Encryption, Authentication, and Distributed System Security
CS239
Advanced Operating Systems
May26, 1997

Outline

- Introduction
- Encryption
- Authentication
- Using encryption and authentication in distributed systems

Why Is Distributed System Security Different?

- You don't necessarily trust everyone completely
- Being distributed makes it easy for intruders
- Distributed problems are always harder than single system problems

The Untrusted Network

In in most distributed systems you don't trust the network

Yet you rely on it for vital services

So how can you operate safely?

Handling Untrusted Networks

- Hide everything from those who shouldn't see it
  - Encryption
- Make sure everything you get over the net came from a proper source
  - Authentication

Encryption

The basic mechanism to hide your data In plain sight

Various *encryption* algorithms can make data unusable by intruders

Typically, encryption uses a secret *key* known only to legitimate users

Without the key, decrypting the data is computationally infeasible

Encryption Example

M is the *plaintext* (data to be encrypted)

E is the encryption algorithm

$\mathbf{K_e}$ is the key

C is the ciphertext (encrypted data)

$$\mathbf{C = E(M, K_e)}$$

Decrypting the Ciphertext

C is the ciphertext

D is the decryption algorithm

$\mathbf{K_d}$ is the decryption key

$$\mathbf{M = D(C, K_d)}$$

M had better be the same as it was before encryption

Symmetrical Encryption

Many common encryption algorithms are symmetrical

I.e., **E** = **D** and $\mathbf{K_e} = \mathbf{K_d}$

Some important encryption algorithms are asymmetrical, however

Encryption Security Assumptions

Assume that someone trying to break the encryption knows-

- The algorithms **E** and **D**
- Arbitrary amounts of matching plaintext and ciphertext **M** and **C**

But does not know the keys $\mathbf{K_e}$ and $\mathbf{K_d}$

Evaluating Security of Encryption

Given these assumptions, and a new piece of ciphertext $\mathbf{C_n}$, how hard is it to discover $\mathbf{M_n}$?

Either by figuring out $\mathbf{K_d}$ or some other method

What if $\mathbf{M_n}$ matches one of the known pieces of plaintext?

Practical Security of Encryption

Most encryption algorithms can be broken

If by no other means, exhaustively trying all possible keys

Goal is to make breaking them too expensive to bother

How do we protect our encryption?

Key Issues in Encryption

Security often depends on length of key
- Longer keys give better security
- But slows down encryption

The more data sent with a given key, the greater the chance of compromise

The more data sent with a given key, the greater the value of deducing it

One-Time Pads

Theoretically unbreakable security

A symmetrical encryption system

Use one bit of key for each bit of plaintext

Never reuse any key bits

Generate key bits truly randomly

Advantages of One-Time Pads

- Proved secure (in information theoretic sense)
- Encryption algorithm is cheap
  - XOR message with key
- Required procedures for proper use well understood

Problems With One-Time Pads

- They burn keys like crazy
- Need to keep key usage in sync
- If the keys aren’t truly random, patterns can be deduced in the bits
- Distribution of pads

Practical Symmetrical Encryption Methods

Use various manipulations to conceal data using secret key

DES is best known example

Encryption for Distributed Systems

Can protect secrecy of data while on insecure links

Can also prevent modification and many forms of fabrication attacks

But keys are a tricky issue

Encryption Keys and Distributed System Security

To gain benefit from encryption, communicating entities must share a key

Each separate set of entities needs at least one different key

And they must be changed periodically

How do you securely distribute keys?

Problems of Key Distribution

- Key must be kept secret
- Key must be generated by trusted authority
- Must be sure key matches intended use
- Must be sure keys aren't improperly reused
- Must be quick and automatic

Key Distribution Schemes

- Manual distribution by one party
- Manual distribution by third party
- Use existing key to send new key
- Key servers

Key Servers

Trusted third party that can provide good keys on demand on line

Must be a program, for full utility

Typically on a separate machine

Tremendous care must be taken to ensure secure communications with the key server

Asymmetrical Cryptography

Key distribution is a major problem for symmetric cryptography

Can we design a system where we can distribute something that isn't secret?

Yes, using asymmetrical cryptography

Different keys for encryption and decryption

Public Key Cryptography

An asymmetrical form of cryptography

Different keys for encrypt/decrypt

$$C = P(M, K_P)$$

$$M = P(C, K_S)$$

Either key can encrypt

Important insight - only one key need be secret!

The Public Key

Everyone has one key that is known by all - it's not a secret

For user X,

$\mathbf{K_{P,X}}$

Encrypt all messages to X using the public key

Since only owner knows private key, only owner can decrypt

The Private Key

The secret part of the key pair

Only known to owner X

$\mathbf{K_{S,X}}$

Used to decrypt all messages sent to that user

Public Key Cryptography For Encryption

To send an encrypted message to Y,

$\mathbf{C = P(M, K_{P,Y})}$

To decrypt it,

$\mathbf{M = P(M,K_{S,Y})}$

So Is Public Key Cryptography Always the Answer?

Only the public key is sent to anyone, and it's not secret

So, no key distribution problem?

Wrong!

How do I know a particular public key really belongs to my partner?

Answer - Key servers again

Authentication

If a system supports more than one user, it must be able to tell who's doing what

I.e., all requests to the system must be tagged with user identity

Authentication is required to assure system that the tags are valid

Especially important for distributed systems

Authentication Mechanisms

- Passwords
- Cryptographic authentication
- Other authentication mechanisms

Passwords

A fundamental authentication mechanism

A user proves his identity by supplying a secret

Either at login or other critical time

The secret is the password

Password Security

- Password selection
- Password storage and handling
- Password aging

Selecting a Password

Desirable characteristics include:

- Unguessable
- Easy to remember (and type)
- Not in a dictionary
- Too long to search exhaustively

Password Storage and Handling

Passwords are secrets, so their security depends on careful handling

But seemingly the system must store the password

To compare when users log in

If system storage is compromised, so is all authentication

Securely Storing Passwords

Store only in encrypted form

To check a password, encrypt it and compare to the encrypted version

Encrypted version can be stored in a file

But there are tricky issues

Tricky Issues in Storing Encrypted Passwords

- What do I encrypt them with?
  - If I use single key to encrypt them all, what if the key is compromised?
  - That key must be stored in the system
- What if two people choose the same password?

Example: The Unix Password File

Each password has an associated *salt*

Unix encrypts a block of zeros

- Key built from password plus 12-bit salt
- Encryption done with DES

Stored information = E(zeros, salt + password)

To check password, repeat operations

How Does This Help the Problems?

- No single key for encryption
  - So can't crack that key
  - And needn't ever store it
- Each encryption (probably) performed with a different key
  - So two people with the same password have different encrypted versions

Does This Solve the Problem?

Not entirely

Passwords exist in plaintext in process checking them

Passwords may travel over communications lines in plaintext

- Especially for remote logins
- Or logins over modems

Problems With Passwords

- People choose bad ones
- People forget them
- People reuse them
- People rarely change them

How To Deal With Bad Passwords

- Educate users so they choose good ones
- Automatic password generation
- Check when changed
- Periodically run automated cracker

Any solution must balance user needs, password security, and resources

Public Key Cryptography For Authentication

Public key cryptography can also be used for authentication

Use it the other way around

Authenicating With Public Key Methods

Originator X of message encrypts it with his private key

$$C = P(M, K_{S,X})$$

Receiver Y decrypts with the public key

$$M = P(C, K_{P,Y})$$

Since only owner knows private key, only he could create proper C

Other Authentication Mechanisms

- Challenge/response
- Smartcards
- Other special hardware
- Detection of personal characteristics

All have some drawbacks

Some are combined with passwords

Authentication For Distributed Systems

When a message comes in over the net, how do you tell who sent it?

Generally with some form of *digital signature*

Must be unique to signing user

And also unique to the message

Digital Signatures

A *digital signature* is a guarantee that an electronic document was created by a particular individual

Basic mechanism for authentication

Vital for electronic commerce, secure electronic mail, etc.

S = signature(M)

Desirable Properties of Digital Signatures

- Easy to generate & verify
- Non-forgeable
- Unique (or close to)
- Non-repudiable
- Storable

Providing Digital Signatures

Encryption with a shared secret key has some of these properties

- Encrypt entire message
- Check signature by decrypting

$$S = E(M, K_e)$$

But normal encryption has problems for digital signatures

Problems of Using Encryption for Digital Signatures

- Both parties can create same message
  - With same signature
- One key per pair of users required
- Signature is as large as message
  - Poor storage properties
- Hard to handle multiple signatures per message

Public Key Cryptography and Digital Signatures

User X wants to sign a message M sent to user Y

Calculate a characteristic Z of message M (checksum or something similar)

$S = P(Z, K_{e,X})$

Send both M and S to Y

Checking a Public Key Digital Signature

Y calculates the characteristic $\mathbf{Z_M}$ of **M**

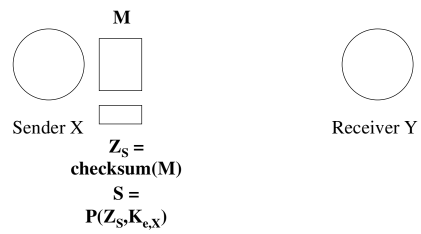
Then Y checks the signature

$Z_S = P(S, K_{d,X})$

If $\mathbf{Z_M == Z_S}$, the signature is valid

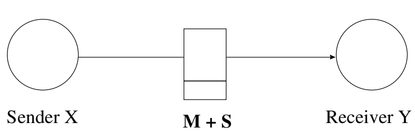
Public Key Digital Signature Diagram, Step 1

Sign the message



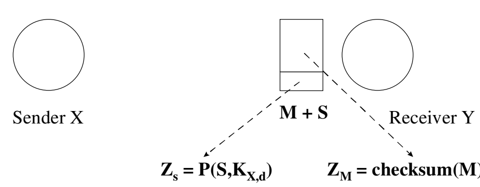
Public Key Digital Signature Diagram, Step 2

Send the message plus the signature



Public Key Digital Signature Diagram, Step 3

Check the signature



If $Z_S == Z_M$, the signature is valid

How Does This Scheme Handle Various Attacks?

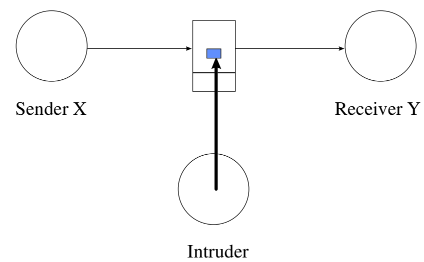
What if an intruder changes the message?

What if someone replays a message?

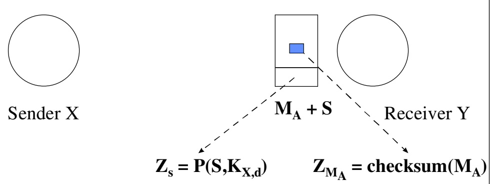
What if the sender denies a message he sent?

What if the receiver tries to alter the message?

Intruder Alteration Diagram

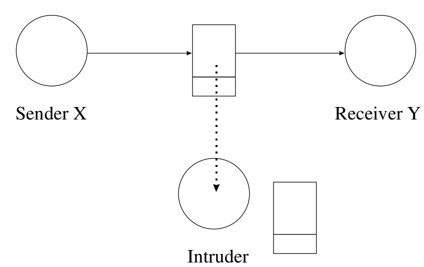


Discovering the Alteration

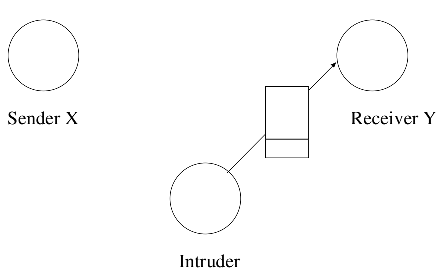


Z_S does not equal Z_{M_A}, so the signature is invalid

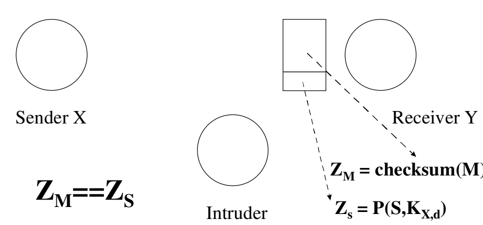
Replay Diagram



Replay Occurs



Checking the Replayed Message's Signature



How To Handle This Replay?

- Sequence numbers in messages
- Challenge/response to sender
- Timestamp messages and discard “old” ones
- Don’t worry about it

Major Challenge in Public Key Cryptography

How do I find out someone’s public key?

If not done securely, the system is totally compromised

Must also be efficient

And how do I securely store and manage public keys?

Authentication Servers

Like key servers, trusted third parties

An authentication server can produce a ticket that guarantees the identity of a user

Generally tickets expire

Also used with symmetric cryptography

Kerberos

Most popular authentication server

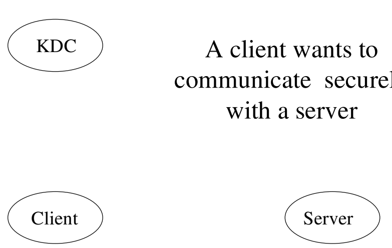
Uses symmetric cryptography

Servers are trusted by all parties

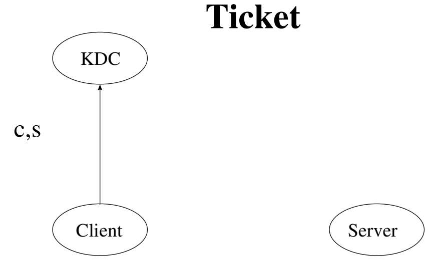
Issues tickets that provide secure communications between clients and servers

Tickets have a lifetime, then expire

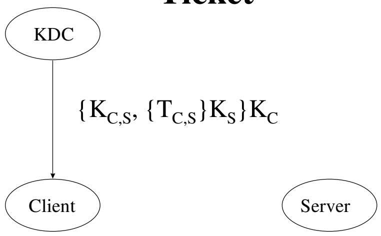
Kerberos in Action



The Client Asks Kerberos for a Ticket



Kerberos Gives the Client a Ticket



What's Going On Here?

What is in this message?

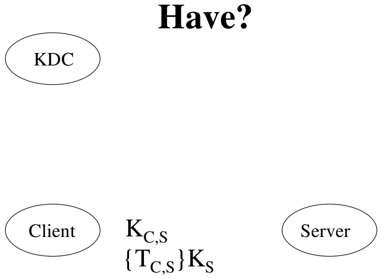
$T_{C,S}$ is the ticket that allows the client to communicate with the server

It's encrypted with K_S (so only the server can read it)

Message contains a new key $K_{C,S}$

Entire message encrypted in C's key

Now What Does the Client Have?



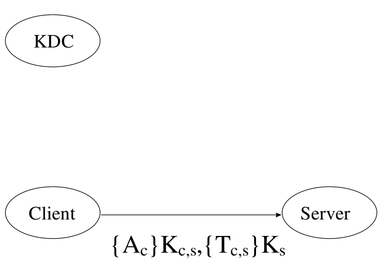
Why the Extra Key $K_{C,S}$?

For authentication purposes

It's also contained within the ticket

Server can authenticate himself to client using that key

Client Sends Ticket to Server



What Does the Client Send?

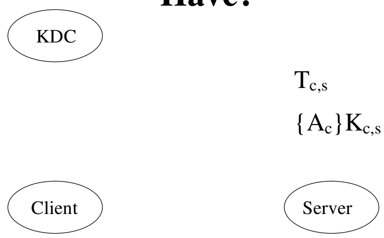
Sends encrypted ticket from Kerberos server

Which only server can read

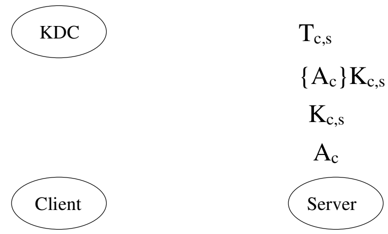
Also sends authenticator A_C in session key $K_{C,S}$

Server gets $K_{C,S}$ from ticket, sends back altered version encrypted with $K_{C,S}$

Now What Does the Server Have?

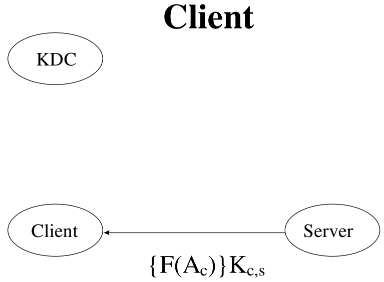


And Also . . .



Since the ticket contains the session key

Server Authenticates Itself to Client



So Who Knows What Now?

- The server knows the client is who he claimed to be
- The client knows the server is who he claimed to be
- Both have a private session key
  - Kerberos server knows it too, but they trust it

Firewalls

A program to allow selective access to the network

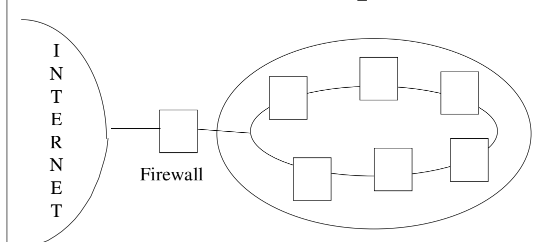
In both directions

Typically, firewalls protect entire networks

They must examine everything that tries to pass into the protected domain

Only authorized transmissions permitted

Firewall Example



What Do Firewalls Do Well?

Prevent intruders from accessing machines on your network

Prevent your users from inadvertently compromising security

What Do Firewalls Do Badly?

Prevent many forms of legitimate access

May get in the way of other forms of security

Often, there's no further security behind the firewall

So if it fails . . .

Firewalls, Eskimos, Polar Bears, and Igloos

Firewalls are like igloos to polar bears - hard, crunchy outsides; soft, chewy insides

If the polar bear breaks through your firewall, expect him to chew up your users