text

- Memory access permissions: Text usually is read-only, and the others are writable.

  Address space is divided into user space and kernel space (which has most restrictive access privileges).

  And a process usually is not allowed to access the address spaces of others. Address boundaries are used to avoid other processes to change a process.
- Virtual memory: use memory management unit to arrange memory so that processes are allowed to act as if their memory is contiguous (actually not). Each virtual address is mapped to a real memory address. Sometimes external drive can be used to store data not using.
- Page faults: when a block of the address space is not accessed for an extended period of time, it may be paged out and written to disk. When a page fault occurs, paging supervisor reads the block back into RAM.

• Virtual machine: host OS, guest OS and hypervisor (or virtual machine monitor). There are two main implementations of VMs.

  - emulation: the host OS simulates virtual interfaces that the guest OS interacts with.
  - virtualization: the virtual interfaces within the VM are matched with the actual hardware on the host machine.

• Cloud computing

  The National Institute of Standards and Technology's definition of cloud computing identifies "five essential characteristics":

- On-demand self-service. A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider.
- Broad network access. Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, tablets, laptops, and workstations).
- Resource pooling. The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand.
- Rapid elasticity. Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear unlimited and can be appropriated in any quantity at any time.
- Measured service. Cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service.

2.2 Process security

Monitor and protect the processes that are running on the computer.

2.2.1 Inductive Trust

The trust that we place on the processes is an inductive belief based on the integrity of the processes. This state is maintained even if the computer is shut down or put into a hibernation state.

- The boot sequence: BIOS(basic input/output system) executes first. BIOS loads second-stage boot loader to memory, that handles loading the rest of the OS.

- Use BIOS password to prevent a second-stage boot loader to be executed without proper authentication (boot by a live-CD).
- Boot device hierarchy determines the order of precedence of booting devices. A customizable hierarchy (the first available device in the list is used for booting) may cause security problem. Utilize second-stage boot loaders that require password protections that only allow authorized users to boot from external storage media.
- Hibernation file: used to stores entire contents of the machine's memory on disk so that the state of the computer can be quickly restore when the system is powered back on (`C:\hiberfil.sys`). This file may be attacked by recovering the RAM content or modifying the hibernation file. Encryption should be used to protect hibernation files and swap files.

2.2.2 Monitoring

- Event logging: `Windows Event Log (Isass.exe)`, Unix `/var/log` using `syslog` daemon.

  Windows defines 3 possible sources of logs: System, Application and Security. Unix produces log files with text format.
- Processing monitoring: task manager in windows and `ps, top, pstree, kill` in Linux.
- Process explorer: customizable and useful tool for task manager in the Microsoft Windows system (free downloaded).

2.3 Memory and file system security

2.3.1 Virtual memory security

- Swap files: `pagefile.sys` for windows or `swap partition` for Linux, used for page file (virtual memory files). Usually prevent users from viewing the contents of virtual memory files.
- Attacks on virtual memory: suddenly power off the machine and boot to another OS via external media may be possible to view the swap

files and expose sensitive information. Hard disk encryption is used to prevent such an attack.

2.3.2 Password

- Hash function and salt: password file stores $(U, S, h(S||P))$ where U is userid, S is salt (random string), P is password. Salt helps to prevent dictionary attack.

- Password authentication

  Windows: `NTLM` algorithm which is challenge-response protocol. Password are hashed and then used as key to encrypt (using DES) the random challenge value. File: `security accounts manager (SAM)`

  Unix: Using salt in password and MD5 or DES. Files: `/etc/passwd` `/etc/shadow`.

- Crack, John the Ripper or other password crackers.

  Crackers are also used to check the security of a password.

- One-time password

  Use security tokens or use hash chains (RFC 2289).

- OAuth: open standard for authorization, used for users to log into third party websites without exposing their passwords. The security of the protocol is unsolved.

2.3.3 Access control

Access control decides what folders and files a user can access.

- Access control entries (ACE) and lists (ACL): ACE is a triplet `(principal, type, permission)`, where principal is either a user or a group, type is either allow or deny, permission is read, write, execution, etc. ACL is an ordered list of ACEs for each object. Disadvantage is they do not provide an efficient way to enumerate all the access rights for a given subject (e.g. when a user is removed)

- Capabilities: For each subject (user or group), list the objects for which it had nonempty access control rights, together with the specific rights. Disadvantage is the subject is not associated directly with objects.

- Linux permissions: Use file permission matrix. Path based access control principal. Owners of files can change the permissions on those file (discretionary access control DAC). Append-only (a user only can write to the end of the file) or immutable attributes are used for some Linux OS. `chmod` command. Security-Enhanced Linux (SELinux) developed by NSA tried to use the principal of least privilege. In SELinux, users are not given the power to decide security attributes of their own files. They are decided by a central security policy administrator.

- Windows permissions: Use ACL model. Use inherited ACEs (opposed to explicit ACEs that are specifically set) that may cause problem.

- How to set permission policies.

- The SetUID bit: used in the file permission matrix. If this bit is set, then that program runs with the effective user ID of its owner, rather than the process that executed it. This is useful for some cases. For example, the `etc/passwd` file belongs to root, but a user may need to change his password. So although the program `passwd` is owned by the root, it has the setuid bit set.

  `setuid seteuid` are used in Linux. This may cause security problem if the setuid program are not using safe programming practices. The following example explains usage of these functions. The following program owned by the `admin`, but can be executed by other user. When a user executes the program, the `uid` is the user but the `euid` is `admin`.

  ```
  #include<stdio.h>
  #include<sys/types.h>
  #include<unistd.h>
  #include<stdlib.h>

  static uid_t euid, uid;

  int main(int argc, char* argv[])
  {
  ```

```
 FILE *file;
 uid = getuid();
 euid = geteuid();
 seteuid(uid); /*drop privileges */
 ... ...
 seteuid(euid);  /*raise privileges*/
 file = fopen("home/admin/log", "a");
 seteuid(uid); /*drop privilege again*/
 fprintf(file, "Someone used this program. \n")'
 fclose(file);
 return 0;
}
```

The `fprintf` can use the permission of the owner of this program, not the user running this program.

2.3.4 Role-based access control (RBAC)

Access control based on roles instead of individual users, which can simplify the access control in many cases. However, most OS does not implement this.

- Core RBAC: users are assigned to roles, permissions are assigned to roles and users acquire permissions by being members of roles. Many-to-many assignment. User-role and permission-role reviews. User sessions.
- Hierarchical RBAC: roles are arranged in hierarchies. Senior roles acquire the permission of their juniors and junior roles acquire the user membership of their seniors.
- Static separation of duty (SSD): place constraints on the assignment of users to roles. Membership in one role may prevent the use from being a member of one or more other roles. SSD in the presence of a hierarchy is also considered.
- Dynamic separation of duty (DSD): limits the availability of the permissions by placing constraints on the roles that can be activated within or across a user's sessions.

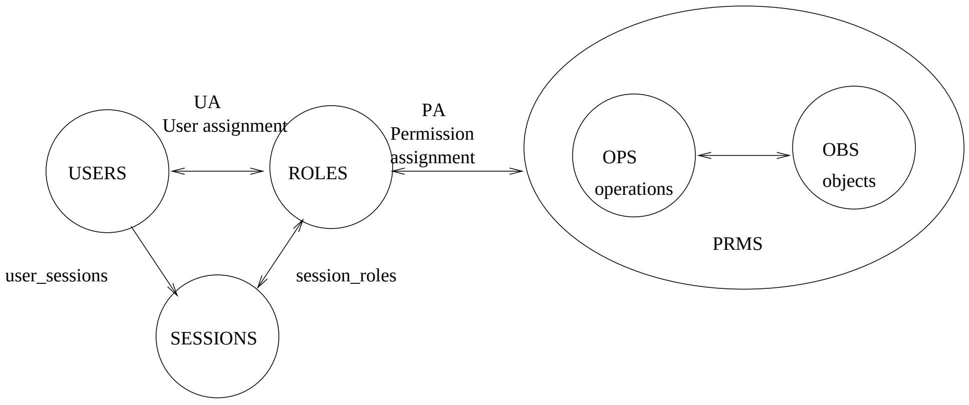

Figure 2.1: The core RBAC

2.3.5 File descriptors

In order to efficiently handle work with files, the operating system uses a mechanism known as file descriptors.

- When a program needs to access a file, a call is made to the `open` system call, which results in the kernel creating a new entry in the `file descriptor table` which maps to the file's location on the disk. When receiving a read or write system call, the kernel looks up the file descriptor in the table and perform the read or write at the appropriate location on disk. After operations, the program should issue the `close` system call to remove the file descriptor.

  The operating system should do several security checks (access permission) during the above procedure.

- When a process creates a child process, that child process inherits copies of all of the file descriptors that are open in the parent. The operating system only checks whether a process has the permissions to a file at the moment of creating a file descriptor. This may cause a security problem known as a file descriptor leak.

  Example

  ```
  #include <stdio.h>
  #include <unistd.h>
  ```

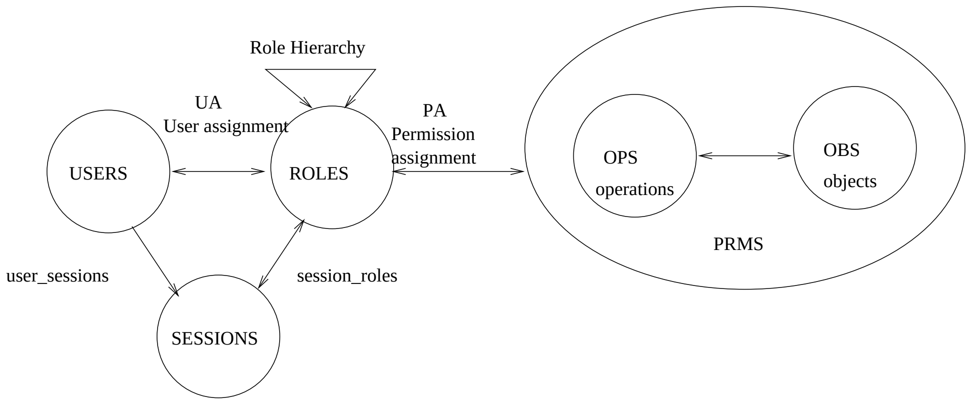

Figure 2.2: The Hierarchical RBAC

```
int main( int argc, char*argv[])
{
 FILE *passwords;
 passwords = fopen("/home/admin/passwords", "r");
 .......
 .......
 execl("/home/joe/shell", "shell", NULL);
 ......
}
```

In this example, the new process has the permission to write to the `passwords` file, because the new process will inherit all copies of the file descriptors that are opened in the parent. To fix the vulnerability, a call to `fclose` should be made before executing the new program.

2.3.6 Symbolic links

Symbolic links (symlink) can be a chain. The link is completely transparent to applications. An attacker could trick a program by creating a symlink to a secure file and specifying the path of the symlink instead. That may cause wrong result of the permission check. To solve this aliasing problem, when open files, the program should either check if the provided filename refers to a symlink, or use a `stat` system call to retrieves information on files.

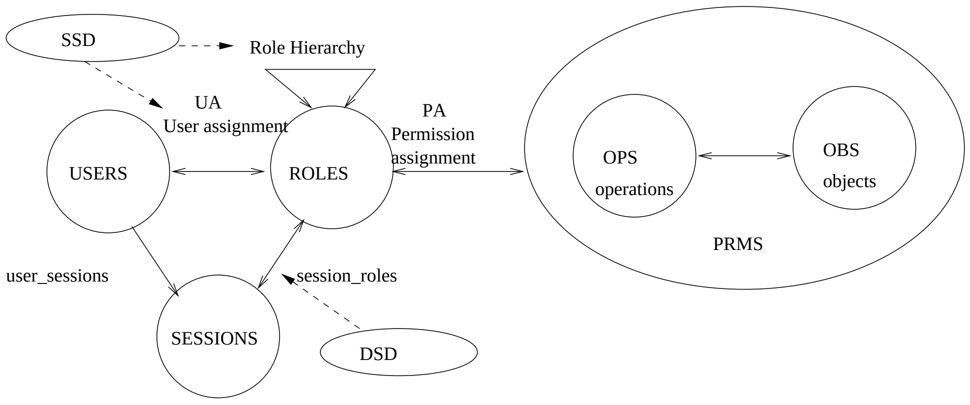

Figure 2.3: The Constrained RBAC

Windows uses shortcuts. Only a program that specifically identify them as shortcuts can follow them to the referenced files. This prevents of most symlink attacks that are possible on unix based systems, but also limits their power and flexibility.

2.4 Application program security

2.4.1 Compiling and linking

- Statically linking: all shared libraries such as operating system functions, that a program needs during its execution are copied into the compiled program on disk. Safer but require more disk space and difficult debugging.

- Dynamically linking: when the program is executed, the loader determines which shared libraries are needed for the program, finds them on disk, and imports them into the process's address space.

- DLL injection: injecting codes to DLL (dynamic linking library) so that no recompiling of the main program is needed after change the codes. Dll injection is very useful to do the debugging. May cause security problem: a malicious parties may inject some code into legitimate program, that redefine a function called by a system administrator program.

2.4.2 Simple buffer overflow attacks

When a program allocates a fixed-size buffer in memory in which to store information, it is important that copying user-supplied data to this buffer is done securely and with boundary check.

Arithmetic overflow: caused by the representation of integers in memory. Add a large positive integer might result a negative value.

Example:

```
#include<stdio.h>

int main(int argc, char * argv[])
{
 unsigned int connections = 0;
 ... ...
 connections++;
 if(connections < 5)
   grant_access();
 else
  deny_access();
 return1;
}
```

Making huge number of connections will cause counter overflows and wraps around to zero.

Add the following before `connections++;` will prevent that kind attacks.

```
if(connections < 5)
```

2.4.3 Stack-based buffer overflow

A stack is a contiguous block of memory containing data. A register called the stack pointer (SP) points to the top of the stack. The bottom of the stack is at a fixed address. Its size is dynamically adjusted by the kernel at run time. The CPU implements instructions to PUSH onto and POP off of the stack.

The stack is implementation dependent. (Here we mostly use Linux on Intel as examples.)

The stack consists of logical stack frames that are pushed when calling a function and popped when returning. A stack frame contains the parameters to a function, its local variables, and the data necessary to recover the previous stack frame, including the value of the instruction pointer at the time of the function call.

Usually, there is a frame pointer (FP) which points to a fixed location within a frame. Many compilers use FP for referencing both local variables and parameters because their distances from FP do not change with PUSHes and POPs.

The first thing a procedure must do when called is save the previous FP (so it can be restored at procedure exit). Then it copies SP into FP to create the new FP, and advances SP to reserve space for the local variables. This code is called the procedure prolog. Upon procedure exit, the stack must be cleaned up again, something called the procedure epilog.

Whenever a function call is made, the function parameters are pushed onto the stack. Then the return address (address to be executed after the function returns), followed by a frame pointer (FP), is pushed on the stack. A frame pointer is used to reference the local variables and the function parameters, because they are at a constant distance from the FP. Local automatic variables are pushed after the FP. In most implementations, stacks grow from higher memory addresses to the lower ones.

The main idea for the buffer overflow attack is trying overflow the buffer so that the return address are changed to the malicious code, as show in the Figure 2.4.

Let us see some examples.

Example 1

```
void function (int a, int b, int c)
{
   char buffer1[5];
   char buffer2[10];
}
int main()
{
  function(1,2,3);
}
```

	buffer2	buffer1	FP		a	b	c	

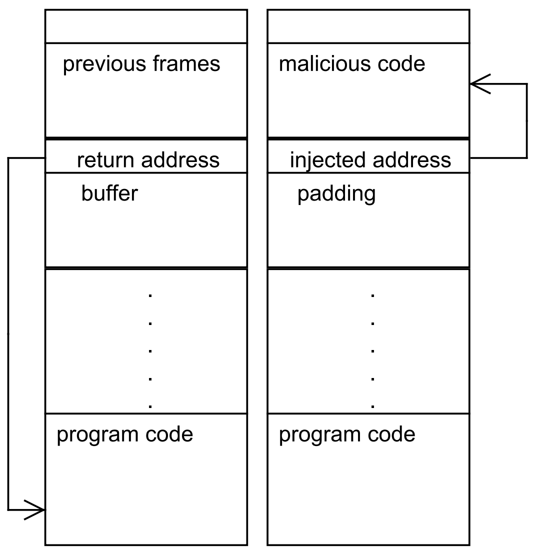

Figure 2.4: Stack-based buffer overflow

buffer1 takes eight bytes and buffer2 takes 12 bytes, as memory can be addressed only in multiples of word size (four bytes). In addition, an FP is needed to access a, b, c, buffer1 and buffer2 variables. All these variables are cleaned up from the stack as the function terminates. These variables take no space in the executable disk copy.

We can use `gcc -S -o exmap1.s exmap1.c` to get assemble codes.

We can use gdb to get more information. In the following, we use `disassemble` command. This specialized command dumps a range of memory as machine instructions. The default memory range is the function surrounding the program counter of the selected frame. A single argument to this command is a program counter value; GDB dumps the function surrounding this value. Two arguments specify a range of addresses (first inclusive, second exclusive) to dump.

```
$gdb a.out
GNU gdb 6.2.1
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General
```

```
Public License, and you are
welcome to change it and/or distribute copies of
it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type
"show warranty" for details.
This GDB was configured as "sparc-sun-solaris2.10"...
(no debugging symbols found)...

(gdb) disassemble function
Dump of assembler code for function function:
0x00010664 <function+0>:       save  %sp, -136, %sp
0x00010668 <function+4>:       st  %i0, [ %fp + 0x44 ]
0x0001066c <function+8>:       st  %i1, [ %fp + 0x48 ]
0x00010670 <function+12>:      st  %i2, [ %fp + 0x4c ]
0x00010674 <function+16>:      ret
0x00010678 <function+20>:      restore
End of assembler dump.

(gdb) disassemble main
Dump of assembler code for function main:
0x0001067c <main+0>:   save  %sp, -112, %sp
0x00010680 <main+4>:   mov  1, %o0
0x00010684 <main+8>:   mov  2, %o1
0x00010688 <main+12>:  mov  3, %o2
0x0001068c <main+16>:  call  0x10664 <function>
0x00010690 <main+20>:  nop
0x00010694 <main+24>:  nop
0x00010698 <main+28>:  ret
0x0001069c <main+32>:  restore
0x000106a0 <main+36>:  retl
0x000106a4 <main+40>:  add  %o7, %l7, %l7
End of assembler dump.
$
```

A buffer overflow is the result of stuffing more data into a buffer than it can handle. How can this often found programming error can be taken advantage to execute arbitrary code? Lets look at another example:

Example 2.

```
void function(char *str) {
   char buffer[16];

   strcpy(buffer,str);
}

void main() {
  char large_string[256];
  int i;

  for( i = 0; i < 255; i++)
    large_string[i] = 'A';

  function(large_string);
}
```

This is program has a function with a typical buffer overflow coding error. The function copies a supplied string without bounds checking by using strcpy() instead of strncpy(). If you run this program you will get a segmentation violation. Lets see what its stack looks when we call function:

bottom of memory					top of memory
	buffer	sfp	ret	*str	
⟵	[ ]	[ ]	[ ]	[ ]	
top of stack					bottom of stack

What is going on here? Why do we get a segmentation violation? Simple. strcpy() is coping the contents of *str (larger_string[]) into buffer[] until a null character is found on the string.

As we can see buffer[] is much smaller than *str. buffer[] is 16 bytes long, and we are trying to stuff it with 256 bytes. This means that all 250 bytes after buffer in the stack are being overwritten. This includes the SFP, RET, and even *str! We had filled large_string with the character 'A'. It's hex character value is 0x41. That means that the return address is now 0x41414141. This is outside of the process address space. That is why when the function returns and tries to read the next instruction from that address you get a segmentation violation.

Lets try to modify our first example so that it overwrites the return address, and demonstrate how we can make it execute arbitrary code. Just before buffer1[] on the stack is SFP, and before it, the return address. That is 4 bytes pass the end of buffer1[]. But remember that buffer1[] is really 2 word so its 8 bytes long. So the return address is 12 bytes from the start of buffer1[]. We'll modify the return value in such a way that the assignment statement 'x = 1;' after the function call will be jumped. To do so we add 8 bytes to the return address.

Example 3

```
void function(int a, int b, int c){
  char buffer1[5];
  char buffer2[10];
  int *ret;
  ret=buffer1+12;
  (*ret)+=8;
}

void main(){
 int x;
 x=0;
 function(1,2,3);
 x=1;
 printf("%d\n",x);
}
```

What we have done is add 12 to buffer1[]'s address. This new address is where the return address is stored. We want to skip pass the assignment to the printf call. How did we know to add 8 to the return address? We used a test value first (for example 1), compiled the program, and then started gdb:

```
[aleph1]$ gdb example3
GDB is free software and you are welcome to
distribute copies of it under certain conditions; type "show
copying" to see the conditions. There is absolutely no warranty for
GDB; type "show warranty" for details. GDB 4.15
(i586-unknown-linux), Copyright 1995 Free Software Foundation,
Inc... (no debugging symbols found)...
```

```
(gdb) disassemble main
Dump of assembler code for function main:
0x8000490 <main>:       pushl  %ebp
0x8000491 <main+1>:     movl   %esp,%ebp
0x8000493 <main+3>:     subl   $0x4,%esp
0x8000496 <main+6>:     movl   $0x0,0xfffffffc(%ebp)
0x800049d <main+13>:    pushl  $0x3
0x800049f <main+15>:    pushl  $0x2
0x80004a1 <main+17>:    pushl  $0x1
0x80004a3 <main+19>:    call   0x8000470 <function>
0x80004a8 <main+24>:    addl   $0xc,%esp
0x80004ab <main+27>:    movl   $0x1,0xfffffffc(%ebp)
0x80004b2 <main+34>:    movl   0xfffffffc(%ebp),%eax
0x80004b5 <main+37>:    pushl  %eax
0x80004b6 <main+38>:    pushl  $0x80004f8
0x80004bb <main+43>:    call   0x8000378 <printf>
0x80004c0 <main+48>:    addl   $0x8,%esp
0x80004c3 <main+51>:    movl   %ebp,%esp
0x80004c5 <main+53>:    popl   %ebp
0x80004c6 <main+54>:    ret
0x80004c7 <main+55>:    nop
```

We can see that when calling function() the RET will be 0x8004a8, and we want to jump past the assignment at 0x80004ab. The next instruction we want to execute is the at 0x8004b2. A little math tells us the distance is 8 bytes.

Because we know it is easy to overwrite a function's return address, an intelligent hacker might want to spawn a shell (with root permissions) by jumping the execution path to such code. But, what if there is no such code in the program to be exploited? The answer is to place the code we are trying to execute in the buffer's overflowing area. We then overwrite the return address so it points back to the buffer and executes the intended code. Such code can be inserted into the program using environment variables or program input parameters. An example code that spawns a root shell can be found in a classic paper written by Aleph One for Phrack Magazine :

`www.phrack.org/archives/49/P49-14`

Example, suppose a program is waiting for a user to enter his or her name.

Rather than enter the name, the hacker would enter an executable command that exceeds the stack size. The command is usually something short. In a Linux environment, for instance, the command is typically EXEC(”sh”), which tells the system to open a command prompt window, known as a root shell in Linux circles.

Overflowing the buffer with an executable command doesn’t mean that the command will be executed. The attacker must then specify a return address that points to the malicious command. The program partially crashes because the stack overflowed. It then tries to recover by going to the return address, but the return address has been changed to point to the command specified by the hacker. Of course this means that the hacker must know the address where the malicious command will reside.

Some techniques used by the attackers.

- NOP sledding: To get around needing the actual address, the malicious command is often padded on both sides by NOP (no-op) instructions, a type of pointer. The NOP instruction does nothing but tell the processor to proceed to the next instruction. Padding on both sides is a technique used when the exact memory range is unknown. Therefore, if the address the hacker specifies falls anywhere within the padding, the malicious command will be executed. Usually, the hacker will be difficult to know the exact address, but just some reference address. So NOP sledding will help.

- Trampolining: Try to use common external libraries (which may contain instructions that are commonly used by many processes, system calls and other low-level operating system code) which usually loaded into predictable memory locations. For example, an attacker might know a particular assembly code instruction in a Windows core system DLL, and the attacker might find instruction tells the processor to jump to the address stored in one of the processor’s registers, such as ESP. If the attacker can place his malicious code at the address pointed to by ESP, then the application will jump and execute the malicious codes.

- The return-to-libc attack: the attacker found the address of a C library function, such as `system()` or `execv`. The attacker then use buffer overflow to overwriting the return address with the address of the library function. Following this address, the attacker provide a new address that the libc function will return to when it is finished. When the vulnerable stack frame returns, it will call the chosen function with the arguments provided, potentially giving full control to the attacker.

Most modern operating systems have some sort of mechanism to control the access level of the user who's currently logged on and executable programs typically require a higher level of permissions. These programs therefore run either in kernel mode or with permissions inherited from a service account. When a stack-overflow attack runs the command found at the new return address, the program thinks it is still running. This means that the command prompt window that has been opened is running with the same set of permissions as the application that was compromised. Generally speaking, this often means that the attacker will gain full control of the operating system.

- Shellcode: Once an attacker has successfully done the buffer overflow, they wants to execute arbitrary code on the machine. Many attackers choose to execute code that spawns a terminal or shell, allowing them to issue further commands. This kind code is known as shellcode. Since the code is executed directly on the stack by the CPU, it needs to be in assembly language, low-level processor instructions. Therefore usually such kind of attacks is platform dependent.

  Ordinary assembly code may contain many null character, `0x00`. However, this code cannot be used in most buffer overflow exploits, because this character typically denotes the end of a string, which will prevent the attacker from copying his payload into a vulnerable buffer. So the attacker needs to use some tricks to avoid null characters. For example, some one uses perl as part of the shellcode.

  `setuid()` system call are commonly used in buffer overflow attack. An attacker may use that call to gain a shell with the permissions of the exploited process's owner, may be allowed for full system compromise.

2.4.4 Heap-based buffer overflow

It is often desirable to give programmers the power to allocate memory dynamically and have it persist across multiple function calls (hence multiple processes). This memory is allocated in a large portion of unused memory known as heap.

Memory leak problems: if programs allocate memory on the heap and do not explicitly deallocate (free) that block, it remains allocated but are not actually being used, which allowing an attacker to execute arbitrary code by buffer overflow.

Heap-based overflows are generally more complex than stack-based buffer overflows and require a more in-depth understanding of how garbage collection and heap are implemented.

Example

```
#include <stdio.h>
#include <stdlib.h>
#include <sting.h>

int main(int argc, char *argv[])
{
 char *buf = malloc(256);
 char *buf2 = malloc(16);
 strcpy(buf, arbv[1]); //this may cause heap buffer overflow.
 printf("argument: %s\n", buf);
 ......
}
```

Instead of using `strcpy()`, we can use `strncpy()`, which is a safer equivalent function.

2.4.5 Prevent buffer overflow attacks

As we have seen, a buffer overflow attack requires two things. First, a buffer overflow must occur in the program. Second, the attacker must be able to use the buffer overflow to overwrite a security sensitive piece of data (a security flag, function pointer, return address, etc).

If we want to prevent buffer overflows completely we must stop one of these two things, i. e. either:

1. Prevent all buffer overflows or

2. Prevent all sensitive information from being overwritten

Both these solutions are costly in terms of efficiency and many programs therefore settle for a partial goal, such as:

- Prevent use of dangerous functions: gets(), strcpy(), etc.
- Prevent return addresses from being overwritten
- Prevent data supplied by the attacker from being executed (stops the attacker from jumping into his own buffer)

There are several possible levels where a defense mechanism can be inserted. At the language level we can make changes to the C language itself to reduce the risk of buffer overflows. At the source code level we can use static or dynamic source code analyzers to check our code for buffer overflow problems. At the compiler level we can change the compiler so that it does bounds checking or protects certain addresses from overwriting. At the operating system level we can change the rules for which memory pages that should be allowed to hold executable content.

Language Tools

The simplest solution at the language level is to switch to a language that provides automatic bounds checking of buffers, such as Java, Perl or Python. However, in most projects this is not an option.

A better solution is to use a library module that implements “safe”, bounds-checked buffers, such as the standard C++ string module or `libmib` (Software Component Library). There are two problems with this solution. First, it requires a complete rewrite of all the source code for the project. This alone makes the solution improper for anything but recently started projects. Second, most programs have to interface with prewritten library code. Since this code will use the ordinary “unsafe” buffers, the program will not be able to completely avoid them. Instead it will constantly have to convert between “safe” and ”unsafe” buffers. Whenever a buffer is in the “unsafe” mode, buffer overflow problems can occur. There is also a risk that programmer’s will forget to convert buffers back to the “safe” mode.

An alternative to making every buffer access safe is to target only those specific functions in C which are known to be dangerous, e. g. strcpy(), strcat(), gets() and sprintf(). This takes care of buffer overflows caused by these functions, but doesn’t handle buffer overflows caused by other code, such as user written functions.

The simplest solution to securing the dangerous function is to disallow them. They all have ”safe” counterparts that can be substituted, such as strncpy() and snprintf(), which in addition to the buffers also take a size parameter. (Of course, these functions are only ”safe” if the size parameter is specified correctly.) Replacing the functions with safe calls is probably the best solution provided that the source code to the program is available. To make sure that the unsafe versions are not used by mistake, their prototypes can be removed from the header files.

The `libsafe` library from Bell Labs provides a way of securing calls to these functions, even if the source code is not available. It does this by replacing the implementation of the dangerous functions in the shared libc library with safe versions.

`libsafe` makes use of the fact that stack frames are linked together by frame pointers (this is implementation dependent, but many C compilers on many platforms use this solution). When a buffer is passed as argument to one of the unsafe functions, libsafe follows the frame pointers to find the stack frame where the buffer was allocated (if it is not found, it is assumed that the buffer resides on the heap). It then checks the distance to the closest return address on the stack. When the function executes it makes sure that this address is not overwritten. If an attempt to overwrite this address is made, the program terminates with a vulnerability warning.

Finding the correct stack frame and protecting the return address requires some overhead which depends on how deep on the stack buffers are buried and how many calls to unsafe functions the program makes. Usually the overhead is quite small.

Source Code Tools

A source code tool analyzes the source code of a program and tries to determine whether it contains any dangerous constructions that could lead to buffer overflows.

We cannot expect a source code tool to be able to detect all possible instances of buffer overflow while at the same time yielding no false positives, since that is a task of the same difficulty as solving the halting problem. Constructing good source code tools will therefore always be a heuristic task.

The source code tools available today make only a limited, local analysis of the code and are therefore heavily restricted in the types of buffer overflow problems that they can detect.

ITS4 (Software Security Tool) is a source code tool that checks for the use of dangerous function. It is a bit smarter than a plain grep search in that it can rule out some cases where the use of a dangerous function usually does not pose a problem.

An analyzer that could perform more extensive, cross-function analysis would be a valuable tool in a security audit. Constructing such a tool would however be a major undertaking. The tool would have to try to keep track of the size of all buffers and the possible ranges of variables to be able to detect when a buffer overflow might occur.

Dynamic analysis tools such as `Purify` are an alternative to static analysis tools. A dynamic analysis tool analyzes the memory use of a program as it is run. Dynamic analysis tools can detect buffer overflow problems if they occur in a test run of a program. However, errors that occur in test runs can usually be detected even without an analysis tools, since they typically cause the program to crash. The main advantage of dynamic analyzers is that they allow for the error to be swiftly located once it has been detected.

Compiler Tools

A compiler tool changes the way a program is compiled, so that protection against buffer overflow is automatically compiled in with the program. No changes to the program's source code are thus necessary.

Buffer overflows can be prevented by adding bounds checking to all buffers. To do this, the compiler must add code for keeping track of the size of buffers and for checking that every buffer access falls within the allocated size. Herman ten Brugge has written a patch that adds bounds checking to the gcc compiler. Another project for adding bounds checking to gcc is managed by Greg McGary.

The problem with adding bounds checking to every pointer is that it results in a great performance hit. Code size and execution time may grow by 200 or more.

Instead of preventing all buffer overflows we might try to protect the return address from being overwritten. This does not protect against variable attacks, only against the more common stack attacks.

One possible way of doing this is to write the return address to a "safe" place (far from the local buffers) at the start of a function and then restore it just before the function returns. Since the function can call other functions which in turn need to store their return addresses we will need a stack to keep track of all the stored return addresses. In practice this solution thus means separating the stack used for return addresses from the stack used for local variables. This requires a lot of changes to the compiler. A program called StackShield implements this approach, which is also a GNU C compiler extension.

Instead of moving the return address, we can move the buffers. For example, we could allocate all buffers in heap space instead of in stack space. The disadvantage of this solution is that the heap is substantially slower than the stack.

A slightly different approach is taken by the StackGuard program. StackGuard does not prevent the return address from being overwritten, instead it tries to detect when it happens and take the appropriate action (terminating the program before any damage is done).

StackGuard accomplishes this in an ingenious way. Whenever a function is called, code is added for pushing a small value, called a "canary" value, to the stack. This value thus ends up between the local variables and the return address.

	buffer2	buffer1	FP	ca	ret	a	b	c	

When the function exits it checks that the canary value has not been modified before returning. The idea is that a buffer overflow in one of the local variables cannot overwrite the return address without simultaneously destroying the integrity of the canary value. It thus becomes possible to detect whether a buffer overflow has occurred before the function returns.

For this to work, the attacker must not be able to guess the canary value. If the attacker can correctly guess the canary value, he can overwrite the return address without being detected. StackGuard can set the canary value in one of two possible ways. A random value may be used, which is hard for the attacker to guess or the value 0 may be used, which is easy to guess but hard for the attacker to put into his buffer (since most sensitive buffer operations, such as string copying, are terminated by a zero value).

StackGuard can only protect against stack attacks, not against variable attacks, but there is ongoing work to add canary values to function pointers too, since they are also a vulnerable target. The added canary checks increase function call and return times with 40 - 80 %. If the program contains many small function calls and inlining is not used, the total overhead can be in this range. If the program does not contain as many function calls, the overhead will be smaller.

IBM's stack-smashing protector (ssp), originally named ProPolice, is a variation of StackGuard's approach. Like StackGuard, ssp uses a modified compiler (gcc) to insert a canary in function calls to detect stack overflows. However, it adds some interesting twists to the basic idea. It reorders where local variables are stored, and copies pointers in function arguments, so that they're also before any arrays. This strengthens the protection of ssp; this means a buffer overflow can't modify a pointer value (otherwise an attacker who can control a pointer can control where the program saves data using the pointer). By default, it doesn't instrument all functions, only those that it deems as being in need of protection (mainly functions with character arrays).

In theory, this could weaken the protection slightly, but this default improves performance while still protecting against most problems. As a practical matter, they implemented their approach using gcc in a way that is architecture-independent, making it easier to deploy. The widely-respected OpenBSD, which concentrates on security, uses ssp (also known as ProPolice) across their entire distribution as of their May 2003 release.

Microsoft has added a compiler flag (/GS) to implement canaries in its C compiler, based on the StackGuard work.

Operating System Tools

Some people have tried to solve the buffer overflow problem at the level of the operating system. The OS can impose some restrictions which make buffer overflow attacks harder.

Solar Designer has created a patch for Linux that makes the stack non-executable. This means that the attacker can no longer inject his own code into the stack and run it. The patch also maps libc to the 0x00... memory range, so that it is impossible to call libc functions without having a zero in the buffer.

It is known that attackers often can't insert the ASCII NUL character (0) using typical buffer overflow attacks. That means that attackers find it difficult to make a program return to an address with a zero in it. Since that's the case, moving all executable code to addresses with a 0 in it makes attacking the program far more difficult.

The largest contiguous memory range with this property is the set of memory addresses from 0 through 0x01010100, so that's been christened the "executable region" (there are other addresses with this property, but they're scattered). Combined with non-executable stacks, this is pretty valuable: non-executable stacks prevent attackers from sending new executable code, and "executable region" makes it hard for attackers to work around it by exploiting existing code. This protects against stack, buffer, and function pointer overflows, all without recompilation.

However, this method doesn't work for all programs; big programs may not fit in that region (so the protection will be imperfect), and sometimes attackers can get a 0 into their destination. Also, some implementations don't support trampolines, so the protection may have to be disabled for programs that need them. Red Hat's Ingo Molnar implemented this idea in his"exec-shield" patch, which is used by Fedora core (the freely available distribution available from Red Hat). The latest version OpenWall GNU/Linux (OWL) uses an implementation of this approach by Solar Designer.

The disadvantage of having a non-executable stack is that some legitimate programs (though not very many) actually execute content on the stack. These programs will cease to work if the patch is applied.

Another approach is called "split control and data stack" – the idea is to split the stack into two stacks, one to store control information (such as the "return" address) and the other for all the other data. Xu et al. implement this in gcc, and StackShield implements it in the assembler. This makes it much harder to manipulate the return address, but it doesn't defend against buffer overflow attacks that change the data of calling functions.

In fact, there are other approaches as well, including randomizing the locations of executables; Crispen's "PointGuard" extends the canary idea to the heap, and so on. Figuring out how to defend today's computers has become an active research task.

Statically and dynamically allocated buffers

Buffers have a limited amount of space. So there are really two major possibilities for dealing with running out of space.

"Statically allocated buffer" approach: When the buffer runs out, that's it; you complain and refuse to add anything more to the buffer. "Dynamically allocated buffer" approach: When the buffer runs out, you dynamically resize the buffer to a larger size until you run out of total memory. There are disadvantages to the static approach. In fact, the static approach may sometimes create a different vulnerability! Static approaches basically throw away "excess" data. If the program uses the resulting data anyway, an attacker will try to fill up the buffer so that when the data is truncated, the attacker will fill the buffer with what the attacker wanted. If you're using a static approach, you should ensure that the worst an attacker could do won't invalidate some assumption, and a few checks on the final result would be a good idea too.

Dynamic approaches have lots of advantages: they can scale up to larger problems (instead of creating arbitrary limits), and they don't have the problem of truncations causing security problems. But they have problems of their own. When arbitrarily sized data is accepted, you may run out of memory and that may not happen just during input. Any memory allocation can fail, and it's not easy to write C or C++ programs that truly handle that well. Even before truly running out of memory, you can cause the computer to get so busy that it becomes useless. In short, dynamic approaches often make it much easier for attackers to create denial-of-service attacks. So you'll still need to limit inputs. What's more, you'll have to carefully design your program to handle memory exhaustion in arbitrary places, which is not easy.

The most important thing for prevent buffer overflow attacks is based on the programmer to use safe coding practices.

2.4.6 Format string attacks

The `printf` family of C library functions can be used to write the numbers of bytes to memory. The format type `%n` (pointer to `int`) is used to store count of characters written so far in the location pointed to by the next argument (which is the address of an `int`). When a program does not supply a format string, but is user-supplied, then an attacker could carefully craft an input that uses format strings to write to arbitrary locations in memory (return address, function pointer, etc.).

```
#include <stdio.h>
int main(int argc, char *argv[])
{
 printf("Your argument is : \n");
 printf(argv[1]); //should be: printf("%s", argv[1]);
}
```

2.4.7 Race conditions

A race condition is any situation where the behaviour of the program is unintentionally dependent on the timing of certain events. In the following program, `access()` function checks if the real user has permission to access the specified file.

```
#include <stdio.h>
......
int main(int argc, char *argv[])
{
 int file;
 char buf[1024];
 memset(buf, 0, 1024);
 if(argc < 2) {
  printf("Usage: printer [filename]\n");
  exit(-1);
  }
 if(access(argv[1],R_OK) !=0) {
  printf("Cannot access file.\n");
  exit(-1);
  }
 file = open(argv[1],O_RDONLY);
 read(file,buf, 1023);
   close(file);
 printf("%s\n",buf);
 return 0;
}
```

There is a tiny time delay between the calls to `access()` and `open`. An attacker could exploit this small delay by changing the file in question between the two calls. For example, suppose the attacker provided a innocent text file `/home/jeo/try.txt` as an argument. After the call to `access()` returns 0, the attacker can quickly replace `/home/jeo/try.txt` with a symbolic link to `/etc/passwd` (use a program running at background to switches the link between these files repeatedly.)

The call of `access()` should be avoided:

```
#include <stdio.h>
......
int main(int argc, char *argv[])
{
 int file;
 char buf[1024];
 uid_t uid, euid;
 memset(buf, 0, 1024);
 if(argc < 2) {
  printf("Usage: printer [filename]\n");
  exit(-1);
  }
  euid = geteuid();
  uid = getuid();
 seteuid(uid); //drop privileges
 file = open(argv[1],O_RDONLY);
 read(file,buf, 1023);
   close(file);
seteuid(euid);
 printf("%s\n",buf);
 return 0;
}
```

2.5 Security standards

- Orange book or Trusted Computer System Evaluation Criteria (TCSEC). Developed in 1983, and updated in 1985 is a standard for evaluating the security of computers storing classified information. Two sets of requirements are defined:

    1. specific security feature requirements which define the operating system features that are necessary to enforce the security requirements.
    2. assurance requirements which specify the effort necessary to verify the correct implementation of the specific security features.

    The Orange book defines divisions of security criteria:

    - Division D: represents a system with minimal protection. This status is assigned to systems that have been evaluated by TCSEC, but fail to meet the requirements of nay higher security class.
    - Division C: guarantees discretionary protection, including the system makes use of some type of discretionary access control system. This division includes two classes: C1 (discretionary security protection) and C2 (controlled access protection).
    - Division B: guarantees mandatory protection, including the system implements mandatory access control. This division includes three classes: B1 (labeled security protection), B2 (structured protection) and B3 (security domains).

  - Division A: guarantees verified protection, demonstrating that a system has a formal process for verification of security.

- Common criteria or Common Criteria for Information Technology Security Evaluation published in 2008. It is a set of international standards describing a computer security certification. Common Criteria consists of four major concepts:

  1. The target of evaluation (TOE) is the system subject to evaluation.
  2. The protection profile (PP) specifies a required set of security requirements for a broad class of security devices, such as an operating system or firewall.
  3. The security target (ST) which defines the functional and assurance measures of the TOE security that are to be evaluated, perhaps satisfying one or more PPs.
  4. The evaluation assurance level (EAL) which identifies the level of assurance that the TOE satisfies the ST.

In the early 1990s, independent approaches for system assurance were developed in Europe and Canada. The Information Technology Security Evaluation Criteria (ITSEC) 1.2 was released in 1991 as a joint standard used by France, Germany, the Netherlands, and the United Kingdom. ITSEC defines a set of criteria for evaluating a system, called the target of evaluation, to verify the presence of a set of security features and to verify its defenses against a comprehensive set of penetration tests. In Canada, the CanadianTrusted Computer Product EvaluationCriteria (CTCPEC) defined its evaluation approach in 1993. CTCPEC leverages the security feature requirements of the Orange Book while incorporating the notion of distinct assurance levels of the ITSEC approach. Inspired by these evaluation approaches, the Common Criteria approach was developed in 2008. At the same time, the Common Methodology for Information Technology Security Evaluation (CEM) (the technical basis for an international agreement), and the Common Criteria Recognition Arrangement (CCRA) were also developed.

Common Criteria is not a certification that vouches for the security of a product. Some researchers have criticized the Common Criteria as expensive, time-consuming, and not particularly effective at guaranteeing functional security.

Chapter 3

Virtual Private Networks

A VPN is a means of carrying private traffic over a public network.

- Authentication
- Access control
- Confidentiality and data integrity

3.1 VPN types

- Encrypted VPNs use various types of encryption mechanisms to secure the traffic flowing across the public network. IPsec is a common protocol of such VPNs.
- Nonencrypted VPNs are constructed to connect two or more private networks so that users on both networks can seamlessly access resources sitting on either network. The security of the traffic that flows from one private network to another is either not ensured at all or is ensured using means other than encryption. These means include route segregation, in which only the traffic flowing between the two private networks can be routed to either of them. One such example is GRE-based VPNs, in which often higher-layer encryption is used.
- Intranet VPNs are created to connect two or more private networks within the same organization. These often come into existence when a remote office needs to be connected to headquarters or when a company

is acquired and needs to have its network integrated into the acquirer's main network.

- Extranet VPNs are used to connect private networks belonging to more than one organizational unit. These are often used in B-2-B scenarios in which two companies want to conduct business together. A company's partners can also be allowed to use the company's resources by giving them access through a VPN across the Internet.

3.2 Data link layer VPNs

With data link layer VPNs, two private networks are connected on Layer 2 of the OSI model using a protocol such as Frame Relay or ATM. Although these mechanisms provide a suitable way of creating VPNs, they are often expensive, because they require dedicated Layer 2 pathways to be created.

3.2.1 Generic routing encapsulation (GRE)

GRE is a protocol often used in networks to tunnel traffic from one private network to another. It does not provide encryption services, but it does provide low overhead tunneling. The entire encapsulated packet uses the form:

Delivery Header
GRE Header
Payload packet

GRE is used to encapsulate an arbitrary layer protocol over another arbitrary layer protocol. In general, GRE allows a tunnel to be created using a certain protocol, which then hides the contents of another protocol carried within the tunnel.

GRE is often used in conjunction with another encryption protocol to provide security. The VPNs set up using GRE are insecure because GRE does not provide a means of securely encrypting its payload. Means of providing this encryption often reside on the application layer, allowing GRE to create the tunnel needed to connect the private networks while the application layer encryption protocol secures the data. In such situations, GRE mainly acts as a transport medium for carrying the traffic from one private network to another.

3.2.2 PPTP and L2TP

PPTP and L2TP are mainly for dial-up VPNs. Point-to-Point Tunneling Protocol (PPTP) uses a modified GRE to encapsulate PPP packets at the link layer. PPTP does not provide stong encryption nor authentication.

The Layer 2 Tunnel Protocol (L2TP) is an Internet Engineering Task Force (IETF) standard that combines the best features of two existing tunneling protocols: Cisco's Layer 2 Forwarding (L2F) and Microsoft's Point-to-Point Tunneling Protocol (PPTP). L2TP is an extension to the Point-to-Point Protocol (PPP), which is an important component for VPNs.

L2TP supports multiple protocols and unregistered and privately administered IP addresses over the Internet. This allows the existing access infrastructure, such as the Internet, modems, access servers, and ISDN terminal adapters(TAs), to be used. It also allows enterprise customers to outsource dialout support, thus reducing overhead for hardware maintenance costs and 800 number fees, and allows them to concentrate corporate gateway resources.

In running L2TP over an IP backbone, UDP is used as the carrier of all L2TP traffic, including the control traffic used to set up the tunnel between the LNS and the LAC. Of course, the LNS and the LAC need to have IP connectivity between them to be able to set up the tunnel between themselves. The initiator of the tunnel uses UDP port 1701 to send its packets.

3.3 Network layer VPNs

Network Layer VPNs are created using Layer 3 tunneling and/or encryption techniques. An example is the use of the IPsec tunneling and encryption protocol to create VPNs. Network layers provide a suitable place to do encryption.

3.3.1 IPSec

Network security encryption can be classified into two types.

- End-to-end encryption: The encryption process is carried out at the two end systems.
- Link encryption: The encryption process is carried in each link.

Figure 3.1 is a simple example of these two types of encryption.

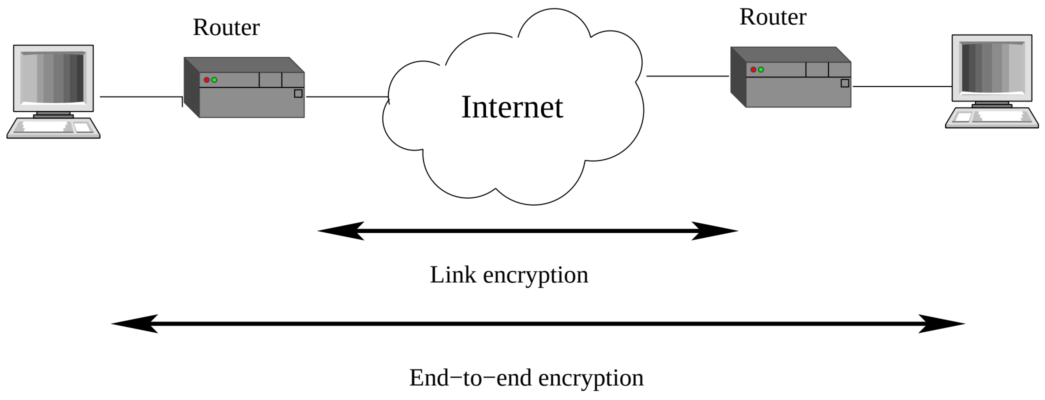

Figure 3.1: Types of Encryption

End-to-end encryption is simple, but it cannot perform at a low level of the communication hierarchy. The address of the message cannot be encrypted, otherwise the packet-switching node cannot route the packet. So end-to-end encryption cannot protect against the traffic analysis attack.

The link encryption can encrypt most data except the link control protocol header. However, the router has to decrypt the data and then encrypt it again.

IPSec considers the security at IP layer. It can be used in a firewall or router. It also can be used for individual users. So IPSec can be used for both end-to-end encryption or link encryption. Since IPSec is below the transport layer, it can be transparent to end users. So there is no need to change the application software or train users on security mechanisms.

IP-level security encompasses three functional areas: authentication, confidentiality and key management.

IPSec documents

Documents of IPSec are developed by IETF as RFC standard. The documents are divided into seven groups as follows.

- Architecture: Covers general concepts, security requirements, definitions and technology.
- Encapsulating Security Payload(ESP): Covers the packet format, packet encryption and, optionally, authentication.
- Authentication Head (AH): Covers the packet format and packet authentication.
- Encryption Algorithm: Describes various encryption algorithms used for ESP.
- Authentication Algorithm: Describes various authentication algorithms for AH and ESP.
- Key Management: Describes key management schemes.
- Domain of Interpretation (DOI): Contains values needed for the other documents to related to each other.

We will not discuss all these documents, but select several most important specifications.

An important concept that appears in IPSec is the security association (SA). An association is a one-way relationship between a sender and a receiver. If a two-way exchange is needed, then two SA are required. An SA is uniquely identified by three parameters.

- Security Parameters Index (SPI): A bit string assigned to this SA, which is carried in AH and ESP headers to enable the receiving system to select the SA under which a received packet will be processed.
- IP Destination Address: This is the address of the destination endpoint of the SA (end user, firewall or router).
- Security Protocol Identifier: This indicates whether the SA is an AH or ESP security association.

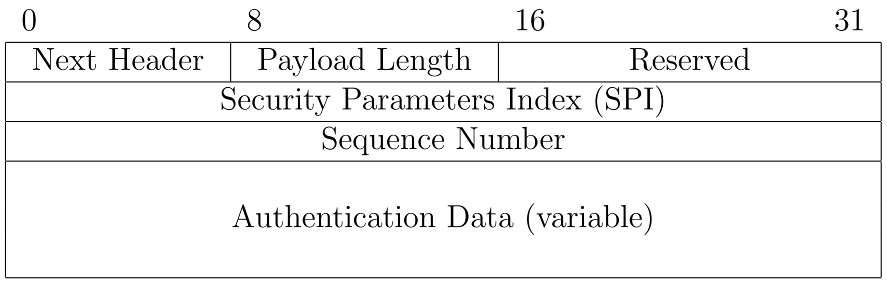
0
8
16
31
Next Header
Payload Length
Reserved
Security Parameters Index (SPI)
Sequence Number
Authentication Data (variable)

Figure 3.2: IPSec Authentication Header

Authentication Header

The Authentication Header (AH) provides support for data integrity and authentication of IP packets. The authentication header is defined as in Figure 3.2

The fields of AH are as follows.

- Next Header (8 bits): Identifies the type of header immediately following this header.
- Payload Length (8 bits): Length of AH in 32-bit words, minus 2. The default length of the authentication data field is three 32-bit words. So the default value of Payload Length is 4.
- Reserved (16 bits): For future use.
- Security Parameters Index (32 bits): Identifies a security association.
- Sequence Number (32 bits): A monotonically increasing counter value to prevent replay attack.
- Authentication Data (variable, must be an integral number of 32-bit words): Contains the Integrity Check Value (ICV), or MAC, for this packet. The default length of this field is 3 32-bit word.

Sequence numbers are used for anti-replay service. An attacker might obtain a copy of a packet and later transmits it to the destination. Thus when a new SA established, sender initializes a sequence number counter to 0. Each time that a packet is sent on this SA, the sender increases the counter and place the value in the Sequence Number field. If the number reaches to $2^{32} - 1$, a new SA with a new key needs to be set.

Note that IP is connectionless. So the packets may not be delivered in order and some packets might be missing. So the IPSec authentication document dictates that the receiver should implement a window of size W (default value of W is 64). The right edge of the window represents the largest sequence number, N, received so far. For any packet with a sequence number in the range from $N - W + 1$ to N that has been correctly received, the corresponding slot in the window is marked. When a packet is received, the receiver does the following.

- If the received packet falls within the window and is new, the MAC is checked and the corresponding slot in the window is marked if the authentication is good.
- If the received packet is to the right of the window, The MAC is check. If the MAC is good, the window is advanced so that this sequence number is the right edge of the window.
- If the received packet is to the left of the window, or if the authentication fails, the packet is discarded.

A receiver's window is shown in Figure 3.3

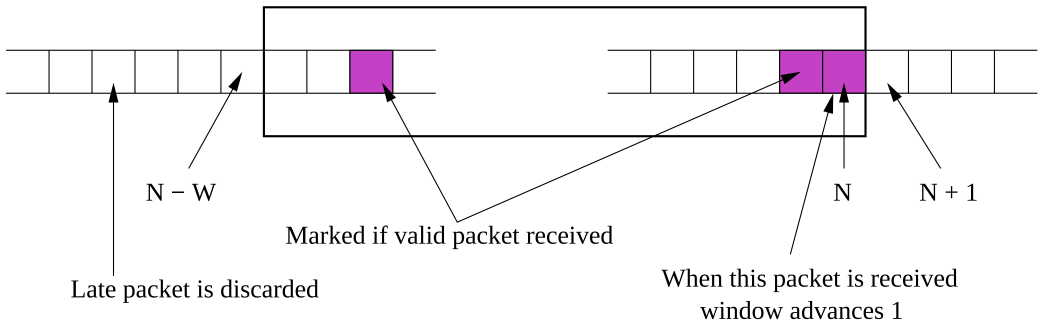
N − W
N
N + 1
Marked if valid packet received
Late packet is discarded
When this packet is received window advances 1

Figure 3.3: A Receiver's Window

An ICV is a MAC (or HMAC) or a truncated version of a code produced by a MAC (or HMAC) algorithm. For example, an HMAC-MD5 or HMAC-SHA-1 is used to produce the code and then the first 96 bits is truncated. The input of the MAC algorithm includes those fields which will not be changed in transit (immutable) or that are predictable in value upon arrival at the endpoint. That includes immutable fields in IP header, AH header other than Authentication data and upper-level protocol data. The mutable fields are set to zero when an ICV is computed.

For example, in IPv4 header, the Internet Header Length (IHL) and Source Address are immutable. The Destination Address is predictable field. The Time to Live and Header Checksum fields are mutable fields, which are zeroed prior to compute ICV.

Note that the AH uses MAC, so there is a shared secret key. We will discuss the key management later.

There are two modes in which the IPSec authentication service can be used.

- Transport Mode: Used in end-to-end authentication (e.g., server to client).
- Tunnel Mode: Used in end-to-intermediate authentication (e.g., workstation to firewall).

These two modes are explained in Figure 3.4

The position of AH at the packet is as follows. A simple packet before applying AH looks like the following.

IPv4	IP-H	TCP-H	Data	

IPv6	IP-H	Extension H	TCP-H	Data

Then the packet after applying AH in transport mode is

IPv4	IP-H	AH	TCP-H	Data

IPv6	IP-H	ext H	AH	dest	TCP-H	Data

For the tunnel mode, the packet after applying AH looks as

IPv4	New IP-H	AH	orig IP-H	TCP-H	Data

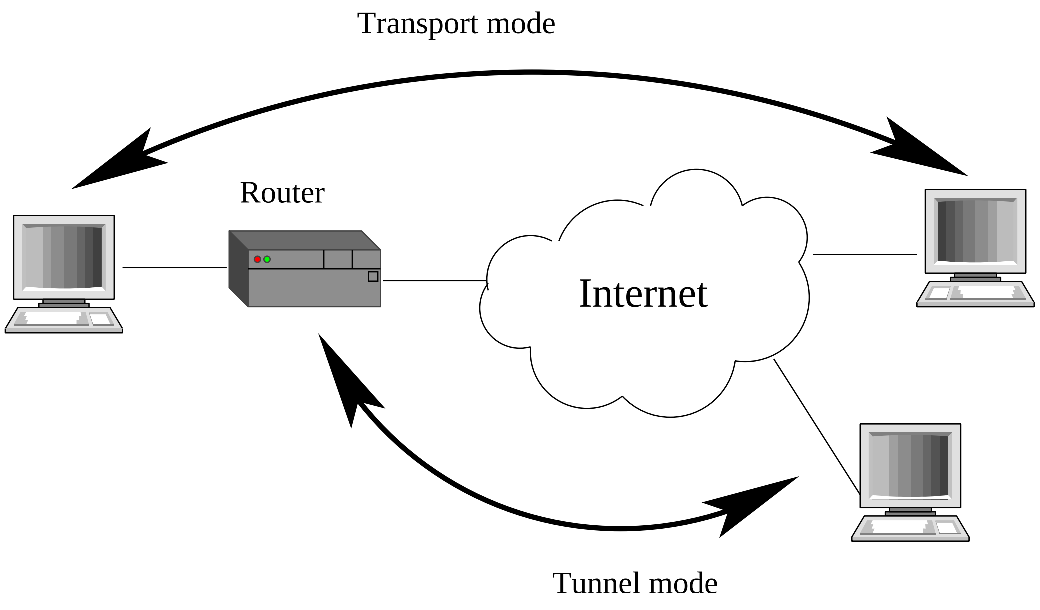

Figure 3.4: IPSec modes

IPv6	New IP-H	ext H	AH	orig IP-H	dest	TCP	Data

Basically, all the fields are authenticated except for mutable fields which are set to be zero values before using HMAC.

Encapsulating Security Payload (ESP)

The ESP provides confidentiality services. As an optional feature, ESP can also provide authentication service.

The format of ESP is as in Figure 3.5

SPI identifies a security association. Sequence number is similar to that of AH. These two 32-bit words are the head of ESP. Payload Data is a transport level segment (transport mode) or IP packet (tunnel mode) that is protected by encryption. The length of payload data is variable. The length of padding is between 0 to 255 bytes. The padding is used to satisfies the requirement of encryption function and the requirement of the alignment of the ESP format. Sometimes, padding also can be used to provide partial traffic flow confidentiality by concealing the actual length of the payload. Pad length (8 bites) indicates the number of pad bytes. Next header (8 bits)

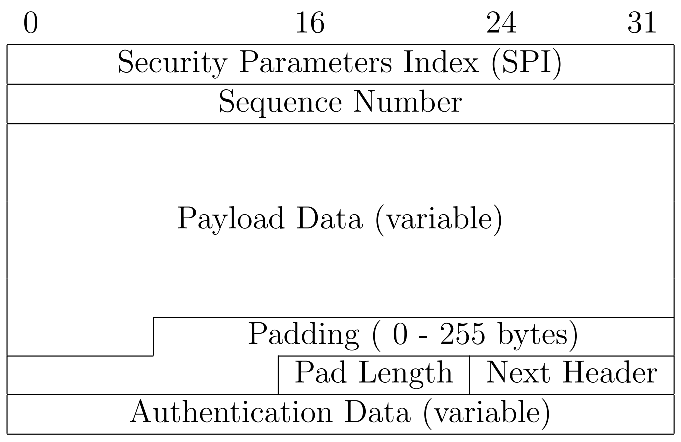

Figure 3.5: IPSec ESP Format

identifies the type of data contained in the payload data field (first header in that payload). The Authentication Data field contains the ICV.

The Payload Data, Padding, Pad Length and Next Header fields are encrypted. If the encryption algorithm needs some initialization vector (IV), then this data may be carried explicitly at the beginning of the Payload Data field. In that case, the encryption is usually started right after IV.

Then the packet after applying ESP in transport mode is

		⟵	Authenticated		⟶	
			⟵	Encrypted	⟶	
IPv4	IP-H	ESP-H	TCP-H	Data	ESP-T	ESP auth

			⟵	Authenticated		⟶		
				⟵	Encrypted	⟶		
IPv6	IP -H	exten -H	ESP -H	dest -H	TCP -H	Data	ESP tailer	ESP auth

For the tunnel mode, the packet after applying ESP looks as

		⟵	Authenticated			⟶	
			⟵	Encrypted		⟶	
IPv4	New IP -H	ESP -H	orig IP -H	TCP -H	Data	ESP tailer	ESP auth

$\longleftarrow$ Authenticated $\longrightarrow$ (ESP-H through ESP-T)

$\longleftarrow$ Encrypted $\longrightarrow$ (orig IP-H through ESP-T)

IPv6	New IP -H	exten -H	ESP -H	orig IP -H	dest -H	TCP -H	Data	ESP -T	ESP auth

When the authentication option is chosen, the authenticated part covers from the ESP header to the ESP trailer.

In practice of using IPSec, we can use different combinations of AH and ESP. We can use ESP with authentication. In this case, the authentication is applied to the ciphertext instead of plaintext. We also can use two SAs. One is for AH and one is for ESP. The inner is used for ESP without authentication and the outer is for AH. Sometimes, we can use several SAs. The IPSec architecture documents lists four examples of combinations of SAs that must be supported by compliant IPSec hosts.

Example 1 implements IPSec for both end systems. Several SA may be used. They can use AH in transport mode, ESP in transport mode, AH followed by ESP in transport mode, etc. (see Figure 3.6).

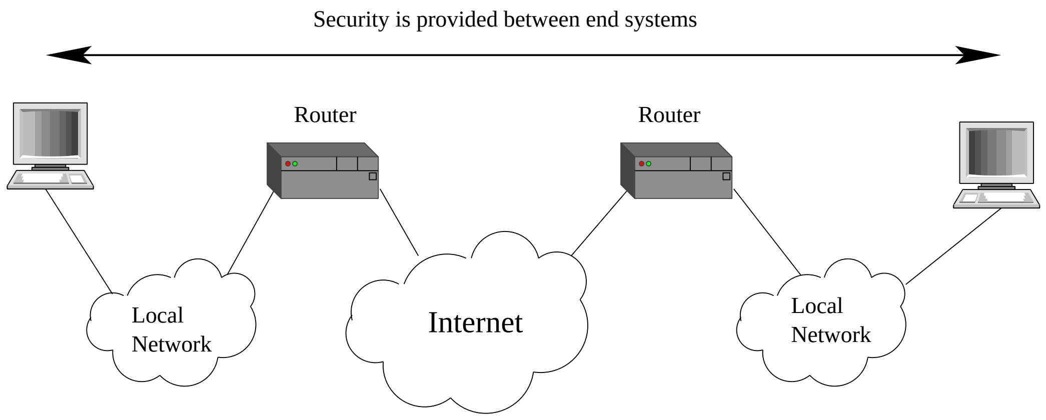

Figure 3.6: Example 1

Example 2 implements IPSec only for gateways such as routers, firewalls etc. In this case, usually only a single tunnel SA is used which supports AH, ESP or ESP with the authentication option. This implementation can be used to support simple virtual private network (see Figure 3.7).

Example 3 combines example 2 with example 1. One or several end-to-end SA is added to the gateway-to-gateway security (see Figure 3.8).

Example 4 supports a remote host to reach an organization's firewall and

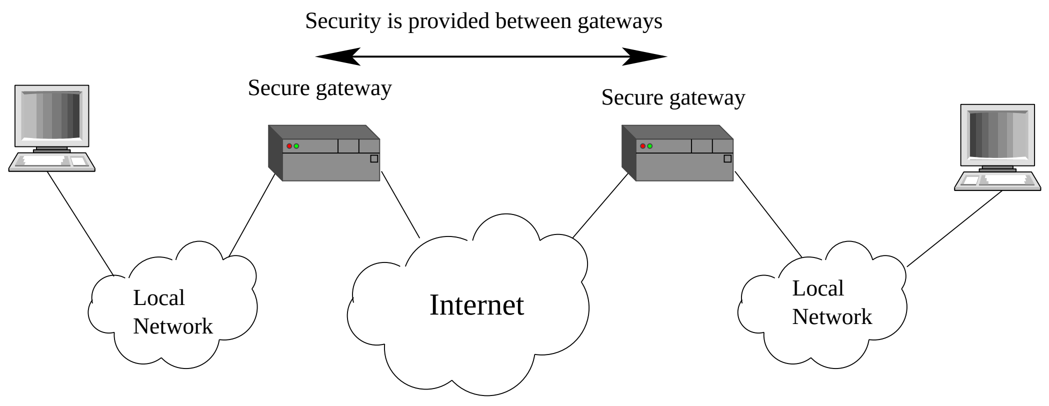

Figure 3.7: Example 2

then to access the end system behind the firewall. A tunnel mode is used between the remote host and the firewall. (see Figure 3.9).

Key Management

In AH and ESP, secret keys are required for communications. An important part of IPSec is key management.

IPSec supports two types of key management for both AH and ESP:

- Manual: A system administrator manually configures each system with its own keys and with the keys of other communicating systems.
- Automated: An automated system enables the on-demand creation of keys for SA and facilities the use of keys in a large system.

IPSec's default automated key management protocol is ISAKMP/Oakley. The Oakley key exchange protocol was discussed before, which is based on Deffie-Hellman key exchange scheme. Now we introduce ISAKMP key management. ISAKMP provides a framework for internet key management. It does not dictate a specific key exchange algorithm. Other key exchange algorithms can also be used with ISAKMP.

ISAKMP (Internet Security Association and Key Management Protocol) defines procedures and packet formats to establish, negotiate, modify and delete security associations.

The ISAKMP header format is shown in Figure 3.10

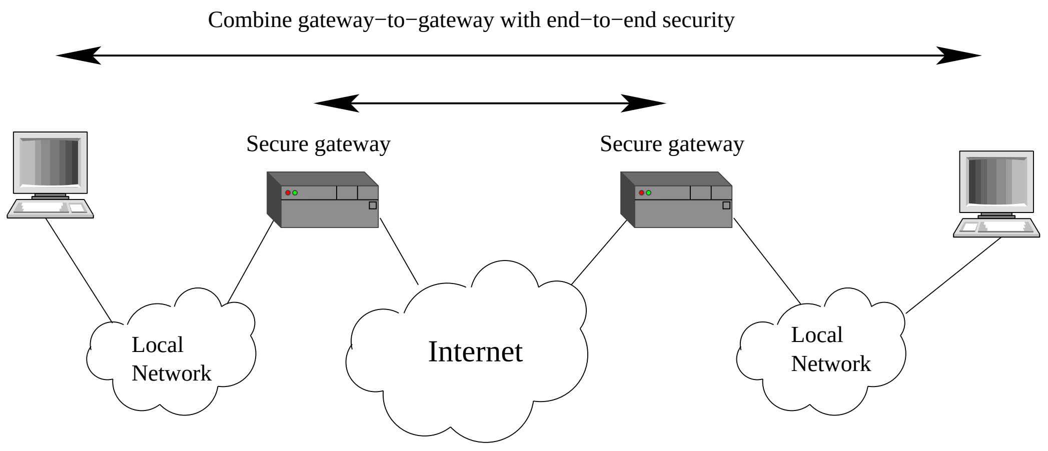

Figure 3.8: Example 3

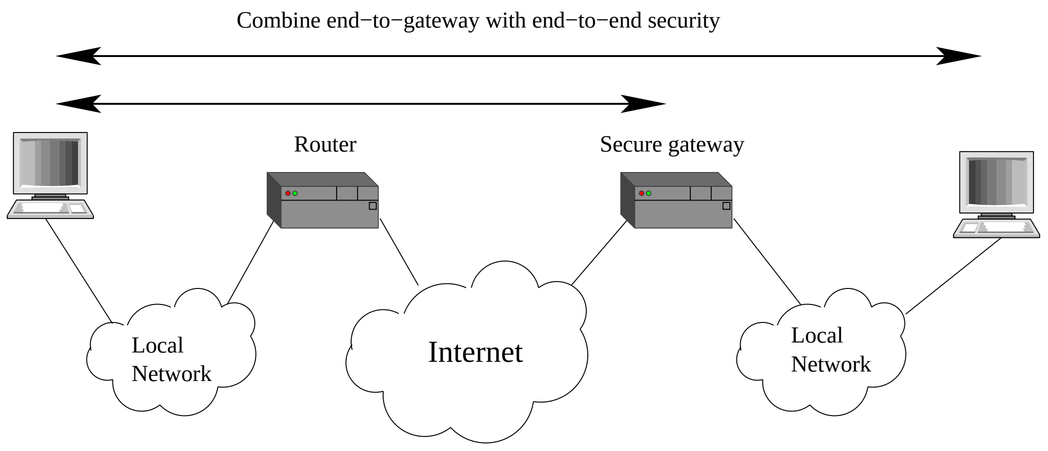

Figure 3.9: Example 4

0	8		16	24 31
Initiator Cookie				
Responder Cookie				
Next payload	MjVer	MnVer	Exchange Type	Flags
Message ID				
Length				

Figure 3.10: ISAKMP header

Initiator Cookie (64 bits) initiated SA establishment, SA notification, or SA deletion. Responder Cookie(64 bits) is used for responding; null in the first message from initiator. Next Payload (8 bits) indicates the type of the first payload in the message, which is followed the ISAKMP header. Major Version (4 bits) indicates major version of ISAKMP in use and Minor Version (4 bits) indicates minor version in use. Exchange Type (8 bits) indicates the type of exchange. Flags (8 bits) indicates specific options set for this ISAKMP exchange. Message ID (32 bits) is the unique ID for this message. Length (32 bits) is the length of total message (header plus all payloads) in octets.

ISAKMP Payload is followed the ISAKMP header. All ISAKMP payloads begin with the same generic payload header which is shown in Figure 3.11

0	8	16 31
Next Payload	Reserved	Payload Length

Figure 3.11: ISAKMP Payload Header

Next Payload field uses 8 bits which has value 0 if this is the last payload in the message, otherwise its value is the type of the next payload. The Payload Length field indicates the length in octets of this payload, including the generic payload header.

The payload types defined for ISAKMP are as follows.

- SA payload (SA): Used to begin the establishment of an SA. Parameters include Domain of Interpretation (e.g., IPSec DOI), and a situation parameter which defines the security policy for this negotiation.

- Proposal payload (P): Contains information used during SA negotiation. The payload indicates the protocol for this SA (ESP or AH), entity's SPI and the number of transforms.

- Transform payload (T): Includes a transform number which identifies this particular payload. The payload also contains the transform ID and Attributes to specify the transform (e.g., 3DES for ESP, HMAC-SHA-1-96 for AH) with its associated attribution (e.g., hash length).

- Key Exchange payload (KE): Used for key exchange (Oakley, PGP etc.). The payload contains the data required to generate a session key.

- Identification payload (ID): Used to identify the communicating peers. The Data field will contain an IPv4 of IPv6 address.

- Certificate payload (CERT): Transfers public-key certificate. The payload indicates the type of certificate of certificate-related information.

- Certificate Request payload (CR): Requests the certificate of the other communicating entity. Lists certificate types and certificate authorities that are acceptable.

- Hash payload (HASH): Contains data generated by a hash function over some message.

- Signature payload (SIG): Contains data generated by a digital signature scheme.

- Nonce payload (NONCE): Contains random data to protect against replay attacks.

- Notification payload (N): Contains either error or status information associated with this SA.

- Delete payload (D): Indicates one or more SAs that sender has deleted from its database.

RFC 2408 lists five default exchange types which are described in Figure 3.12. In the table SA refers to an SA payload with associated Protocol and transform payload. NONCE is a random number used to ensure against replay attacks. AUTH payload is used to authenticate keys, identities and the nonce. In the Identity Protection Exchange, the two parties identities are protected by encryption. The Aggressive Exchange minimizes the number of exchanges at the expense of not providing identity protection.

We use the diagram in Figure 3.13 to illustrate the payloads exchanged between the two parties in the first round trip exchange. In this example, the initiator propose two proposals. The responder should reply with one proposal. This round is to initiate an SA. This example is the Identity Protection Exchange in ISAKMP.

The second round trip exchange is to generate key and send nonce. We explain it in Figure 3.14

We omit the third trip round. It is not difficult to describe this round now.

The Internet Key Exchange (IKE, RFC 2409) further detailed the key exchange scheme. The IKE uses part of Oakley and part of SKEME (Secure Key Exchange MEchanism protocol, a versatile key exchange technique which provides anonymity, repudiability and quick key refreshment) in conjunction with ISAKMP to obtain authenticated keying material. Two phases of exchange are defined in IKE. Phase 1 is where the two ISAKMP peers establish a secure, authenticated channel with which to communicate. Phase 2 is where Security Associations are negotiated on behalf of services such as IPsec or any other service which needs key material and/or parameter negotiation. Two modes of phase 1, Main mode and Aggressive mode, are described in IKE. For phase 2, IKE describes Quick mode. There are different authentication options for each mode. We will not discuss all the details, but give some examples to explain.

An example of Main Mode is authenticated with a revised mode of public key encryption. This mode is defined as follows.

1. $I \rightarrow R$: HDR, SA.

   Where HDR is the header of ISAMKP whose exchange type is the mode.

2. $I \leftarrow R$: HDR, SA.

(a) Base Exchange

(1) $I \rightarrow R$: SA; NONCE	Begin ISAKMP-SA or Proxy negotiation
(2) $I \leftarrow R$: SA; NONCE	Basic SA agreed upon
(3) $I \rightarrow R$: KE;IDi; AUTH	Key Generated (by responder) Initiator Identity Verified by Responder
(4) $I \leftarrow R$: KE; IDr; AUTH	Responder Identity Verified by Initiator Key Generated (by initiator) SA established

(b) Identity Protection Exchange

(1) $I \rightarrow R$: SA	Begin ISAKMP-SA or Proxy negotiation
(2) $I \leftarrow R$: SA	Basic SA agreed upon
(3) $I \rightarrow R$: KE; NONCE	Key generated
(4) $I \leftarrow R$: KE; NONCE	Key generated
(5)*$I \rightarrow R$: IDi; AUTH	Initiator Identity Verified by Responder
(6)*$I \leftarrow R$: IDr; AUTH	Responder Identity Verified by Initiator SA established

(c) Authentication Only Exchange

(1) $I \rightarrow R$: SA;NONCE	Begin ISKMP-SA or Proxy negotiation
(2) $I \leftarrow R$: SA;NONCE;IDr;AUTH	Basic SA agree upon Responder identity verified by Initiator
(3) $I \rightarrow R$: IDi;AUTH	Initiator identity verified by responder; SA established

(d) Aggressive Exchange

(1) $I \rightarrow R$: SA;KE;NONCE;IDi	Begin ISKMP-SA or Proxy negotiation and key exchange
(2) $I \leftarrow R$: SA;KE;NONCE;IDr;AUTH	Initiator identity verified by responder; Key generated; Basic SA agreed upon
(3)* $I \rightarrow R$: AUTH	Responder identity verified by initiator; SA established

(e) Informational Exchange

(1) $I \rightarrow R$: N/D	Error notification or deletion

Notation:
I = initiator
R = responder
* = payload encryption after the ISAKMP header

Figure 3.12: ISAKMP Exchange Types

ISAKMP Header with Exchange Type of Main Mode and Next Payload of ISA-SA		
0	Reserved	Payload Length
Domain of Interpreting		
Situation		
0	Reserved	Payload Length
Proposal # 1, PROTO-ISAKMP, SPI size=0 || # of Transforms		
ISA-TRANS	Reserved	Payload Length
Transform # 1, KEY-OAKLEY, preferred SA attributes		
0	Reserved	Payload Length
Transform # 2, KEY-OAKLEY, preferred SA attributes		

Figure 3.13: ISAKMP exchange round 1

ISAKMP Header with Exchange Type of Main Mode and Next Payload of ISA-KE		
ISA-NONCE	Reserved	Payload Length
D-H public Value (α^x from initiator α^y from responder)		
0	Reserved	Payload Length
$NONCE_I$ or $NONCE_R$		

Figure 3.14: ISAKMP exchange round 2

3. $I \rightarrow R$: HDR, $[HASH(1)]$, $E_{Pub_R}(NONCE_I)$, $E_{K_I}(KE)$, $E_{K_I}(ID_I)$, $[E_{K_I}(CERT_I)]$.

   Where [x] indicates that x is optional. $HASH(1)$ is a hash (using the negotiated hash function) of the certificate which the initiator is using to encrypt the nonce and identity. E_{Pub_R} is the encryption function using R's public key Pub_R. $NONCE_I$ is I's nonce payload. E_{K_I} is a symmetric encryption algorithm (from the SA payload) with I's secret key K_I which is derived from the nonce.

4. $I \leftarrow R$: HDR, $E_{Pub_I}(NONCE_R)$, $E_{K_R}(KE)$, $E_{K_R}(ID_R)$.

   E_{Pub_I} is the encryption function using I's public key Pub_I. $NONCE_R$ is I's nonce payload. E_{K_R} is a symmetric encryption algorithm (from the SA payload) with R's secret key K_R which is derived from the nonce.

5. $I \rightarrow R$: HDR*, $HASH_I$.

   HDR* indicates payload encryption. $HASH_I$ is the hash payload produced from HMAC. The key is derived from I's cookie and the nonce. The hash function is performed on the Diffie-Hellman key exchange values, cookies, SA and I's ID.

6. $I \leftarrow R$: HDR*, $HASH_R$.

   $HASH_R$ is similar to $HASH_I$.

The Quick Mode is defined as follows. The payload in this mode is encrypted. Since this mode is basically in phase 2, the encryption is possible.

1. $I \rightarrow R$: HDR*, $HASH(1)$, SA, $NONCE_I$, [, KE], [, ID_{cI}, ID_{cR}].

   If ISAKMP is acting as a client negotiator on behalf of another party, then the identities are passed as ID_{cI} and ID_{cR}.

2. $I \leftarrow R$: HDR*, $HASH(2)$, SA, $NONCE_R$, [, KE], [, ID_{cI}, ID_{cR}].

   $HASH(2)$ is the hash payload of message ID, SA, $NONCE_R$ and optional KE, ID_{cI}, ID_{cR}

3. $I \rightarrow R$: HDR*, $HASH(3)$

   $HASH(3)$ is the hash payload of message ID, $NONCE_I$ and $NONCE_R$.

ISAKMP is not only for IPSec. It can be used for other security issues of internet. For example, is it used for VPN (virtue private network).

3.3.2 MPLS

PPTP, L2TP, IPSec are not all interoperable and may be tied to one equipment or a single service provider vendor. IP is a connectionless protocol so each separate packet moves through the network to its destination along a path determined by a distributed set of routing tables and the current network topology. The Multi-Protocol Label Switching (MPLS) attaches labels to each data packet and forwards data packets based on these labels. Intermediate MPLS nodes do not need to look at the content of the data in each packet. In particular the destination IP addresses in the packets are not examined, which enables MPLS to offer an efficient encapsulation mechanism for private data traffic traversing through networks thus creating an effective tunnel for standards-based VPNs.

MPLS provides high-speed data forwarding between Label Switched Routers (LSRs) together with reservation of bandwidth for traffic flows. As a packet enters the network, an edge router looks up the destination address of the packet and tags it with a small and fixed-format label that specifies the route and optionally specifies Class of Service (CoS) attributes. At each hop across the network, the packet is routed based on the value of the incoming interface and label, then dispatched to an outwards interface with a new label value. The Label Switched Path (LSP) that a packet takes across the MPLS network is defined by the transition in label values, as the label is swapped at each LSR. As the labeled packet moves across the network, each router uses the label to choose the destination rather than looking up the destination address for each packet in a routing table. As the packet leaves the MPLS network, an edge router uses the destination address to direct the packet to its final destination. Subsequent packets in the data stream are quickly and automatically labled in this way.

The Label Edge Routers (LERs), operate at the edge of the access network and MPLS network and play a critical role in the assignment and removal of labels as traffic enters or exits an MPLS network. The set of all packets that are forwarded in the same way is called Forwarding Equivalence Class (FEC).

The exact format of a label and how it is added to the packet depends on the data link technique used in the MPLS network. So it works conjunction with the layer 2 and layer 3.

There are two standardized protocols for managing MPLS paths: LDP (Label Distribution Protocol) and RSVP-TE, an extension of the Resource

Reservation Protocol (RSVP) for traffic engineering. Furthermore, there exist extensions of the BGP protocol that can be used to manage an MPLS path.

An MPLS header does not identify the type of data carried inside the MPLS path. If one wants to carry two different types of traffic between the same two routers, with different treatment by the core routers for each type, one has to establish a separate MPLS path for each type of traffic.

MPLS is currently in use in IP-only networks and is standardized by RFC 3031. It is deployed to connect as few as two facilities to very large deployments. For example, in the retail sector, it is not uncommon to see deployments of 2000 to 5000 locations to communicate transaction data to a headquarters data center.

3.4 Application layer VPNs

Application layer VPNs are created to work specifically with certain applications.

One of the main drawbacks of application layer VPNs is that often they are not seamless. The user must perform an action to enable the end devices for creating the VPN for each of the various applications. As new services and corresponding applications are added, support for them must be developed as well. This is unlike network layer and link layer VPNs, which provide seamless VPN connectivity for all application after the basic VPN has been set up.

Chapter 4

Database Security

4.1 Database

- Database management system (DBMS): a suite of programs for constructing and maintaining the database and for offering ad hoc query facilities to multiple users and applications. A query language provides a uniform interface to the database for users and applications.

  Figure 4.1 provides a diagram of a simplified DBMS architecture. Developers use DDL (data definition language) to define the database logical structure and procedural properties, which are presented by a set of database description tables. A data manipulation language (DML) provides a powerful set of tools for application developers. Query language are declarative languages designed to support users. The database description tables are used to manage the physical database. The interface to the database is through a file manager module and a transaction manager module. The DBMA uses authorization tables to ensure the user has permission to execute the query language statement on the database. The concurrent access table prevents conflicts when simultaneous, conflicting commands are executed.

  Database systems provide efficient access to large volumes of data, which requires security mechanisms different from operating system. OS system usually allows a user to access a entire file for read or write, but not to limit access to specific records or fields in that file. A DBMS typically allows this type of more detailed access control and also provides a wider range of commands, such as to select, insert, update, or

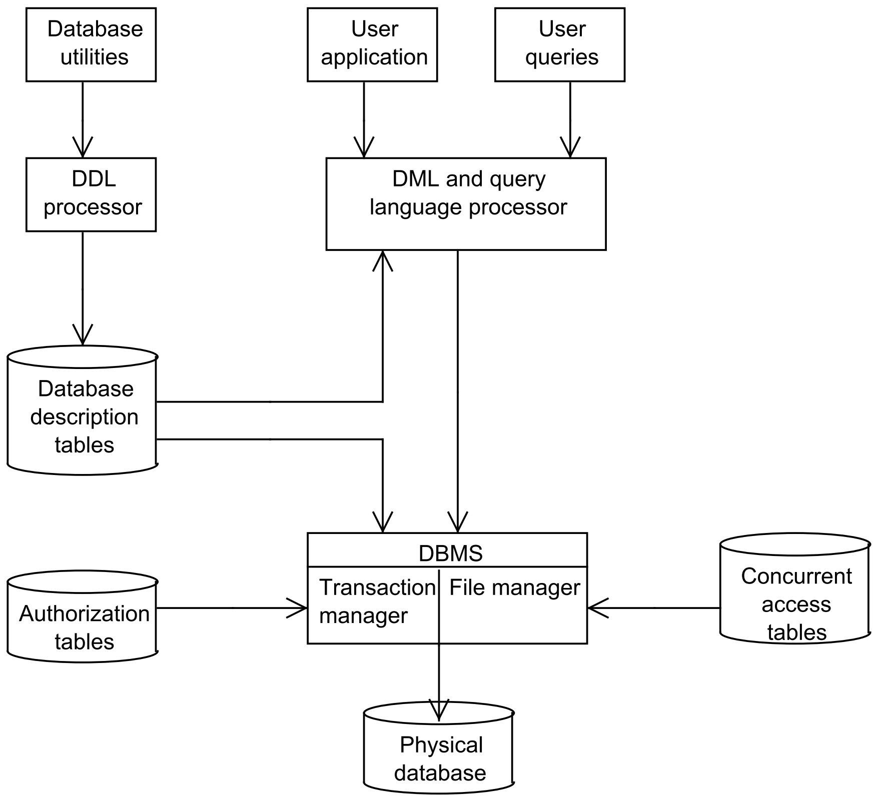

Figure 4.1: DBMS Architecture

delete specified items in the database.

- Relational database:
  - The basic building block of a database is a table of data, consisting of rows and columns, similar to a spreadsheet. Drawback: possible blank cells and difficult to add new columns. The relational database structure enables the creation of multiple tables tied together by a unique identifier that is present in all tables.
  - The basic building block of a relational database is a relation, which is a flat table. Rows are referred to as tuples and columns are referred to as attributes. A primary key is used to uniquely identify a row in a table. The primary key consists of one or more column names.
  - To create a relationship between two tables, the attributes that define the primary key in one table must appear as attributes in another table, where they are referred as a foreign key.

Did	Dname	Dacctno
4	account	528221
13	education	202035
15	services	223945
...	...	...

Ename	Did	S code	Eid	Ephone
Robin	15	23	2345	6127092485
Neil	13	12	5088	6127092246
Code	15	22	9664	6127093148
Holly	4	26	7712	6127099348
...	...	...	...	...

Table 4.1: Two tables in a relational database

In Table 4.1, one table is Department Table and another is Employee Table. Did is the primary key of Department Table. Eid is the primary key of Employee Table, while Did is the foreign key of Employee Table. A foreign key value can appear multiple times in a table.

  - A view is a virtual table which is a result of a query. Table 4.2 is an example of a view. Views are often used for security purpose. A view can provide restricted access to a relational database so that a user only has access to certain rows or columns.
  - Structured query language: There are several versions of the ANSI/ISO standard and a variety of different implementations of SQL, but the basic syntax and semantics are the same. Example:

Dname	Ename	Eid	Ephone
service	Robin	2345	6127092485
education	Neil	5088	6127092246
service	Code	9664	6127093148
account	Holly	7712	6127099348
...	...	...	... ...

Table 4.2: A view derived from the database

```
CREATE TABLE department (
  Did INTEGER PRIMARY KEY,
  Dname CHAR (30),
  Dacctno CHAR(6) )

CREATE TABLE employee (
  Ename CHAR (30),
  Did INTEGER,
  Scode INTEGER,
  Eid INTEGER PRIMARY KEY,
  Ephone CHAR (10),
  FOREIGN KEY (Did) PREFERENCES department (Did) )
```

The basic command for retrieving information is the SELECT statement. The following query returns the Ename, Did and Ephone fields from the Employee table for all employees assigned to department 15.

```
SELECT Ename, Eid, Ephone
  FROM Employee
  WHERE Did = 15
```

The view in Figure 4.2 is created using the following statement:

```
CREATE VIEW newtable (Dmane, Ename, Eid, Ephone)
AS SELECT D.Dmane, E.Ename, E.Eid, E.Ephone
FROM Department D Employee E
WHERE E.Did = D.Did
```

Other common commands are `INSERT, UPDATE, DELETE, AND, OR, UNION` etc.

```
DELETE FROM employee WHERE Scode < 22

INSERT INTO employee
VALUES  ('David', 4, 25, 4576, '6127099234')

UPDATE employee
SET Scode=25
WHERE Emane='Holly'
```

- Google big tables: Bigtable is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers. Many projects at Google store data in Bigtable, including web indexing, Google Earth, and Google Finance. These applications place very different demands on Bigtable, both in terms of data size (from URLs to web pages to satellite imagery) and latency requirements (from backend bulk processing to real-time data serving). Despite these varied demands, Bigtable has successfully provided a flexible, high-performance solution for all of these Google products. A simple data model provided by Bigtable gives clients dynamic control over data layout and format.

  In many ways, Bigtable resembles a database: it shares many implementation strategies with databases. Parallel databases and main-memory databases have achieved scalability and high performance, but Bigtable provides a different interface than such systems. Bigtable does not support a full relational data model; instead, it provides clients with a simple data model that supports dynamic control over data layout and format, and allows clients to reason about the locality properties of the data represented in the underlying storage. Data is indexed using row and column names that can be arbitrary strings. Bigtable also treats data as uninterpreted strings, although clients often serialize various forms of structured and semi-structured data into these strings. Clients can control the locality of their data through careful choices in their schemas. And Bigtable schema parameters let clients dynamically control whether to serve data out of memory or from disk.

  A Bigtable is a sparse, distributed, persistent multidimensional sorted map. The map is indexed by a row key, column key, and a timestamp; each value in the map is an uninterpreted array of bytes.

```
(row:string, column:string, time:int 64) ↦ string
```

The row keys in a table are arbitrary strings. The row range for a table is dynamically partitioned. Each row range is called a tablet, which is the unit of distribution and load balancing. Every read or write of data under a single row key is atomic (regardless of the number of different columns being read or written in the row), a design decision that makes it easier for clients to reason about the system's behavior in the presence of concurrent updates to the same row. Column keys are grouped into sets called column families, which form the basic unit of access control. All data stored in a column family is usually of the same type. A column family must be created before data can be stored under any column key in that family; after a family has been created, any column key within the family can be used. In contrast, a table may have an unbounded number of columns. Each cell in a Bigtable can contain multiple versions of the same data; these versions are indexed by timestamps.

row keys (sorted)		*column family*	*column family*	
	language:	contents:	anchor: cnnsi.com	anchor: mylook.ca
com.aaa	EN	⟨!DOCTYPE...		
com.cnn.www	EN	⟨!DOCTYPE... ⟨!DOCTYPE ...	`"CNN"`	`"CNN.com"`
com.cnn.www/TECH	EN	⟨!DOCTYPE ...		
com.weather	EN	⟨!DOCTYPE...	`"WEATHER"`	`"Weather"`

Table 4.3: Bigtable column families and columns

Table 4.3 shows a slice of a Bigtable. A column key uses the syntax: `family:qualifier`. The family names must be printable, but qualifiers may be arbitrary strings. This example table stores Web pages. The row name is a reversed URL. The `contents` column family contains the page content, and the `anchor` column family contains the text of any anchors that reference the page. CNN's home page is referenced by both the Sports Illustrated and the MY-look home pages, so the row contains `anchor` columns. Each anchor cell has one version, but the contents column has two versions (we omitted the timestamps in the table).

Bigtable comprises a client library (linked with the user's code), a master server that coordinates activity, and many tablet servers. Tablet servers can be added or removed dynamically. The master assigns tablets to tablet servers and balances tablet server load. It is also responsible for garbage collection of files in GFS and managing schema changes (table and column family creation). Each tablet server manages a set of tablets (typically 10-1,000 tablets per server). It handles read/write requests to the tablets it manages and splits tablets when a tablet gets too large. Client data does not move through the master; clients communicate directly with tablet servers for reads/writes. The internal file format for storing data is Google's SSTable, which is a persistent, ordered, immutable map from keys to values. Bigtable uses the Google File System (GFS) for storing both data files and logs. A cluster management system contains software for scheduling jobs, monitoring health, and dealing with failures.

- Data warehouse: A centralized data warehouse is usually used to store enterprise data. The data are organized based on a star schema, which usually has a fact table with part of the attributes called dimensions and the rest called measures. Each dimension is associated with a dimension table indicating a dimension hierarchy. The dimension tables may contain redundancy, which can be removed by splitting each dimension table into multiple tables, one per attribute in the dimension table. The result is called a snowflake schema. Figure 4.2 is a simple example of star schema. A data warehouse usually stores data collected from multiple data sources, such as transactional databases throughout an organization. The data are cleaned and transformed to a common consistent format before they are stored in the data warehouse. Subsets of the data in a data warehouse can be extracted as data marts to meet the specific requirements of an organizational division. Unlike in transactional databases where data are constantly updated, typically the data stored in a data warehouse are refreshed from data sources only periodically.

  OLAP stands for On-Line Analytical Processing. Unlike statistical databases which usually store census data and economic data, OLAP

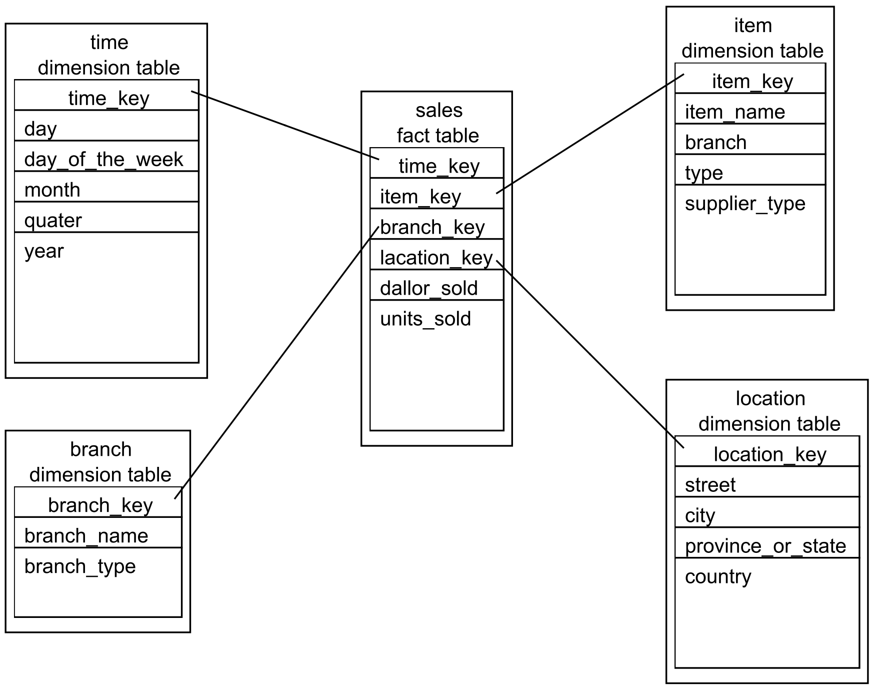

Figure 4.2: Star schema of a data warehouse for sales

is mainly used for analyzing business data collected from daily transactions, such as sales data and health care data. The main purpose of an OLAP system is to enable analysts to construct a mental image about the underlying data by exploring it from different perspectives, at different level of generalizations, and in an interactive manner. Popular architectures of OLAP systems include ROLAP (relational OLAP) and MOLAP (multidimensional OLAP). ROLAP provides a front-end tool that translates multidimensional queries into corresponding SQL queries to be processed by the relational backend. MOLAP does not rely on the relational model but instead materializes the multidimensional views. Using MOLAP for dense parts of the data and ROLAP for the others leads to a hybrid architecture, namely, the HOLAP or hybrid OLAP.

As a component of decision support systems, OLAP interacts with other components, such as data mining, to assist analysts in making business decisions. While data mining algorithms automatically produce knowledge in a pre-defined form, such as association rule or classification. OLAP does not directly generate such knowledge, but instead relies on human analysts to observe it by interpreting the query results. On the other hand, OLAP is more flexible than data mining in the sense that analysts may obtain all kinds of patterns and trends rather than only knowledge of fixed forms. OLAP and data mining can also be combined to enable analysts in obtaining data mining results from different portion of the data and at different level of generalization. The requirements on OLAP systems have been defined differently, such as the FASMI (Fast Analysis of Shared Multidimensional Information) test and the Codd rules. First, to make OLAP analysis an interactive process, the OLAP system must be highly efficient in answering queries. OLAP systems usually rely on extensive pre-computations, indexing, and specialized storage to improve the performance. Second, to allow analysts to explore the data from different perspectives and at different level of generalization, OLAP organizes and generalizes data along multiple dimensions and dimension hierarchies. The data cube model we shall address shortly is one of the most popular abstract models for this purpose. Data cube was proposed as a SQL operator to support common OLAP tasks like histograms and sub-totals. Even though such tasks are usually possible with standard SQL queries, the queries may become very complex. The number of needed unions is exponential in the number of dimensions of the base table. Such a complex query may result in many scans of the base table, leading to poor performance. Because sub-totals are very common in OLAP queries, it is desired to define a new operator for the collection of such sub-totals, namely, data cube. Figure 4.3 displays the lattice of cuboids, making up a 4-D data cube for the dimensions time, item, location, and supplier. Each cuboid represents a different degree of summarization. The apex cuboid (0-D), typically denoted by all, summarized over all four dimensions. The 4-D cuboid is the base cuboid.

- Outsourced database.

  Recently, there has been growing interest in outsourcing database services in both the commercial world and the research community. Although the outsourced database service model is emerging as an efficient replacement solution for traditional in-house database management systems, its clients, however, have to store their private data at

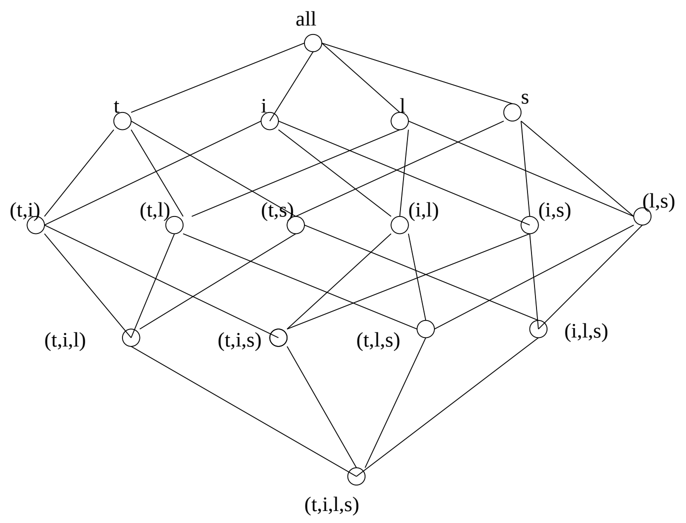

Figure 4.3: Lattice of cuboids, making up a 4-D datacube

an external service provider, who is typically not fully trusted, and so it introduces numerous security research challenges.

Cloud computing: A model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storages, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model promotes availability and is composed of five essential characteristics, three service models and four deployment models.

The essential characteristics are: broad network access, rapid elasticity, measured service, on-demand self-service, and resource pooling.

The three service models are: software as a service (SaaS), Platform as a service (PssS), and infrastructure as a service (IaaS).

The deployment models are: public cloud, private cloud, community cloud, and hybrid cloud.

- Geospatial database

When the location of facilities is established, spatial coordinates such as latitude and longitude are generated from global positioning systems

(GPS) or from Geographic Information Systems (GIS). The information including latitude and longitude is aggregated into a set of database records, the result is the creation of a Geospatial Database. Geospatial databases are very useful when used to support single or multiple well permitting projects. What is really powerful about using geospatial database files is that they are easily shared amongst construction service companies, the operators and with other agencies without the need to send hardcopy or scanned image maps. Using just geospatial database files, any or all of the information in the file can be easily plotted to create a variety of maps as determined by the database record values.

The vector and raster models are two principal spatial data organization scheme.

- The vector model uses the geometry shapes such as points, lines, polygons, etc. These spatial attributes record data about the location, topology and geometry of geospatial data. This model also consists some thematic attributes such as annual rainfall, vegetation type, census tracts, etc.
- Under the raster data model, the spatial data, such as satellite images, elevation maps, or digitized maps, is represented as a grid of columns and rows, i.e. as a matrix of cells (called pixels). Each layer of grid cells records a separate attribute. Each cell carries the non-spatial data too. Spatial coordinates are not usually explicitly stored for each cell, but implicitly represented with the ordering of the pixels. Typically, each layer contains information about the number of rows and columns and the geographic location of the origin.

• Hippocratic Databases

Hippocratic databases (HDBs) are a class of database systems that accept responsibility for the privacy and security of information they manage without impeding legitimate use and disclosure. HDBs ensure that only authorized individuals have access to sensitive information and that any disclosure of this information is for proper purposes. They empower individuals to consent to specific uses and disclosures of their information and to verify the enterprise's compliance with its privacy policies. HDBs also employ technical safeguards to ensure the security of the information they manage. Further, they use advanced information sharing and analytics to enable enterprises to gain maximum value from information without compromising security or individual privacy.

Active enforcement technology supports the principles of purpose specification, consent, limited use and limited disclosure for relational database and XML-based systems. Compliance auditing, query ranking, and duration auditing technologies support the compliance principle. Sovereign Information Integration enables limited use and limited disclosure in distributed environments with no trusted third parties, while order-preserving encryption and the Partition Plaintext Ciphertext storage model promote the safety principle. The growth of electronic medical and financial records accompanied by many highly publicized privacy breaches in recent years underscore the importance of continuing research in HDB technology.

4.2 Database access control

The DBMS operates on the assumption that the computer system has authenticated each user. As an additional line of defence, the computer system may use the overall access control system to determine whether a user may access to the database as a whole. For users that are granted access to the database, a database access control system provides a specific capability that controls access to portions of the database.

Commercial DBMSs provide discretionary or role-based access control. A DBMS can support a range of administrative policies, such as

- Centralized administration: a small number of privileged users may grant and revoke access rights.
- Ownership-based administration: The creator of a table may grant and revoke access right to the table.
- Decentralized administration: The owner of the table may grant and revoke access rights to other uses.

A database access control system usually distinguishes different access rights, such as create, insert, delete, update, read and write. Access rights can be to the entire database, to individual tables, of to selected rows or columns within a table.

There is a problem for multiple agents accessible data base. For example, if two users need to update the same record, then the operation might be conflict. Even if the updates are not conflict, a computer failure may cause the database in an inconsistent state. Most database uses two-phase commit to solve this problem.

1. Request phase: all the parts that involve this update are identified and flagged as being intended for this change. The result of this phase is either successful or abort. The abort may caused by not able to flag some parts or system failure.

2. Commit phase: database locks itself into other changes and perform the changes identified in the request phase. If the changes complete successfully, then all the flags are cleared and the database is unlocked. If some operation fails, then the database reverses all the changes back to the state the database was in just after completing the first phase.

4.2.1 SQL-based access definition

Two commands are used for managing access rights, GRANT and REVOKE. Examples of syntax are as follows:

-
  ```
  GRANT           {privileges | role}
  [ON             table]
  TO              {user | role | PUBLIC}
  [IDENTIFIED BY  password]
  [WITH           GRANT OPTION]
  ```

  This command can be used to grant one or more access rights or can be used to assign a user to a role. For access rights, the command can optionally specify that it applies only to a specified table. The TO clause specifies the user or role to which the rights are granted. A PUBLIC value indicates that any user has the specified access tights. The optional IDENTIFIED BY clause specifies a password that must be used to revoke the access rights of this GRANT command. The GRANT OPTION indicates that the grantee can grant this access right to other users, with or without the grant option.

Typically, SQL provides different ranges of access rights, such as: Select, Insert, Update, Delete, References, etc.

The statement: `GRANT SELECT ON ANY TABLE TO rwei`

enables user `rwei` to query any table in the database.

-
  ```
  REVOKE         {privileges | role}
  [ON            table]
  FROM           {user | role | PUBLIC}
  ```
  The statement: `REVOKE SELECT ON ANY TABLE FROM rwei`

  will revoke the access rights of the preceding example.

- The grant option enables an access tight to cascade through a number of users. When user A revokes an access tight, any cascaded access tight is also revoked, unless that access right would exist even if the original grant from A had never occurred.

  Figure 4.4 is an example, where *t* is the time. To implement that rule,

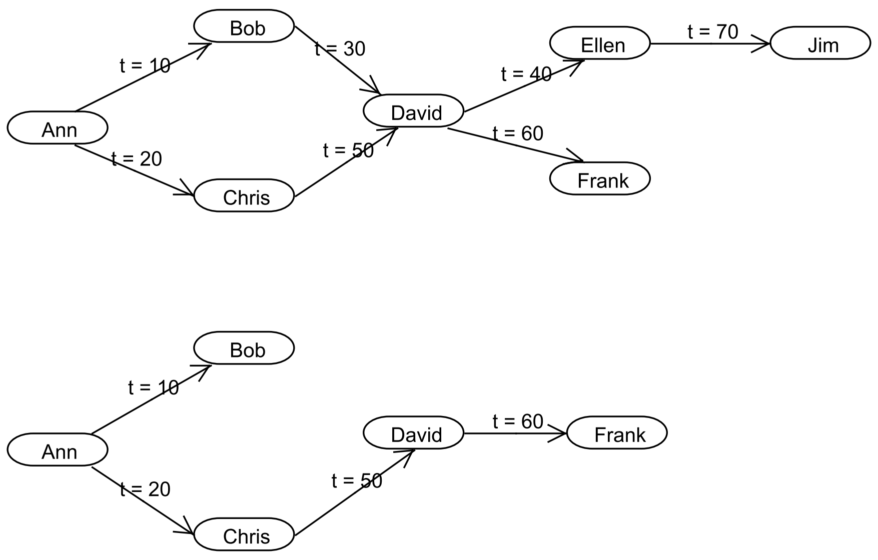

Figure 4.4: Bob revokes privilege from David

a time stamp for each privilege granting action is maintained. For example, for the example in Figure 4.4 the database will keep a table `grants` as follows:

Grantor	Grantee	Privilege	Timestamp
Ann	Bob	P	10
Ann	Chris	P	20
Bob	David	P	30
David	Ellen	P	40
Chris	David	P	50
David	Frank	P	60
Ellen	Jim	P	70

The pseudo code for the algorithm of revocation is expressed below:

```
REVOKE(record $X$)
  let $X = (A, B, P, t)$
  delete record $X$ from grants
  $t^* \leftarrow t$
  for each record $R$ such that $R.grantee = B$ and $R.privilege = P$
    if $R.timestamp < t^*$ then $t^* \leftarrow R.timestamp$
     // $t^*$ is the earliest time stamp of a grant of $P$ to $B$.
  for each record $R$ such that $R.grantor = B$ and $R.privilege = P$
    if $R.timestamp < t^*$ then REVOKE($R$)
```

4.2.2 SQL injection attacks

Many web sites use databases which may in the same machine of the web server or in a separated, dedicated server. Usually, unknown users are not allowed to access the database directly. Most web-based database interaction is carried out on the server side, invisible to the user. In this way, the interactions between users and the database can be carefully controlled. The goal of an attacker is to breach this controlled database interaction to get direct access to a database.

A typical SQL injection lets an attacker to access, or even modify, arbitrary information from a database by inserting his own SQL commands in a data stream that is passed to the database by a web server.

For example, the following is a sample of PHP script:

```
<?php
 //Create SQL query
 $qeury = 'SELECT * FROM news WHERE id = '.$_GET['ID'];
 //Execute sql query
 $out = mysql_query($qeury) or die('Query failed: '. mysql_error());
 //Display query results
 echo "<table border=1>\n"
 //Generate header row
 echo "<tr>
       <th>id</th><th>title</th><th>author</th><th>body</th>
       </tr>";
While ($row = mysql_fetch_array($out)) {
 //Generate row
 echo " <tr>\n"
 echo "  <td>:" $row['id']."/td>\n";
 echo "  <td>:" $row['title']."/td>\n";
 echo "  <td>:" $row['author']."/td>\n";
 echo "  <td>:" $row['body']."/td>\n";
 echo " </tr>\n";
}
 echo "</table>\n";
?>
```

The above code builds an SQL query that retrieves from table `news` the record with `id` given by a GET variable. The query is stored at `$query`. Then the script executes the SQL query and stores the results in `$out`. Finally, a web page is formed to display the results. If the `id` number is 3, then the following URL will be generated:

`http://www.example.com/news.php?id=3`

The problem of the above code is that the server does not check if the GET variable is a valid input.

Suppose the database contains another table `users` which stores account information with attributes `first, last, email, cardno`, where `cardno` is the credit card number. The attacker could request the URL:

```
http://www.example.com/news.php?id=NULL UNION SELECT
cardno,first,last,email FROM users
```

In SQL, `UNION` command joins the results of two queries into a single result, the injected query might show Table 4.4

id	title	author	body
1234-2222-1111-3333	Alice	All	alice@exapmle.com
1234-2323-1643-4444	Bob	Brown	bob@example.com

Table 4.4: SQL injection result

In table 4.4, the names of attributes are from table `news`. But the contents are actually card number, first name, last name, email address from the table `users` (the `news` table and `users` table have the same number of attributes). By using the `UNION` operator, the attacker can inject an SQL query that reads off the entire table, and cause information disclosure.

In another example, the attacker tried to bypass authentication. the code of the web site is as follows.

```
<?php
$query = 'SELECT * FROM users WHERE email = "'.$_POST['email'].
'"'. ' AND pwdhash = "' . hash('sha256'<$__POST['password']). '"';
$out = mysql_query($query) of die('Query failed:' . mysql_error());
if (mysql_num_rows($out) > 0) {
  $access = true;
  echo "<p>Login successful!</p>";
 }
else {
  $access = false;
  echo "<p>Login failed.</p>";
}
?>
```

The server uses POST variables `email` and `password`, which will be specified in the login page. If the number of rows returned by this query is greater than zero (i.e., there is an entry in the `user` table that matches the entered user name and password), access is granted.

Now suppose the attacker input the following information in the login page:

Email:	`"OR 1=1;--`
Password:	`(empty)`

Then the following SQL query will generated:

```
SELECT * FROM users WHERE email= "" OR 1=1;--" AND pwhash="..."
```

An SQL query statement is terminated by a semicolon. The `--` characters denote a comment in MySQL, which results in the rest of the line being ignored. Since the statement `1=1` is true, the query returns the entire `users` table. So the attacker will successfully login.

The above two examples are about an attacker to grain accessing to a database. There are more serious attacks. Some SQL attacks allow for inserting new records, modifying existing records, deleting records, or even deleting entire tables. In addition, some databases have build-in features that allow execution of operating system commands via the SQL interface, enabling an attacker to remotely control the database server.

There are other kind SQL injection attacks:

- Blind SQL injection attack: the attacker uses multiple injected queries and exams how they affect error messages and the contents of a page. It may be possible to deduce some contents of the database without seeing any query results.
- Cross-site scripting: the attacker injects Javascript cookie-stealing code into the database, and when a user visits a page that retrieves the now malicious data, the malicious code will be executed on the user's browser.
- SQL injection worm: the worm propagates automatically by using the resources of a compromised server to scan the Internet for other sites vulnerable to SQL injection. After finding targets, the worms will exploit the found vulnerabilities, install themselves on the compromised database servers and repeat the process.

Basically, the SQL injection vulnerabilities are the result of programming failing to check user input before using that input to construct database queries. This can be avoided if the programmer can be more careful. Most language have built-in functions that strip input of dangerous characters. For example, `mysql_real_escape_string` is provided by PHP.

4.2.3 Multilevel security

Multilevel security is of interest when there is a requirement to maintain a resource, such as a file system or database in which multiple levels of data sensitivity are defined.

The addition of multilevel security to a database system increases the complexity of the access control function and of the design of the database itself. The following are possible methods of imposing multilevel security on a relational database, in terms of the granularity of classification.

- Entire database
- Individual tables (relations)
- Individual columns (attributes)
- Individual rows (tuples)
- Individual elements

The granularity of the classification scheme affects the way in which access control is enforced. For example, suppose the following SQL query is issued:

```
SELECT Ename
 FROM employee
 WHERE Salary > 50K
```

If the classification granularity is the entire database or at the table level, then it is straightforward. However, if it is at the column level, then it will not reject only if both the column of salary and column of name are accessible.

When a user with a low clearance (unrestricted) requests the insertion of a row with the same primary key as an existing row where the row or one of its elements is at a higher level. The DBMS has essential three choices:

- Notify the user that a row with the same primary key already exists and reject the insertion. This will cause some inference problem.
- Replace the existing row with the new row classified at the lower level. This will result overwrite data not visible to the user.
- Insert the new row at the lower level without modifying the existing row at the higher level. This is know as polyinstantiation, which will creates a database with conflicting entries.

4.2.4 Classical access control models

The basic elements of access control are: subject, object, and access right. Usually access control systems define three subjects: owner, group and world (users not included in the categories owner and group). An object is a resource to contain or receive information (records, blocks, pages, segments, files, . . . etc.). Access right can be read, write, execute, delete, create, search, etc.

- Discretionary access control

  While convenient for their expressiveness and flexibility, in high security settings discretionary access control results limited for its vulnerability to Trojan horses. The reason for this vulnerability is that discretionary access control does not distinguish between users (i.e., human entity whose identity is exploited to select the privileges for making the access control decision) and subjects (i.e., process generated by a user and that makes requests to the system). A discretionary access control system evaluates the requests made by a subject against the authorizations of the user who generated the corresponding process. It is then vulnerable from processes executing malicious programs that exploit the authorizations of the user invoking them. Protection against these processes requires controlling the flows of information within processes execution and possibly restricting them. Mandatory policies provide a way to enforce information flow control through the use of labels.

- Mandatory Access Control

  Mandatory security policies enforce access control on the basis of regulations mandated by a central authority. The most common form of mandatory policy is the multilevel security policy, based on the classifications of subjects and objects in the system. Each subject and object in the system is associated with an access class, usually composed of a security level and a set of categories. Security levels in the system are characterized by a total order relation, while categories form an unordered set. As a consequence, the set of access classes is characterized by a partial order relation, denoted $\geq$ and called dominance. Given two access classes c_1 and c_2, c_1 dominates c_2, denoted $c_1 \geq c_2$, if and only if the security level of c_1 is greater than or equal to the security level of c_2 and the set of categories of c_1 includes the set of categories of c_2.

Access classes together with their partial order dominance relationship form a lattice. Mandatory policies can be classified as secrecy-based and integrity-based, operating in a dual manner.

The main goal of secrecy based mandatory policies is to protect data confidentiality. As a consequence, the security level of the access class associated with an object reflects the sensitivity of its content, while the security level of the access class associated with a subject, called clearance, reflects the degree of trust placed in the subject not to reveal sensitive information. The set of categories associated with both subjects and objects defines the area of competence of users and data. The main goal of integrity-based mandatory policies is to prevent subjects from indirectly modifying information they cannot write. The integrity level associated with a user reflects then the degree of trust placed in the subject to insert and modify sensitive information. The integrity level associated with an object indicates the degree of trust placed on the information stored in the object and the potential damage that could result from unauthorized modifications of the information.

A major drawback of mandatory policies is that they control only flows of information happening through overt channels, that is, channels operating in a legitimate way. As a consequence, the mandatory policies are vulnerable to covert channels, which are channels not intended for normal communication but that still can be exploited to infer information. For instance, if a low level subject requests the use of a resource currently used by a high level subject, it will receive a negative response, thus inferring that another (higher level) subject is using the same resource.

- Role-based access control

  A RBAC scheme for database will simplify the administrative work and improve the security. However, not all the database softwares provide RBAC mechanism. For example, current MySQL dose not accommodate that. Microsoft SQL Server supports three types of roles: server roles, database roles and user-defined roles.

  It is important to note that roles and groups of users are two different concepts. A group is a named collection of users and possibly other groups, and a role is a named collection of privileges, and possibly other roles. Furthermore, while roles can be activated and deactivated directly by users at their discretion, the membership in a group cannot be deactivated. The main advantage of RBAC, with respect to DAC and MAC, is that it better suits to commercial environments. In fact, in a company, it is not important the identity of a person for her access to the system, but her responsibilities. Also, the role-based policy tries to organize privileges mapping the organization's structure on the roles hierarchy used for access control.

4.2.5 Credential-based access control

A client and a server, interacting through a predefined negotiation process. The server is characterized by a set of resources. Both the client and the server have a portfolio, which is a collection of credentials (i.e., statements issued by authorities trusted for making them) and declarations (statements issued by the party itself). Credentials correspond to digital certificates and are guaranteed to be unforgeable and verifiable through the public key of the issuing authority. To the aim of performing gradual trust establishment between the two interacting parties, the server defines a set of service accessibility rules, and both the client and the server define their own set of portfolio disclosure rules.

The service accessibility rules specify the necessary and sufficient conditions for accessing a resource, while portfolio disclosure rules define the conditions that govern the release of credentials and declarations. Both the two classes of rules are expressed by using a logic language. A special class of predicates is represented by abbreviations. Since there may exist a number of alternative combinations of certificates allowing access to a resource, abbreviation predicates may be used for reducing the communication cost of such certificates. The predicates of the language adopted exploit the current state (i.e., parties characteristics, certificates already exchanged in the negotiation, and requests made by the parties) to take a decision about a release. The information about the state is classified as persistent state, when the information is stored at the site and spans different negotiations, and negotiation state, when it is acquired during the negotiation and is deleted when the same terminates.

A number of trust negotiation strategies have been proposed in the literature, which are characterized by the following steps.

- The client first requests to access a resource.

- The server then checks if the client provided the necessary credentials. In case of a positive answer, the server grants access to the resource; otherwise it communicates the client the policies that she has to fulfill.
- The client selects the requested credentials, if possible, and sends them to the server.
- If the credentials satisfy the request, the client is granted access to the resource.

The main advantage of this proposal is that it maximizes both server and client's privacy, by minimizing the set of certificates exchanged. In particular, the server discloses the minimal set of policies for granting access, while the client releases the minimal set of certificates to access the resource. To this purpose, service accessibility rules are distinguished in prerequisites and requisites. Prerequisites are conditions that must be satisfied for a service request to be taken into consideration (they do not guarantee that it will be granted); requisites are conditions that allow the service request to be successfully granted. Therefore, the server will not disclose a requisite rule until the client satisfies a prerequisite rule.

4.2.6 Policy Composition

In many real word scenarios, access control enforcement needs to take into consideration different policies independently stated by different administrative subjects, which must be enforced as if they were a single policy. As an example of policy composition, consider an hospital, where the global policy may be obtained by combining together the policies of its different wards and the externally imposed constraints (e.g., privacy regulations). Policy composition is becoming of paramount importance in all those contexts in which administrative tasks are managed by different, non collaborating, entities.

Policy composition is an orthogonal aspect with respect to policy models, mechanisms, and languages. As a matter of fact, the entities expressing the policies to be composed may even not be aware of the access control system adopted by the other entities specifying access control rules. The main desiderata for a policy composition framework can be summarized as follows.

- Heterogeneous policy support. The framework should support policies expressed in different languages and enforced by different mechanisms.

- Support of unknown policies. The framework should support policies that are not fully defined or are not fully known when the composition strategy is defined. Consequently, policies are to be treated as black-boxes and are supposed to return a correct and complete response when queried at access control time.
- Controlled interference. The framework cannot simply merge the sets of rules defined by the different administrative entities, since this behavior may cause side effects. For instance, the accesses granted/denied might not correctly reflect the specifications anymore.
- Expressiveness. The framework should support a number of different ways for combining the input policies, without changing the input set of rules or introducing ad-hoc extensions to authorizations.
- Support of different abstraction levels. The composition should highlight the different components and their interplay at different levels of abstraction.
- Formal semantics. The language for policy composition adopted by the framework should be declarative, implementation independent, and based on a formal semantic to avoid ambiguity.

Different solutions have been proposed for combining different policies. Various models have been proposed to reason about security policies.

- Focus on the secure behavior of program modules.
- Algebra of security, which is a Boolean algebra that enables to reason about the problem of policy conflict, arising when different policies are combined. However, even though this approach permits to detect conflicts between policies, it does not propose a method to resolve the conflicts and to construct a security policy from inconsistent subpolicies.
- Meta-policies, which are defined as policies about policies. Metapolicies are used to coordinate the interaction about policies and to explicitly define assumptions about them. Some researcher formalizes the combination of two policies with a function, called policy combiner, and introduces the notion of policy attenuation to allow the composition of conflicting security policies.

- Other approaches are targeted to the development of a uniform framework to express possibly heterogeneous policies.

A different approach has been illustrated in 2002, where the researchers propose an algebra for combining security policies together with its formal semantics. Here, a policy, denoted P_i, is defined as a set of triples of the form (s, o, a), where s is a constant in (or a variable over) the set of subjects S, o is a constant in (or a variable over) the set of objects O, and a is a constant in (or a variable over) the set of actions A. Policies of this form are composed through a set of algebra operators whose syntax is represented by the following BNF(Backus-Naur Form):

$$\begin{aligned} E &::= \mathbf{id}|E+E|E\&E|E-E|E\wedge C|o(E,E,E)|E*R|T(E)|(E) \\ T &::= \tau\mathbf{id}.E \end{aligned}$$

where **id** is a unique policy identifier, E is a policy expression, T is a construct, called template, C is a construct describing constraints, and R is a construct describing rules. The order of evaluation of algebra operators is determined by the precedence, which is (from higher to lower) $\tau, ., +$ and $\&$ and $-, *$ and $\wedge$.

In BNF "::=" means that the symbol on the left must be replaced with the expression on the right and "|" means a choice.

The semantic of algebra operators is defined by a function that maps policy expressions in a set of ground authorizations (i.e., a set of authorization triples). The function that maps policy identifiers into sets of triples is called environment, and is formally defined as follows.

Definition An environment e is a partial mapping from policy identifiers to sets of authorization triples. By $e[X/S]$ we denote a modification of environment e such that

$$e[X/S](Y) = \begin{cases} S & \text{if } Y = X \\ e(Y) & \text{otherwise} \end{cases}$$

The semantic of an identifier X in the environment e is denoted by $[[X]]e = e(X)$. The operators defined by the algebra for policy composition basically reflect the features supported by classical policy definition systems. As an example, it is possible to manage exceptions (such as negative authorizations), propagation of authorizations, an so on. The set of operators together with their semantic is briefly described in the following.

- Addition (+). It merges two policies by returning their union.

  $$[[P_1 + P_2]]_e = [[P_1]]_e \cup [[P_2]]_e$$

  Intuitively, additions can be applied in any situation where accesses can be authorized if allowed by any of the component policies (maximum privilege principle).

- Conjunction (&). It merges two policies by returning their intersection.

  $$[[P_1 \& P_2]]_e = [[P_1]]_e \cap [[P_2]]_e$$

  This operator enforces the minimum privilege principle.

- Subtraction (−). It deletes from a first policy, all the authorizations specified in a second policy.

  $$[[P_1 - P_2]]_e = [[P_1]]_e \setminus [[P_2]]_e$$

  Intuitively, subtraction operator is used to handle exceptions, and has the same functionalities of negative authorizations in existing approaches. It does not generate conflicts since P_1 prevails on P_2 by default.

- Closure (∗). It closes a policy under a set of derivation rules.

  $$[[P * R]]_e = closure(R, [[P]]_e)$$

  The *closure* of policy P under derivation rules R produces a new policy that contains all the authorizations in P and those that can be derived evaluating R on P, according to a given semantics. The derivation rules in R can enforce, for example, an authorization propagation along a predefined subject or object hierarchy.

- Scoping Restriction (∧). It restricts the applicability of a policy to a given subset of subjects, objects, and actions of the system.

  $$[[P \wedge c]]_e = \{t \in [[P]]_e | \ t \text{ satisfy } c\}$$

  where c is a condition. It is useful when administration entities need to express their policy on a confined subset of subjects and/or objects (e.g., each ward can express policies about the doctors working in the ward).

- Overriding (o). It overrides a portion of policy P_1 with the specifications in policy P_2; the fragment that is to be substituted is specified by a third policy P_3.

$$[[o(P_1, P_2, P_3)]]_e = [[(P_1 - P_3) + (P_2 \& P_3)]]_e$$

- Template (τ). It defines a partially specified (i.e., parametric) policy that can be completed by supplying the parameters.

$$[[\tau X.P]]_e(S) = [[P]]_{e[S/X]}$$

  where S is the set of all policies, and X is a parameter. Templates are useful for representing policies as black-boxes. They are needed any time when some components are to be specified at a later stage. For instance, the components might be the result of a further policy refinement, or might be specified by a different authority.

Due to the formal definition of the semantic of algebra operators, it is possible to exploit algebra expressions to formally prove the security properties of the obtained (composed) policy. Once the policies have been composed through the algebraic operators described above, for their enforcement it is necessary to provide executable specifications compatible with different evaluation strategies. To this aim, the authors propose the following three main strategies to translate policy expressions into logic programs.

- Materialization. The expressions composing policies are explicitly evaluated, by obtaining a set of ground authorizations that represents the policy that needs to be enforced. This strategy can be applied when all the composed policies are known and reasonably static.

- Partial materialization. Whenever materialization is not possible since some of the policies to be composed are not available, it is possible to materialize only a subset of the final policy. This strategy is useful also when some of the policies are subject to sudden and frequent changes, and the cost of materialization may be too high with respect to the advantages it may provide.

- Run-time evaluation. In this case no materialization is performed and runtime evaluation is needed for each request (access triple), which is checked against the policy expressions to determine whether the triple belongs to the result.

The authors then propose a method (*pe2lp*) for transforming algebraic policy composition expressions into a logic program. The method proposed can be easily adapted to one of the three materialization strategies introduced above. Basically, the translation process creates a distinct predicate symbol for each policy identifier and for each algebraic operator in the expression. The logic programming formulation of algebra expressions can be used to enforce access control. As already pointed out while introducing algebra operators, this policy composition algebra can also be used to express simple access control policies, such as open and closed policy, propagation policies, and exceptions management. For instance, let us consider a hospital composed of three wards, namely Cardiology, Surgery, and Orthopaedics. Each ward is responsible for granting access to data under its responsibility. Let $P_{Cardiology}, P_{Surgery}$ and $P_{Orthopaedics}$ be the policies of the three wards. Suppose now that an access is authorized if any of the wards policies state so and that authorizations in policy $P_{Surgery}$ are propagated to individual users and documents by classical hierarchy-based derivation rules, denoted R_H. In terms of the algebra, the hospital policy can be represented as follows.

$$P_{Cardiology} \& P_{Surgery} * R_H \& P_{Orthopaedics}.$$

4.3 Inference

Inference is the process of performing authorized queries and deducing unauthorized information from the legitimate responses received. Inference problem arises when the combination of a number of data item is more sensitive than the individual items, or when a combination of data item can be used to infer data of a higher sensitivity. The attacker may use nonsensitive data as well as metadata to infer sensitive information. Metadata refers to knowledge about correlations of dependencies among data items that can be used to deduce information not otherwise available to a particular user. The information transfer path by which unauthorized data is obtained is referred to as an inference channel.

Two inference techniques can be used to derive additional information: analyzing functional dependencies between attributes within a table, or across tables, and merging views with the same constraints.

Suppose general user are not authorized to access the relationship between Name and Salary in the Employee table (Figure 4.5). However, users can perform the following two queries:

Name	Position	Salary($)	Department	Dept.Manager
Andy	senior	43,000	strip	Cathy
Calvin	junior	35,000	strip	Cathy
Cathy	senior	48,000	strip	Cathy
Dennis	junior	38,000	panel	Herman
...		...		...

Table 4.5: Employee table

```
CREATE view V1 AS
SELECT positon, Salary
FROM Employee
WHERE Department = "strip"

CREATE view V2 AS
SELECT Name, Department
FROM Employee
WHERE Department = "strip"
```

Position	Salary($)
senior	43,000
junior	35,000
senior	48,000

Name	Department
Andy	strip
Calvin	strip
Cathy	strip

Table 4.6: Two views

The result views are displayed in Table 4.6. If a user who knows the structure of the Employee table and who knows that the view tables maintain the same order as the Employee table is then able to merge the two views to construct a table show in Figure 4.7. This violates the access control policy that the relationship of attributes Name and Salary must not be disclosed.

In general, there are two approaches to dealing with the threat of disclosure by inference:

- Inference detection during database design. This approach removes inference channels by altering the database structure (such as split a table) or by changing the access control regime. Techniques in this

Name	Position	Salary($)	Department
Andy	senior	43,000	strip
Calvin	junior	35,000	strip
Cathy	senior	48,000	strip

Table 4.7: Combination of two views

category often result in unnecessary stricter access controls that reduce availability.

- Inference detection at query time. This approaches seeks to eliminate an inference channel violation during a query or series of queries. If an inference channel is detected, the query is denied or altered. This method usually needs to record user query history.

In both approaches, the important thing is that some detection algorithms for inference channels are needed. This is a difficult problem. The inference problem is even more difficult if we consider multiple databases with metadata.

Figure 4.5 shows medical records from a fictitious hospital. Although the name of patients are omitted in the table, people can still inference some information about the sensitive value of the disease condition. For example, if we know someone at the hospital who is less than 30 years old, then we know that his disease is not cancer.

This example shows that even we hid some identities from the database, it is still possible leaking some sensitive data. Some data anonymization techniques are used to prevent some inference problems. We will discuss them later.

4.3.1 Statistical database

A statistical database (SDB) is one that provides data of a statistical nature, such as counts and averages.

- Pure statistical database: only stores statistical data, such as a census database.

- Ordinary database with statistical access: contains individual entries. The database supports a population of nonstatistical users who are

ID	QI			SV
	Zip Code	Age	Nationality	Condition
1	13053	28	Russian	Heart Disease
2	13068	29	American	Heart Disease
3	13068	21	Japanese	Viral Infection
4	13053	23	American	Viral Infection
5	14853	50	Indian	Cancer
6	14853	55	Russian	Heart Disease
7	14850	47	American	Viral Infection
8	14850	49	American	Viral Infection
9	13053	31	American	Cancer
10	13053	37	Indian	Cancer
11	13068	36	Japanese	Cancer
12	13068	35	American	Cancer

Figure 4.5: Inpatient microdata

allowed access to selected portions of the database. In addition, the database supports a set of statistical users who are only permitted statistical queries. For the latter users, aggregate statistics based on the underlying raw data are generated in response to a user query, or may be pre-calculated and stored as part of the database.

The access control objective for an SDB system is to provide users with the aggregate information without compromising the confidentiality of any individual entity represented in the database. The database administrator must prevent or at least detect, the statistical user who attempts to gain individual information through one or a series of statistical queries.

An example of Table 4.8 is a database containing 13 confidential records of students in a university that has 50 departments.

Statistics are derived from a database by means of a characteristic formula, C, which is a logical formula over the values of attributes. A characteristic formula users the operations OR, AND, and NOT $(+, \cdot, \sim)$, written here in order of increasing priority. A characteristic formula specifies a subset of the records in the database. For example, the formula

$$(Sex = Male) \cdot ((Major = CS) + (Major = EE))$$

(a) Database with statistical access with $N = 13$ students

Name	Sex	Major	Class	SAT	GP
Allen	Female	CS	1980	600	3.4
Baker	Female	EE	1980	520	2.5
Cook	Male	EE	1978	630	3.5
Davis	Female	CS	1978	800	4.0
Evans	Male	Bio	1979	500	2.2
Frank	Male	EE	1981	580	3.0
Good	Male	CS	1978	700	3.8
Hall	Female	Psy	1979	580	2.8
Iles	Male	CS	1981	600	3.2
Jones	Female	Bio	1979	750	3.8
Kline	Female	Psy	1981	500	2.5
Lane	Male	EE	1978	600	3.0
Moore	Male	CS	1979	650	3.5

(b) Attribute values and counts

AttributeA_j	Possible values	$\lvert A_j \rvert$
Sex	Male, Female	2
Major	Bio, CS, EE, Psy,...	50
Class	1978, 1979, 1980, 1981	4
SAT	$310, 320, \ldots 790, 800$	50
GP	$0.0, 0.1, 0.2, \ldots, 3.9, 4.0$	41

Table 4.8: Database with statistical access

specifies all male students majoring in either CS or EE. For numerical attributes, relational operators may be used. For example, $(GP > 3.7)$ specifies all students whose grade point average exceeds 3.7. For simplicity, attribute names are omitted in what follows. Thus, the preceding formula becomes $Male \cdot (CS + EE)$.

The `query set` of characteristic formula C, denoted as $X(C)$, is the set of records matching that characteristic. For example, for $C = Female \cdot CS$, the set $X(C)$ consists of records 1 and 4, the records of Allen and Davis.

A statistical query is a query that produces a value calculated over a query set. Table 4.9 lists some simple statistics that can be derived from a

Name	Formula	Description
count(C)	$\|X(C)\|$	Number of records in the query set
sum(C, A_j)	$\sum_{i \in X(C)} x_{ij}$	Sum of the values of numerical attribute A_j over all the records in $X(C)$
rfreq(C)	$\frac{\text{count}(C)}{N}$	Fraction of all records that are in $X(C)$
avg(C, A_j)	$\frac{\text{sum}(C,A_j)}{\text{count}(C)}$	Mean value of numerical attribute A_j, over all the records in $X(C)$
median(C, A_j)		The $\lceil \|X(C)\|/2 \rceil$ largest value of attribute over all the records in $X(C)$. Note that when the query set size is even, the median is the smaller of the two middle values.
max(C, A_j)	$\text{Max}_{i \in X(C)}(x_{ij})$	Maximum value of numerical attribute A_j over all the records in $X(C)$
min(C, A_j)	$\text{Min}_{i \in X(C)}(x_{ij})$	Minimum value of numerical attribute A_j over all the records in $X(C)$

Table 4.9: Some queries of a statistical database

query set. Examples, count(Female·CS) = 2; sum(Female·CS,SAT) = 1400.

4.3.2 Inference from a statistical database

The inference problem in an SDB is that a user may infer confidential information about individual entities represented in the database. Such an inference is called a compromise.

The compromise is positive if the user deduced the value of an attribute associated with an individual entity and is negative if the user deduces that a particular value of an attribute is not associate with an individual.

For example, the statistic sum$(EE \cdot Female, GP) = 2.5$ compromises the database if the user knows the Baker is the only female EE student (using metadata). Or by the sequence of two queries:
count$(EE \cdot Female) = 1$
sum$(EE \cdot Female, GP) = 2.5$.

Another example: consider a personal database in which the sum of salaries of employees may be queried. Suppose a questioner knows the following information

Salary range for a new systems analyst with BS degree is \$[50K, 60K]

Salary range for a new systems analyst with MS degree is \$[60K, 70K]

Suppose two new systems analysts are added to the payroll and the change in the sum of the salaries is \$130K, Then the questioner knows that both new employees have an MS degree.

In general terms, the inference problem for an SDB can be stated as follows. A characteristic function C defines a subset of records (rows) within the database. A query using C provides statistics. If the subset is small enough, or even a single record, the questioner may be able to infer characteristics of a single individual or a small group.

Two distinct approaches can be used to protect inference attacks for SDB:

1. Query restriction: Rejects a query that can lead to a compromise. The answers provided are accurate.

   Some query restriction techniques:

   - Query size restriction: For a database of size N (number of rows, or records), a query $q(C)$ is permitted only if the number of records that match C satisfies

     $$k \leq |X(C)| \leq N - k,$$

     where k is a fixed integer greater than 1. Note that since $q(C) = q(All) - q(\sim C)$, the upper bound is necessary.

     A questioner may divides information of an individual into parts, such that queries can be made based on the parts without violating the query size restriction. The combination of parts is called a tracker if it can be used to track down characteristics of an individual. For example, if the formula $C \cdot D$ corresponds to zero or one record, so that the query count$(C \cdot D)$ is not permitted. Suppose that the formula C can be decomposed into two parts $C = C_1 \cdot C_2$, such that the query sets for both C_1 and $T = (C_1 \cdot \sim C_2)$ satisfy the query size restriction. Then count$(C) =$ count$(C_1)-$ count(T). And count$(C \cdot D) =$ count$(T + C_1 \cdot D)-$ count(T). In Table 4.8, let $C =$ Male·Bio·1979 and $D =$(SAT $\geq$ 600) Suppose the restriction $k = 3$. We can use $T = (C_1 \cdot \sim C_2) =$ Male· $\sim$(Bio· 1979). Both C_1 and C_2 satisfy the query size restriction. We can obtain count(Male·Bio·1979) $=$ count$(C_1)-$ count$(T) = 7 - 6 = 1$.

And count((Male·Bio·1979)·(SAT $\geq$ 600)) = count($T + C_1 \cdot D$)− count(T) = 6 − 6 = 0.

- Query set overlap control: A query $q(C)$ is permitted only if

$$|X(C) \cap X(D)| \leq r$$

  for all $q(D)$ that have been answered for this user, where r is a fixed integer. This method may limiting the usefulness of the database. It also cannot prevent the cooperation of several users to compromise the database and a user profile has to be kept up to date.

- Partition: The records in the database are partitioned into a number of mutually exclusive groups. The user may only query the statistical properties of each group as a whole (the user may not select a subset of a group). The drawbacks of this method are that the user's ability to extract useful statistics is reduced, and there is a design effort in constructing and maintaining the partitions.

A general problem with query restriction techniques is that the denial of a query may provide some clues that an attacker can deduce underlying information.

2. Perturbation: Provides answers to all queries, but the answers are approximate. The data in the SDB are modified or the system can generate statistics that are modified from that the original database would provide.

   - Data perturbation: Modify the data to prevent inference.
     - Data swapping. Attribute values are exchanged (swapped) between records in sufficient quantity so that nothing can be deduced from the disclosure of individual records. The swapping is done in such a way that the accuracy of at least low-order statistics is preserved. Table 4.10 is a simple example, transforming the database D into database D'. D' has the same statistics as D for statistics derived from one or two attributes. However, count(Female·Bio·4.0) has the value 1 in D but the value 0 in D'.

D			D'		
Sex	Major	GP	Sex	Major	GP
Female	Bio	4.0	Male	Bio	4.0
Female	CS	3.0	Male	CS	3.0
Female	EE	3.0	Male	EE	3.0
Female	Psy	4.0	Male	Psy	4.0
Male	Bio	3.0	Female	Bio	3.0
Male	CS	3.0	Female	CS	3.0
Male	EE	4.0	Female	EE	4.0
Male	Psy	4.0	Female	Psy	4.0

Table 4.10: Data swapping

  - Using the estimated underlying probability distribution of attribute values to modify database. For each confidential or sensitive attribute, determine the probability distribution function that best matches the data and estimate the parameters of the distribution function. Then generate a sample series of data from the estimated density function for each sensitive attribute. Finally, substitute the generated data of the confidential attribute for the original data in the same rank order (smallest value replace the smallest original value).

- Output perturbation: Database is not modified, but the output of a query is modified.
  - Random-sample query: suitable for large database. Suppose a user's query is $q(C)$ that is to return a statistical value. The query set is $X(C)$. Then the system replaces $X(C)$ with a sampled query set, which is a properly selected subset of $X(C)$. The system calculates the requested statistic on the sampled query set and returns the value.
  - Calculate the statistics on the requested query set and then adjusting the answer up or down by a given amount in some systematic or randomizes fashion.

To using perturbation techniques, there is a potential loss of accuracy as well as the potential for a systematic bias in the results. The main challenge in the use of perturbation techniques is to determine the average size of the error to be used. If there is too little error, a user can infer close approximations to protected values. If the error is too great, the resulting statistics may be unusable. For a small database, it is difficult to add sufficient perturbation to hide data without badly distorting the results.

Some researchers reported the following. Assume the size of the database (number of data items or records) is n. If the number of queries from a given source is linear to n, then a substantial amount of noise (at least of order $\sqrt{n}$) must be added to the system in terms of perturbation, to preserve confidentiality. This amount of noise may be sufficient to make the database effective unusable. However, if the number of queries is sublinear (e.g., of order $\sqrt{n}$), then much less noise must be added to the system to maintain privacy. For a large database, limiting queries to a sublinear number may be reasonable.

4.4 Database encryption

Encryption is a popular technique for ensuring confidentiality of sensitive data. While data encryption is able to enhance security greatly, it can impose substantial overhead on the performance of a system in terms of data management. Management of encrypted data needs to address several new issues like choice of the appropriate encryption algorithms, deciding the key management architecture and key distribution protocols, enabling efficient encrypted data storage and retrieval, developing techniques for querying and searching encrypted data, ensuring integrity of data etc. In this Section, we introduce some of these areas using the Database As a Service (DAS) as the prototype application. We especially concentrate on techniques for querying encrypted data and summarize the basic techniques proposed for SQL queries over encrypted relational data.

There are some disadvantages to database encryption:

- Key management: Because a database is typically accessible to a wide range of users and a number of applications, providing secure keys to selected parts of the database to authorized users and applications is complex task.

- Inflexibility: When part or all of the database is encrypted, it becomes more difficult to perform record searching.

Encryption can be applied to the entire database, at the record level, at the attribute level, or at the level of the individual field.

We use outsourced database as an example. There are four entities involved.

- Data owner: An organization that produces data.
- User: The entity that presents queries to the system.
- Client: Front end that transforms user queries into queries on the encrypted data stored on the server.
- Server: An organization that receives the encrypted data from a data owner and makes them available for clients.

For the security reason, the database are encrypted, but the encryption/decryption keys are not available for the server. So the data are secure at the server. The client system has a encryption key. A user at the client can retrieve a record from the database as follows.

1. The user issues an SQL query for fields from one or more records with a specific value of the primary key.
2. The query processor at the client encrypts the primary key, modifies the SQL query accordingly, and transmits the query to the server.
3. The server processes the query using the encrypted value of the primary key and returns the appropriate records.
4. The query processor of the client decrypts the data and returns the results.

The above method is straightforward but lacks flexibility. For example, suppose a Employee table contains a salary attribute and the user wishes to retrieve all records for salaries less than $70K. Then the above method does not work.

To provide more flexible queries, the following approach is taken. Each row (record) in the database is encrypted as a block. To assist in data retrieval, attributes indexes are associated with each table. We use an example to explain. The Table 4.11 is a simple table. Suppose that the employee ID

eid	ename	salary	addr	did
23	Tom	70K	Maple	45
860	Mary	60K	Main	83
320	John	50K	river	50
875	Jerry	55K	Hopewell	92

Table 4.11: Employee Table

(eid) values lie in the range [1, 1000]. We divide these values into five partitions: [1, 200], [201,400], [401,600], [601, 800] and [801,1000], and then assign index values 1, 2, 3, 4 and 5, respectively. For the attribute emame, we assign index 1 to values starting with A or B, index 2 to values starting with C and D, and so on. Similar partitioning schemes can be used for each of the attributes. Table 4.12 shows the result table. The values in the first

E(K,B)	I(eid)	I(ename)	I(salary)	I(addr)	I(did)
1100100011001010...	1	10	3	7	4
1011010100101010...	5	7	2	7	8
1110010100010101...	2	5	1	9	5
0011100000101001...	5	5	2	4	9

Table 4.12: Encrypted Employee Table with Indexes

column represent the encrypted values for each row in the original table. The remaining columns show index values for the corresponding attribute values. The mapping function between attribute values and index values constitute metadata that are stored at the client and data owner locations but not at the server.

This arrangement provides for more efficient data retrieval. Some randomized index can be used to further secure the database. For example, the eid values could be partitioned by mapping [1, 200], [201, 400], [401, 600], [601, 800] and [801, 1000] into 2, 3, 5, 1 and 4, respectively.

Other features may be added to this scheme. The system could use the encrypted value of the primary key attribute values, or a hash value to encrease the efficiency of accessing records by means of the primary key. Different portions of the database could be encrypted with different keys, so that users would only have access to that portion of the database for which they had the decryption key.

Approaches based on new encryption techniques are also discussed: These techniques can support either arithmetic and/or comparison operators directly on encrypted representation. Encryption techniques that support limited computation without decryption have been explored in cryptographic literature in the past. Amongst the first such technique is the privacy homomorphism (PH) developed that supports basic arithmetic operations. While PH can be exploited to compute aggregation queries at the remote server, it does not allow comparison and, as such, cannot be used as basis for designing techniques for relational query processing over encrypted data. Some researchers developed a data transformation technique that preserves the order in the original data. Such a transformation serves as an order-preserving encryption and can therefore support comparison operators. Techniques to implement relational operators such as selection, joins, sorting, grouping can be built on top of the order preserving encryption. The encryption mechanism, however, cannot support aggregation at the server. While new cryptographic approaches are interesting, one of the limitation of such approaches has been that they are safe only under limited situations where the adversary's knowledge is limited to the ciphertext representation of data. These techniques have either been shown to break under more general attacks (e.g., PH is not secure under chosen plaintext attack), or the security analysis under diverse types of attacks has not been performed.

For special database, such as text database, XML database, etc., more specific methods are discussed in literatures.

4.5 Data anonymization

Given a microdata table T, the objective of privacy preserving publication is to release a distorted version T' of T such that T' does not allow an adversary to confidently derive the sensitive data of any individual, and yet, T' can be used to analyze the statistical patterns significant in T. The existing methods of privacy preserving publication is essentially the integration of an anonymization framework and an anonymization principle. Specifically, a framework describes how anonymization is performed, whereas a principle measures whether a sufficient amount of anonymization has been applied.

4.5.1 Linking attacks

As a concrete application example, consider that the publisher is a hospital, and T is given in Table 4.13. Here, T' has three non-sensitive attributes $A_1^q = Age, A_2^q = Sex, A_3^q = Zipcode$, and a sensitive attribute $A^s = Disease$. The column Name specifies the owners of the tuples, e.g., Tuple 1 indicates that Andy, aged 5, lives in a neighborhood with Zipcode 12000, and he contracted gastric-ulcer. Obviously, Name should not be published along with T, since it explicitly reveals the identities of all individuals.

row#	name	age	sex	zipcode	disease
1	Andy	5	m	12000	gastric ulcer
2	Bill	9	m	14000	dyspepsia
3	Ken	6	m	18000	pneumonia
4	Nash	8	m	19000	bronchitis
5	Sarah	28	f	37000	pneumonia
6	Mary	56	f	33000	flu
7	Jame	26	f	36000	Alzeimer
..	...	...	...	..	...

Table 4.13: Microdata for a hospital

Let T' be the resulting table after removing Name from T. At first glance, it appears that we can simply release T', which, by itself, does not contain any hint about who is the owner of each tuple. This naive approach fails, because an adversary may combine T' with certain additional information, to recover the owner of a tuple. For instance, imagine that a neighbor of Sarah knows the age 28 of Sarah, and that Sarah has been hospitalized before, and thus, must have a record in T'. Since this neighbor has all the non-sensitive values of Sarah, s/he easily finds out that the last-but-one tuple in T' belongs to Sarah. In this way, the neighbor has successfully linked Sarah to her sensitive value pneumonia. The above process exemplifies a type of privacy inferences called linking attacks, where an adversary accurately infers the sensitive value of a victim, via the victim's non-sensitive values. Since the non-sensitive attributes may be utilized to pinpoint the tuple owned by a person, they are commonly referred to as the *quasi-identifier (QI)* attributes.

In reality, the QI-values of an individual may be acquired by an adversary through several channels. Knowing the victim is an obvious channel, as the earlier example where the adversary is the neighbor of Sarah. Alternatively, an adversary may also obtain the QI-values from an external database, which can be completely separate from T, and its accessibility to the public cannot be controlled by the publisher of T. For instance, a worker in the government may have access to the voter registration list in Table 4.14, which includes some of the QI-attributes in the microdata of Table 4.13, together with people's names.

name	age	sex	zipcode
Andy	5	m	12000
Bill	9	m	14000
Ken	6	m	18000
Nash	8	m	19000
Sarah	28	f	37000
...	...	...	...

Table 4.14: Voting registration list

4.5.2 k-anonymity

Given a microdata table T, the generalization anonymization framework replaces each QI-value with a less specific form, such that the QI-values of a tuple become indistinguishable from those of some other tuples. Table 4.15 demonstrates a generalized version of the microdata in Table 4.13. For example, the age 5 of Tuple 1 in table has been generalized to an interval [1, 10] in Table 4.15. Semantically, the interval indicates that the original age of Tuple 1 may be any value in the range of [1, 10]. Notice that Tuples 1 and 2 have exactly the same generalized value on every QI attribute, and therefore, constitute a "QI-group". Formally, a QI group is a group resulting from grouping the tuples in a relation by all the QI attributes. Clearly, Table 4.15 involves several QI-groups: 1, 2 (indicated by tuple IDs), 3, 4, and so on. It is worth mentioning that the notion of "QI-group" is also known by several other names, such as "equivalent class" "q-block", etc.

Assume that the publisher releases Table 4.15. Consider the linking attack launched by the neighbor of Sarah who possesses the QI values 28, F, 37000 of Sarah. To guess which tuples may belong to Sarah, the adversary settles on a fuzzy fact: Sarah may have got flu, pneumonia, or Alzeimer's.

row#	age	sex	zipcode	disease
1	[1,10]	m	[10001,15000]	gastric ulcer
2	[1,10]	m	[10001,15000]	dyspepsia
3	[1,10]	m	[15001,20000]	pneumonia
4	[1,10]	m	[15001, 20000]	bronchitis
5	[21,60]	f	[30001,60000	pneumonia
6	[21,60]	f	[30001,60000]	flu
7	[21,60]	f	[30001,60000]	Alzeimer
...	...	...	..	...

Table 4.15: 2-anonymous version of Table 4.13

Formally, a generalized table is k-anonymous if each QI-group contains at least k-tuples.In general, a higher k provides stronger protection because, in general, k-anonymity guarantees that an adversary has at most $1/k$ probability of finding out the actual tuple owned by the victim individual. As a trade-off, however, increasing k also brings down the utility of the generalized table, since more information must be lost in generalization.

There are different generalization methods discussed and different metrics to calculate information loss. However, researchers found that the optimal k-anonymity generalization is NP-hard, even for simple information-loss metrics. This fact forces a practitioner to accept an approximate algorithm. However, the existing solutions cannot ensure a small approximation ratio (which varies according to the information-loss norm). In particular, the best known ratio is $O(k)$, which implies that the output of an algorithm may considerably deviate from the optimal quality, when strong privacy protection is required (e.g., $k = 20$). It is also known that, when the number of QI attributes is large, k-anonymous generalization inevitably lose a huge amount of information, even for $k = 2$.

4.5.3 l-diversity

There are some crucial defects for k-anonymity. For example, Figure 4.6 shows the microdata in Figure 4.5 has been anonymized for $k = 4$.

The sensitive values in last q-block are all "Cancer". That means it is possible for people to use QI values to determine the sensitive attribute. This is called Homogeneity attack. Another kind of attack is called background

ID	QI			SV
	Zip Code	Age	Nationality	Condition
1	130**	2*	*	Heart Disease
2	130**	2*	*	Heart Disease
3	130**	2*	*	Viral Infection
4	130**	2*	*	Viral Infection
5	1485*	> 40	*	Cancer
6	1485*	> 40	*	Heart Disease
7	1485*	> 40	*	Viral Infection
8	1485*	> 40	*	Viral Infection
9	130**	3*	*	Cancer
10	130**	3*	*	Cancer
11	130**	3*	*	Cancer
12	130**	3*	*	Cancer

Figure 4.6: 4-anonymous inpatient microdata

attack. Now let us consider, once again, the neighbor of Sarah. As explained before, from Table 4.15, the neighbor can only figure out that Sarah may have had flu, Alzeimer's, or pneumonia. This conjecture, however, may be further improved, if the neighbor utilizes her/his "background knowledge". For example, s/he may know that a flu-vaccine shot had been offered to all the residents in the neighborhood a month before Sarah was hospitalized. Hence, it is rather unlikely that the hospitalization was caused by flu. Furthermore, obviously, Sarah is too young to get infected with Alzeimer's disease. Thus, the adversary becomes (almost fully) confident that Sarah contracted pneumonia.

l-diversity was exactly motivated by this observation. It requires that, after generalization, every QI-group should contain at least l "well represented" sensitive values. Intuitively, this requirement does not allow an adversary to accurately recover the sensitive value of any individual o, provided that the adversary can exclude up to $l-2$ values (i.e., leaving at least 2 possibilities for o). Thus, with a sufficiently large l, l-diversity can effectively prevent privacy breaches. There are multiple ways to interpret the meaning of "well-represented".

Definition 4.5.1 A QI-group fulfills distinctness l-diversity, if it contains at least l different sensitive values.

Although this interpretation can be easily understood, it does not offer strong privacy guarantees from a probabilistic point of view. For example, imagine a QI-group with 1000 tuples, 900 of which carry the same sensitive value HIV, and the remaining 100 tuples have distinct values different from HIV. Clearly, the QI-group satisfies distinctness 101-diversity. Nevertheless, the privacy of HIV patients is poorly preserved. Specifically, consider an adversary who aims at inferring the disease of such a patient o, and has no background knowledge, i.e., s/he cannot exclude any disease before studying the published table. With a random guess, the adversary concludes that o had HIV with probability $900/1000 = 90\%$. Notice that this process of privacy inference essentially captures homogeneity attacks as a special case; hence, we refer to the process as a probabilistic homogeneity attack. This phenomenon leads to an improved version of l-diversity.

Definition 4.5.2 A table fulfills frequency l-diversity if, in each QI-group, at most $1/l$ of the tuples carry the sensitive value.

Frequency l-diversity does not provide adequate protection to background attacks. To understand this, consider a QI-group with 1000 tuples, 500 of which have the sensitive value HIV, 499 tuples have pneumonia, and the remaining tuple carries flu. This QI-group qualifies frequency 2-diversity, since the most frequent value HIV belongs to 50% of the tuples. Let o be an HIV patient. Now, imagine an adversary who knows that this group contains the record of o, and that o does not have pneumonia. As a result, the record of o must be one of the 500 HIV-tuples, or the flu-tuple. At this point, the adversary cannot exclude any other disease; hence, taking a random guess, s/he conjectures that o contracted HIV with an exceedingly high probability $500/501 > 99.8\%$. The cause of the above problem is as follows: after removing the 2nd frequent sensitive value (i.e., pneumonia) in a QI-group, the most frequent sensitive (HIV) value accounts for an excessively high proportion of the remaining tuples in the group. The implication is that, it is not enough to limit the frequency of the most popular sensitive value with respect to the QI-group size (as is the case in frequency l-diversity). Instead, we should limit the frequency according to the number of remaining tuples, after eliminating those having the 2nd frequent sensitive value. Carrying the reasoning to the general scenario, if an adversary can exclude at most $l - 2$

values, we ought to constrain the frequency of the most sensitive value, with respect to the remaining tuples, after discarding the 2nd, 3rd, ..., $(l-1)$-th most frequent sensitive values. This leads to the next version of l-diversity.

Definition 4.5.3 Given a QI-group, use $n_1, n_2, \ldots, n_m$ to denote the number of tuples having the most, 2nd most, ..., the least frequent sensitive values in the group, respectively. The QI-group obeys recursive (c, l)-diversity, if the next inequality holds:

$$n_1 \le c \cdot (n_l + n_{l+1} + ... + n_m),$$

where c is a certain constant, and l is an integer at most m.

A QI-group obeys (c, l)-diversity, if and only if, after eliminating the tuples with any l different sensitive values, the remaining set of tuples still obeys frequency $c/(c+1)$-diversity. This connection leads to a crucial property: if all the QI-groups in a generalized table satisfy recursive (c, l)-diversity, an adversary can correctly discover the sensitive value of an individual with probability at most $c/(c+1)$, provided that the adversary can preclude at most $l-2$ values as belonging to the victim individual.

The Figure 4.7 demonstrate a 3-diversity generalization. The q-blocks are divided by a line.

4.5.4 Anatomy

So far our discussion has employed generalization as the underlying anonymization framework. In this section, we proceed to introduce another framework: anatomy. As with generalization, anatomy can be combined with k-anonymity and l-diversity. The following analysis focuses on l-diversity, due to its obvious advantages over k-anonymity. In particular, we adopt frequency l-diversity, to avoid the complication of recursive (c, l)-diversity. Accordingly, we will assume that probabilistic homogeneity attacks are the objective of privacy protection.

Although generalization preserves privacy, it often loses considerable information in the microdata, which severely compromises the accuracy of data analysis.

Anatomy overcomes the above defect of generalization, by releasing the exact QI-distribution without compromising the quality of privacy preservation. Specifically, anatomy releases a quasi-identifier table (QIT) and a

ID	QI			SV
	Zip Code	Age	Nationality	Condition
1	1305*	< 40	*	Heart Disease
4	1305*	< 40	*	Viral Infection
9	1305*	< 40	*	Cancer
10	1305*	< 40	*	Cancer
5	1485*	> 40	*	Cancer
6	1485*	> 40	*	Heart Disease
7	1485*	> 40	*	Viral Infection
8	1485*	> 40	*	Viral Infection
2	1306*	< 40	*	Heart Disease
3	1306*	< 40	*	Viral Infection
11	1306*	< 40	*	Cancer
12	1306*	< 40	*	Cancer

Figure 4.7: 3-diversity inpatient microdata

sensitive table (ST), which separate QI-values from sensitive values. For example, Tables 4.17 (a) and (b) demonstrate the QIT and ST obtained from the microdata Table 4.16 (a), respectively.

Construction of the anatomized tables can be (informally) understood as follows. First, we partition the tuples of the microdata into several QI-groups, based on a certain strategy. We use an example in Table 4.16 to explain.

Here, following the grouping in Table 4.16, let us place tuples 1-4 (or 5-8) of Table 4.16(b) into QI-group 1 (or 2). Then, we create the QIT. Specifically, for each tuple in Table 4.16 (a), the QIT (Table 4.17 (a)) includes all its exact QI-values, together with its group membership in a new column Group-ID. However, QIT does not store any Disease value. Finally, we produce the ST (Table 4.17 (b)), which retains the Disease statistics of each QI-group. For instance, the first two records of the ST (to avoid confusion, we use record instead of couple for the data of an ST) indicate that, two tuples of the first QI-group are associated with dyspepsia, and two with pneumonia. Similarly, the next three records imply that, the second QI-group has a tuple associated with bronchitis, two with flu, and one with gastritis.

Anatomy preserves privacy because the QIT does not indicate the sensitive value of any tuple, which must be randomly guessed from the ST. To

tuple ID	age	sex	zipcode	disease
1	23	m	12000	pneumonia
2	27	m	14000	dyspepsia
3	35	m	18000	dyspepsia
4	59	m	19000	pneumonia
5	61	f	37000	flu
6	65	f	33000	gastritis
7	65	f	36000	flu
8	70	f	30000	bronchitis

(a) The microdata

tuple ID	age	sex	zipcode	disease
1	[21,60]	m	[10001,60000]	pneumonia
2	[21,60]	m	[10001,60000]	dyspepsia
3	[21,60	m	[10001,60000]	dyspepsia
4	[21,60]	m	[10001,60000]	pneumonia
5	[61,70]	f	[10001,60000]	flu
6	[61,70]	f	[10001,60000]	gastritis
7	[61,70]	f	[10001,60000]	flu
8	[61,70]	f	[10001,60000]	bronchitis

(b) A 2-diverse table

Table 4.16: A generalization example

tuple ID	age	sex	zipcode	group ID
1	23	m	12000	1
2	27	m	14000	1
3	35	m	18000	1
4	59	m	19000	1
5	61	f	37000	2
6	65	f	33000	2
7	65	f	36000	2
8	70	f	30000	2

(a) The quasi-identifier table (QIT)

group ID	disease	count
1	dyspepsia	2
1	pneumonia	2
2	bronchitis	1
2	gastritis	1
2	flu	2

(b) The sensitive table (ST)

Table 4.17: The anatomized table

explain this, consider the adversary who has the age 23 and zipcode 11000 of Bob. Hence, from the QIT (Table 4.17 (a)), the adversary knows that tuple 1 belongs to Bob, but does not obtain any information about his disease so far. Instead, s/he gets the id 1 of the QI-group containing tuple 1. Judging from the ST (Table 4.17 (b)), the adversary realizes that, among the 4 tuples in QI-group 1, 50% of them are associated with dyspepsia (or pneumonia) in the microdata. Note that s/he does not gain any additional hints, regarding the exact diseases carried by these tuples. Hence, s/he arrives at the conclusion that Bob could have contracted dyspepsia (or pneumonia) with 50% probability. This is the same conjecture obtainable from the generalized Table 4.16 (b).

By announcing the QI values directly, anatomy permits more effective analysis than generalization. Given query for pneumonia and age ≤ 30, we know, from the ST, that 2 tuples carry pneumonia in the microdata, and they are both in QI-group 1. Hence, we proceed to calculate the probability p in the age-zipcode plane. This calculation does not need any assumption about the data distribution in the Age-Zipcode plane, because the distribution is precisely released. Specifically, the QIT shows that tuples 1 and 2 in QI-group 1, leading to the exact $p = 50\%$.

4.6 Database watermarking

The motivation for database watermarking is to protect databases, especially those published online (e.g., parametric specifications, surveys, and life sciences data), from tampering and pirated copies. A watermark can be considered to be some kind of information that is embedded into underlying data for tamper detection, localization, ownership proof, and/or traitor tracing purposes.

Database watermarking consists of two basic processes: watermark insertion and watermark detection.

The existing database watermarking schemes can be classified along various dimensions, such as

- Data type: Different schemes are designed for watermarking different types of data, including numerical data and categorical data.
- Distortion to underlying data: While some watermarking schemes inevitably introduce distortions/errors to the underlying data, others are distortion-free.
- Sensitivity to database attacks: A watermarking scheme can be either robust or fragile to database attacks. A scheme is robust (fragile, respectively) if it is difficult to make an embedded watermark undetectable (unchanged, respectively) in database attacks, provided that the attacks do not degrade the usefulness of the data significantly.
- Watermark information: The watermark information that is embedded into a database can be a single-bit watermark, a multiple-bit watermark, a fingerprint, or multiple watermarks in different watermarking schemes.
- Verifiability: A watermark solution is said to be private if the detection of a watermark can only be performed by someone who owns a secret

key and can only be proved once to the public (e.g., to the court). After this one-time proof, the secret key is known to the public and the embedded watermark can be easily destroyed by malicious users. A watermark solution is said to be public if the detection of a watermark can be publicly proved by anyone, as many times as necessary.

- Data structure: Different watermarking schemes are designed to accommodate different structural information of the underlying data, including relational databases (with or without primary keys), data cubes, streaming data, and XML data.

In this section, we introduce some examples of watermarking schemes for database.

4.6.1 Watermarking numerical data

The fundamental assumption is that the watermarked database can tolerate a small amount of errors: it is acceptable to change a small number of ξ least significant bits in some numeric values; however, the value of data is significantly reduced if a large number of the bits are changed. The basic idea is to ensure that those bit positions contain specific values determined by a secret key K. The bit pattern constitutes a watermark.

For watermark insertion, the scheme scans each tuple r in a relation R and seeds a cryptographically secure pseudo-random sequence generator S with the secret key K in concatenation with the tuples primary key $r.P$. Alternatively, we can use HMAC and select random keys $K_1, K_2, \ldots$ and let $S_i = HMAC(K_i, r.P)$ as the ith number generated by S.

Let S_i be the i-th number generated by S. If S_1 satisfies ($S_1 \bmod \gamma = 0$), then the current tuple r is selected, otherwise the tuple is ignored, where γ is a watermarking parameter used to control the percentage of tuples being selected. Because S_1 is pseudo-random, roughly η/γ tuples are selected, where η is the total number of tuples in relation R. Then, for each selected tuple, the scheme selects one attribute with index ($S_2 \bmod \nu$) out of ν watermarkable numerical attributes indexed from 0 to $\nu - 1$. For the selected attribute of a selected tuple, the scheme selects one bit with index ($S_3 \bmod \xi$) out of ξ least significant bits indexed from 0 to $\xi - 1$, where ξ is a watermarking parameter used to control the error that each numerical value can tolerate. The scheme then assigns the selected bit of the selected attribute in the selected tuple with a mark value ($S_4 \bmod 2$). With a probability of 1/2, the underlying bit value is changed in this process. Due to the use of a cryptographically secure pseudo-random sequence generator, it is computationally infeasible for an attacker, without knowing the secret key, to derive where the watermark bits are embedded, what the mark bits are, and the correlations among the embedded locations and the embedded values.

For watermark detection, the scheme scans all the tuples in a suspicious database relation R', locates the marked bit positions, and computes the mark values at those bit positions exactly as in watermark insertion. To detect a watermark, the scheme compares the computed mark values to the corresponding bit values stored in R'. A watermark is detected if the percentage of matches in such comparison is greater than τ , where $\tau \geq 0.5$ is a parameter that is related to the assurance of the detection process.

4.6.2 Distortion-free watermarking

In this solution, all η tuples in a database relation R are first securely divided into g groups according to a secret key K. A different watermark is embedded and verified in each group independently. As a result, any modifications to the watermarked data can be detected and localized to the group level with high probabilities.

First, a (keyed) tuple hash and a (keyed) primary key hash are computed for each tuple r_i using a HMAC function. The tuple hash values are computed based on a fixed order of attributes. Based on the primary key hash values, all tuples are securely divided into g groups. The grouping is only a virtual operation, which means that it does not change the physical position of the tuples. After grouping, all tuples in each group are sorted according to their primary key hash. Like grouping, the sorting operation does not change the physical position of tuples either. Each group is then watermarked independently.

Algorithm 1 Watermark embedding

```
For all $k = 1, \dots g, q_k = 0$
  for $i = 1$ to $\eta$ do
    $h_i = HMAC(K, r_i)$ // tuple hash
    $h_i^p = HMAC(K, r_i.P)$ // primary key hash
    $k = h_i^p \bmod g$
    $r_i \to G_k$ // Virtual operation: assign tuple $r_i$ to group $k$
    $q_k++$
```

```
end for
for $k = 1$ to $g$ do
 watermark embedding in $G_k$ // See algorithm 2
end for
```

Algorithm 2 shows the embedding process in each group. A (keyed) group hash value is computed based on the tuple hash values in a sorted order. A watermark, the length of which is equal to the number of tuple-pairs in the current group, is extracted from the group hash value. To embed the watermark, for each tuple pair, the order of the two tuples are changed or unchanged (physically in the original database) to represent a corresponding watermark bit 1 or 0, where 0 is encoded by the ascendant order and 1 by the descendant order. Since only the order of the tuples is changed, the watermark insertion does not introduce any error to the underlying data.

Algorithm 2 Watermark embedding in G_k

```
sort tuples in $G_k$ in ascendant order according to their primary key hash values// Virtual operation
$H = HMAC(K, h'_1); H = HMAC(K, H|h'_2); \ldots H = HMAC(K, H|h'_{qk})$ /* group hash, where $h'_i (i = 1, \ldots q_k)$ is the tuple hash of the $i$th tuple after ordering */
$W = extractBits(H, q_k/2)$ // See function below
  for $i = 1, i < q_k, i = i + 2$ do
   if $W[i/2] == 1$ then
    switch the position of $r_i$ and $r_i + 1$ physically in DB
   end if
  end for

function $extractBits(H, \ell)${
  if length($H$) $\geq \ell$ then
   W = concatenation of first $\ell$ selected bits from $H$ /* in most cases, $H$ is longer than $\ell$ */
  else
   $m = \ell-$ length($H$)
   W = concatenation of $H$ and extractBits($H, m$)
  end if
return $W$}
```

Algorithms 3 and 4 describe the watermark detection process. As in watermark insertion, the primary key hash is computed for each tuple and all tuples are divided into groups. Each group is processed independently. In a group, the tuples are first sorted according to their primary key hash values. Like watermark insertion, the sorting is a virtual operation and does not involve order change of any tuples. Based on the tuple hash of the sorted tuples, a group hash value is computed. Then, a watermark W is extracted from the group hash. The watermark W is the one that is supposed to have been embedded if the underlying data were watermarked. On the other hand, a binary string W' is extracted from the tuples in this group. For every tuple pair, if their tuple hash values are in ascendant order, the corresponding bit in W' is extracted to be zero; otherwise, it is one. If W matches W, the data in the group are authentic; otherwise, the data in this group have been modified or tampered with.

Algorithm 3 Watermark detection

```
For all $k = 1, \ldots g, q_k = 0$
   for $i = 1$ to $\eta$ do
    $h_i = HMAC(K, r_i)$
    $h_i^p = HMAC(K, r_i.P)$
    $k = h_i^p \bmod g$
    $r_i \rightarrow G_k$
    $q_k++$
   end for
   for $k = 1$ to $g$ do
    watermark verification in $G_k$ // See algorithm 4
   end for
```

Algorithm 4 Watermark verification in G_k

```
sort tuples in $G_k$ in ascendant order according to their primary key hash values// Virtual operation
$H = HMAC(K, h'_1); H = HMAC(K, H|h'_2); \ldots H = HMAC(K, H|h'_{qk})$
$W = extractBits(H, q_k/2)$ // See function in algorithm 2
   for $i = 1, i < q_k, i = i + 2$ do
    if $h_i < h_{i+1}$ then
     $W'[i/2] = 0$
    else
     $W'[i/2] = 1$
```

```
 end if
end for
if $W' == W$ then
 $\nu = TRUE$
else
 $\nu = FALSE$
end if

  switch the position of $r_i$ and $r_i + 1$ physically in DB
 end if
end for
```

The number g of groups is used to make a trade-off between security and localization. On the one hand, the smaller the value of g, the larger the probability of detecting modifications in watermark detection, and the more secure the proposed scheme. On the other hand, this leads to a larger group size; thus, one can localize modifications less precisely as there are more tuples in each group.

4.6.3 Robust watermarking

To confuse watermark detection, various database attacks may be launched to watermarked databases. Typical database attacks include tuple/ attribute insertion/ deletion/ reorganization, value modification/ suppression (including random/ selective bit-flipping/ value-rounding), invertibility attack (attacker successfully discovers a fictitious watermark which is in fact a random occurrence from a watermarked database), additive attack (attacker embeds some additional watermarks into a watermarked database), and the brute force attack against the secret key. The brute-force attack can be thwarted by assuming that the key is long enough (e.g., 160 bits) in watermarking.

- False hit: the probability of the original watermark being detected from unmarked data or a fictitious watermark being detected from watermarked data (i.e., invertibility attack).

- False miss: the probability of not detecting the original watermark from watermarked data.

The robustness of watermarking can be achieved by using majority vote in watermark detection.

Consider Bernoulli trials with probability p of success and q of failure. Let $b(k; m, p) = \binom{m}{k} p^k q^{m-k}$ be the probability that m Bernoulli trials result in k successes and $m - k$ failures, where $\binom{m}{k} = m!/k!(m-k)!, 0 \leq k \leq m$. Let $B(k; m, p) = \sum_{i=k+1}^{m} b(i; m, p)$ be the probability of having more than k successes in m Bernoulli trials. A watermark is detected if more than τ in percentage of the embedded bits are detected correctly. If the watermark detection is applied to unmarked data (or watermarked data with a different secret key), the detection can be considered as Bernoulli trials with a probability of 1/2 that a correct value will be found in a specific bit position. Assuming ω bits are checked in watermark detection, then the false hit rate is $B(\lfloor \tau\omega \rfloor; \omega, 0.5)$. The false hit rate is extremely low if τ and ω are reasonably large. For example, the false hit rate can be as low as 10^{10} for $\tau \geq 0.6$ and $\omega \geq 1000$.

The false miss rate can be analyzed under various attack scenarios. A typical modification attack is that an attacker randomly flips every least significant bit with a probability $p < 0.5$ (if $p \geq 0.5$, one can flip every bit back before watermark detection). For the detection algorithm to fail to recover the correct watermark, at least $\omega - \lfloor \tau\omega \rfloor$ embedded bits must be toggled. Thus, the false miss rate is $B(\omega - \lfloor \tau\omega \rfloor - 1; \omega, p)$. An attacker has to flip a significant portion of tuples in order to get a high probability of success in this attack. This scheme relies on the following assumptions to maintain its robustness. First, the watermarked relation has a primary key attribute that either does not change or else can be recovered. The rationale behind this is that a primary key attribute contains essential information and that modification or deletion of this information will substantially reduce the value of the data. With this assumption, the watermark detection is robust against tuple insertion/deletion and it is not affected by tuple reorganization. Second, the names of some, if not all, of the watermarked attributes either do not change or else can be recovered in watermark detection. Under the above two assumptions, the scheme is robust against attribute operations including insertion, deletion, and reorganization.

4.7 Trust management and trust negotiation

Authorization is one of the most important problems in computer security and privacy. Within a single organization, pre-established trust relationships are used to assign authorizations and prearranged information such as login names and passwords can serve as the basis for making authorization decisions at run time. However, when resource provider and resource requester belong to different organizations or have no prior relationship whatsoever, there are no pre-existing trust relationships. X.509 certificates can be used for cross-domain authentication or authorization. But because the rapidly changing organizational structure and partnerships, it becomes entirely ad hoc, chaotic, and unmanageable when the requirements for authorization have nothing to do with formal organizational affiliations, such as a senior citizen discount or letting family and friends access an on-line photo album. This is because the approach relies too heavily on pre-established trust relationships.

4.7.1 Trust management

Traditional access control models base authorization decisions on the identity of the principal who requests a resource (PKI certificate, passwords, access control list, stc). However, the number of autonomous services that are administered in a decentralized manner (i.e., within different security domains) has increased enormously on the Internet. As a result, services are often provided to clients whose identities are not previously known to the service provider. Similarly, the participants in a peer-to-peer system need to establish mutual trust in one another. In such a decentralized environment, the traditional access control mechanisms may not be used.

The trust management (TM) approach aims to provide a basis for authorization in highly decentralized environments by enabling resource owners to delegate authority to other entities who will help them identify the appropriate requesters to authorize. Trust management relies on digital credentials, which are unforgeable statements signed by their issuer. Typically, a digital credential contains an assertion about the properties of one or more principals mentioned in the credential. Most of these schemes rely on public key cryptography: the credential issuer signs the credential using its private key, and anyone can verify the contents of the credential by obtaining the corresponding public key and checking the signature. However, current credential standards already support the inclusion of additional information describing a principal's properties, such as one would need for a digital employee ID, driver's license, or birth certificate.

In TM systems, security policy is made by local administrators to specify access control rules on local resources. TM systems combined the notion of specifying security policy with the mechanism for specifying security credentials. The authorization semantics of most TM systems is monotonic in the sense that if any given action is approved when given a set of evidence E (i.e., policies and credentials), then it will also be approved when given any superset of E. This means that no negative evidence is allowed in the system.

Recent TM systems use credentials to characterize the holders of the credentials. These credentials need not contain specific authorizations, but provide more general attributes of the credential holders (e.g., student, US citizen, licensed driver born in 1960, etc.), which can be reused by various resource owners to make their access control decisions. This enables much more scalable policy definition.

SPKI/SDSI

SPKI/SDSI merged the SDSI and the SPKI efforts together to achieve an expressive and powerful trust management system. SDSI (Simple Distributed Security Infrastructure) was proposed as a new public-key infrastructure by Rivest and Lampson. Concurrently, Carl Ellison et al. developed SPKI (Simple Public Key Infrastructure). SDSIs greatest contribution is its design of local and extended names, which are bound to keys through the use of SDSI name certificates (see below), and which solve the problem of globally unambiguous naming. The owner of each public key can define names local to a name space that is associated with and identified by that key. For example, "K_{Alice} bob" is an example of a local name in which bob is an identifier and K_{Alice} is a globally unique key that we assume here belongs to a specific principal, Alice, who has sole authority to define bindings for the local name. Alice can define "K_{Alice} bob" to refer to a particular key "K_{Bob}" by issuing a tuple of the form (K_{Alice}, bob, K_{Bob}, 1). This in effect says that the principal that Alice refers to as bob has the key K_{Bob}. (The 1 just indicates that the certificate is valid.) Given such a binding, a reference to the SDSI name "K_{Alice} bob" can be resolved to the key K_{Bob}, which Bob can prove he controls when he needs to prove that he is the referenced principal.

Whereas a local name is a key followed by an identifier, an extended name is a key followed by two or more identifiers. The meaning of these are the result of multiple bindings of local names. For instance, if Bob were to issue the certificate (K_{Bob}, friend, K_{Carol}, 1), then the extended name "K_{Alice} bob friend" could be resolved to K_{Carol}. This brings up another important point about SDSI names; they can refer to more than one principal. For instance, Bob could also issue (K_{Bob}, friend, K_{Dave} friend, 1) with the effect that "K_{Alice} bob friend" would refer not only to Carol, but to all of Daves friends as well. Thus, SDSI names (both local and extended) can denote groups of keys and, equivalently, properties of key owners.

In general, SDSI name certificates are 4-tuples of the form (K, A, S, V), in which K is the key used to issue the certificate, "KA" is the local name being defined, S is either a key, a local name, or an extended name, and V is a certificate validity bit. A key point about SDSIs use of name spaces is that names that start with different keys are different names, so there is no danger of controllers of different public keys accidentally trying to bind the same name in conflicting ways.

SPKI contributed authorization certificates. These are 5-tuples of the form (K, A, D, T, V) in which K is the key issuing the certificate, A is the subject of the certificate, D is a delegation bit which indicates whether the authorization being conveyed to A can be further delegated by A, T is a tag that specifies the authorization being granted, and V is a certificate validity bit. While in the original design of SPKI, A was required to be a key, in SPKI/SDSI, A can also be a SDSI name. For example, a certificate such as (K_{Alice}, K_{Dave} friends, 1, downloadPhotos, 1) might indicate that Alice allows Daves friends to download photos and to delegate the permission to others. Notice that as principals are added to or removed from the group of Daves friends, they automatically gain or lose this permission.

QCM and SD3

QCM (Query Certificate Manager) was designed at the University of Pennsylvania as part of the SwitchWare project on active networks. It was designed specifically to support secure maintenance of distributed data sets. For example, QCM can be used to support decentralized administration of distributed repositories housing public key certificates that map names to public keys. In the sense of access control, QCM provides security support for the query and retrieval of ACLs. Although QCM is not designed to be a trust management system, it had significant impact on the TM system SD3 proposed by

Trevor Jim. One of the main contributions of QCM that can be adopted by other TM systems is its design of a policy directed certificate retrieval mechanism, which enables the TM evaluator to automatically detect and identify missing but needed certificates and to retrieve them from remote certificate repositories. It uses query decomposition and optimization techniques, and discusses its novel solutions in terms of network security, such as private key protection methods.

SD3 is the successor of QCM and inherits design features from QCM, such as the certificate retrieval mechanism in a dynamic decentralized certificate storage system. The SD3 project aimed to make trust management systems easy for applications to use. To this end, SD3 is responsible for verifying cryptographic signatures. In addition, SD3 has a credential retrieval mechanism that enables the evaluation of authorization decisions in the context of distributed credential storage. Finally, in order to guarantee returning a correct answer, SD3 implements certified evaluations, in which a checker checks the evaluator's outcome before passing it to the calling application. Together these features ensure that calling applications need only specify policy, without worrying about how it is enforced.

SD3 enables application developers to write policy statements in an extended Datalog (a declarative logic programming language used extensively in deductive database systems) that introduces a notion of name space in which predicates and relations are defined. It extends Datalog with SDSI names. In Datalog, "ancestor(x, y) :- parent(x, z), parent(z, y)" gives the rule: if x is z's parent and z is y's parent, then x is y's ancestor. The left part of rule before the symbol :- is called the head and the right part is called the body.

As an example of SD3, consider the following SD3 rule, which expresses the recursive case in the definition of the transitive closure (T) of the edge relation E : "$T(x, y)$:- $K\$E(x, y), T(y, z)$". Here K is a public key, E is a local relation name, defined in K's name space, and $K\$E$ is a global relation name, the definition of which is independent of the point of evaluation. The presence of this rule in a rule base associated with a given name space says that the pair (x, y) is in the the local relation T if it is in K's E relation. SD3 also allows an IP address A to be paired with its global name, such as $(K@A)\$E$, in which A is the IP address of an evaluation service operated by the principal that has public key K. The address assists in locating the evaluation agent and rule base associated with K, though the authenticity of the rule base is ensured by using K.

RT

The RT framework is a family of Role-based Trust-management languages that combines the strengths of RBAC (Role-Based Access Control) and the strengths of trust-management systems. Different languages in the family incorporate different features, but all members are designed to permit efficient (polynomial time) evaluation of ordinary authorization queries. Like SD3, RT is based on Datalog. However, rather than writing arbitrary Datalog clauses, the RT policy author uses a distinct RT syntax organized around RT language abstractions whose semantics is given by a formal translation of RT statements (i.e., credentials) into Datalog. This approach enforces an orderly policy-definition discipline while obtaining significant benefits from using what is in effect a subset of Datalog:

- the semantics are unambiguous and can be constructed in several well understood and equivalent manners (logical entailment, fixpoint, top down, bottom up, etc.);

- authorization queries are easily generalized to ask, for example, which principals are authorized to access a given resource, or which resources a given principal is authorized to access;

- the complexity of the RT features is easily determined by making use of established complexity results for evaluation of Datalog queries.

In addition, the way in which the Datalog clauses generated from RT statements are restricted enables RT credentials to be stored in a manner that is more flexible than is possible with QCM or SD3. Because of these restrictions, RT credentials that are stored with either their subject or their issuer can be located and retrieved as needed during authorization query evaluation. In QCM and SD3, credentials must be stored with their issuers.

The definition and use of roles in RT is based on and extends that of groups in SDSI. Keys are called principals. Each principal A controls the definition of a collection of roles of the form $A.R$ in which R is called a role name and is either an identifier r or, in members of the RT family of languages that support parameterized roles, an identifier applied to a list of parameters, as in $r(t_1, \ldots, t_k)$. Parameters are quite helpful for the purpose of expressing quantitative attributes, such as age or budget, as well as for enabling roles to express relationships between principals and data objects.

For instance, $Alice.read('/usr/alice/research/')$ might represent principals allowed to read Alices research directory.

Certificates in RT are called statements or credentials. For concreteness, we consider the forms these can take in RT_0. There are four types of credentials that an entity A can issue, each corresponding to a different way of defining the membership of one of A's roles, $A.r$.

- Simple Member: $A.r \leftarrow D$. With this credential A asserts that D is a member of $A.r$.
- Simple Inclusion: $A.r \leftarrow B.r_1$. With this credential A asserts that $A.r$ includes (all members of) $B.r_1$. This represents a delegation from A to B, as B may cause new entities to become members of the role $A.r$ by issuing credentials defining (and extending) $B.r_1$.
- Linking Inclusion: $A.r \leftarrow A.r_1.r_2$. $A.r_1.r_2$ is called a linked role. With this credential A asserts that $A.r$ includes $B.r_2$, for every B that is a member of $A.r_1$. This represents a delegation from A to all the members of the role $A.r_1$.
- Intersection Inclusion: $A.r \leftarrow B_1.r_1 \cap B_2.r_2$. $B_1.r_1 \cap B_2.r_2$ is called an intersection. With this credential A asserts that $A.r$ includes every principal who is a member of both $B_1.r_1$ and $B_2.r_2$. This represents partial delegation from A to B_1 and to B_2.

Again to illustrate the technique by which semantics are given to a set of RT_0 credentials, we now present the translation to Datalog. Given a set $\mathcal{C}$ of RT_0 credentials, the corresponding semantic program, $SP(\mathcal{C})$, is a Datalog program with one ternary predicate m. Intuitively, $m(A, r, D)$ indicates that D is a member of the role $A.r$. Given an RT statement c, the semantic program of $c, SP(c)$, is defined as follows, where identifiers starting with the "?" character are logic variables:

$$\begin{aligned}
SP(A.r \leftarrow D) &= m(A, r, D). \\
SP((A.r \leftarrow B.r_1) &= m(A, r, ?X) \text{ :- } m(B, r_1, ?X). \\
SP((A.r \leftarrow A.r_1.r_2) &= m(A, r, ?X) \text{ :- } m(A, r_1, ?Y), m(?Y, r_2, ?X). \\
SP((A.r \leftarrow B.r_1 \cap B_{2,r_2}) &= m(A, r, ?X) \text{ :- } m(B_1, r_1, ?X), m(B_2, r_2, ?X).
\end{aligned}$$

SP extends to the set of statements in the obvious way: $SP(\mathcal{C}) = \{SP(c) | c \in \mathcal{C}\}$. Now to determine whether a principal D belongs to role $A.r$, one simply evaluates a query (according to any one of a variety of evaluation mechanisms) to determine whether it is the case that $SP(\mathcal{C}) \models m(A, r, D)$.

RT_1 adds parameterized roles to RT_0, and RT_2 adds logical objects to RT_1. Just as roles group together related entities so that their authorizations can be assigned in fewer statements, logical objects logically group together objects so that their permissions can be assigned together. RT^C incorporates constraint systems, carefully selected to preserve query-answering efficiency. Constraints are very helpful for representing ranges of quantitative values and object specifiers such as directory paths. For instance, they can very concisely express policies such as "anyone over 65 is entitled to a senior citizen discount" and "Alice can access the entire directory subtree of /usr/home/Alice". RT^T provides manifold roles and role-product operators, which can express threshold policies and separation-of-duty policies. RT^D provides delegation of role activations, which can express selective use of capacities and delegation of these capacities. RT^D and RT^T can be used, together or separately, with each of RT_0, RT_1, or RT_2. The resulting combinations are written RT_i, RT_i^D, RT_i^T , and RT_i^{DT} for $i = 0, 1, 2$.

SDSI extended names and RT's linked roles both rely on agreement among principals as to the intended meaning of role names (identifiers in SDSI). For instance, a linked name such as ABET.accreditedUniversity.student is only meaningful if there is some agreement among ABET-accredited universities as to what it means to be a student.

PCA and TPL

PCA (Proof Carrying Authorization) was designed primarily for access control in web services. HTTP proxies are used to make the whole process of accessing a web page transparent to the web browser. The web browser only knows the final result of either a displayed web page that it attempted to access, or a denial message. The proxy is designed to be portable and easily integrated into the client system without changing anything inside the original web browser. The client is responsible for constructing a proof of authorization, which the server need only check for correctness. This substantially reduces the burden imposed on the server by the authorization process.

PCA uses higher-order logic to specify policies and credentials, so that it can be very expressive. Indeed, in general the determination of whether a proof of authorization exists is undecidable, much less tractable. PCA overcomes this issue as follows. First, as mentioned earlier, the server only has to check the proof constructed by the requester, and the checking process is decidable and tractable. Second, on the client side, the proxy is responsible for discovering and retrieving credentials, computing proofs, and communicating with the server. To avoid undecidable computations on the client side, the client proxy does not use the full logic; instead, it uses a limited, application specific logic, in which authorization decisions are tractable.

TPL (Trust Policy Language), designed at IBM Haifa Research Lab, was proposed specifically for trust establishment between strangers. TPL is based on RBAC and extends it by being able to map strangers automatically to roles. Unlike other trust management systems, TPLs efforts are put only into mapping users to roles, and not into mapping roles to privileges, which simplifies the design. The latter is the responsibility of the application. The concrete syntax of TPL uses XML to represent security rules. These are then translated by TPL into a standard logic programming language, Prolog.

Using different transcoders, TPL is certificate format independent: rules written in XML can be translated and reorganized by the transcoders into any certificate formats, such as X.509 or PGP. In each certificate, the certificate type field points to its certificate profile, which selects the proper transcoder to interpret that certificate into its XML rules.

The mandatory components of each certificate are the issuers public key, the subjects public key, the certificate type, the version of the certificate, the profile URL, the issuer certificate repository, and the subject certificate repository. The last two components were innovative considerations with respect to credential retrieval. First, to enable the TPL system to automatically retrieve relevant certificates from remote repositories, the certificate that is currently being processed should specify the locations of the repositories where the relevant certificates are housed. Second, certificates can be referenced negatively in TPL, which means that TPL is non-monotonic in the sense that adding certificates can diminish authorizations. Thus TPL cannot rely on requesters to present certificates that are referenced negatively. Instead the resource owner specifies a credential collector, which is a software module configured to know about trusted repositories of negative certificates.

4.7.2 Trust negotiation

Trust negotiation is the process of establishing bilateral trust at run time. Trust negotiation uses verifiable, unforgeable digital credentials that describe principals properties, together with explicit policies that describe the properties that a principal must possess in order to gain access to a particular resource. When Alice wishes to access a resource owned by Bob, she and Bob follow one of many proposed trust negotiation protocols to determine whether she has the right properties, i.e., whether her credentials satisfy the policy that Bob has specified for access to that resource.

There are many different algorithms that a set of autonomous parties can follow to establish trust at run time. However, all recent approaches to trust negotiation do share the following advantages over traditional identity-based approaches to authorization:

- Two previously unacquainted principals can establish bilateral trust between themselves at run time.
- The authorization policy for a resource can specify the properties that authorized parties must possess, removing the administrative burden of maintaining access control lists of authorized identities.
- Trust establishment does not rely on the existence of any trusted third parties, other than credential issuers.
- In trust negotiation approaches that involve direct disclosure of credentials, trust can be built up gradually through an iterative process, starting with less sensitive properties and moving on to more sensitive ones after a certain level of trust has been established.
- In trust negotiation approaches that do not involve direct disclosure of credentials, trust can be established without either principal learning exactly which properties the other principal possesses.

All approaches to trust negotiation also share a reliance on policy languages with certain properties, including the following:

- The policy language must possess a well-defined semantics. This implies that the meaning of the policy in that language must be independent of any particular implementation of the language. Otherwise, two negotiating parties can disagree on whether a particular policy has been satisfied by a set of credentials, leading to chaos.

- As hinted earlier, the language and runtime system should be monotonic in the sense that once a particular level of trust has been reached (e.g., access has been granted), the disclosure of additional credentials should not lower the level of trust. This limits the use of negation in policies in a pragmatic manner. For example, suppose that convicted felons cannot buy guns. This policy can be used as is in trust negotiation, as long as the store owner checks the negated construct (not a convicted felon) by conducting the appropriate credential discovery process himself. In other words, the store owner cannot decide that it is okay to sell Alice a gun, just because she has not supplied a convicted felon credential. The store owner must go out to the national registry of criminals and see whether Alice is listed there. If the runtime system does not support credential discovery and the store owner has not cached the list of criminals before negotiation starts, then the policy cannot be used as written.

- At a minimum, the policy language should also support conjunction, disjunction, transitive closure, constraints on attribute values, and constraints that restrict combinations of multiple credentials (theta-joins, in database terminology).

Avoiding Information Leakage during Trust Negotiation

Researchers recognized early on that negotiation strategies that directly disclose credentials may leak information about credentials and policies that are never disclosed. By observing the behavior of a party, one may also be able to determine what strategy they are using, which can be used as leverage in extracting additional information.

A credential may contain more information than needed to satisfy a policy. For example, Alice can prove that she is over 21 by presenting a digital drivers license. However, the license also gives her home address, exact date of birth, weight, and other details that are not needed to prove that she is over 21. To address these shortcomings, researchers have proposed versions of digital credentials that allow one to hide information that is irrelevant to the negotiation at hand, such as Alices home address. More sophisticated (and more expensive) schemes provide even more privacy, by avoiding direct disclosure of credentials. For example, Alice can prove that she is over 21, without disclosing her exact age. These schemes allow Alice to prove to Bob that she has the properties specified in his policy, without Bob learning exactly what properties she has. For example, in the pharmacy example, Bob might learn that Alice is authorized to place an order, without learning who her doctor is. Bob only learns that Alice has some combination of properties that satisfy his policy.

Often, possession or non-possession of a sensitive credential is itself sensitive information. For example, suppose that Alice is a CIA employee, and Bob is looking for people who might be such agents. Bob might query people for their CIA credentials. Even if Alice has a policy to protect the credential, her response for Bobs credentials on receipt of such a request can indicate that she has the credential. In other words, a request for such a credential may cause the recipient to issue counter-requests for credentials needed to satisfy disclosure of the sensitive credential. This, in turn, may indicate that the recipient possesses the sensitive credential. Non-possession may also be sensitive, and termination of a negotiation upon request for a credential can indicate non-possession. If the value of an attribute in a credential is sensitive, then it is possible for a principal to determine ownership and value of the attribute by the other negotiating principal based on her replies. For example, suppose that Alice has a sensitive date of birth field in her drivers license. Now, if Bobs policy has a constraint on age, and upon receipt of Bobs policy, Alice responds by asking for any further credentials from Bob, then Bob can assume that Alice has the attribute that satisfies the constraint. By using a scheme similar to binary search, it is possible for Bob to determine Alices age, without Alice revealing it to him.

Under many proposed approaches to trust negotiation, an attacker can even use a need-to-know attack to systematically harvest information about an arbitrary set of credentials that are not even relevant to the clients original request. To do this, the attacker rewrites her policies in such a way that they are logically equivalent to the original policies, but when used during negotiation, they force the victim into a series of disclosures related to the credentials being harvested. Once the harvest is over, the negotiation completes as it would have with the original policies. The most complete solution to these problems is to adopt a negotiation approach that does not involve direct disclosure of credentials. While these approaches vary in the degree of privacy that they provide, all of them can avoid the leaks cataloged in this section. The price of this improved protection, of course, is significantly longer execution times; thus one may wish to reserve these expensive strategies for policies that are particularly sensitive, and use direct disclosure elsewhere. In general, these TN approaches replace direct disclosure with sophisticated cryptography, usually coupled with special-purpose formats for credentials.

In some instances, less expensive forms of protection can be effective against leakage. One approach is that when Bob queries Alice about a sensitive attribute, she does not respond, whether she has that attribute or not. Only after Bob satisfies the conditions to allow disclosure does Alice would disclose the credential or disclose the fact that she does not possess it. This approach is also effective if non-possession is sensitive. However, it relies on the willingness of individuals to behave in the same manner whether or not they possess the sensitive attributeand for those who do not possess it, there may be little incentive to behave in this manner, as the negotiation will progress faster if they immediately confess that they do not have the attribute.

Another solution with moderate runtime costs involves the use of acknowledgement policies. In this scheme, Alice has an acknowledgement policy (ack-policy) for each possible sensitive credential, regardless of whether she has that credential or not. She only discloses whether she has the credential after the ack-policy has been satisfied. This approach also relies on the willingness of people who do not possess a sensitive attribute to act as though they did, even though it will prolong negotiations. The other disadvantage of this approach is that users will have many more policies, and policy specification and maintenance is a huge practical challenge.

Another way to address the problem is to abstract away from requesting specific credentials, and instead request a particular attribute. For example, one can request age instead of a drivers license. With the help of an ontology of concepts and credential contents, a party can choose which credential to disclose to prove possession of the desired attribute, in such a manner that as little sensitive information as possible is disclosed in the process. For example, Alice might choose to prove her age by disclosing her passport rather than her drivers license, as the latter includes her home address and other sensitive information not present in a passport. The ontology can also be used to help respond to requests for a particular attribute by disclosing either more specific or more general information than was requested. For example, if asked to prove North American residency, a party might instead prove that they live in Mexico.

Chapter 5

Wireless Network Security

A wireless network communication takes place over a wireless channel (which is usually a radio channel, or sometimes an infrared channel).

- The channel can be eavesdropped.
- The data can be altered.
- The radio channel can be overused.
- The channel can be jammed.
- The mobility may cause security problem.

5.1 Cellular networks

Originally, cellular networks provided only voice communications services and they could also be used to send and receive short text messages. Today, the range of application is much wider, including data communications, Internet access, multimedia applications (video telephony), and mobile payment services, etc.

Cellular networks are infrastructure-based networks. The infrastructure consists of base stations and a wired backbone network that connects the base stations together, as well as to the wired telephone system and to the Internet. Each base station serves only a limited physical area, called a cell. All the base stations of a given network operator together can cover a large area. Different network operators can jointly provide ubiquitous coverage and enable continent wide and ever worldwide mobility for users. In cellular networks, the only wireless part in the system is the link between the mobile phone and the base station. The rest is wired network. Base stations are connected to the mobile switching center (MSC) which is connected to the public switched telephone network (PSTN). The frequency spectrum allocated to wireless communications is very limited. Each cell is assigned a certain number of channels. To avoid radio interference, the channels assigned to one cell must be different from the channels assigned to its neighboring cells. However, the same channels can be reused by two cells that are far apart.

5.1.1 GSM

Global System for Mobile Communications (GMS) is a European initiated standard which is a prominent example of cellular network. One important security requirement of GSM is subscriber authentication. In addition to subscriber authentication, GSM also provides some countermeasures for the inherent weaknesses of the wireless channel. GSM provides confidentiality for voice communications and signaling over the wireless interface, and it protects the privacy of subscribers by hiding their identity from eavesdroppers. Being a wide area system, GSM security services operate in a multi-party environment.

In GSM, a subscriber and a network operator (called the home network operator) have a contractual relationship which is represented as a long-term secret key. The secret key and other identity related information of the subscriber are not stored in the mobile phone, but in a separate security unit, called the SIM (Subscriber Identity Module).

Subscriber authentication in GSM is based on a challenge-response principle. The subscriber receives an unpredictable random number as a challenger, and she must compute a correct response in order to be authenticated. The correct response is computed from the challenge and the secret key of the subscriber, which is shared with the home network.

The GSM subscriber authentication protocol can be described as follows. Assume that the subscriber roams into a foreign network, usually refereed to as the visited network. First the mobile reads the IMSI (International Mobile subscriber Identity) from the SIM, and sends it to the visited network. Based on the IMSI, the visited network determines the identity of the home network of the subscriber. Then the visited network forwards the IMSI to the home network via the backbone. The home network looks up the se-

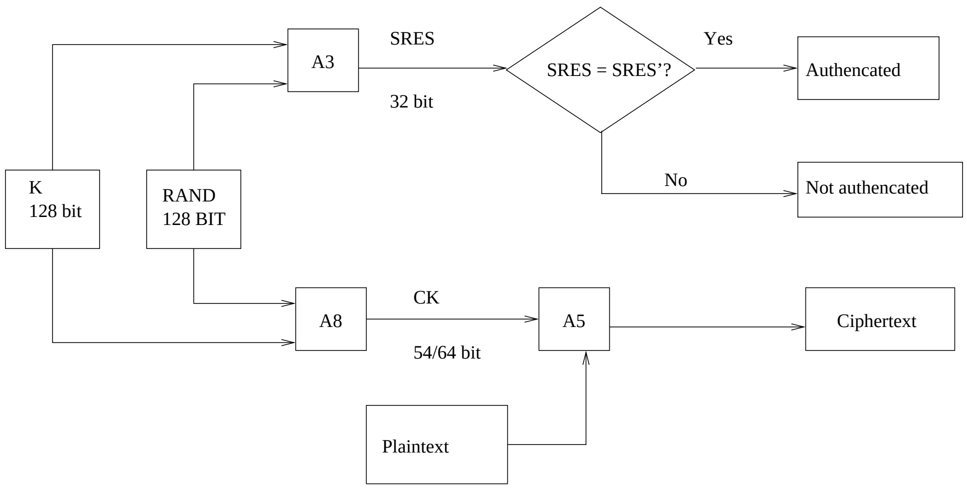

Figure 5.1: GMS security model

cret key K that corresponds to the subscriber by the IMSI. It then creates a triplet $(RAND, SRES, CK)$, where $RAND$ is an unpredictable random number used as the challenge, $SRES$ is the correct response to the challenge, and CK is a key to be used for encrypting communications over the wireless interface between the mobile phone and the base station of the visited network. $RAND$ is generated by a pseudo random number generator. $SRES$ and CK are computed from $RAND$ and K using the algorithms denoted by A3 and A8, respectively, in the GSM specifications. The triplet is sent to the visited network, which challenges the mobile phone with $RAND$. After the successful authentication of the subscriber, the communication between the mobile phone and the base station of the visited network are encrypted and decrypted with CK by using the stream cipher denoted by A5 in the GSM specification. It requires some trust in the home network operator by the visited network operator, which is established by signing roaming agreements between the two operators. In practice, the home network can transfer several triplets to the visited network when the subscriber first authenticates herself. In this way, the visited network does not need to contact the home network every time the subscriber needs to be authenticated.

The identity of the subscriber is hidden from eavesdroppers on the wireless interface as follows. After each successful authentication, the subscriber receives a temporary identifier called TMSI (Temporary Mobile Subscriber

Identifier) from the visited network. The TMSI is encrypted with the freshly established key CK. In the next authentication request, the mobile phone uses the TMSI, instead of IMSI, to identify the subscriber. The TMSI is mapped to the IMSI by the visited network.

When the subscriber moves into another visited network, the new network contacts the previous one and sends it the TMSI received from the mobile phone. The previous network looks up the data associated with the TMSI and transfers the IMSI of the subscriber and the remaining triplets (if any) to the new network, so that the new network can continue serving the subscriber. In case the data associated with the TMSI are no longer available in the previous network, the new network requests the mobile phone to send the IMSI in order to bootstrap the TMSI mechanism again.

Some flaws and attacks for GSM:

- No authentication of the network is provided to the user. It is possible for an attacker to set up a false base station with the same mobile network code as the subscriber's network.

- Common implementation of A3/A8 is flawed. This is because of the algorithms use the procedure COMP128.

- Vulnerabilities in the subscriber identity confidentiality mechanism. When a network loses track of a TMSI, it must ask the subscriber to submit its IMSI over the radio link, using a special mechanism for identity request. Thus the IMSI is sent in plain text. An attacker may use this to map a TMSI to its IMSI.

- Over the air cracking of K. An attacker first obtains the identity of a mobile phone (from TMSI and IMSI). Then the attacker requests a lot $SRES$ by sending $RAND$s. In this way, the attacker collects the $(RAND, SRES)$ pairs until he gains enough information to derive the key K (known plaintext attack).

5.1.2 UMTS

Universal Mobile Telecommunications System (UMTS) is an extension and improvement of GSM, which is the third generation cellular network in Europe. It provides necessary mechanisms for authenticating the network to the subscriber and considering integrity protection over the wireless interface.

In UMTS, the GSM triplet are replaced by authentication vectors that have five elements: $(RAND, XRES, CK, IK, AUTN)$. $RAND$ is an unpredictable random number used as a challenge in the subscriber authentication protocol, $XRES$ is the expected response to $RAND$ and CK is an encryption key to be used between the mobile phone and the base station of the visited network. Both $XRES$ and CK are computed from $RAND$ and the long-term secret key K of the subscriber. In addition, IK is an integrity protection key and $AUTN$ is a token that authenticates the home network to the subscriber and proves the freshness of $RAND$. $AUTN$ consists of three fields: $AUTN = (SQN \oplus AK, AMF, MAC)$, where

- SQN is a sequence number maintained synchronously by both the subscriber and the home network;
- AK is called the anonymity key, and it is used to hide the value of SQN from eavesdroppers. AK is generated from $RAND$ and K;
- AFM is an authentication and key management field used to pass parameters form the home network to the subscriber, but is is not fully specified in the UMTS standard;
- MAC is a message authentication code computed over $RAND, SQN$, and AMF using the long-term key K.

The subscriber authentication protocol is modified in such a way that, upon request, the visited network receives an authentication vector from the home network and it passes not only the challenge $RAND$ to the subscriber, but also the authentication token $AUTN$. The subscriber first generates the anonymity key AK and decodes the sequence number SQN received in $AUTN$. SQN is encoded with AK to protect the privacy of the subscriber. Otherwise, an eavesdropper could associate different executions of the authentication protocol with consecutive sequence numbers to the same subscriber. Once SQN is obtained, the subscriber verifies the MAC. If this verification is successful, then she knows that $RAND$ originates from her home network. Then the subscriber verifies if SQN is greater than the last sequence number stored by the subscriber. If it does not hold, then the protocol fails. This prevents the subscriber from accepting an old challenge. Finally, the subscriber computes a response RES to $RAND$ and sends it

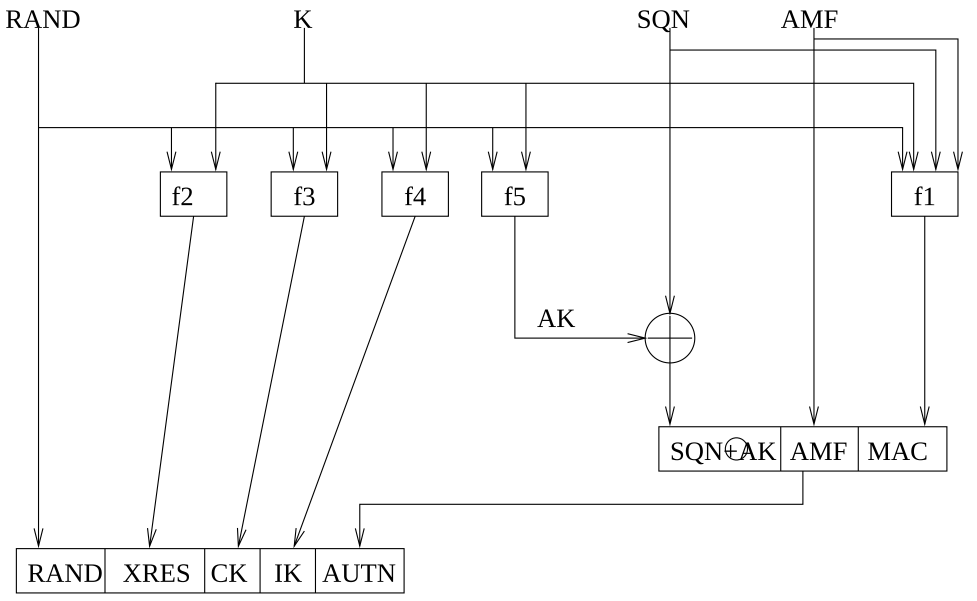

Figure 5.2: UMTS authentication vector

back to the visite network. The subscriber also computes CK and IK. Naturally, these computations are not performed by the subscriber herself, but the security unit of her mobile phone, which is called USIM.

The visited network compares RES to $XRES$, and if they are equal, then the authentication of the subscriber succeeds. After that, the mobile phone and the base station of the visited network protect the integrity and the confidentiality of their communications with IK and CK, respectively. Figure 5.2 describes the construction of UMTS authentication vector.

One weakness of the UMTS subscriber authentication protocol is that the home network does not pass any confirmation regarding the identity of the visited network to the subscriber in the authentication token. Therefore the visited network is not authenticated to the subscriber. This allows a malicious network operator X to masquerade as network Y to the subscriber. It would still authenticate itself as X to the home network, but the subscriber would not know this, and she would believe that she is served by Y. This can be a problem, as X and Y could use different tariffs and the subscriber would learn that she actually used a more expensive network when she receives her bill.

5.2 Wireless local area networks

The IEEE 802.11 wireless LAN standard featured a security architecture called WEP (Wired Equivalent Privacy). WEP is intended to increase the level of difficulty of attacking wireless LANs such that it becomes comparable to the difficulty of attacking wired LANs. However, researchers discovered the weakness of WEP and tools that automate the cracking of WEP keys appeared on the web.

IEEE came up with a new security architecture for wireless LANs, described in an extension to the 802.11 standard, which is called IEEE 80211i.

5.2.1 WEP

There are two basic security problems in wireless LANs:

- Due to the broadcast nature of radio communications, wireless transmissions ca be easily eavesdropped.

- Connecting to the network does not require physical access to the network Access Point (AP), thus any device can try to illegitimately use the services provided by the network.

WEP attempts to solve the first problem by encrypting messages. The second problem is addressed by requiring the authentication of the mobile stations (STAs) before allowing their connect to the network.

The authentication of the STA is based on a simple challenge-response protocol, similar to that used in GSM systems. Once authenticated, the STA communicates with the AP by encrypted messages. The key used for encryption is the same as the one used for authentication. The encryption algorithm specified by WEP is based on the RC4 stream cipher.

One problem of the encryption is addressed in WEP: if the same key are used to encryption two messages M_1 and M_2, then it is easy to get $M_1 \oplus M_2$: which is equal to $(M_1 \oplus K) \oplus (M_2 \oplus K)$. Definitely this is a weak encryption, which is likely to be broken by the statistical method. Therefore, WEP uses an IV (Initialization Vector) to the secret key before initializing the RC4 algorithm, where the IV changes for every message. The receiver should also know the IV. Thus the IV is sent in clear together with the encrypted message.

The sender also attaches an integrity check value(ICV) to the clear message. The purpose of this value is to enable the receiver to detect any malicious modifications of the message by an attacker. In WEP, the ICV is a CRC (cyclic redundancy check) value computed for the clear message. The CRC value is also encrypted by the secret key.

The key distribution in WEP is as follows. The standard states that each STA has its own key, know only to that STA and the AP. This makes the key management on the AP's side complicated, because the AP must store a key for every STA. For this reason, most implementations do not actually support this option. The standard also specifies a default key, known to every STA and the AP. Originally, this key was intended to be used for the encryption of broadcast messages originated by the AP. But most WEP implementations support only this default key. Therefore, in many WLANs, there is only one single common key, which only can be used to protect the communications from an outside attacker.

Some security problems of WEP.

- Authentication problems.
  - Authentication is not mutual, meaning that the AP does not authenticate itself to the STA.
  - Authentication and encryption use the same secret key.
  - STA is authenticated only at the time when it tries to connect to the network.
  - WEP uses RC4 in the authentication protocol for encrypting the random challenge. The attacker can easily obtain the challenge C and the encrypted challenge $R = C \oplus K$ by overhearing the exchange. Thus the attacker can easily obtain K which can be used to impersonate the STA later on. the IV mechanism of WEP does not mitigate this problem, because the value of IV is selected by the sender and the attacker can always use the same IV as R. It will make things worse, if any STA uses a same secret key.
- Integrity problems.

  The encrypted message in WEP can be written as $(M||CRC(M)) \oplus K$ where M is the message, K is the pseudo-random sequence produced by RC4, and $(CRC(M)$ is the ICV. CRC is linear with respect to the XOR operation, i.e., $CRC(X \oplus Y) = CRC(X) \oplus CRC(Y)$. Based on this, an attacker can manipulate a WEP message. Suppose the attacker knows $(M||CRC(M)) \oplus K$ and she wants to change the message to $M \oplus \Delta M$. Then she can just XOR the $\Delta M||CRC(\Delta M)$ to the original message. In fact,

$$\begin{aligned} & ((M||CRC(M)) \oplus K) \oplus (\Delta M||CRC(\Delta M)) \\ = \; & ((M \oplus \Delta M)||CRC(M) \oplus CRC(\Delta M))) \oplus K \\ = \; & ((M \oplus \Delta M)||CRC(M \oplus \Delta M)) \oplus K \end{aligned}$$

Another related problem is WEP does not considered a replay attack.

- Confidentiality problems.

  In WEP, the IV is only 24 bits long. A WiFi device can transmit approximately 17 million possible 500 full-length frames in a second, thus the whole IV space is used up in a few hours. Once all the IV values have been used, they start to repeat. That means a same key sequence will be repeated.

  Another practical problem is that in many WEP implementation, the IV is initialized with 0 at startup, and then incremented by one after each message sent. So several devices may use the same IV value. If they use a same key, then the things are even worse.

  The weakness of RC4 cipher causes the serious problem of WEP. It is known that there exist weak RC4 keys which let RC4 produces an output that does not look random. Security experts suggest always throwing away the first 256 bytes of the RC4 output, but WEP did not adopt it. Also due to the ever changing IV value, a weak key can be encountered sooner or later, and the attacker can easily know that a weak key is used, because the IV is transmitted in clear. Based on this, some cryptographers constructed a method that break the full 104-bit secret key by eavesdropping on only a few hundred thousands messages.

5.2.2 IEEE 802.11i

IEEE began to develop a new security architecture for WiFi networks described in the 802.11i specification. The new concept is called RSN (Robust

Security Network). It includes a new method for authentication and access control, which is based on the model defined in the 802.1X standard. The AES is used instead of RC4. However, since the old devices only support RC4 and not AES, the 802.11i included an optional protocol which still uses the RC4 but fixes the flaws in WEP. This protocol is called TKIP (Temporal Key Integrity Protocol).

Manufacturers immediately adopted TKIP. They did not wait until the 802.11i architecture was finalized by the lengthy standardization procedure, but they issued their own specification, called WPA (WiFi Protected Access), based on TKIP. WPA is a specification supported by WiFi manufacturers, which contains a subset of RSN. WPA can run on old devices that support only the RC4 cipher. Authentication and access control, as well as key management, are the same in WPA and in RSN. The difference between the two concepts lies in the mechanisms used for integrity protection and confidentiality. RSN is also called WPA2 by many manufacturers.

Authentication and access control

The 802.1X model distinguishes three entities in the authentication procedure: the supplicant, the authenticator and the authentication server. The supplicant wants to access the network and wants to authenticate itself. The authenticator controls access to the network, that is represented by controlling the state of a port. The default state of the port is “closed”, which means that data traffic is disabled. The authenticator can “open” the port if this is authorized by the authentication server. So the supplicant tries to authenticate itself to the authentication server, and if this authentication is successful, then the authentication server grants access to the network by instructing the authenticator to open the port.

In the WiFi networks, the supplicant is the mobile device and the authenticator is the AP. The authentication server is a process that can run on the AP in the case of smaller network, or on a dedicated server machine in the case of larger networks. A port in WiFi is not a physical connector, but a logical control implemented in software running on the AP.

In 802.1X standard, a port can only connect to one supplicant. Since in the wireless case, once the STA authenticates itself and associates with the AP, someone else may try to steal its session by spoofing its MAC address. So 802.11i extends 802.1X with the requirement of setting up a session key between the STA and the AP.

The authentication procedure in 802.11i uses EAP (Extensible authentication Protocol, RFC 3748) which has four message types: request, response, success and failure. EAP is only a carrier protocol at data link layer. EAP request and response messages are used to carry the messages of embedded authentication protocol from the STA to the server, and from the server to the STA, respectively. The EAP success and failure messages are used to signal the result of the authentication to the supplicant. The embedded authentication protocol can be higher layer protocols such as the EAP-TLS Handshake and GSM authentication protocols.

The EAP protocol and the embedded authentication protocol are executed by the mobile device and the authentication server. The AP relays messages without interpreting them. The AP only understand the success and failure messages.

EAP messages between the mobile device and the AP are carried by the EAPOL (EAP over LAN) protocol (802.1X). EAP messages between the AP and the authentication server can be carried by various protocols. WPA mandates the use of RADIUS (Remote Authentication Dial In User Service, RFC 3579), but RSN just specifies RADIUS as an option.

The authentication process in WiFi also establishes a session key to protect further communication between the mobile device and the AP. However, as authentication takes place between the mobile device and the authentication server, the session key is also established between them. The RADIUS uses MS-MPPE-Recv-Key RADIUS attribute, which has been specified for key transfer purpose, to securely transfer the session key to the AP. The session key is transferred in encrypted form, where the encryption uses a long-term key shared by the AP and the authentication server.

Key management

The session key established between the mobile and the AP is called the pairwise master key (PMK) which is not used directly for encryption or integrity protection. Both the mobile device and the AP derived four keys from the PMK: a data-encryption key, a data-integrity key, a key-encryption key and a key-integrity key. These four keys together are called the pairwise transient key (PTK). When AES-CCMP is used, the data encryption and integrity use the same key. In this case, PTK contains only three keys. The derivation of the PTK uses PMK together with the MAC addresses of the parties and two random numbers generated by the parties.

The mobile device and PA use four-way handshake protocol to exchange the random numbers, which are carried by the EAPOL protocol in EAPOL message of type Key.

1. The AP sends its random number to the mobile device. With this number, the mobile device can computes the PTK.

2. The mobile device sends its random number to the AP. This message also carries a Message Integrity Code (MIC), computed by the mobile device using the key-integrity. Upon reception of this message, the AP can compute the PTK and then verify th MIC.

3. The AP sends a message that contains a MIC to the mobile device. The MIC is computed using the key-integrity key of the PTK. This message contains the starting value of a sequence number that will be used to number further data packets, and to detect replay attacks. This message signals the mobile device that the AP has installed the keys and will encrypt subsequent data packets.

4. The mobile device acknowledges the reception of the third message.

For the purpose of broadcast messages, AP generates additional key material called the group transient key (GTK). The GTK contains a group encryption key and a group integrity key and it is sent to each mobile device separately encrypted with the key-encryption key of the given mobile device.

TKIP

AES-CCMP (AES CTR mode and CBC MAC Protocol) need new hardware that supports the AES algorithm. TKIP still uses RC4 but tries to fix flaws in WEP.

- TKIP uses a new integrity protection mechanism, called Michael. TKIP also uses the IV as a sequence number.

- In TKIP, the IV size is increased from 24 bits to 48 bits and each message is encrypted with a different key. The message keys are generated from the data-encryption key of the PTK. The 48-bit IV is divided into a 32-bit upper part and a 16-bit lower part. The upper part of the IV is combined with the 128-bit data-encryption key and the MAC address of the device. The result of this computation is then combined with the lower part of the IV in order to obtain the 104-bit message key. The RC4 seed value for TKIP is obtained by concatenating the message-key to the lower part of the IV and a dummy byte.

5.3 Bluetooth

Bluetooth is a wireless technology that uses short-range digital radio communications and offers fast and reliable transmission of both voice and data. Bluetooth incorporates a radio frequency transceiver and a full set of networking protocols on a single chip that is small enough to be included in cellular and cordless phone, portable PCs, headsets, etc.

Bluetooth protocol stack includes:

- Bluetooth Radio layer: This is the lowest defined layer of the bluetooth specification. It is not a protocol, but defines the requirements and operations of the bluetooth transceiver device, transmitting and receiving radio frequency signals in the 2.4GHz ISM band.

- Baseband: This is the physical layer protocol of the bluetooth specification and it lies on top of the bluetooth radio layer.

- Link Manager Protocol(LMP): LMP performs the link setup, configuration and authentication process within the bluetooth stack. After the Link Manager(LM) of one device discovers the LM of another device, the LMP then communicates with the remote LM to establish a link.

- Host Controller Interface (HCI): HCI provides a commend interface between the baseband control and the LM.

- L2CAP: The L2CAP resides on data-link layer (OSI model) and provides the link functions for the baseband protocol.

- RFCOMM: is the cable replacement protocol in the bluetooth stack. It creates a virtual seral port through which RF communications can be passed using standard EIA/TIA (Electronics Industries Association/Telecommunications Industry Association) 232 standard, which is the serial port communication standard.

- Object Exchange Protocol (OBEX): Originally specified by the Infrared Data Association (IrDA), OBEX is used to transfer data, graphics and voice objects between devices. OBEX is used in serval devices, such as PDAs (personal digital assistants), mobile phones and computer systems.
- vCard/vcal: A protocol used to store and transfer virtual business cards and personal calenders on mobile devices.
- PPP
- TCP
- IP
- WAP
- WAE: Create a World Wide Web environment on wireless mobile and handled devices.
- AT command set: It includes the control commands used to control dial-up modem functions and actions.
- Telephone Control Specification–Binary(TCS BIN): It is a bit-oriented protocol used to establish voice and data links between bluetooth devices.
- Service Discovery Protocol (SDP): Bluetooth requires an SDP to identify and manage the services available on portable devices moving into and out of range with each other. The Bluetooth SDP is specifically designed for bluetooth devices.

Unlike WLAN, in the case of bluetooth, communications are between wireless stations (no APs). The operation of bluetooth networks (called piconets) is based on the master-slave principle, where one station is the master and other stations (up to 7) become the slaves. Bluetooth technology can also provide a link to a wired network in a similar way that 802.11 access point do, by installing a bluetooth access point that is connected to a wired network.

An ad-hoc bluetooth network piconet can include up to 8 bluetooth devices. When more than 8 device are attempting to associate with a piconet, the piconet is divided into two or more piconets, and then interconnected into what is called a scatternet.

A bluetooth device cannot act as a master for two piconet. If a poconet master device is also linked to another piconet in a scatternet, it must participate as a master device in one piconet and a slave device in a second piconet. The slaves that share the same master device belong to the same pibonet and to remain as a member of a piconet, each slave must interact with the master on a time interval negotiated between the master and the salve. If the master leaves the piconet, the other devices must elect a new master and the piconet is reformed. When a bluetooth device is a member of two piconets, it can be used as a link between the piconets.

Any bluetooth device is capable of serving as a master or a slave, depending on the networking situation it encounters on-the-fly. A piconet can be formed using one of the following methods:

- A master device actively scans for slave devices and, when it detects one in its range, it can invite the device to join a piconet as a slave.

- A master device can passively wait for a slave to contact it, and then invite the slave to join the piconet as a slave.

Bluetooth devices each have a unique clock signal and device address, which are combined to provide a unique identity. The difference or offset between one device's clock signal and the clock signal of another device is the basis for the FHSS (frequency-hopping spread-spectrum) sequence used between the devices to transmit data. Bluetooth device use frequency hopping in order to avoid interference with other devices that operate in the same unlicensed ISM (Industrial Scientific and Medical) band. The frequency-hopping scheme uses 79 different channels and changes frequency 1600 times per second in a pseudo-random matter. This makes eavesdropping slightly more difficult. Bluetooth is a short range radio technology enabling communications over a few meters only (mostly in class 2 that is for 10 meters). That means that an attacker must be physically close to the victims in order to eavesdrop the communications, which also reduces the likelihood of attacks.

The bluetooth security architecture is concerned with the establishment of a secured wireless link between two bluetooth devices. This involves the authentication of the devices each other and setting up a confidential channel between them. Cryptographic functions used here are E_1, E_{21}, E_{22} and E_3, which are based on the SAFER+ block cipher.

There are two ways to establish a link key. The first method is used when one of the devices has memory limitations and can store only one key, otherwise the second method is used. Both methods start by setting up a temporary initialization key K_{init}. First, one device selects a random number IN_RAND and sends it to the other device. Then both devices compute K_{init} as a function of IN_RAND, a share PIN and the length L of the PIN. The length of PIN can be vary between 1 and 16 bytes. The PIN can be shared between the devices in several ways. If both devices have some input facility, then the user can choose a random PIN and enter it into both devices. If only one device has input facility, then the user can enter the pre-configured PIN of the other device into the first device.

Now suppose one device, A, has memory limitation. A sends its long-term unit key K_A to the other device B encrypted with the key K_{init}. B decrypts the cipher text and K_A become the link key.

When none of the devices has memory limitation, both A and B choose a random number $RAND_A$ and $RAND_B$, respectively. A computes LK_K_A as a function of $RAND_A$ and its unique device address BD_ADDR_A. B does the similar computation. Then they exchange $RAND_A$ and $RAND_B$ encrypted with K_{init}. So A can get LK_K_B and uses $LK\_K_A \oplus LK\_K_B$ as the link key. B can also compute the link key.

After two devices share a link key, they authenticate each other using a simple challenge-response protocol as follows. One of the device, referred as the "verifier", generates a random number AU_RAND and sends it to the other device called " claimant". Both of them compute an authentication response $SRES$ from AU_RAND, BD_ADDR of the claimant, and the link key Key_{link} by security function E_1. The procedure also generate ACO (authentication cipher offset). The claimant sends the value $SRES$ to the verifier who then check its correctness. If the protocol above fails, then the verifier device will wait some time before a new attempt can be made. This makes it impractical for an attacker to defeat authentication by trying different keys in rapid succession.

The encryption key K_{enc} is computed by both devices as a function of three elements: link key K_{link}, the authentication cipher offset ACO generated during the authentication protocol, and a random number EN_RAND generated by the master device. A stream cipher E_0 is used for encryption. Besides the encryption key, E_0 also inputs the address BD_ADDR_{amster} of the master device, and the clock value $CLOCK_{master}$ of the master.

Some weakness of bluetooth:

- The strength of the system is based on PIN which is typically a 4-digit number. An attacker can eavesdrop the communication, then try 10000 possible values off-line.
- For a memory-constrained device, the link key is the long-term unit key. An attacker can obtain the unit key by establishing a link key with it.
- There is also a privacy problem that stems from the use of fixed and unique device addresses. An attacker can track the whereabout of the person by tracking the use of the given device address.
- The encryption algorithm E_0 has some weakness.

5.4 Mobile Sensor Networks

In this section, we consider wireless sensor networks (WSN) which usually integrate a large number of low-power, low-cost sensor nodes. They are largely deployed to monitor a specific environment. Sensor networks often have one or more points of centralized control called base stations. sensor nodes are small in size and able to sense, process data and communicate with each other, typically over an RF (radio frequency) channel. In general, there are three categories of traffic:

- Many-to-one traffic.
- One-to-many traffic.
- Local traffic: The nodes in a limited area send localized messages to discover the neighbouring nodes and coordinate with each other. May be broadcast or send messages intended for a single neighbour.

Some factors of WSNs which make more difficulties for the security of the communication.

- Sensor nodes have limited storage, computation and power resources.

- The WSN does not have a fixed infrastructure and does not have a static topology.
- The sensing and communication tasks are often performed in a hostile environment where the gathered events are subjected to numerous threats.
- The detected events are forwarded through the sensor nodes themselves.

5.4.1 WSN Features

The basic features of WSNs:

- Self-organizing capabilities. The WSNs are able to cope with topology variability and infrastructure variations.
- Short-range broadcast communication and multirouting. The sensor nodes have reduced radio ranges and should cooperate to achieve complete routing of information.
- Dense deployment and cooperative effort of sensor nodes. The shortage of the radio range and the need to have efficient sensing call for a dense deployment of sensors.
- Limitations of energy,transmit power, memory and computing power.WSNs cope with limitation of resources and frequent changes of topology due to fading and node failures.

Some of the challenges for WSNs:

- Extension of lifetime. A typical alkaline battery, for example, provides about 50 watt-hours of energy. Given the expense and the potential infeasibility of monitoring and replacement of batteries for a large WSN, significantly longer lifetimes would be desired.
- Responsiveness. A simple solution to extending network lifetime is to operate the nodes in a duty-cycled manner with periodic switching between sleep and wake-up modes. This causes the time synchronizing requirement, and the responsiveness of and the effectiveness of the sensors.

- Robustness. The use of large number of inexpensive devices characterizes the WSNs. It is important to ensure that the global performance of the system is not sensitive to individual device failure. It is also often desirable that the performance of the system degrade as gracefully as possible with respect to component failure.

- Synergy. Design synergistic protocol, which ensures that the system as a whole is more capable than the sum of the capabilities of its individual component. The protocols must provide an efficient collaborative use of storage, computation and communication resources.

- Self-configuration. WSNs are inherently unattended distributed systems. Autonomous operation of the network is therefore a key design. Nodes in a wireless sensor network have to be able to configure their own network topology, synchronize, and calibrate themselves; coordinate inter-node communication; and determine other important operating parameters.

- Privacy and security. The large scale, prevalence, and sensitivity of the information collected by WSN (as well as their potential deployment in hostile locations) give rise to the final key challenge of ensuring both privacy and security.

5.4.2 WSN security requirements

Data confidentiality

Data confidentiality is the most important issue in network security. Every network with any security focus will typically address this problem first. In sensor networks, the confidentiality relates to the following:

- A sensor network should not leak sensor readings to its neighbours. Especially in a military application, the data stored in the sensor node may be highly sensitive.

- In many applications nodes communicate highly sensitive data, e.g., key distribution, therefore it is extremely important to build a secure channel in a wireless sensor network.

- Public sensor information, such as sensor identities and public keys, should also be encrypted to some extent to protect against traffic analysis attacks.

The standard approach for keeping sensitive data secret is to encrypt the data with a secret key that only intended receivers possess, thus achieving confidentiality.

Data integrity

With the implementation of confidentiality, an adversary may be unable to steal information. However, this does not mean the data is safe. The adversary can change the data, so as to send the sensor network into disarray. For example, a malicious node may add some fragments or manipulate the data within a packet. This new packet can then be sent to the original receiver. Data loss or damage can even occur without the presence of a malicious node due to the harsh communication environment. Thus, data integrity ensures that any received data has not been altered in transit.

Data freshness

Even if confidentiality and data integrity are assured, we also need to ensure the freshness of each message. Informally, data freshness suggests that the data is recent, and it ensures that no old messages have been replayed. This requirement is especially important when there are shared-key strategies employed in the design. Typically shared keys need to be changed over time. However, it takes time for new shared keys to be propagated to the entire network. In this case, it is easy for the adversary to use a replay attack. Also, it is easy to disrupt the normal work of the sensor, if the sensor is unaware of the new key change time. To solve this problem a nonce, or another time-related counter, can be added into the packet to ensure data freshness.

Availability

Adjusting the traditional encryption algorithms to fit within the wireless sensor network is not free, and will introduce some extra costs. Some approaches choose to modify the code to reuse as much code as possible. Some approaches try to make use of additional communication to achieve the same goal. What is more, some approaches force strict limitations on the data access, or propose an unsuitable scheme (such as a central point scheme) in order to simplify the algorithm. But all these approaches weaken the availability of a sensor and sensor network for the following reasons:

- Additional computation consumes additional energy. If no more energy exists, the data will no longer be available.
- Additional communication also consumes more energy. What is more, as communication increases so too does the chance of incurring a communication conflict.
- A single point failure will be introduced if using the central point scheme. This greatly threatens the availability of the network.

The requirement of security not only affects the operation of the network, but also is highly important in maintaining the availability of the whole network.

Self-organization

A wireless sensor network is a typically an ad hoc network, which requires every sensor node be independent and flexible enough to be self-organizing and self-healing according to different situations. There is no fixed infrastructure available for the purpose of network management in a sensor network. This inherent feature brings a great challenge to wireless sensor network security as well. For example, the dynamics of the whole network inhibits the idea of pre-installation of a shared key between the base station and all sensors. Several random key predistribution schemes have been proposed in the context of symmetric encryption techniques. In the context of applying public-key cryptography techniques in sensor networks, an efficient mechanism for public-key distribution is necessary as well. In the same way that distributed sensor networks must self-organize to support multi-hop routing, they must also self-organize to conduct key management and building trust relation among sensors. If self-organization is lacking in a sensor network, the damage resulting from an attack or even the hazardous environment may be devastating.

Time synchronization

Most sensor network applications rely on some form of time synchronization. In order to conserve power, an individual sensor's radio may be turned off for periods of time. Furthermore, sensors may wish to compute the end-to-end delay of a packet as it travels between two pairwise sensors. A more collaborative sensor network may require group synchronization for tracking applications, etc. researchers propose a set of secure synchronization protocols for sender-receiver (pairwise), multihop sender-receiver (for use when the pair of nodes are not within single-hop range), and group synchronization.

Secure localization

Often, the utility of a sensor network will rely on its ability to accurately and automatically locate each sensor in the network. A sensor network designed to locate faults will need accurate location information in order to pinpoint the location of a fault. Unfortunately, an attacker can easily manipulate nonsecured location information by reporting false signal strengths, replaying signals, etc. A technique called verifiable multilateration (VM) is proposed. In multilateration, a device's position is accurately computed from a series of known reference points. Authenticated ranging and distance bounding are used to ensure accurate location of a node. Because of distance bounding, an attacking node can only increase its claimed distance from a reference point. However, to ensure location consistency, an attacking node would also have to prove that its distance from another reference point is shorter. Since it cannot do this, a node manipulating the localization protocol can be found. For large sensor networks, the SPINE (Secure Positioning for sensor NEtworks) algorithm is used. It is a three phase algorithm based upon verifiable multilateration.

A SeRLoc (Secure Range-Independent Localization) can be described as follows. Its novelty is its decentralized, range-independent nature. SeRLoc uses locators that transmit beacon information. It is assumed that the locators are trusted and cannot be compromised. Furthermore, each locator is assumed to know its own location. A sensor computes its location by listening for the beacon information sent by each locator. The beacons include the locator's location. Using all of the beacons that a sensor node detects, a node computes an approximate location based on the coordinates of the locators. Using a majority vote scheme, the sensor then computes an overlapping antenna region. The final computed location is the "center of gravity" of the overlapping antenna region. All beacons transmitted by the locators are encrypted with a shared global symmetric key that is pre-loaded to the sensor prior to deployment. Each sensor also shares a unique symmetric key with each locator. This key is also pre-loaded on each sensor.

Authentication

An adversary is not just limited to modifying the data packet. It can change the whole packet stream by injecting additional packets. So the receiver needs to ensure that the data used in any decision-making process originates from the correct source. On the other hand, when constructing the sensor network, authentication is necessary for many administrative tasks (e.g. network reprogramming or controlling sensor node duty cycle). From the above, we can see that message authentication is important for many applications in sensor networks. Informally, data authentication allows a receiver to verify that the data really is sent by the claimed sender. In the case of two-party communication, data authentication can be achieved through a purely symmetric mechanism: the sender and the receiver share a secret key to compute the message authentication code (MAC) of all communicated data.

Adrian Perrig et al. propose a key-chain distribution system for their μTESLA secure broadcast protocol. The basic idea of the μTESLA system is to achieve asymmetric cryptography by delaying the disclosure of the symmetric keys. In this case a sender will broadcast a message generated with a secret key. After a certain period of time, the sender will disclose the secret key. The receiver is responsible for buffering the packet until the secret key has been disclosed. After disclosure the receiver can authenticate the packet, provided that the packet was received before the key was disclosed. One limitation of μTESLA is that some initial information must be unicast to each sensor node before authentication of broadcast messages can begin.

Liu and Ning propose an enhancement to the μTESLA system that uses broadcasting of the key chain commitments rather than μTESLA's unicasting technique. They present a series of schemes starting with a simple predetermination of key chains and finally settling on a multi-level key chain technique. The multi-level key chain scheme uses pre-determination and broadcasting to achieve a scalable key distribution technique that is designed to be resistant to denial of service attacks, including jamming.

Bibliography

[1] Levente Buttyán and Jean-Pierre Hubaux, Security and Cooperation in Wireless Networks, `http://secowiner.epfl.ch`

[2] M. Gertz and S. Jajodia Edited, Handbook of Database Security, Applications and trends, Springer Science+Business Media, LLC, 2008.

[3] M.T. Goodrich and R. Tamassia, Introduction to Computer Security, Pearson, 2011

[4] W. Stallings, Network and internetwork security, Prentice Hall.

[5] D.R. Stinson, Cryptography: theory and practice, CRC Press.

Index