Security 101
Security Core Concepts...
Where Do You Start?

Denise Hatzidakis

Perficient, Inc.

Director, Chief Technologist

denise.hatzidakis@perficient.com

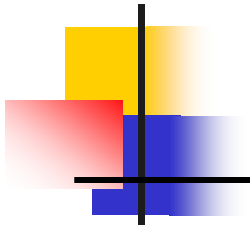
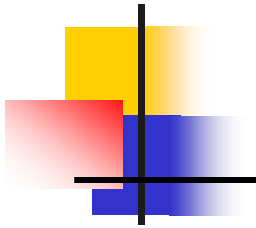


Table of contents



Authentication

"Identifying Users/Entities"

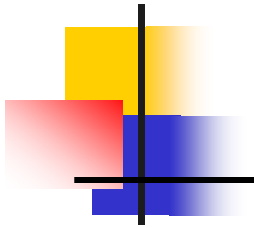
Prevent Impersonation

ISSUES:

- SW *vs*. HW
- Multi-factor authentication
  - What you know
  - What you have
  - What you are
- **Scalability** (ID/key mgmt.)
- **Portability/Mobility**
- **Linking authentication policy to business policy**
- **Single sign-on**

Examples

- Logon IDs and Passwords
- Pass Tickets
- Digital Certificates and Private Keys
- Smart Cards & PINs
- Tokens/fobs (SecureID, USB port fob, . . .)
- Biometric Devices



Access Control

"Selectively Granting/Denying Access to Resources"

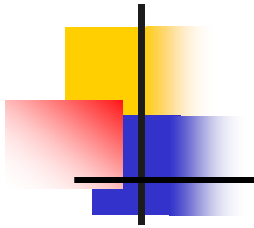
a.k.a. "Authorization"

ISSUES:

- **Granular control** over heterogeneous resources
- **Groups/roles simplify administration**
- **Single, comprehensive policy** versus multiple, disparate approaches
- Control access while **maintaining high availability/performance**
- Ability to **tightly link business policy** to authorization policy

Examples

- Access Control Lists
- Roles
- Security Labels
- Physical Barriers (Locks, Guards)
- Firewalls
- Split Control



Confidentiality

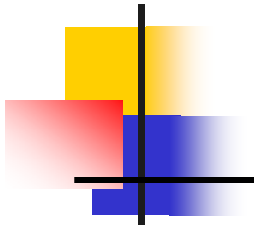
"Preventing Unauthorized Disclosure of Stored and Transmitted Data"

ISSUES:

- **Choice of protocol** (SSL, IPSEC, . . .)
- **Choice of strength** (key/algorithm)
- **Performance** (hardware versus software)
- **Security of keys** (hardware versus software)
- **Scalability** (key management)
- **Ease of implementing, ease of use**

Examples

- Encryption (based on selected algorithms, *e.g.* DES)
- Data masking



Data Integrity

"Detecting Unauthorized Modification of Stored and Transmitted Data"

ISSUES:

- **Choice of protocol** (SSL, IPSEC, . . .)
- **Choice of strength** (key/algorithm)
- **Performance** (hardware versus software)
- **Security of K** (hardware versus software)
- **Scalability** (key management)
- **Ease of implementing, ease of use**

Same as "Confidentiality"

Examples

- Checksums, CRCs, . . .
- Message integrity codes
- Hashes
- Digital signatures
- Anti-virus programs

Non-Repudiation

"Proof of:

- Origin
- Receipt
- Transmission

....

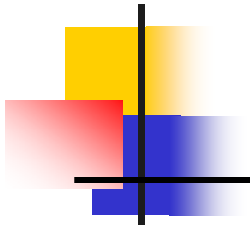
Of a message"

ISSUES:

- Digital signature being written into **laws**
- **Security of keys** (hardware versus software)
- **Scalability** (key management)
- **Integrating** into existing middleware/applications

Examples

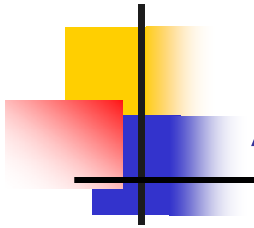
- Message Authentication Codes (MACs)
- Digital signatures
- Audit
- Trusted Time



More on Authentication Types

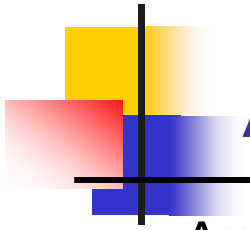
"Advanced" Authentication

- Strong Authentication
  - *a.k.a.* "better than passwords"
- Multi-factor authentication
  - Two or more pieces of authentication data
  - Usually something you know and something you posses (*e.g.*, a certificate, secureID card, *etc*.)
  - If something you posses this dramatically improves security since physical object must be stolen or hacked – changes nature of crime
- Step up authentication
  - Asked to authenticate strongly under certain circumstances when accessing sensitive resources.
  - Rarely works as easily as the sales documents say



Authentication Concerns

- Detection of compromise
  - How easily can one detect compromise?
    - A stolen token is usually easy to detect. What about a stolen password?
- Management difficulty
  - Creating, distributing, and replacing authentication data
- Terminal requirements for authentication
  - *e.g.*, secureid is pretty easy to use nearly anywhere while biometrics requires a reader on the client system
- Credential Revocation
  - When credentials are compromised, how do you fix?
  - With some technologies this can be very hard



Authentication Concerns

- Authenticating the server
  - Are you sure you are authenticating to the right system? What's the risk if you are wrong?
    - False login programs, bogus web sites (can we say IBM's own internal firewalls!!!)
    - Passwords high risk here
- Authentication Session Lifetime
  - How maintain?
    - Some kind of token? *e.g.*, LTPAToken
    - Implicit in connection object – *e.g.*, DBs do this
    - Web services (stateless)
  - When does it expire?
  - Likely not the same as application session (SSO across multiple apps)
    - HTTPSession – idleness based
    - LTPA lifetime – time based

Passwords are fundamentally a problem

- Can be hard to remember
  - You probably shouldn't write them down

- **Problem**: How many sites do you access? How many passwords?
  - Can share a small number of passwords
    - Perhaps varying which password is used based on "importance"
    - Risk as compromise of one site may compromise many
  - Can store site unique password in a password database
    - *e.g.*, PasswordSafe - http://passwordsafe.sourceforge.net/
  - Clever Stanford trick
    - Sends a one way hash of password and domain to site
    - Each site then has a unique password, yet user has to remember only one
    - http://crypto.stanford.edu/PwdHash/

- Even worse, passwords can be captured

Keyboard loggers

- **An actual email…**

I urgently need some assistance with this.

We were called in by a local bank this morning who noticed that some of their branch PCs had these hardware key logger devices attached to them and were being used to capture login details, which are being used to commit fraud.

Losses experienced to date are huge.

Details of the hardware keylogger are at this site:
http://www.keyghost.com/

- **Can someone help with this…**

Is there any means of detecting the presence on these devices on PCs. (the bank security team claim it is not possible, except by physical audit - which is not practical)

What counter measures can we implement, besides physical surveillance or put a physical locked case around the system unit.

Can we put together a forensics investigations team to help the bank find the culprits - (they believe it is an insider)

Advanced Authentication Technologies

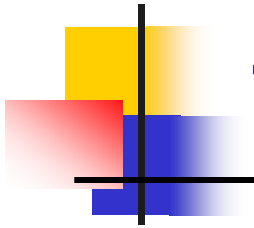
- **Smart cards**
  - *e.g.*, SecureId, cards that hold certificates, *etc.*
  - Seems pretty good if done right
    - Of course anything can be compromised
    - But, so far seems to require stealing the card, thus likely easy to detect quickly – hey, where's my card?
  - US Homeland security cards problematic: http://www.epic.org/privacy/surveillance/spotlight/0405.html
  - Beware that information in RFID enabled devices is likely easily stolen (unless encrypted): http://www.wisdom.weizmann.ac.il/%7Eyossio/rfid/

Advanced Authentication Technologies

- **Biometric**
  - Revocation?
    - IBM claims to have a partial solution:

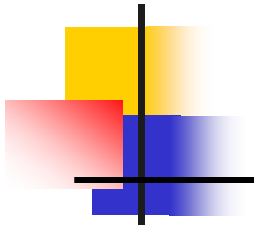
      http://www.computerworld.com/printthis/2005/0,4814,103986,00.html

    - IBM's system wouldn't entirely solve the replaceability problem of biometrics:
      - ✓ If a hacker got hold of a user's fingerprint and made a passable model, he could still wreak havoc with it.



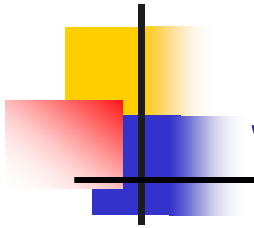
Trust

- Trust Domain
  - a collection of servers, processes, machines, *etc.*, that are part of same security infrastructure and trust same thing.
- Least Trust
  - to limit risk, trust the fewest things possible
  - Work hard to secure them.
- “Chain of Trust”
- Delegation
  - Sending identity information to another.
  - Might also be called identity projection.



Registry

- Users are identified here
  - A single registry might span multiple directories/data stores
- Directory
  - Particular “concrete” instance of user data.
    - *e.g.*, an LDAP server, file based, etc…
- How does registry relate to “system of record”
  - Are all users in this registry?
  - Replica?
  - It is a Subset?
  - Replication considerations (high availability, *etc*.)



J2EE Roles

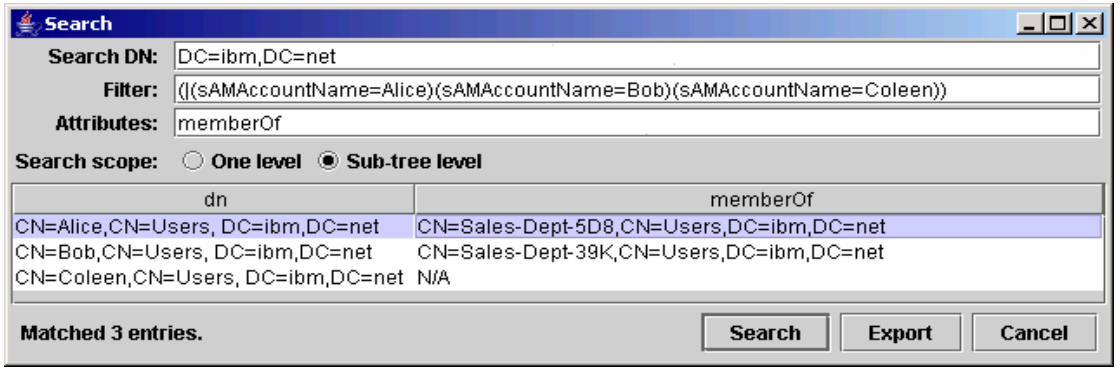
- Role based authorization protects resources by only making them available to users who have been granted those roles

- It is essential to be clear what the word “Role” actually identifies:
  - In the J2EE world, roles are not defined as groups of users. This is a common misconception!
  - J2EE Roles are application defined logical constructs which are bound to Principals at deployment time. A Principal can be a User or a Group.
  - Roles actually conceptually contain a set of permissions (these permissions are specified as constraints)

- Role based authorization in J2EE can be defined using declarative or programmatic means
  - Declarative authorization is configured in web.xml and ejb-jar.xml
  - Programmatic authorization makes use of the standard J2EE APIs

Roles *vs.* Groups

- Groups are a registry concept
  - Registries contain groups, not roles – regardless of what they may call them!!
- Roles are a logical concept that is **application-specific**.
  - Applications can be written completely abstracted from Groups
- Example:
  - Alice is a member of the group sales-dept-5d8, Bob sales-dept-39k, and Coleen none
  - To grant Alice access to something, you need to ensure that the **group** sales-dept-5d8 is bound to the **role** that has the needed permissions



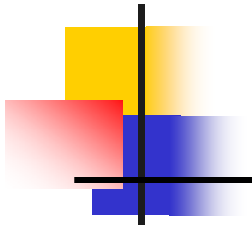
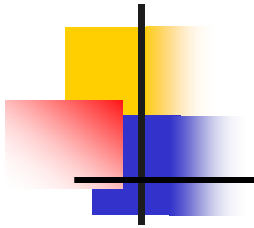


Table of contents



Security Problems

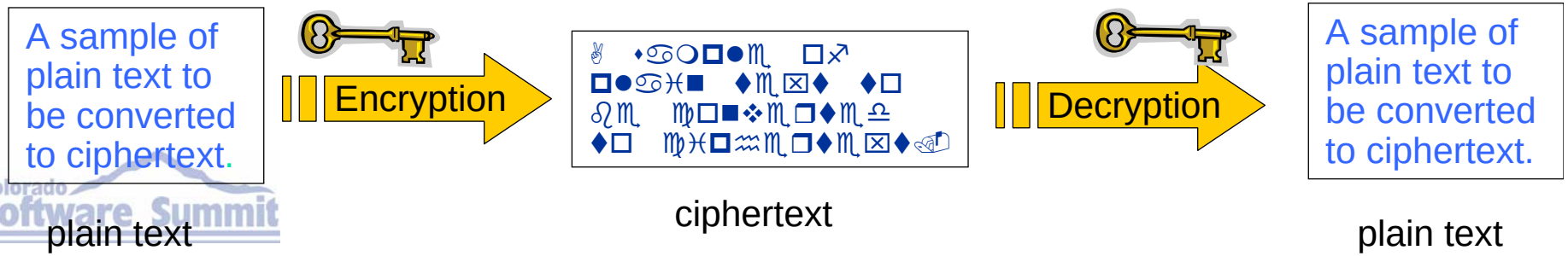
- Eavesdropping –
  - how do I keep anyone from seeing my message
- Tampering –
  - how do I know if anyone has intercepted the message and tampered with it?
- Impersonation –
  - how do I know that the person who I am conversing with is the person that I think it is?

Eavesdropping

So how do I prevent anyone from looking at my message…

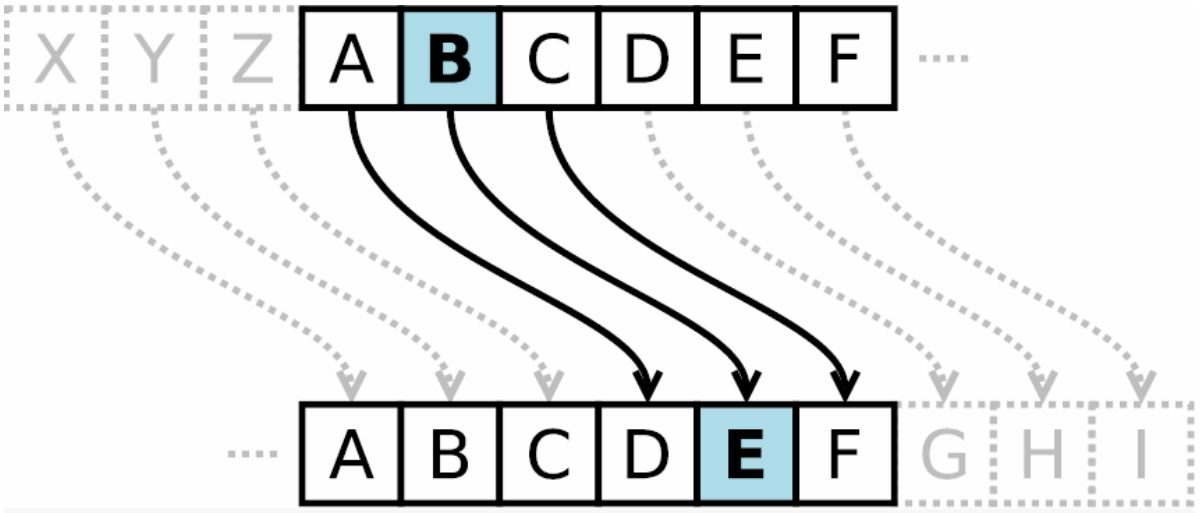
- **Use Cryptography!**
  - Modern cryptography uses algorithms and keys to manipulate data
  - Keys are pieces of data that are used to alter how the algorithm behaves
  - Knowing the algorithm does not help an attacker – they need the key.
    - Most algorithms are published publicly
    - Keys are usually protected and hidden

- **The cryptographic process uses Keys**
  - Encryption – The process of applying a key to create a ciphertext message.
  - Decryption – When an a key is applied to a ciphertext to recreate the original message



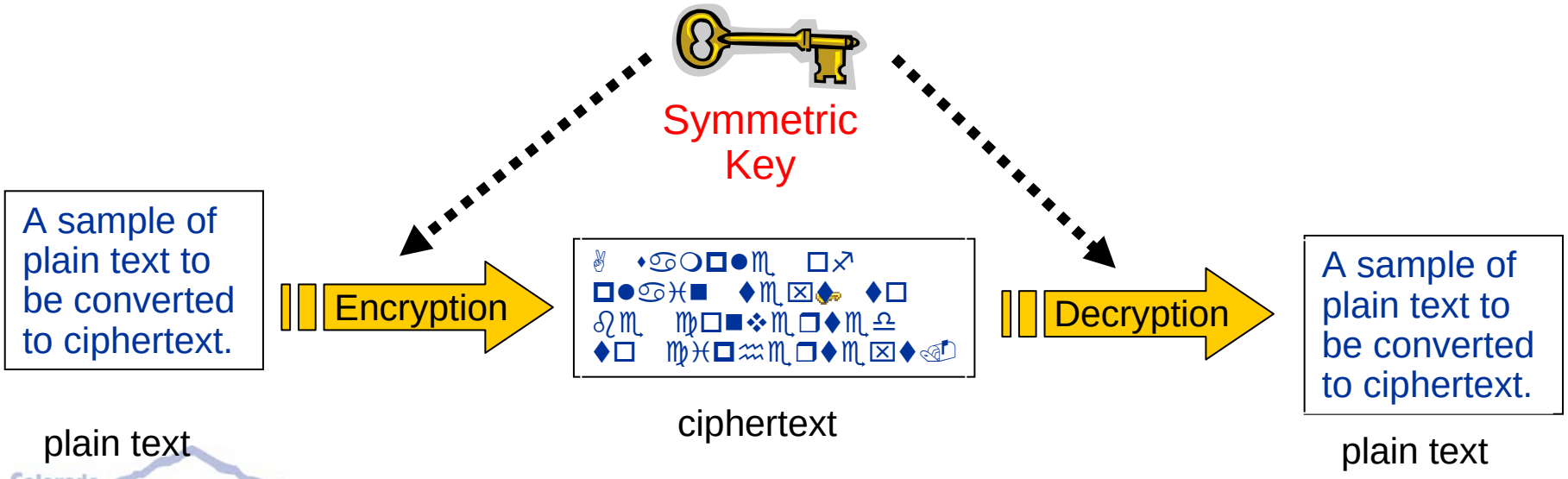
Caesar Cipher

- Rot3 system allegedly used by Julius Caesar
  - Rotated characters in a message by 3 characters
  - Cipher “UHWXUQ WR URPH” = “RETURN TO ROME”
  - Usenet news reading and posting programs include a **rot13** feature.



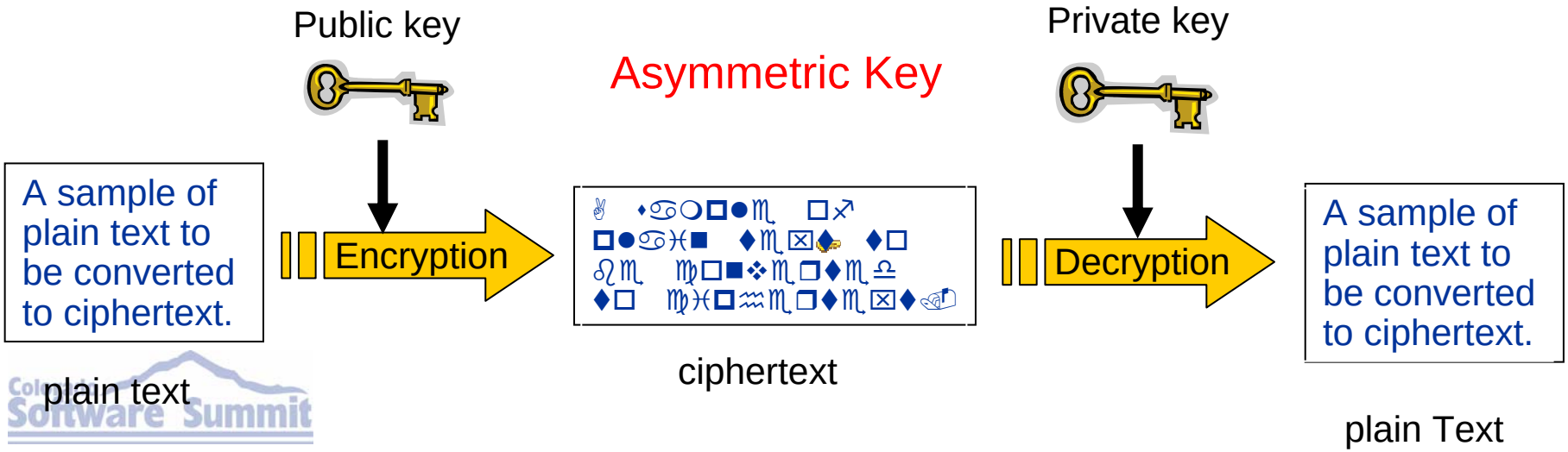
Symmetric Key Encryption

- Symmetric key –
  - a secret key that is used to both encrypt and decrypt messages
- Known only by sender and receiver
- Relatively fast
- Challenges – exchanging keys with many people



Asymmetric Key Encryption

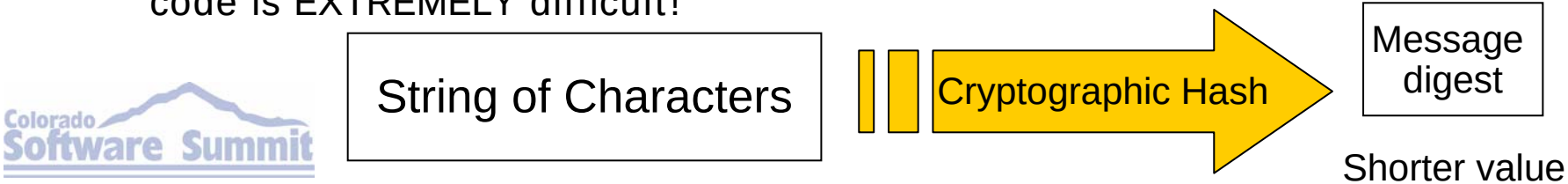
- Asymmetric key – public/private key pairs
  - Also known as public key cryptography
- Message is
  - encrypted with public key and
  - decrypted with private key (or secret key)
- Slower than secret key encryption
- The private key cannot be constructed from the public key
- Either Key can undo what the other does



Tampering

So how do I know if anyone has looked at my message and changed it?

- This is where the use of a cryptographic hash comes into play.
- A cryptographic hash function is an algorithm that transforms a string of characters into a shorter number of a fixed length.
  - This value is called the message digest
    or
  - Message Authentication Code (MAC)
- If anyone tampers with the message
  - this will result in a different message digest or hash number.
- Purposefully creating 2 separate messages that create the same hash code is EXTREMELY difficult!

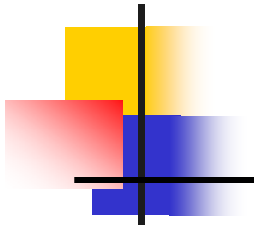


Cryptographic Hash Algorithms

- **Cryptographic hash functions**

  - There are 2 popular families of cryptographic hashes
    - MD4, MD5
      - ✓ 128 bit digest
    - SHA-1, SHA-224, SHA-256, SHA-512
      - ✓ 160, 224, 256, 512 bit digests respectively

  - Questionable hashes
    - MD4 and MD5 are now considered broken
      - ✓ http://www.litfuel.net/plush/?sectionid=7
        (45 minutes to break an MD5 hash!)
    - There are concerns that SHA-1 may also be broken but no evidence to prove that.



Impersonation

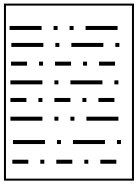
How do I know who the party on the other end is?

- By the use of Digital Signatures
  - Provides the ability to authenticate who the message was sent from.
  - Provided within a Digital certificate
  - Incorporates the use of Asymmetric keys and cryptographic hash functions.
- The signature itself is encrypted with the senders private key.
- The signature is what is verified when the certificate is checked.

Digital Signature

Joe

Joe creates his plain text document

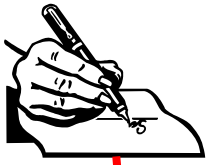


The message digest is created using the Hash function

Message Digest

Private key

The message digest is encrypted using Joe's private key to create the signature

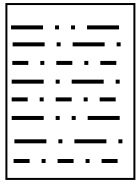


Kate

Public key

Using Joe's public key Kate decrypts the signed hash to get the message digest

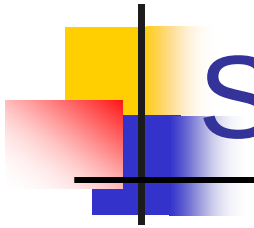
Message Digest



Kate hashes the plain text to produce a message digest

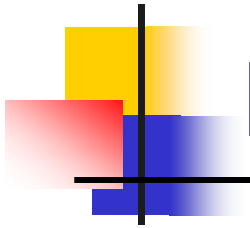
Message Digest

If both message digests are equal, then the message has not been tampered with, and only Joe could have signed it.



Security Problems – Solved

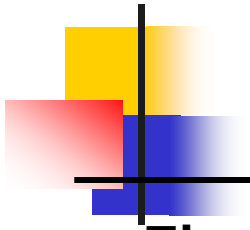
- Eavesdropping – how do I keep anyone from seeing my message?
  - Symmetric and Asymmetric key messages can be encrypted and decrypted.
- Tampering – how do I know if anyone has intercepted the message and tampered with it?
  - Cryptographic Hash Algorithms – these algorithms can be used to detect if a message has been tampered with
- Impersonation – how do I verify the sender and authenticate that they are who they say they are?
  - Digital Signatures – by using a public key, it can be determined who has sent the message.



Problem – Public/Private Keys

- The trouble with public/private key pairs is that they **don't identify** anyone. Somehow you have to manage that manually.
  - Given a message encrypted (or signed) using a private key, you can determine which public key goes with it, but so what?

- **For Example**...
  - If Alice wants others to be able to send her secret messages, she needs only to publish her public key.
  - Anyone possessing it can then send her secure information.
  - Unfortunately, David could publish a different public key (for which he knows the related private key) claiming that it is Alice's public key.
  - In so doing, David could intercept and read at least some of the messages meant for Alice.

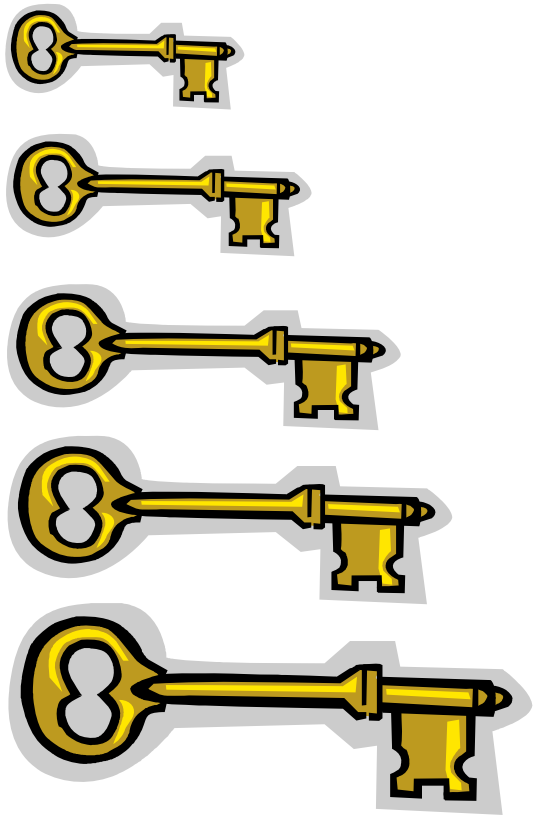


Solution - Digital Certificates

- **The solution is Digital Certificates**
  - A special kind of public key
    - a digital signature to bind together a public key with an identity
  - **The Digital Certificate contains**
    - Your name
    - A serial number
    - Expiration dates
    - A copy of the certificate holders public key
    - And a digital signature, to verify that the certificate has not been altered since it was signed
    - An indication of the signer of the certificate – “the trusted signer”
    - It does not contain the private key, although we often refer to the private key and certificate together simply as “the certificate”

Cryptography Public Key Sizes

- Key sizes (encryption strength)
  - 512 bits – WEAK - - - -
  - 768 bits - - - - - - - - - -
  - 1024 bits - - - - - - - - -
  - 2048 bits – STRONG - -
  - ** 2048+ - - - - - - - - -
    (starting to see use)



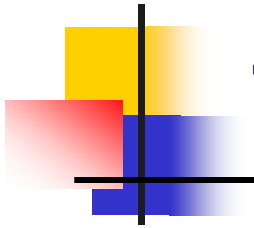
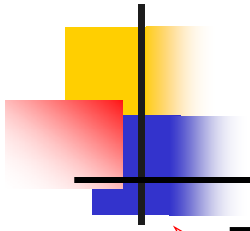


Table of contents

Certificates, where do I get them from?

- Before any encrypted communication using certificates can take place, you must have the certificate containing the digital signature within your keystore!
- So where do I get these certificates?
- They have to be physically received one way or another

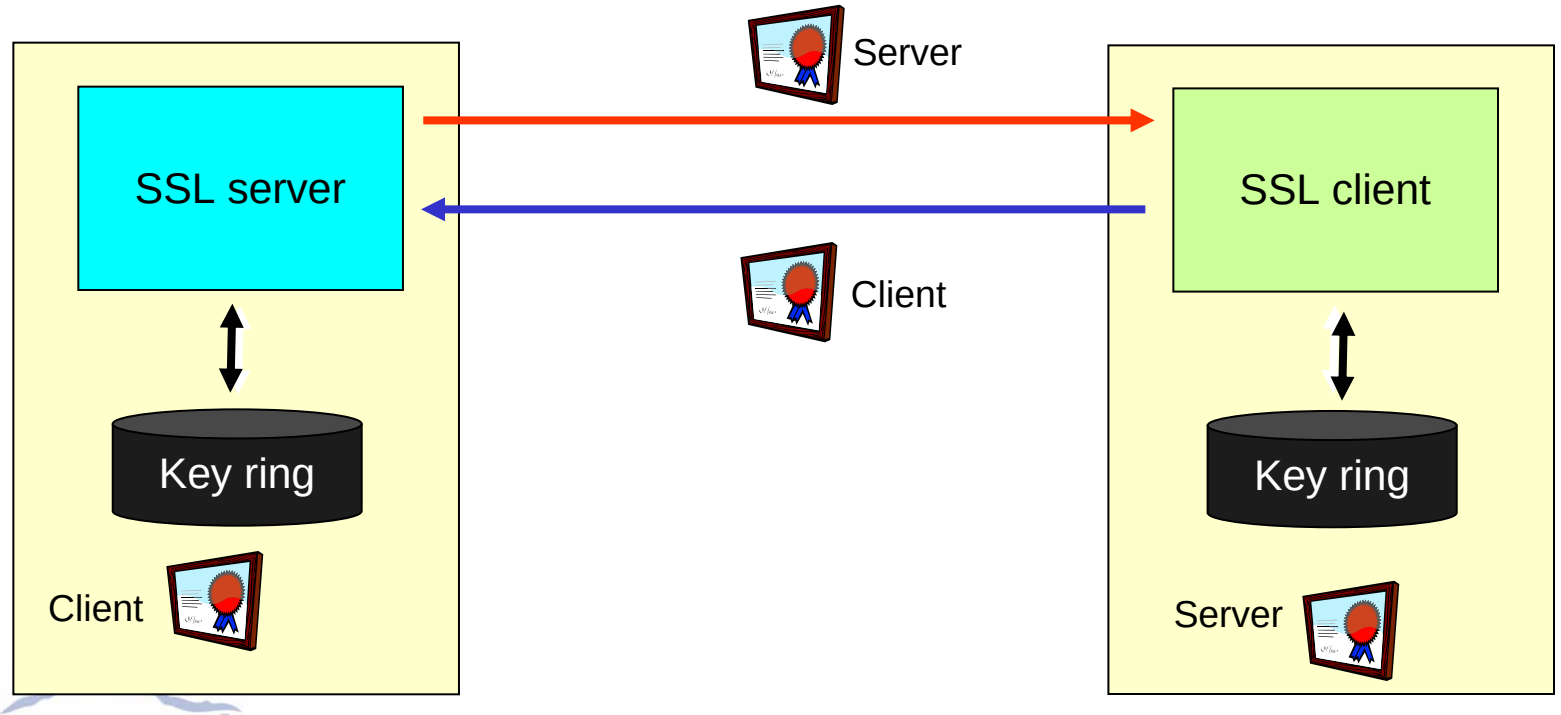


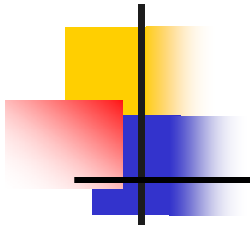
Creating a certificate

- The Certificate issuer first
  - verifies that a public key belongs to a specific company or individual (the "subject"),
  - the validation process it goes through to determine if the subject is who it claims to be depends on the level of certification and the issuer itself.
- The Certificate issuer then creates a certificate that contains Certificate issuer and subject information,
  - including the subject's public key
- The Certificate issuer signs the certificate
  - by creating a digest of all the fields in the certificate
  - and encrypting the hash value with its private key.
  - The encrypted digest is called a "digital signature," and when placed into the X.509 certificate, the certificate is said to be "signed."

Certificate Management, Self signed

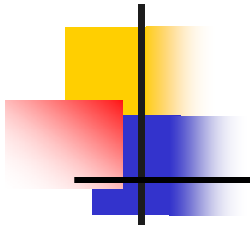
- Before any communication can take place, the client must already have a copy of the server's certificate.
- If you are using client authentication then the server must already have a copy of the client's certificate





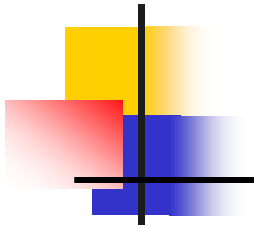
Distribution problem

- 1st we created public/private key pairs so we could communicate securely
- 2nd we created digital certificates that we could send to someone we wanted to communicate with.
  - These certificates contained our digital signature and our public keys.
- Now we have all these Certificates to manage!!!
  - We need to send everyone who wants to communicate a signed certificate with our public key!
  - We also need to get a signed certificate with a public key from everyone who we want to communicate with.
  - What a distribution nightmare!



Distribution solution

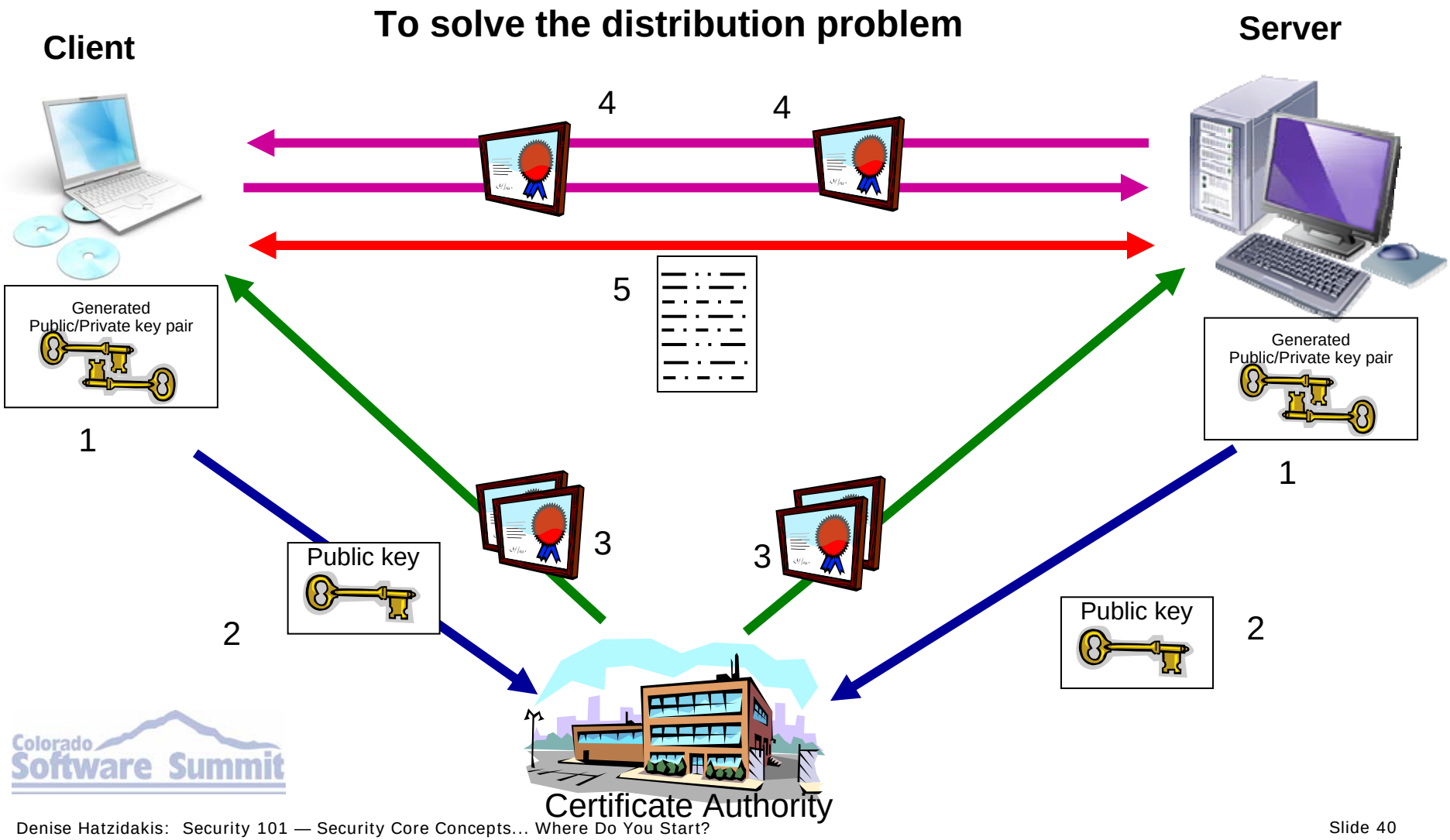
- **Instead, we create a Certificate Authority's (CA) to resolve this issue!**
  - Instead of self signed Digital Certificate to everyone who needs it, they just need to download the CA certificate that signed your certificate!
- **A CA can be thought of as a certificate validator**
  - A certificate that is signed by a CA can be assumed to have undergone some level of validation so we can assume (we hope) that the identity information in the certificate is trustworthy.



CA Certificates

- To obtain a CA certificate you have to send your information to a Certificate-issuing Authority, such as Verisign.
  - Create an Asymmetric key (public/private key pair)
  - Send your identity along with public key to CA
  - The CA then does some appropriate action to verify that you are who you claim to be and then signs the certificate
    - **Beware**: this process is not magic and can in fact be compromised!!!!
  - CA sends you a back a CA signed certificate.
    - **Note**: Depending on who creates the CA cert, it may cost you money.
- The CA then issue you a signed Digital Certificate that contains your public key.
  - This is the certificate that get transmitted during the SSL handshake.

Using CA Certificates



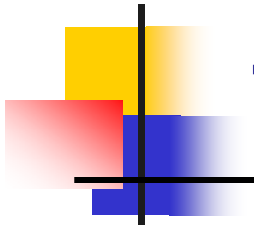
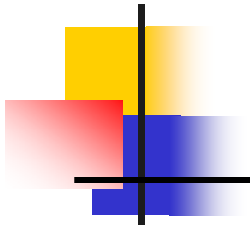
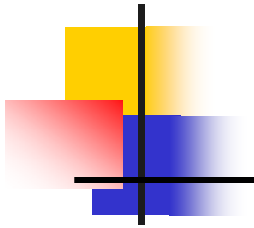


Table of contents



Solving the security problems

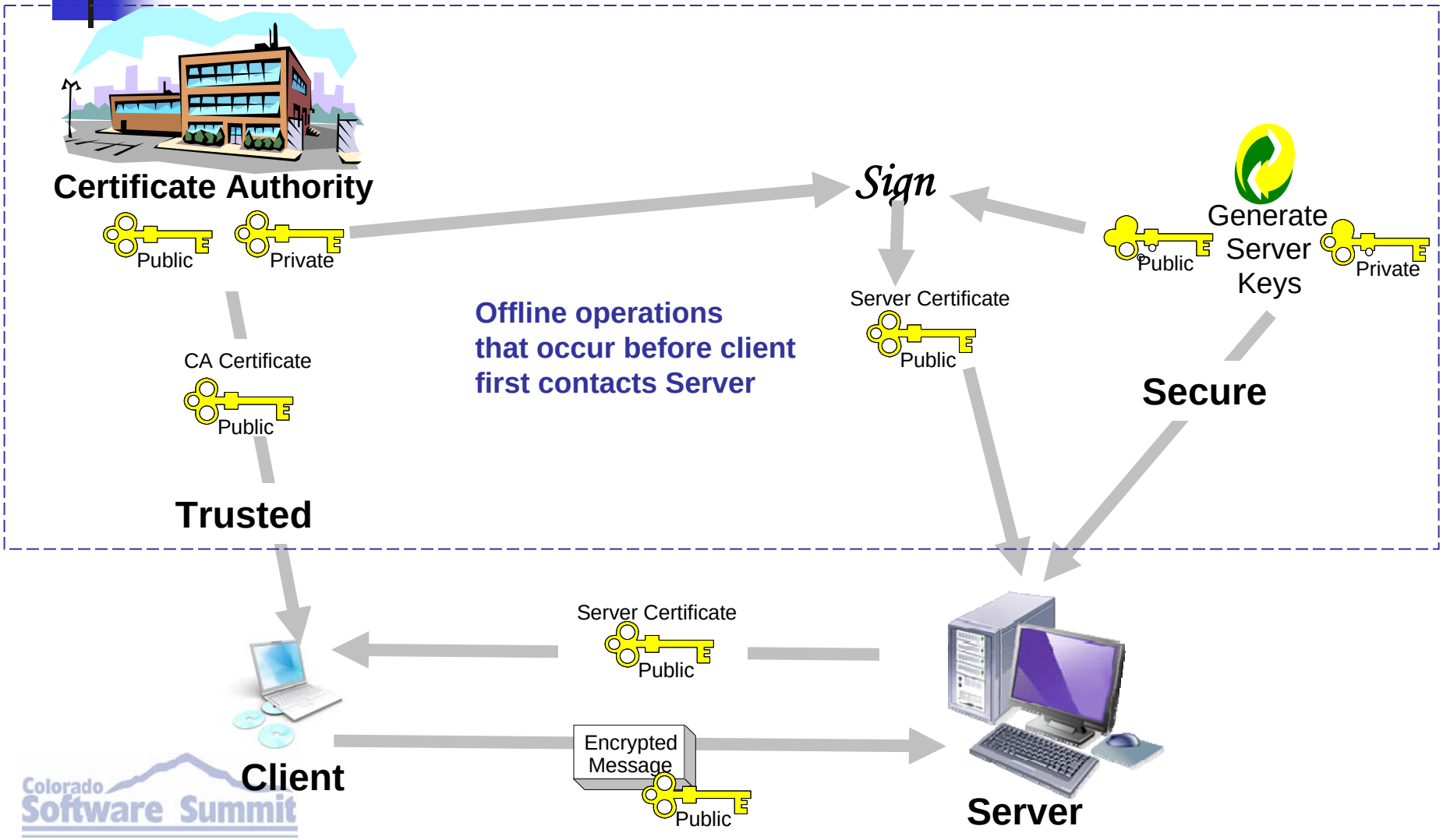
- Solves the following security problems
  - Tampering
  - Impersonation
  - Eavesdropping
- Using the following processes
  - Symmetric and Asymmetric Keys
  - Encryption techniques
  - Digital Signatures
  - Digital Certificates
- These processes are combined together in a protocol called the Secure Sockets Layer

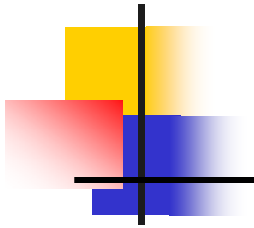


SSL

- SSL provides
  - Message privacy
    - Using asymmetric and symmetric key encryption.
    - Uses a handshake when initiating contact.
    - The handshake establishes a session key and encryption algorithm, between both parties, prior to any messages being sent.
  - Message integrity
    - By using the combination of shared secret key and cryptographic hash functions.
    - This ensures that the content of any messages does not change.
  - Mutual authentication
    - Server always authenticates to client.
    - Client optionally authenticates to server
    - This happens during the handshake.

SSL With CA - Simplified





SSL "Authentication" - Simplified

- Server side SSL authentication in brief
  - Server sends its certificate to client along with secret encrypted using its private key (from key store)
  - Client validate the server's certificate by checking its expiration date and signature
  - Signature is verified using signing certificate in trust store
    - If certificate signature isn't right, connection will be refused
  - Client side authentication is basically the same with the parties reversed

SSL in Detail

- **Cipher Text**
  - Encryption is the process of transforming information (referred to as plaintext) to make it unreadable to anyone except those possessing special knowledge, usually referred to as a key

- **Cipher Spec**
  - A cryptographic hash function (MD4, MD5, SHA-1, ….)
    - used to create the message digest
  - An Encryption method (DES, 3DES, RSA, RC4, Blowfish)
  - Both ends of the channel must use the same CipherSpec

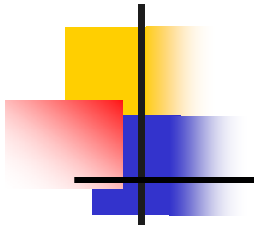
CipherSpec = Encryption Algorithm + Hash Function

SSL in Detail

- **CipherSuite** –
  - The CipherSpec
  - And the Authentication/key exchange algorithm

CipherSuite = CipherSpec + Authentication/key Exchange

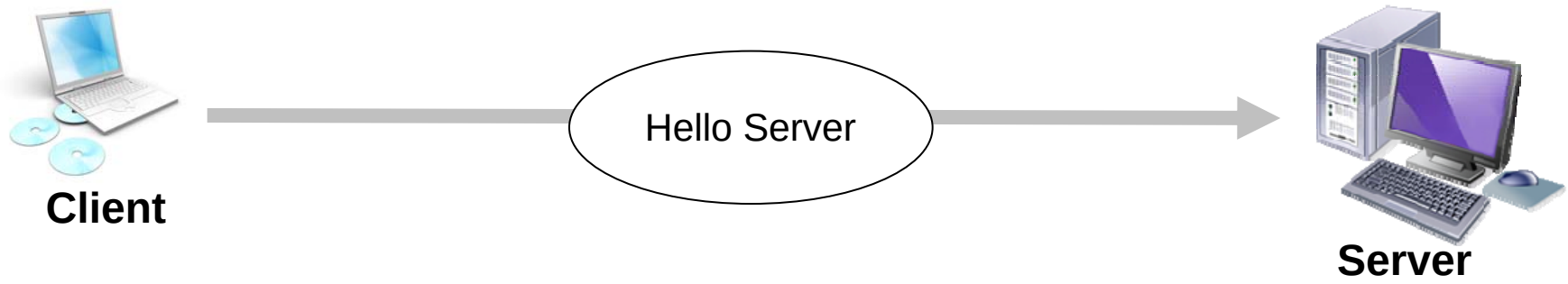
- For example,
  - **SSL_RSA_WITH_RC4_128_MD5**
    - This suite means,
      - ✓ Use the SSL protocol,
      - ✓ use RSA certificates to manage the identification and key exchange,
      - ✓ use 128-bit RC4 as the data stream cipher,
      - ✓ and use MD5 for the Message Digest.



SSL Handshake

- The Handshake
  - The way SSL establishes a secure communication link between a client and a server.
- During this handshake process SSL will
  - Negotiate the level of SSL to use.
  - Decide on a Cipher Suite that both parties can use.
  - Authenticate the server and (optionally) the client.
  - Build a secret key that is to be used for this session only.
- The Handshake is initiated by the client, sending a “hello” message to a server.

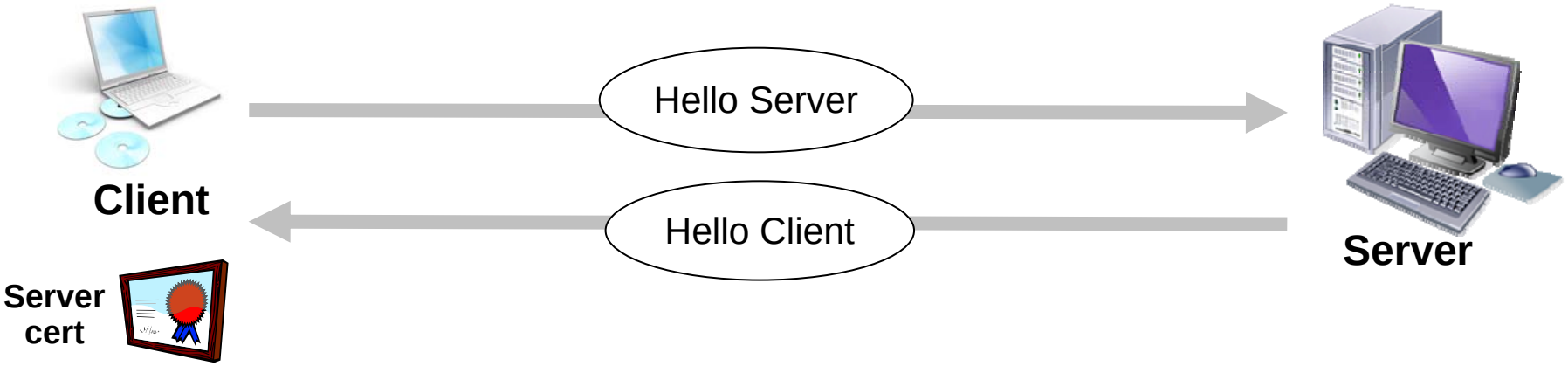
SSL Handshake Client Hello



- **Client** sends **Server** a hello message.
  - Within this initial contact message is a list of Cipher Suites that the client can use.
  - The list is used in client preference.

Note, The client is considered the one who initiated the communication.

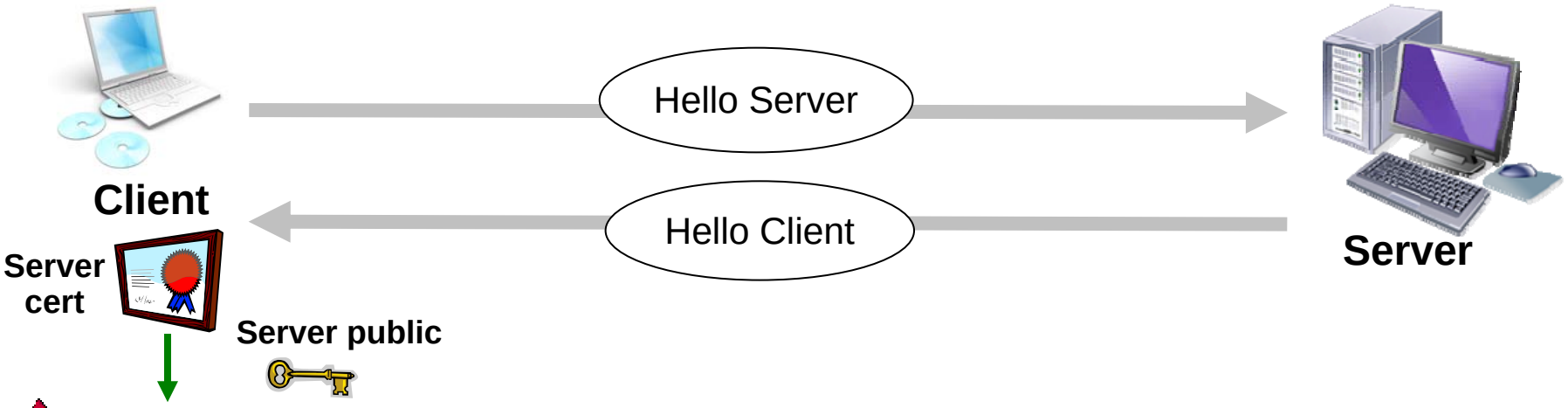
SSL Handshake Server Hello



- **Server** responds with a "Hello"
  - This response contains the CipherSuite that **Server** has selected from the list the **Client** sent.

- **Server** also sends Client a signed Digital Certificate containing the Server's public key as well as a certificate chain of signers
  - **Client** will already have a signed certificate in a key store, to verify Server's signature against

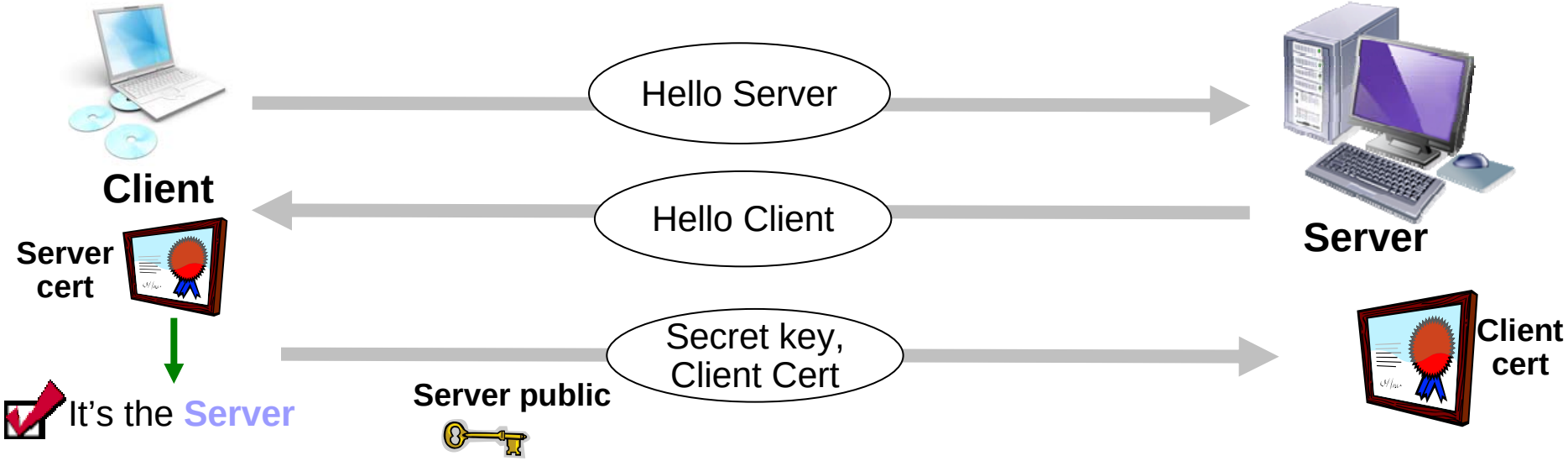
SSL Handshake Verify Server Certificate



It's the **Server**

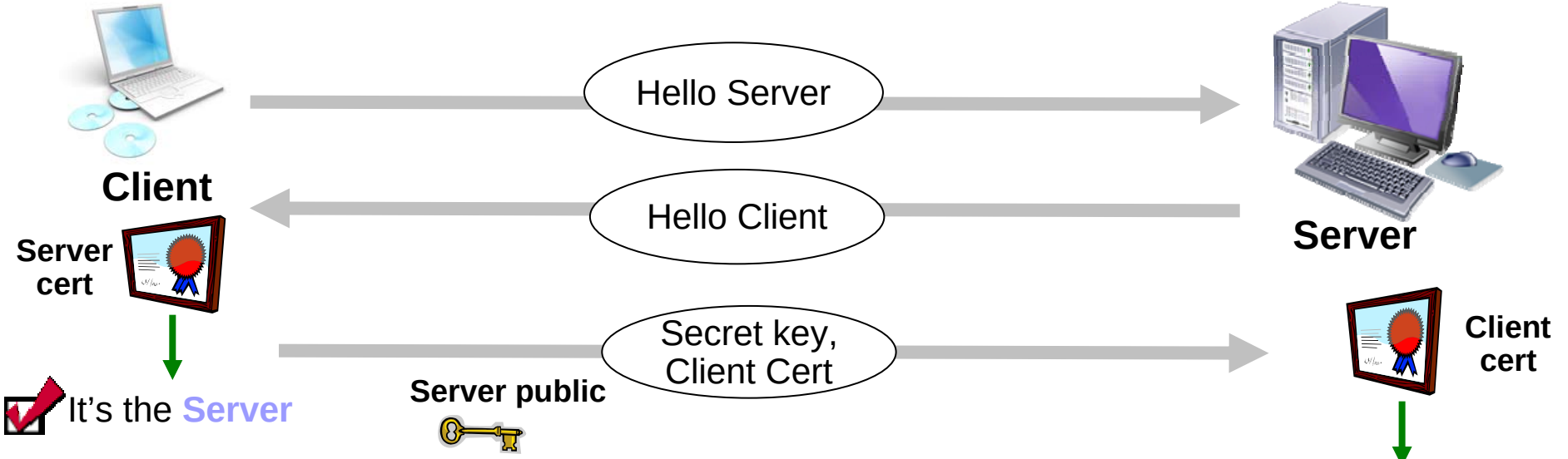
- **Client** checks if certificate has expired
- Verify the certificates signature against a signed certificate in a key store. This may involve following the certificate chain.
- Receive Server's public key from certificate
  - No validation of server identity is actually done!! A browser will check the domain name of the signature against what is in the keystore.
  - This is not part of the SSL process
- If client Authentication is being used, then the **Server** would also have requested a Digital certificate from the client.

SSL Handshake Client Key Exchange



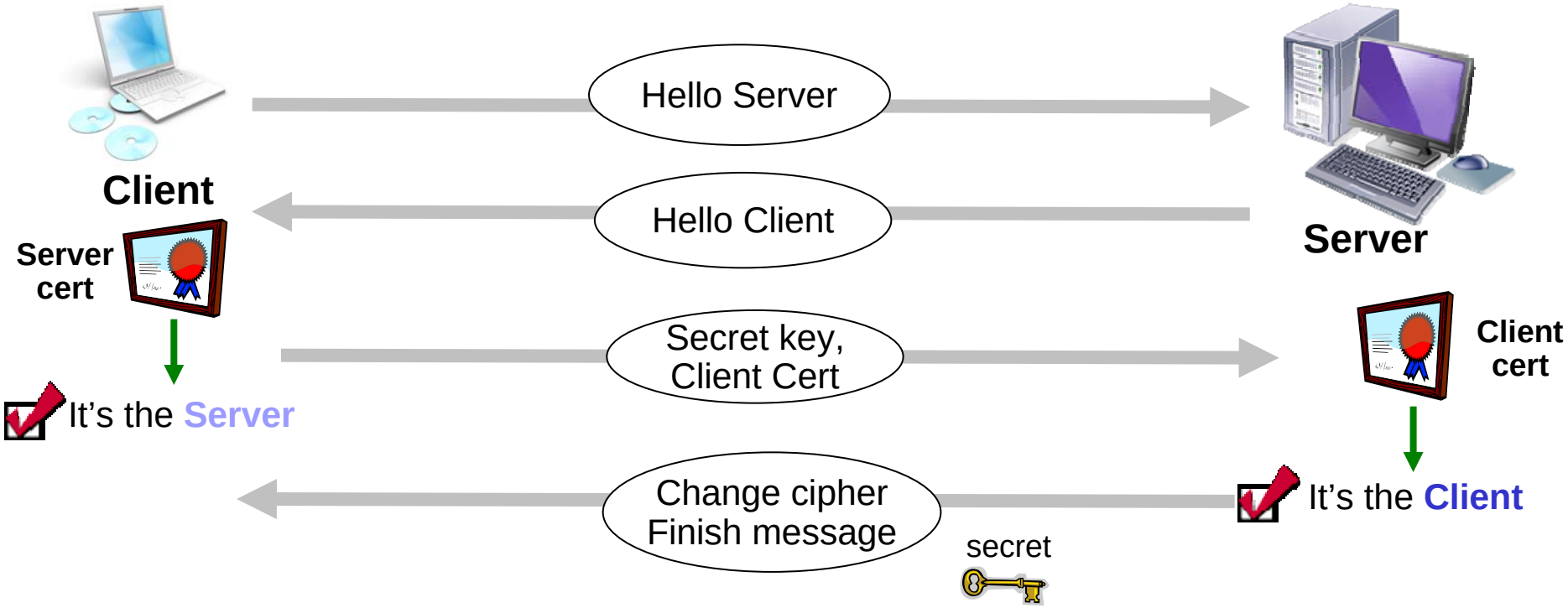
- The **Client** builds and sends the server a secret key that is encrypted using the **Server's** public key.
- If client Authentication was requested, the **Client** would send a client certificate and verify message

SSL Handshake Verify Client Certificate



- **Server** would (optionally) verify the client certificate against a keystore
- Decrypt secret key using **Server's** private key
- **Client** key exchange message,
  - the premaster secret is sent, encrypted using the server's public key.
  - Both the client and the server generate the symmetric encryption keys on their own using the premaster secret and the random data that is generated from the SERVER_HELLO and CLIENT_HELLO commands.

SSL Handshake Reply with Secret Key



- Change CipherSuite message
  - This indicates that the contents of subsequent SSL data sent by the client will be encrypted.
- Finish message
  - Encrypted using the symmetric secret key

SSL Handshake Complete

Client

Ok we have:

- Agreed Upon a CipherSuite
- have set up our CipherSpec
- Exchanged secret keys.

This is now a secure Connection!!

Let's communicate!

Server

Encryption and SSL Term Summary

- Private key –
  - private half of public/private key pair (don't share!!!)
- Public key –
  - public half of public/private key pair
- Certificate – a public key that has been signed
  - We sometimes refer to the signed public key and private key together as the certificate or personal certificate, but that's not really correct
- Personal Certificate
  - Usually used to mean certificate along with private key data
- Signing certificate
  - The certificate that corresponds to the private key that was used to digitally sign a public key
- Certificate Authority
  - A well known entity that signs public keys creating certificates
  - Just a special kind of signer

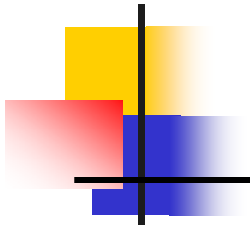


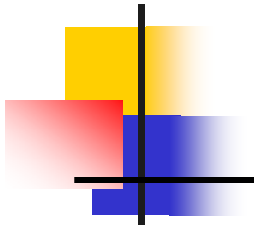
Table of contents

Limiting Connections to Trusted Clients

- Sometimes it is difficult to insert proper authorization into an existing connection path
  - *e.g.* Some App Servers do not provide for authorization of Web Server connections
- If we limit the signers we trust on the server, we can limit the clients that can complete the SSL handshake
  - With self-signed certificates, there is only one signer
  - Hence, only one valid client side private key that can be used to connect
  - We can also limit the signers we trust on the client to ensure the client only connects to the “right” servers
- We can use this technique to limit connections to trusted clients
  - Several situations where this technique is useful

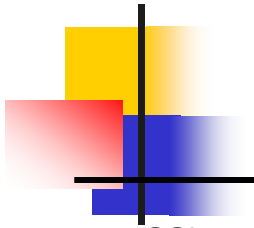
Implementing "Trusted" Tunnel with SSL

- Using certificate tools
  - Create two sets of key stores and trust stores
    - Key store and trust store for client
    - Another key store and trust store for the server
  - Remove all trusted signers from all stores
    - Now none can be used to validate any other certificate
  - Create a self signed certificate in each key store
  - Export the signing certificate from each key store and import into the trust store of the other party (client to server and server to client)
    - Now each trust store can be used to validate exactly one certificate – the one that corresponds to the private key just created in the key store



PKI Caveats

- PKI doesn't solve (and in fact ignores) end to end identity problem!!
  - Still need Kerberos or some other authentication system
- Private key must *never* be exposed
  - Don't store on public file system, certainly not network file system
  - Beware of backups
  - Beware of machine compromise
- Hard to revoke certificates once compromised (easier to reset passwords)
  - Really implies smart cards are needed to protect sensitive information
    - Has added advantage of being 2 factor authentication
  - Certificate Revocation Lists or online revocation aren't easy to implement
    - Very few app servers do



SSL Caveats

- SSL sure sounds magically great. Once I enable SSL, my system is secure then, right? WRONG!
  - Authentication versus completing the SSL handshake
    - SSL handshake validates certificates, it doesn't really authenticate the user.
    - Get subject DN out of the certificate as an additional step to map to user identity
    - Need for proper identity after handshake
      - ✓ Is this really the target server?
      - ✓ Is the really the right client?
  - Too many signers problem
    - Doing any authorization based on a DN without checking for trusted CAs is useless
    - Authorization based on DN means you're trusting *ANY* of the trusted CAs to issue a cert with that DN – lowest common denominator
  - Only terminator of SSL connection can do authentication

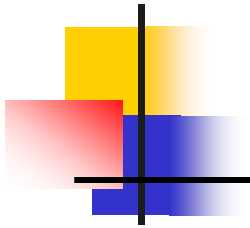
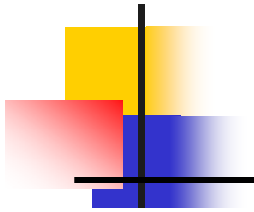


Table of contents



Questions?

And don't forget to do your evaluations... if you liked the session :>)

And don't forget to do your evaluations... if you liked the session :>)

Software Summit