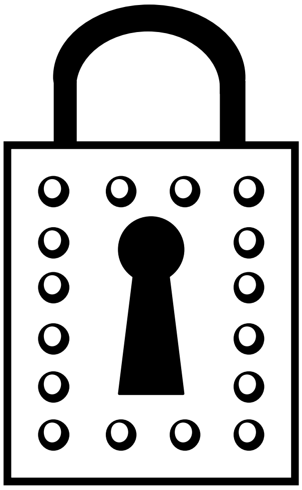
Secure Communication Concepts Explained Simply



A concise, self-teachable training course for people who want to understand the concepts of secure communications but don't need to know the details .

CiaranMcHale.com — Complexity explained simply

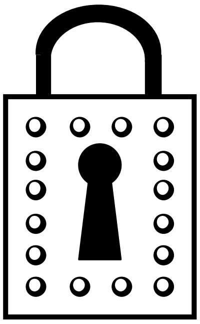
Copyright License

About the Author

Ciaran McHale has a Ph.D. in computer science from Trinity College Dublin. He has been working for IONA Technologies (www.iona.com) since 1995, where he is a principal consultant. His primary talent is the ability to digest complex ideas and re-explain them in simpler ways. He applies this talent to subjects that stir his passion, such as multi-threading, distributed middleware, code generation, configuration-file parsers, and writing training courses. You can find details of some of his work at his personal web site: www.CiaranMcHale.com. You can email him at Ciaran@CiaranMcHale.com.

Table of Contents

Purpose of this Course



CiaranMcHale.com
Complexity explained simply

License

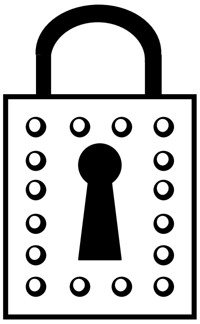
Purpose of this course

- Secure communications is becoming more common in computer systems
  - Between web browsers and websites
  - In distributed middleware (remote procedure call, CORBA, web services, …)
- Some people just want to learn "System X":
  - But System X provides support for secure communications
  - So they need to learn enough about secure communications to be able to configure and administer System X
- This training course:
  - Is aimed at such people
  - Explains the concepts (but not the details) of secure communications
  - Is relatively short (just a few hours of material)

Part I

Cryptography

Introduction to Cryptographic Terminology



CiaranMcHale.com
Complexity explained simply

License

Cryptography

- The term *cryptography* has two parts: *crypt* and *-graphy*
- The word *crypt* comes from the Greek word *kryptos*
  - Means hidden or covered
  - A *crypt* is an underground burial place or a secret meeting place
  - The word *cryptic* means "difficult to understand", that is, a "hidden meaning"
- The *-graphy* suffix denotes a process or science for drawing, writing, representing, describing and so on
  - Biography, choreography, geography, photography, typography, …
- So, cryptography is the science of hidden writing
  - To *encrypt:* to turn a plaintext message into a hidden message
  - To *decrypt:* to turn a hidden message back into a plaintext message

Cipher

- The Arabic number system had some important innovations:
  - Arithmetic is much simpler than arithmetic with Roman numbers
  - The concept of zero (*sifr* in Arabic) has two meanings:
    - It denotes "nothing"
    - It denotes an order of magnitude (10, 100, 1000, …)
- Initially, Europeans were confused by the concept of zero:
  - So *cipher* (*sifr*) was used to refer to something that was a mystery
  - The word *cipher* evolved to mean the deliberate hiding of meaning
- So, *cipher* and *cryptography* are almost synonyms
  - To *encipher* means to *encrypt*
  - To *decipher* means to *decrypt*

Cipher (cont')

- A *cipher* is an algorithm that enciphers and deciphers text
  - Many ciphers take a secret *key* that controls the algorithm
- Example of a (very simple to break) cipher:
  - Algorithm is: rotate each letter "N" places
  - "N" is the secret *key*
    - If "N" is 1 then A → B, B → C, …, Y → Z, Z → A
    - If "N" is 2 then A → C, B → D, …, Y → A, Z → B
- Knowing the cipher is not enough to decode a message
  - You also need to know the key that was used to encode the message

Plaintext and ciphertext

- The term *plaintext* means a readable message:
  - The message does *not* have to be text-based
  - It might be a graphic file or an audio file instead
- Conversely, *ciphertext* means an encrypted message

Security though obscurity

- Security through obscurity:
  - Develop your own cipher (probably with a hardcoded key)
  - Nobody else knows your cipher's algorithm (so it is obscure)
  - You mistakenly think your secrets are safe
- The flaw in is that there are always people smarter than you
  - Smarter people are likely to find flaws in your cipher
  - So they can decode all your secret messages

Well known ciphers

There is a better approach…

- When somebody invents a cipher, he publishes the details:
  - Mathematicians around the world test the cipher for flaws
  - If no flaws can be found then everybody has confidence in the cipher
- A *strong* (that is, good) cipher can be broken only by trying every single possible key value
  - If there, say, 10^{70} possible keys then this approach might take thousands of years of computer time
- All you need to do is:
  - Pick a key (at random) to use with the cipher
  - Keep the key secret

What about Moore's Law?

- In 1965, Gordon Moore (co-founder of Intel) made an observation:
  - Advances in technology mean you can put twice as many transistors onto a chip every 18 months
  - This observation has remained true for over 40 years
- Doubling the transistors usually means doubling the computational power
  - In 15 years time, computers will be 1000 faster than they are today
  - A cipher that takes 1000 years to crack today will take only one year to crack in 15 years' time
- There is no need to panic, because:
  - Most of today's secret messages will be worthless in 15 years' time (so it will not be worthwhile for somebody to crack them)
  - Better ciphers will be developed within 15 years

Key length

- Many ciphers consist of:
  - A well known algorithm, and…
  - A *key* (a number used to prime the algorithm)
- Example of an easy-to-break cipher:
  - Algorithm: rotate each letter "N" places
  - Key: a value in the range 1 to 26
- *Key length* is the number of bits used to represent a key
  - For example, a key length of 128 implies (at most) 2^{128} possible values
  - A value in the range 1 to 26 can be represented in 5 bits (2^5 is 32)
- There is a tradeoff. Longer keys:
  - Make the cipher more secure with only a little extra overhead (which is good)
  - Require more storage space and transmission bandwidth (which is bad)

Key length (cont’)

- Some countries impose legal restrictions on key lengths
  - The term “export cipher” refers to a cipher used with a short key
- The intention is as follows:
  - Strong encryption can be used by the military
  - We do not want to allow strong encryption technologies to be used by foreign militaries (possible enemies)
  - So, we allow only weaker encryption to be exported
- Export restrictions on encryption make international e-commerce more difficult

Misplaced concern about public ciphers

- “Won’t mathematicians working for the <such-and-such> government keep silent about flaws so they can decode your secret messages?”
- No, because:
  - A flaw is likely to be spotted by several people in different countries
    - So keeping silent about flaws just lets somebody else get the credit
  - E-commerce (which is *huge*) cannot work without reliable ciphers
    - E-commerce is not just buying books from Amazon
    - It is also online banking
    - And stock trading
    - And business-to-business transactions
  - E-commerce is driving advances in cryptography much more than the <such-and-such> government’s attempts at spying

Checksums

- When data is being transmitted, it might become corrupted
  - For example, there might be noise on the transmission line
- A checksum is a way to detect accidental corruption. Example algorithm:

```
int calculateChecksum(char* data, int size)
{
    int result = 0;
    for (int i = 0; i < size; i++) {
        result += (unsigned int)data[i];
    }
    return result;
}
```

- Sender transmits data plus the checksum value
  - Receiver calculates checksum value for received data
  - Compares this to the transmitted checksum

Message Authentication Code (MAC)

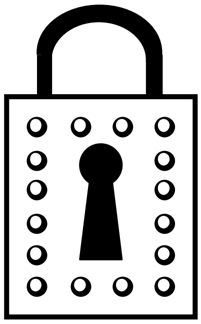
- A checksum is not infallible, but it is a useful check
- A checksum can guard against *accidental* corruption
  - But is too simplistic to guard against deliberate corruption
- A *message authentication code* (MAC) is a kind of checksum:
  - The checksum value is encoded in, say, 20 bytes instead of just 4
  - And it is encrypted
  - These properties make a MAC unfeasibly difficult to deliberately fake
- A MAC is one of several ingredients used to ensure secure communication:
  - A cipher ensures nobody else can understand a secret message
  - A MAC ensures nobody can modify a secret message

8. Summary

Summary

- This chapter has introduced some basic terminology:
  - Cryptography, cipher
  - Encrypt = encipher; decrypt = decipher
  - Plaintext and ciphertext
  - Key length
  - Message authentication code (MAC) is an encrypted checksum
    - Used to detect tampering of messages
- Also explained:
  - Why security through obscurity is a bad idea
  - Well known ciphers tested by mathematicians worldwide are better
  - Lots of people can rely on the same cipher; but they use different (secret) keys

Symmetric and Asymmetric Ciphers



CiaranMcHale.com
Complexity explained simply

License

1. Symmetric Ciphers

Symmetric ciphers

- A symmetric cipher is one in which you can:
  - Encrypt a plaintext message with a secret key
  - Decrypt a ciphertext message with the *same* secret key
- Encrypting a message twice produces the original message
- There are good symmetric ciphers that:
  - Are fast, that is, require an acceptable amount of CPU time to encrypt/decrypt
  - Are strong, that is, take thousands of years to crack
  - Are useful for encrypting files on your computer (you are the only person who needs to know the secret key)

Limitations of symmetric ciphers

- Symmetric ciphers have a significant limitation:
  - Makes them unsuitable for communication with people in remote locations
- How can you negotiate a secret key with the remote party?
  - A telephone call containing the secret key might be intercepted
  - Same for postal mail or email messages that contain a secret key
  - Sending the secret key through a courier means you have to trust the courier. Could he be bribed?
- You have to keep track of multiple secret keys:
  - One secret key for messages to Alice
  - A different secret key for messages to Bob. And so on…
  - In general, a group of N people needs $O(N^2)$ secret keys
- *Asymmetric* ciphers addresses this limitation

2. Asymmetric Ciphers

Asymmetric ciphers and public key cryptography

- Symmetric ciphers were used for thousands of years
  - In the 1970s, Ralph Merkle developed an asymmetric cipher
  - See www.merkle.com for his PhD thesis (of historical interest)
  - Since then, several better asymmetric ciphers have been developed
- An asymmetric cipher uses two keys:
  - A message encrypted with key1 can be decrypted only with key2
  - A message encrypted with key2 can be decrypted only with key1
- Knowing one key does not help you guess the other key
- *Public key cryptography* is another name for asymmetric ciphers
  - One key is called the *public key*
  - The other is called the *private key*

Uses for public key cryptography

- I put my public key on, say, my business card or website
- Use 1:
  - To securely send me a message, encrypt it with my public key
  - Only I can decrypt the message (with my private key)
- Use 2:
  - I make some (compiled) software available from my website
  - I make a checksum of the software files and encrypt the checksum with my private key (this is called *signing*)
  - You download the software and the signed checksum
  - To verify that the software comes from me (rather than a hacker):
    - You calculate a checksum for the software you downloaded
    - You use my public key to decode my checksum, and compare it to your checksum
    - If the checksums match then you know the software is genuine

Combining symmetric and asymmetric ciphers

- A serious limitation of asymmetric ciphers:
  - They can use 100–1000 times more CPU time than symmetric ciphers
- Solution: use a two-step approach for remote communication:
  - Use an asymmetric cipher to securely communicate a private key for use with a symmetric cipher
  - Then switch over to using the symmetric cipher with the agreed-upon private key
- SSL uses a (more elaborate) two-step approach:
  - Initial handshaking is slow due to use of an asymmetric cipher
  - Then communication gets much faster due to use of a symmetric cipher

Securely storing the private key

- I can advertise my public key widely. However…
- I *must* keep my private key private
  - Otherwise, somebody else could pretend to be me
- Advice for storing your private key:
  - Store it in a file on your computer
  - Ensure nobody else can read the file:
    - Example: UNIX file permissions
    - Always lock your computer when you leave your office
    - Be careful who has access to backup disks of your computer
  - Also, encrypt the private key using a *pass phrase* (password) known only to you as the encryption key

A limitation of asymmetric ciphers

- Anyone can create a public-private key pair
  - You just need to run a software utility to create the key pair
  - There are proprietary and open-source utilities, such as OpenSSL
- This creates a problem:
  - A public key enables you to securely communicate with whoever has the corresponding private key
  - The person with the private key *claims* to be, say, Amazon.com
  - How can you be sure?
- The solution is called a *certificate authority* (CA)

3. Certificate Authority (CA)

Driving license authority

- Scenario: you want a driving license to use as a form of identification
  - You go to the *driving license authority* (DLA) building
  - You must prove your identity to the DLA
    - You might use your passport, recent utility bills, and so on
  - The DLA gives you a (difficult-to-forge) driving license document
    - Laminated card containing your photograph, age, height, eye color
    - Start and end validity dates
    - Also contains the DLA logo
- A driving license works as a form of identification because:
  - People can verify that details on the driving license match you
  - Lots of people trust the DLA (can't be bribed to give out fake ids)
- A *certificate authority* (CA) serves a role similar to a DLA

Certificate Authority

- Scenario: you want people to have faith in your public-private key pair, so you can use it as a form of identification
  - You create a public-private key pair yourself
    - This is called a *certificate signing request* (CSR)
  - You go to a *certificate authority* (CA) building
  - You must prove your identity to the CA:
    - Passport, driving license, recent utility bills, and so on
  - The CA gives you an *X509 certificate* (a particular standard)
    - In other words, the CA *signs* your CSR
  - The certificate specifies:
    - Your public key
    - Your details (name, website address, …)
    - Name of the CA
    - A checksum for the certificate, signed by the CA's private key
    - Start and end validity dates

> The purpose of a certificate is to securely associate your details with your public key

Certificate Authority (cont')

- An X509 certificate works as a form of identification because:
  - Lots of software use a library to recognize (check) X509 certificates
  - Software is bundled with copies of public keys for popular CAs
    → the software can verify the signed checksum on the certificate
  - Lots of people trust the CA (can't be bribed to give out fake certificates)
- Example: online shopping with your web browser:
  - When you click on the *Pay Now* button, your browser visits a web page starting with https:// (the "s" denotes a secure web page)
  - The web browser downloads the website's X509 certificate
  - The web browser checks:
    - Has the certificate been signed by a CA trusted by the web browser?
    - Is the name of the website contained in the certificate's details?
  - If the checks are okay then you know the website is not an imposter

Practical details of CAs

- How do you know you can you trust a CA?
  - You just have to trust them
  - Just like you have to trust the driving license authority
  - A CA "self signs" its own X509 certificate
- Can anyone set themselves up as a CA?
  - Yes, but it might take a lot of effort to convince web browser companies to bundle your public key in their products
  - Users can import extra CA certificates into their browser
    (but this is "too much bother" for most users)
- It is common for an organization to have its own *internal* CA:
  - Saves money: don't have to pay an external CA for certificates
  - The internal CA may not be trusted outside of the organization, but that is not a problem for internally-deployed applications

4. Summary

17

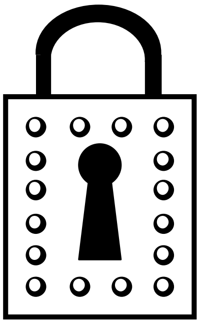
Summary

- Symmetric ciphers are fast, but have some limitations:
  - Difficult to *securely* agree on a secret key with a remote party
  - You need a separate secret key for each person you communicate with
- Asymmetric ciphers are slow but:
  - Enable secure communication without prior agreement of a secret key
  - Make it possible to electronically “sign” a document to prove it came from you
- SSL:
  - Uses an asymmetric cipher initially to agree on a secret key
  - Then switches over to a symmetric cipher (for speed)

Summary (cont')

- X509 is a standard for a security certificate
  - A form of identification, just like a driving license or passport
  - The certificate contains a *public key*, so people can send you messages securely
  - You keep the corresponding *private key* a secret
  - The certificate is signed by a certificate authority
- A certificate authority (CA) is a trusted organization that can sign your X509 certificate
  - There are well known CAs that are trusted worldwide
  - An organization can have its own CA for internally-deployed applications

Goals of Secure Communication



CiaranMcHale.com
Complexity explained simply

License

Goals of cryptographic communication

There are several goals of cipher-based communication…

- Confidentiality
  - This is provided by using a strong cipher and secret key
- Authentication
  - This is provided by the use of digital certificates, such as X509
- Integrity (also known as *message authentication*)
  - This is provided by a MAC (message authentication code)
- Non-repudiation (discussed on the next slide)

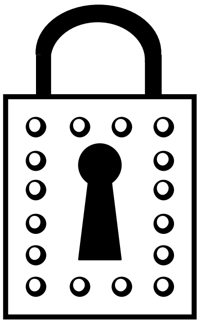
Goals of cryptographic communication (cont')

- Non-repudiation
  - *Repudiate* means to deny, disown or reject as untrue.
  - *Non-repudiation* means the ability to prove whether or not somebody sent a message
- Example of the need for non-repudiation:
  - An investor thinks the IBM share price will drop
  - He tells his stockbroker to sell his IBM shares
  - Soon afterwards, IBM shares increase in value
  - The investor pretends he never told his stockbroker to sell his shares
  - The stockbroker uses non-repudiation to prove the investor is lying

Authorization

- Authorization is an important goal in security
- However, authorization is distinct from cryptography:
  - It is *not* provided by cryptography
  - However, authorization *does* rely upon authentication

SSL and TLS



CiaranMcHale.com

Complexity explained simply

License

What are SSL and TLS?

- In 1994, the world wide web was new and immature:
  - Web browsers and web servers sent only plaintext messages
  - Unsafe to use your credit-card to buy something from a website
- Netscape designed SSL to provide encrypted communication for the web
  - SSL stands for *secure sockets layer*
  - Netscape had a patent for SSL, but made SSL open
- SSL matured quickly with the help of the web community:
  - In 1995, SSL 3.0 was released
  - In 1996, Netscape handed over responsibility for SSL to the IETF
    - IETF = *Internet Engineering Task Force*
    - IEFT is an international standards organization
    - IETF renamed the next version of SSL to TLS 1.0
  - You can think of TLS 1.0 as being SSL 3.1

An extra layer in the protocol stack

- An application-level protocol normally talks directly to TCP/IP
- SSL was designed so:
  - It could be used with HTTP (an application-level protocol)
  - It could be used with other application-level protocols too
- For example:
  - CORBA is a remote procedure call (RPC) mechanism
  - The insecure CORBA protocol is called IIOP
  - The secure CORBA protocol is called IIOP/TLS

Simplified overview of SSL/TLS

- Recall:
  - Symmetric ciphers:
    - Are fast
    - But how do the two parties *securely* agree on a secret key?
  - Asymmetric ciphers have the opposite properties:
    - Are 100–1000 times slower than symmetric ciphers
    - Can safely exchange public keys, even if other people overhear
- Slightly simplified explanation of how SSL works:
  - SSL uses both symmetric and asymmetric ciphers
  - Uses an asymmetric cipher to securely communicate a private key for use with a symmetric cipher
  - Then switches over to using the symmetric cipher with the agreed-upon private key

Simplified overview of SSL/TLS (cont')

- Actually, SSL uses six secret keys rather than just one:
  - Three are for client-generated messages
  - And three are for server-generated messages
  - The use of multiple keys makes life even harder for hackers
- Each group of three secret keys consists of:
  - A key used by the encryption cipher
  - A key used by the MAC cipher
  - A key used to initialize the encryption cipher

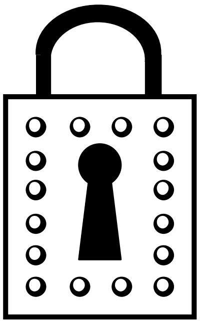
SSL/TLS supports many ciphers

- SSL/TLS uses one of each of the following:
  - A symmetric cipher
  - An asymmetric cipher
  - A MAC cipher
- But there are many competing ciphers in each category
  - Which one should be used?
- During the initial SSL/TLS handshaking:
  - Client sends a list of ciphers it understands to the server
  - The server picks one from each category and notifies client of its choice
- Benefits:
  - SSL/TLS can adapt whenever better ciphers are developed in the future
  - SSL/TLS can adapt to legal restrictions on ciphers in some countries

Summary

- TLS is the new name for SSL
  - TLS 1.0 = SSL 3.1
- SSL was first used to secure communication via HTTP
  - But can be used to secure other protocols
- SSL uses both symmetric and asymmetric ciphers:
  - Uses an asymmetric cipher to securely communicate a private key for use with a symmetric cipher
  - Then switches over to using the symmetric cipher with agreed-upon private keys
- SSL is *not* hardcoded to use a particular set of
  - Client and server negotiate on which set of ciphers to use
  - SSL can evolve to support newer, better ciphers when they are developed

Miscellaneous Terminology



CiaranMcHale.com
Complexity explained simply

1

License

Commonly used names

- Some names often appear in books and articles about cryptography
- *Alice* and *Bob* are two people who want to communicate securely
- *Black hat* wants to intercept and decode or modify messages sent between Alice and Bob
- The term *black hat* comes from old cowboy movies:
  - By convention, the hero wore a white hat
  - And the villain wore a black hat
- The term *white hat* refers to somebody who fixes security loopholes in computer systems

Principal and credentials

- *Principal* is an identity
  - Think of it as being a username
- *Credentials* is the data that prove the *principal* (identity)
  - This might be a password (to go with the username) or an X509 certificate
- Human analogy:
  - "I am Dr. John Smith" (that is my *principal*)
  - "Here is my passport (or driving license)" to prove I am who I say I am (my *credentials*)
  - "Here is my license to practice medicine" to prove that I am a doctor (another set of *credentials*)

PK, PKI and IETF

- Recall:
  - *Public and private key cipher* is another name for *asymmetric cipher*
  - *Public and private key* is often abbreviated to *public key*
- *PK* is an acronym for *public key*
- *PKI* is an acronym for *public key infrastructure*
  - Supporting infrastructure required to use public keys
  - It consists of:
    - Certificate authority (CA) software
    - Procedures used verify a user's identity so the CA is willing to sign the user's certificate
- *IETF* is an acronym for the *Internet Engineering Task Force*
  - An organization that defines standards for Internet-related technologies

RSA, VeriSign and PKCS

- RSA is a public-key encryption algorithm
  - Its name is an acronym of the surnames of its inventors (Ron Rivest, Adi Shamir and Leonard Adleman)
  - It was invented in 1977 and is still widely used today
- *RSA Security* was a company set up to promote and exploit cryptographic technologies (including RSA)
  - RSA is now owned by EMC Corporation
- *PKCS* is an acronym for public *key cryptographic standards*
  - A collection of (pseudo-)standards defined by a RSA Security
  - Not officially standards, because they are defined by a company
  - However, several have been adopted by formal standards organizations
- *VeriSign* was a spin-off company from RSA Security
  - It is the largest certificate authority for the Internet

Some well-known ciphers used in SSL and TLS

- Asymmetric ciphers:
  - RSA, Diffie-Hellman, DSA, SRP, PSK
- Symmetric ciphers:
  - RC2, DES, IDEA (used only in old versions of SSL)
  - RC4, Triple DES, AES (also called Rijndael), Camellia
- Cryptographic hash functions:
  - MD2, MD4 (used only in old versions of SSL)
  - MD5, SHA-1
- An SSL *cipher suite* consists of:
  - One asymmetric cipher, plus
  - One symmetric cipher, plus
  - One cryptographic hash function

  It is negotiated during the initial SSL handshaking

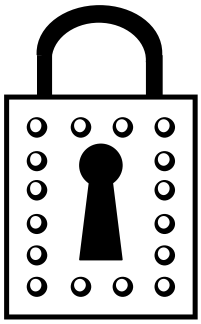
Some other standards

- PKCS#11 is an API used to obtain cryptographic tokens from hardware
  - For example, from a smart card
- PKCS#12 ("`.p12`") is a file format:
  - Used to store private keys with accompanying public key certificates
  - The file is encrypted for security
  - Used widely
- Privacy Enhanced MAIL (PEM)
  - An IETF proposed standard for using public key cryptography in email
  - Not widely deployed
  - PEM ("`.pem`" file extension) is used by OpenSSL
  - OpenSSL can convert between "`.pem`" and "`.p12`" file formats

Part II

Access Control List

Access Control List (ACL)



CiaranMcHale.com
Complexity explained simply

License

Access control list

- An *access control list* (ACL) is an *authorization* mechanism
  - Specifies what access permissions different users have for a resource
  - Examples of resources: a file, a printer, operations on an object in a server application
- Many systems provide ACLs, but the mechanisms vary
- Example using a pseudo-code syntax for an RPC system:

```
user Fred can execute:
    SessionManager.login
    SessionManager.logout
    Session.*
user Mary can execute:
  ...
```

<interface>.<operation>
"*" matches all operation names

Role-based access control

- Problem: some systems have thousands of users:
  - Tedious and error prone to specify ACLs for each user individually
- Solution: role-based access control (RBAC):
  - Assign each user to one or more roles
  - Then write ACLs in terms of roles rather than individual users
- Example using a pseudo-code syntax for an RPC system:

```
user Fred belongs to employee, manager;
user Mary belongs to employee;
user Sam  belongs to customer;
...
role employee can execute: ...
role manager can execute: ...
```

Role-based access control (cont')

- Benefit of role-based access control:
  - There may be thousands or even millions of users
  - But usually only a very small number of roles
  - So ACL maintenance is easy

Prerequisites for authorization

- A prerequisite for access control lists (or any other authorization mechanism) is *authorization*
  - No point in specifying what user Fred can do if we cannot verify whether or not a user *is* Fred
- Question: can a system provide authentication without (the overhead of) secure communications (such as SSL/TLS)?
- Answer:
  - Probably not, because…
  - Authentication is often done via username and password
  - If usernames and passwords are transmitted without encryption then a hacker can easily capture them

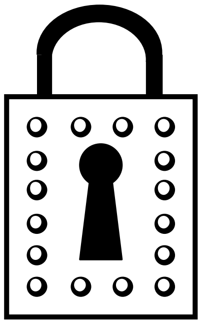
Summary

- Access control lists (ACLs) are widely used for authorization
- Can be tedious to specify an ACL for each of 1000's of users
  - Better to map many users to a small number of *roles*
  - Then define ACLs for the roles
  - This is called role-based access control (RBAC)
- Authentication is a prerequisite for authorization
  - And encryption is a prerequisite for authentication
    (so people cannot snoop on usernames and passwords)

Part III

LDAP

Introduction to LDAP



CiaranMcHale.com
Complexity explained simply

License

What is LDAP?

- LDAP = Lightweight Directory Access Protocol
  - Let's look at each of those words
- A *directory* is a collection of information you can look up to find a person, organization, …
- You have probably encountered many directories:
  - A "telephone directory" book
  - A "directory enquiries" telephone service
  - A directory of sports clubs, embassies, local businesses, …
- UNIX uses the term *directory* in the same way that Windows uses the term *folder*
  - Enables you to look up a file by its name

What is LDAP? (cont')

- LDAP is *lightweight* in comparison to its predecessor (the X.500 directory service)
  - Implemented on top of TCP/IP rather than with the 7-layer OSI stack
  - Omits many operations that were rarely used in X.500
- LDAP is a *protocol,* that is, a specification for how clients communicate with servers
  - You can implement LDAP clients and servers in many programming languages and on many operating systems
- So, LDAP is a *lightweight protocol* that enables clients to *access directory* services.

Relevance of LDAP to security

- Some knowledge of LDAP is useful when working with secure communications
- An X509 certificate contains a *distinguished name*
  - This term comes from LDAP
- Some organizations use LDAP to centralize:
  - Usernames and passwords
  - Public key certificates
  - User → role mappings (for access control lists)

Typical use of LDAP

- Multi-user systems require access to directory information:
  - Operating system: usernames and passwords, user-specific information (home directory, default shell, …)
  - Mail client: usernames and passwords, email addresses
  - Wiki
  - Bug-tracking application
- It is error-prone to update a user's details if each system has its own directory service
- If each system can act as an LDAP client then:
  - You can centralize directory information → easier administration
  - Applications can use LDAP to:
    - Check login details
    - Perform auto-completion of, say, names or email addresses
    - Retrieve user → role mappings for access control lists

Typical use of LDAP (cont')

- LDAP is of limited benefit if you have just one multi-user system
  - The multi-user system might provide its own built-in directory service that is easier to use
- Benefits of LDAP grow quickly as an organization gets several multi-user systems
  - As already discussed, LDAP offers centralized administration
  - LDAP also offers replication and federation (splitting a directory's contents over several, inter-connected LDAP servers)
- Because of this:
  - People with a standalone computer, for example, home users, are unlikely to use LDAP or even know what it means
  - Administrators in large organizations are more likely to be familiar with it

LDAP schemas

- A schema is meta-information:
  - Often written in the syntax of the thing it describes
- Example:
  - A database schema describes the structure of a database:
  - Names of tables
  - Names and types of columns within each table
- LDAP uses schemas:
  - You can define an LDAP schema for the information you want to store
  - The schema syntax is a bit obscure and outside the scope of this course

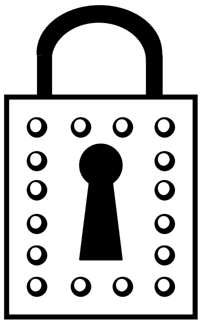
LDAP and databases

- LDAP and databases have some characteristics in common:
  - They can perform searches quickly
  - They have extensible schemas
- However, there are some differences:
  - LDAP assumes that reads are much more frequent than updates
    - In contrast, a database assumes that reads and updates occur with similar frequency
  - LDAP does not support transactions
- However, remember that LDAP is an on-the-wire protocol
  - An LDAP server can use any technology it wants to store its data
  - It might use text files
  - It might use a database
    (but it won't expose the database's transaction capability to clients)

Summary

- LDAP = Lightweight Directory Access Protocol
- LDAP is useful when you have *several* multi-user systems
  - Use LDAP to centralize directory information → easier administration
- LDAP is relevant to security because:
  - An X509 certificate contains an LDAP *distinguished name*
  - LDAP can be used to centralize usernames, passwords, public key certificates and user → role mappings

Organization of Data in LDAP



CiaranMcHale.com
Complexity explained simply

License

LDAP Directory Information Tree (DIT)

- Data in an LDAF server is organized as a hierarchical tree
  - It is usually a tree, but *alias* entries can introduce cyclic loops
  - This tree is called an *LDAP Directory Information Tree* (DIT)
  - Often, *directory information tree* is abbreviated to *directory tree*
- Each entry in the tree can be uniquely addressed by its *distinguished name* (DN):
  - Conceptually similar to /path/to/a/unix/file or C:\path\to\a\windows\file
  - However, there are differences:
    - The separator at each level is a comma (",") rather than "/" or "\"
    - Within a level, there is *name=value* instead of just *name*
    - A distinguished name is written with the most significant piece first, like in a postal address
  - Example of a distinguished name:

    cn=John Smith,ou=staff,dc=example,dc=com

Attribute names

- Let's consider that example of a distinguished name:

  cn=John Smith,ou=staff,dc=example,dc=com
- What are "cn", "ou" and "dc"?
  - They are names of *attributes*
    (similar to Java fields or C++ instance variables)
  - What follows "=" is the value of the specified attribute
- Many attributes have confusingly short names. Examples:
  - cn = common name
  - sn = surname
  - ou = organizational unit
  - dc = domain component, that is, a component in a DNS domain name
- Attribute names are *not* case sensitive
  - Example: "CN", "cn", "cN" and "Cn" are equivalent

Entries, objectClasses and attributes

- Each *entry* in an LDAP server is an object
  - An entry (object) can contain many *attribute-name=value* pairs
  - The *objectClass* attribute specifies the entry's class (that is, its type)
- Each objectClass is defined in an LDAP schema:
  - The LDAP schema language supports single inheritance for classes
  - The definition of an objectClass specifies which attributes are optional and which are mandatory
- The schema definition of an attribute specifies if it can have one value or multiple values:
  - Example of an attribute that has multiple values:

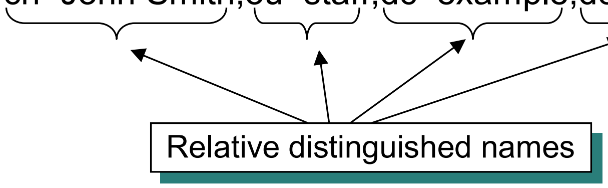
    telephoneNumber: +1 555 967-1432

    telephoneNumber: +1 555 967-5634
  - An entry can have multiple values for its *objectClass* attribute

Relative distinguished name (RDN)

- A relative distinguished name (RDN) is a *attribute-name=value* that identifies an entry at one level in the hierarchy
- An an example, consider the following distinguished name:

  cn=John Smith,ou=staff,dc=example,dc=com



- An LDAP schema does *not* specify that a particular attribute must be used in the RDN
  - Instead, you can use whatever *attribute-name=value* you prefer (as long as it uniquely identifies one entry)
  - If needed for uniqueness, you can use a "+" separated list of *attribute-name=value*
  - Example: cn=John Smith+telephoneNumber= +1 555 967-1432

LDIF Data

- An LDAP server:
  - May store data in whatever format it wants: text files, relational database, …
  - Must be able to import and export data in LDIF format
- LDIF = LDAP Interchange Format
  - It is a text-file format
- There are typically two ways to enter data into an LDAP server:
  - Use a (proprietary or open-source) GUI client that uses the LDAP protocol
  - Use an LDIF file
- Many administrators prefer using LDIF files instead of GUIs

Example LDIF data

```
dn: uid=jsmith,ou=Marketing,dc=example,dc=com
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
uid: jsmith
cn: John Smith
ou: Marketing
# rest of entry deleted for brevity
```

- Notes:
  - Comments lines start with a hash sign (“#”)
  - Blank lines are used to separate entries
  - Attributes are specified as *attribute-name* followed by a colon (“:”) and a space, and then the *value*
  - The *dn* (distinguished name) pseudo-attribute specifies the entry’s location within the directory tree

Suggested reading

- An incomplete but informative online LDAP manual:
  http://www.zytrax.com/books/ldap
- The following book:
  LDAP System Administration by Gerald Carter. O'Reilly, 2003
  - Gives an overview of LDAP
  - Explains how to install and administer OpenLDAP
    (an open-source implementation)