Complications with Digital Certificates

Nalini Elkins

Inside Products, Inc.

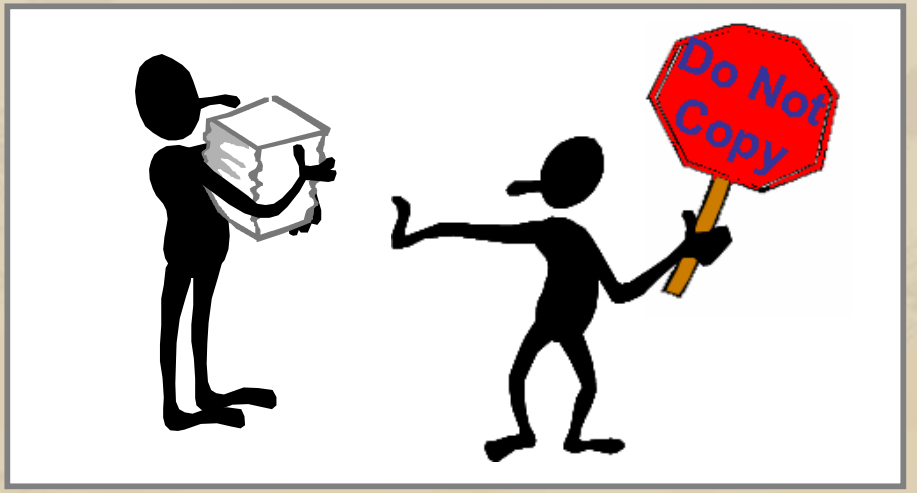
Nalini.elkins@insidethestack.com

Inside Products, Inc.

(831) 659-8360

www.insidethestack.com

Copyright



Agenda

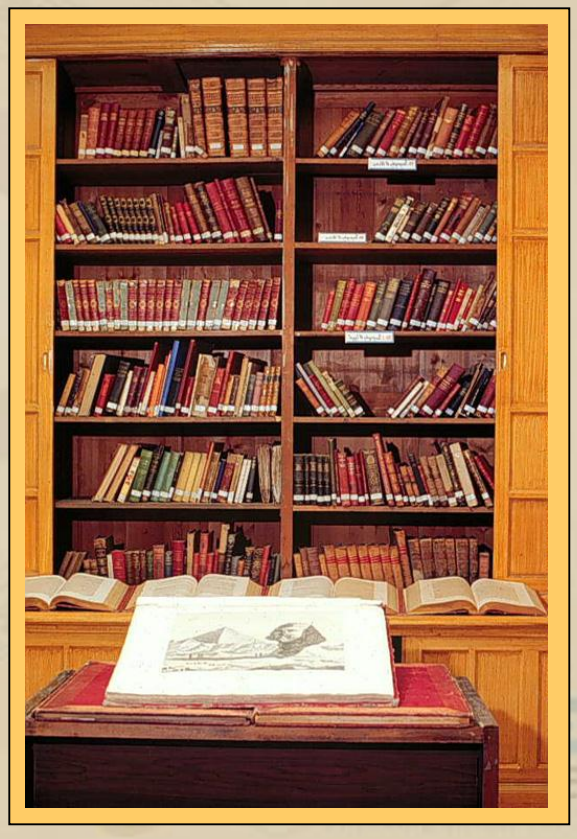
- What is Secure Sockets Layer (SSL) and why do we need it?
- What are certificates and why do we need them?
  - Public Key Infrastructure (PKI)
  - Asymmetric / symmetric algorithms
  - Certificate authorities
- Where are certificates used?
- How are certificates used in SSL?
  - Server only certificates
  - Server and client certificates
  - Server and client certificates in XML
  - Performance issues
- How can we control this whole nightmare?

SSL Applications

- To implement SSL, the application program must use special SSL socket calls.
- As far as the TCP stack is concerned, SSL is just a TCP application. It is transparent to the stack.
- Languages such as C/C++ or Java provide application programming interfaces that interface with the sockets APIs for the platform (z/OS, Windows, Unix) to allow applications to establish secure sockets communications.
- SSL is available for TCP applications only. UDP, ICMP or other higher level protocols are not supported.

SSL Socket Library:

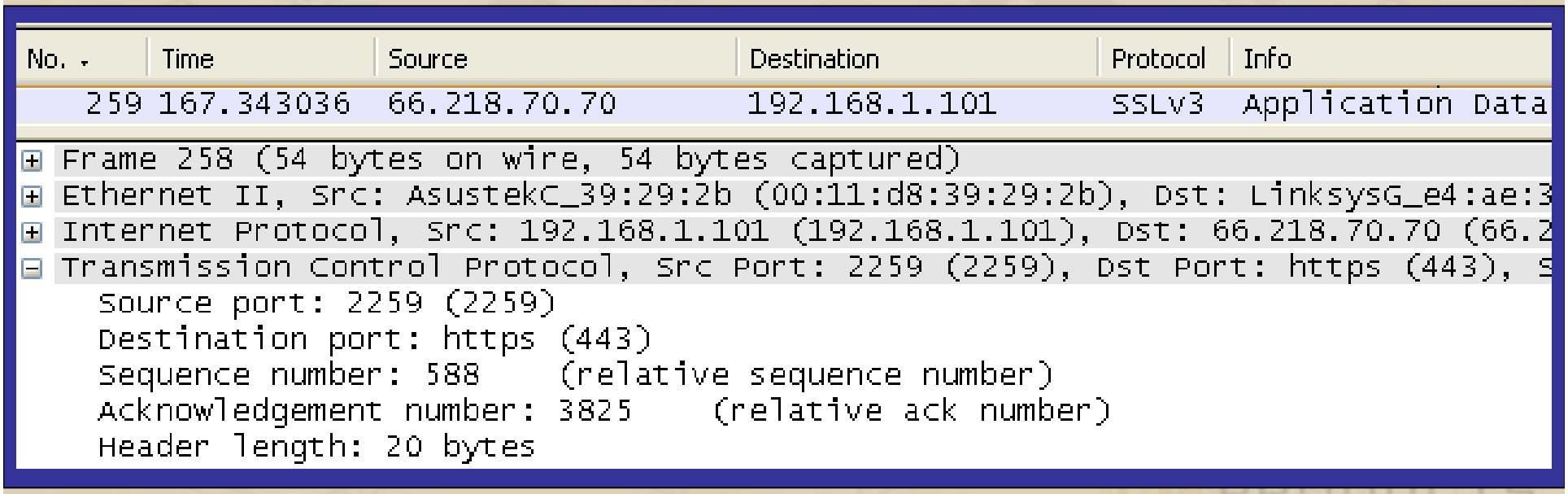
TCP Only



SSL Packet Encryption

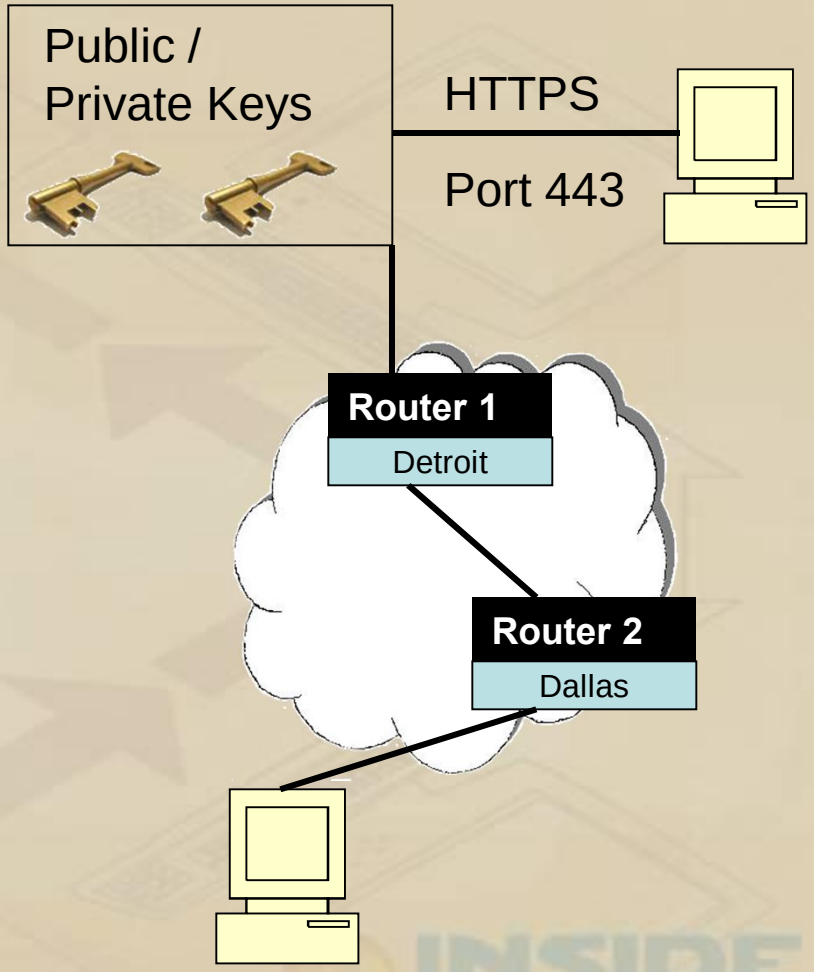
- SSL protects the TCP packet data. The IP header and TCP header are sent unencrypted. The IP header includes the IP addresses. The TCP header includes the port numbers and important error and flow control parameters.

- One of the benefits of using SSL is that if you do a packet trace, you will be able to see the above header, which means that you can see what application is being used. Some protocols such as IPSec encrypt the whole packet.



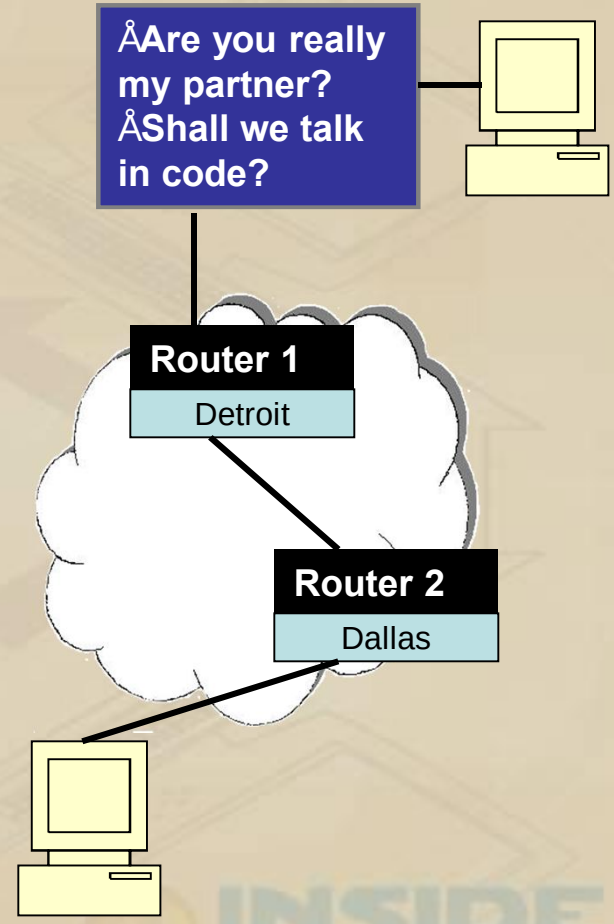
How Does SSL Work?

- Usually, SSL uses two keys to encrypt data ī a public key known to everyone and a private or secret key known only to the recipient of the message.
- All known browsers support SSL, and many Web sites use the protocol to obtain confidential user information, such as credit card numbers.
- By convention, URLs that require an SSL connection start with *https:* instead of *http*:.
- An SSL-protected HTTP transfer uses port 443 instead of HTTP's normal port 80. Ports for SSL for TN3270 or FTP are assigned by the user.



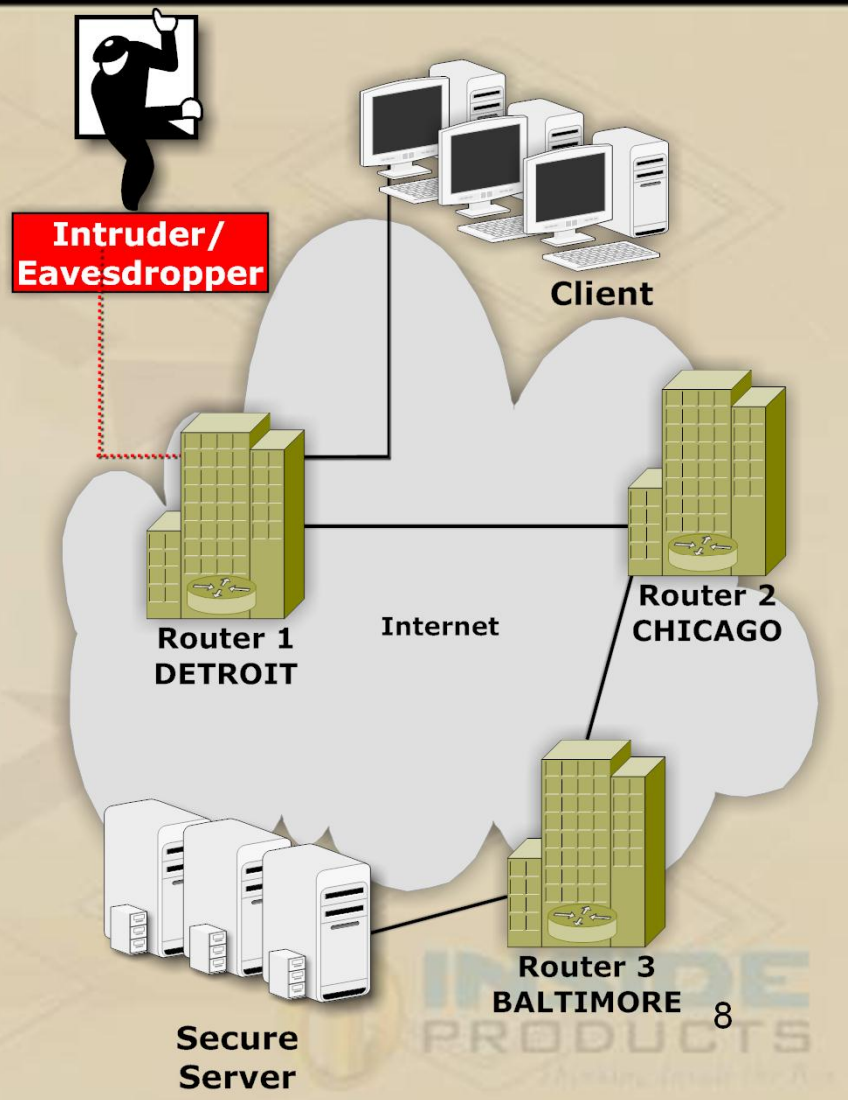
What is Secure Sockets Layer?

- Secure Sockets Layer (SSL) is a widely used protocol first developed by Netscape for transmitting private documents via the Internet.
- The main functions of SSL are:
  - Server authentication
  - Data privacy and integrity
  - Optional client authentication via digital certificate
- Multiple versions of SSL exist: SSL V2.0 and SSL V3.0.
- The SSL protocol became the Internet standard Transport Layer Security (TLS) described in RFC 2246 and updated in RFC 3546.
- TLS V1.0 is the latest version of the secure sockets layer protocol.
- There are slight differences between SSL 3.0 and TLS 1.0, but the protocol remains substantially the same.



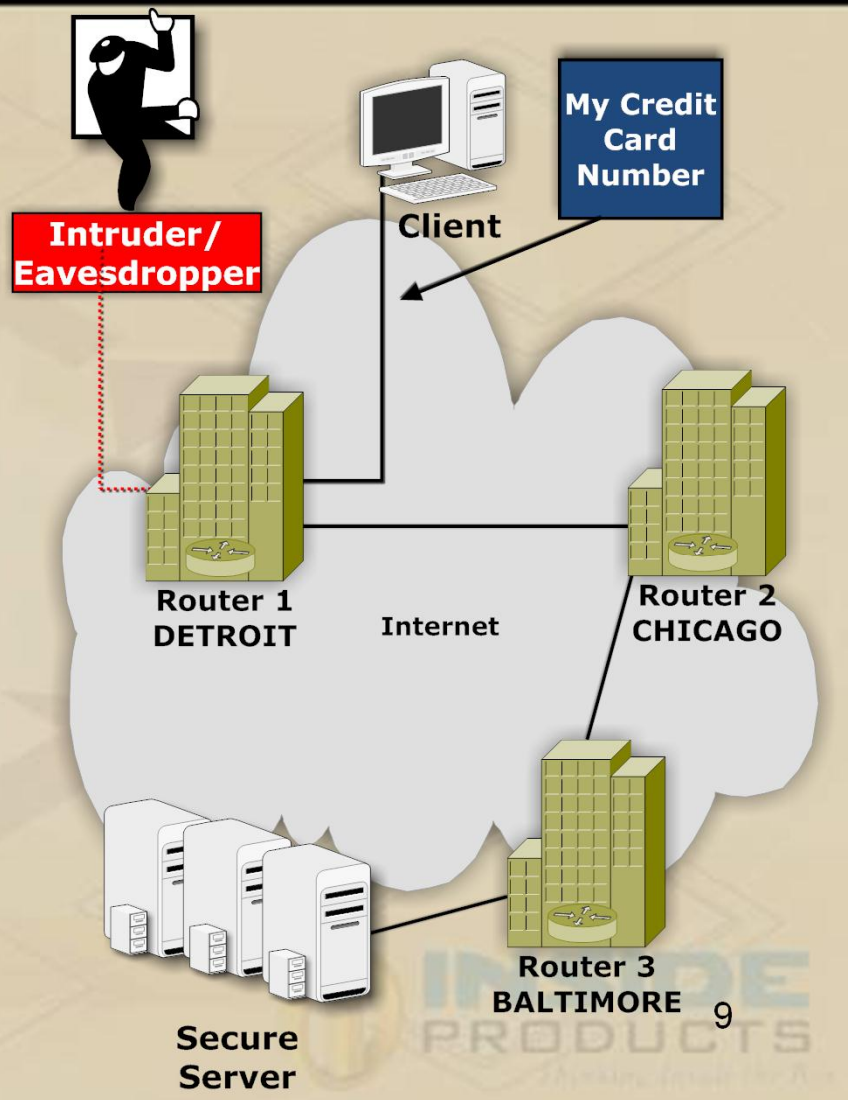
Why Security?

- Today, we have vast communications networks (Internet, Digital (GSM), cell phones, Automatic Teller Machines (ATM)) offering instant ósecureô communication.
- The future of Electronic Commerce and, in fact, the electronic world, rests on secure digital communication.
- Unfortunately, so does the success of terrorists, drug rings, people smugglers, child porn, organized crime, spy rings, and ócyber crimeô



Cryptography

- Certificates fall into the area of Cryptography.
- You can think of Cryptography as the art of keeping secrets.
- In computer systems, what we want to do is transmit data from one point to another in a way that snoopers can't see what we are sending
- Easier said than done!
- Two of the major components of Cryptography are:
  - Encryption and
  - Decryption



Key Based Algorithms

- Most modern algorithms are *key-based.*
- A *key-based algorithm* uses an *encryption key* to encrypt the message. This means that the encrypted message is generated using not only the message, but also using a 'key':
- The receiver can then use a *decryption key* to decrypt the message. Again, this means that the decryption algorithm doesn't rely only on the encrypted message. It also needs a 'keyó
- Some algorithms use the same key to encrypt and decrypt, and some do not.

Encryption Key: **TIGER**

Encrypted Message:

Scr qul wdjj gmkr mus smkmppmw

Decryption Key: **TIGER**

Decrypted Message:

The sun will come out tomorrow

?????

Encryption Key: **FLOWER**

Decryption Key: **TIGERS**

Asymmetric Algorithms

- Algorithms which use the same key are are called *symmetric algorithms*.
- Secure systems also use *asymmetric algorithms*, where a different key is used to encrypt and decrypt the message.
- *Public-key algorithms* are the most commonly used type of asymmetric algorithms.
- In public-key cryptography, the two keys are called the *private key* and the *public key*
- **Private key**: This key must be know *only* by its owner.
- **Public key**: This key is known to everyone (it is *public*)
- **Relation between both keys**: What one key encrypts, the other one decrypts, and vice versa. That means that if the sender encrypts something with a public key, the receiver needs his private key to decrypt the message.

Encryption Key: **TIGER**

Encrypted Message:

Scr qul wdjj gmkr mus smkmppmw

Decryption Key: **TIGER**

Decrypted Message:

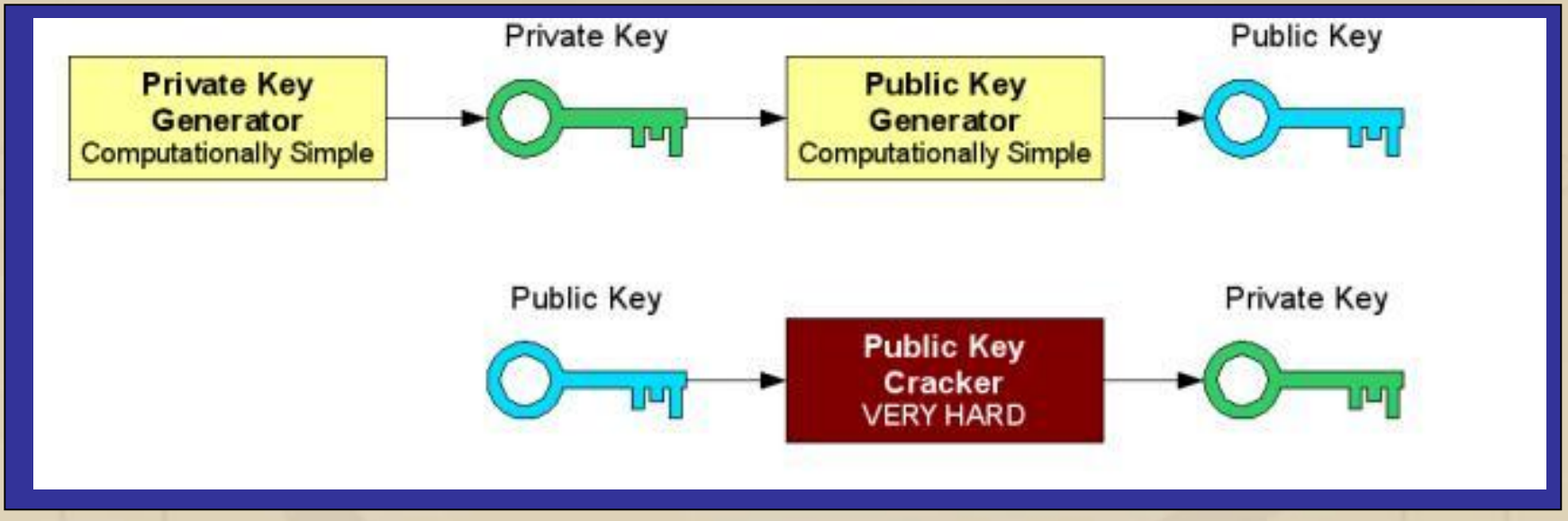
The sun will come out tomorrow

?????

Public Key: **FLOWER**

Private Key: **TIGERS**

Key Generation



- In public-key systems, it is relatively easy to compute the public key from the private key, but *very hard* to compute the private key from the public key (which is the one everyone knows).
- In fact, some algorithms need several *months* (and even years) of constant computation to obtain the private key from the public key.
- The public key algorithm most often used is called the RSA algorithm which was invented in 1978 by Ron Rivest, Adi Shamir, and Leonard Adleman.
- These algorithms rely on the hard to invert functions of factoring prime numbers and modulus.

Hard to Invert Functions

- What you want is a function which is hard to undo.
- That is, if you give me a number, I can compute the result easily. But if I give you just the result, you can't tell me the original number very easily.

Easy to Invert

Function 1:

You give me 2
I give you 4.
You give me 5.
I give you 10.
You give me 9.
I give you 18.

Now, to invert it, if I give you 22, you can quite easily tell me the answer. (11)!

Hard to Invert

Function 2:

You give me 2

I give you 1.

You give me 5.

I give you 2.

You give me 56.

I give you 35.

Now, if I give you 3, it is not so easy to see the pattern. I am using the modulus or remainder function.

Answers to above:

1. **Start with 707 / 2 remainder =1**
2. **Start with 707 / 5 remainder = 2**

Many possibilities for a number which will give a remainder of 3 when divided into 707.

Factoring Prime Numbers

- A **prime number** (or a **prime**) has exactly two *distinct* divisors: 1 and itself.
- The smallest twenty-five prime numbers (all the prime numbers under 100) are:

  2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83,89, 97

- Prime factorization is a list of all the prime-number factors of a given number.

- The prime factorization does not include 1, but does include every copy of every prime factor. For instance, the prime factorization of 8 is 2×2×2, not just "2". Yes, 2 is the only factor, but you need three copies of it to multiply back to 8, so the prime factorization includes all three copies

Example of RSA

- P = 61 <- first prime number (destroy this after computing E and D)
- Q = 53 <- second prime number (destroy this after computing E and D)
- PQ = 3233 <- modulus (give this to others)
- E = 17 <- public exponent (give this to others)
- D = 2753 <- private exponent (keep this secret!)

- Your public key is (E,PQ).
- Your private key is D.

- The encryption function is: encrypt(T) = (T^E) mod PQ = (T^17) mod 3233
- The decryption function is: decrypt(C) = (C^D) mod PQ = (C^2753) mod 3233

- To encrypt the plaintext value 123, do this: encrypt(123) = (123^17) mod 3233 = 337587917446653715596592958817679803 mod 3233 = 855

- To decrypt the ciphertext value 855, do this: decrypt(855) = (855^2753) mod 3233 = 123

Cracking RSA?

- A challenge was placed in Martin Gardnerôs column in 1977 in Scientific American in which the readers were invited to factor

  C=114,381,625,757,888,867,669,235,779,976,146,612,010,218,296,721,242,362,562,561,842,935,706,935,245,733,897,830,597,123,563,958,705,058,989,075,147,599,290,026,879,543,541

  into its two prime number factors and the encryption key = 9007

- The first solver was to win one hundred dollars.

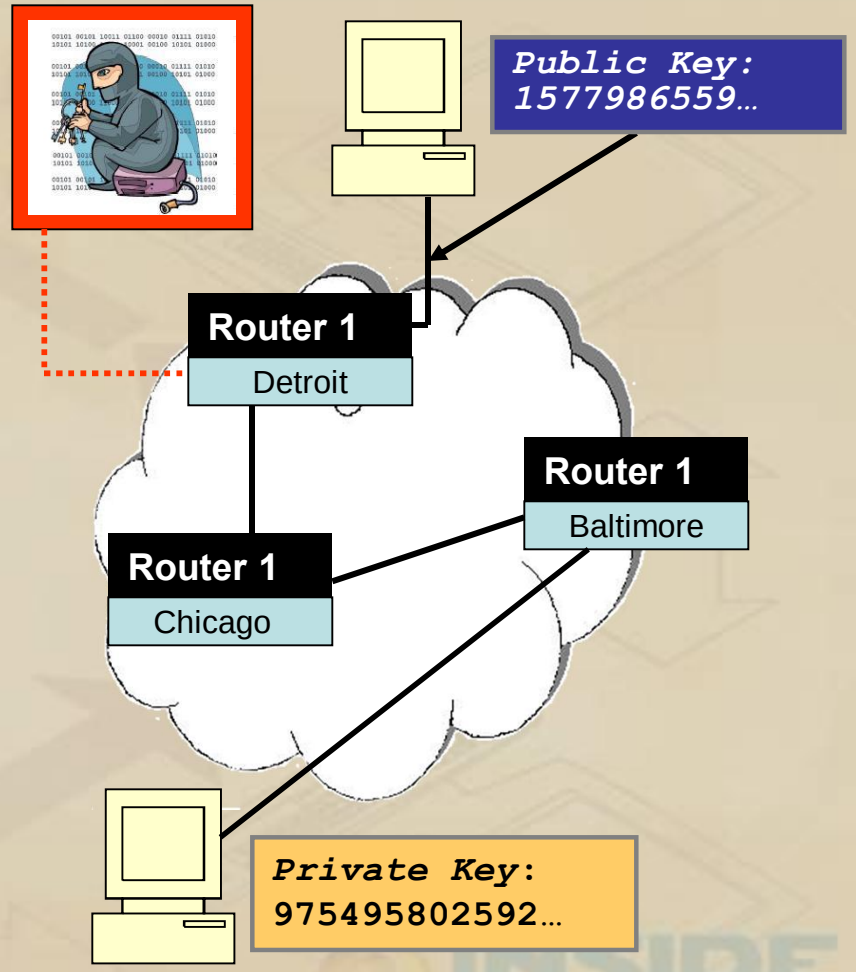
- This was solved 17 years later in April 26, 1994, cracked by an international effort via the internet with the use of 600 volunteers, workstations, mainframes, and supercomputers. They attacked the number above for eight months before finding its factors.

- The RSA algorithm is seen as safe.

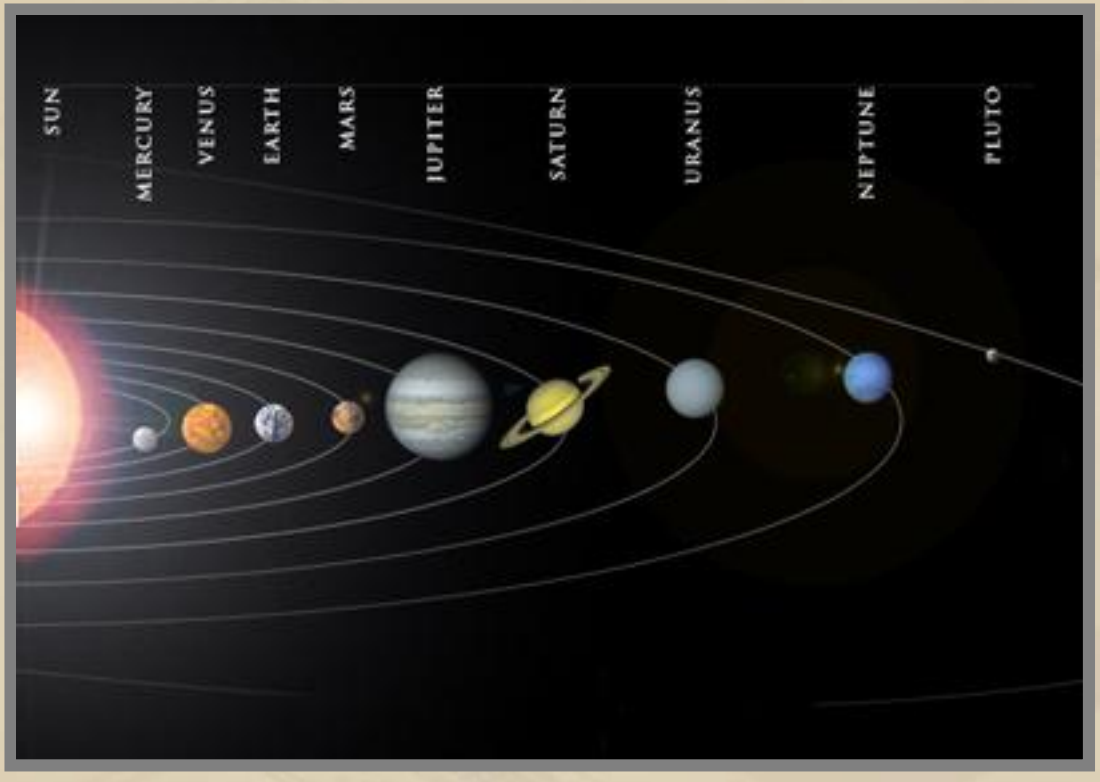
- Today, the numbers used for c are much larger.

Pros and Cons of PKI

- In public-key systems, there is no need to agree on a common key for both the sender and the receiver.
- Publishing the *public* key in no way compromises the secure transmission.
- If the private key is kept secret, no one but the receiver will be able to decrypt the messages encrypted with the corresponding public key.
- Public-key systems can guarantee integrity and authentication, not only privacy.
- The main disadvantage of public-key systems is that they are not as fast as symmetric algorithms



Elliptic Curve Cryptography

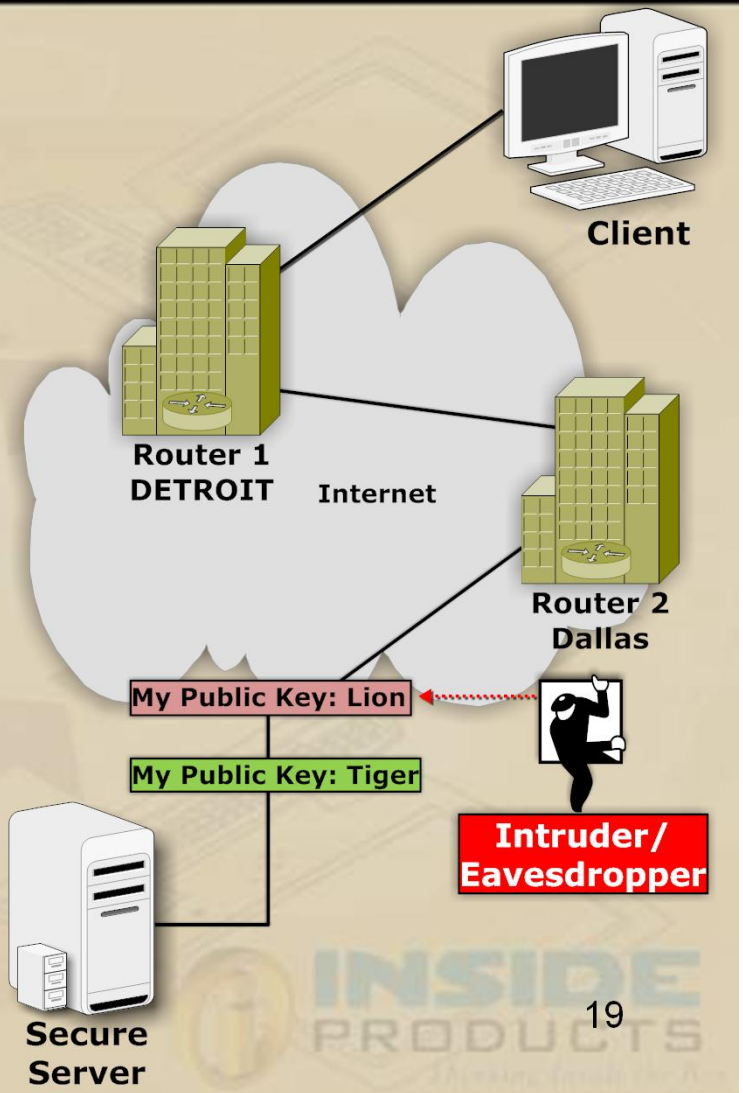


Thanks to Johannes Kepler, we know that the planets move in an elliptic orbit with the sun at one focus.

- One of the advances in cryptography is called Elliptic Curve Cryptography.
- In theory, elliptic-curve cryptography can use smaller keys and is more efficient than other public-key methods.
- With more and more devices requiring secure communications, speed and size matter.

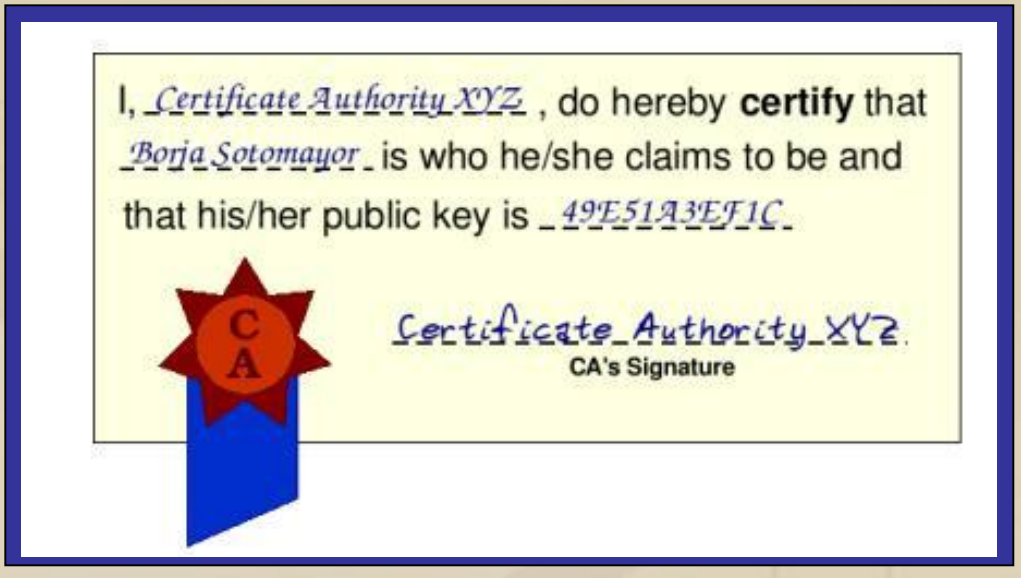
Why Digital Certificates?

- PKI does guarantee the authenticity to an extent.
- But this only means that whoever sent the message has the private key corresponding to the public key we used to decrypt the digital signature.
- How can we be sure that this public key really belongs to the sender?
- Maybe it is just someone impersonating the sender.
- When you want to be absolutely sure about a user's identity, you use a *digital certificate*.



Digital Certificates

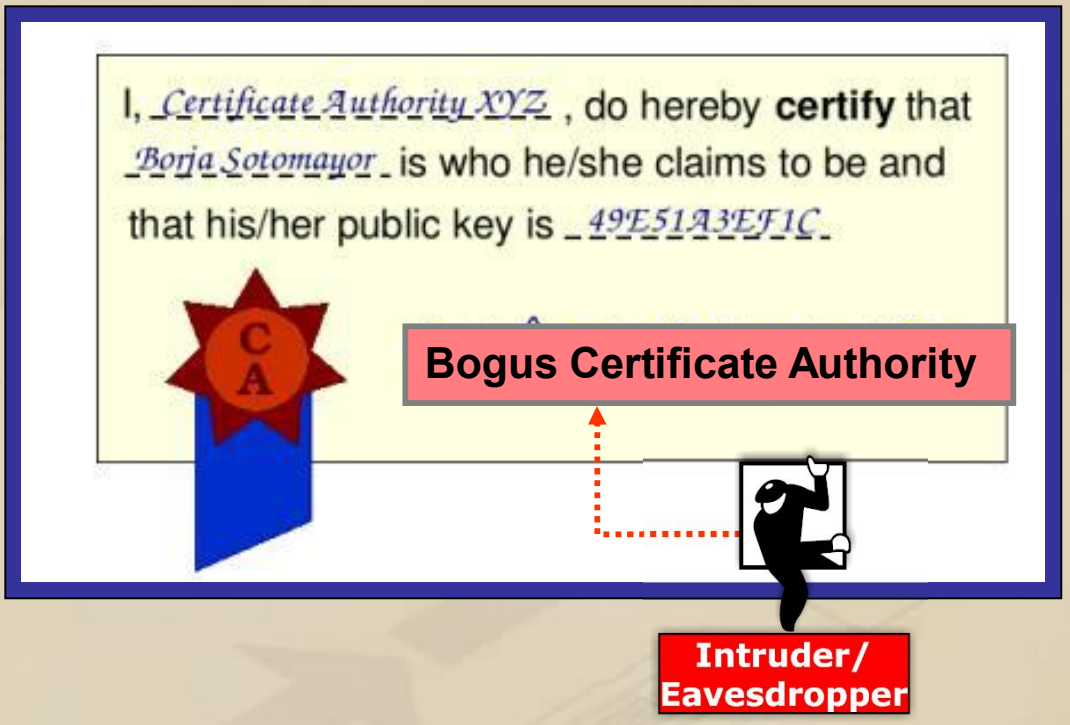
- A *digital certificate* is a digital document that *certifies* that a certain public key is owned by a particular user. This document is signed by a third party called the *certificate authority* (or CA).
- Of course, the certificate is encoded in a digital format.
- The important thing to remember is that the certificate is *signed* by a third party (the certificate authority) which does not itself take place in the secure conversation.



- The signature is actually a digital signature generated with the CA's private key.
- Therefore, we can verify the integrity of the certificate using the CA's public key.

Certificate Authorities

- Now, how can you *trust* the certificate? To be more exact, how can you *trust the CA that signs the certificate*.
- Believe it or not, there are no fancy algorithms to decide when a CA is trustworthy... you must decide by yourself whether you trust or don't trust a CA.
- This means that the public-key system you use will generally have a list of 'trusted CAs', which includes the digital certificates of those CAs you will trust (each of these certificates, in turn, include the CA's public key, so you can verify digital signatures).
- You have to decide.



Well-Known Certificate Authorities

- Some CAs are so well known that they are included by default in many public-key systems (for example, web browsers usually include VeriSign and GlobalSign certificates, because many websites use certificates issued by those companies to authenticate themselves to web browsers).
- Of course, you can add other CAs to the 'trusted list'.
- For example, if your department sets up a CA, and you *trust* that the department's CA will only issue certificates to trustworthy people, then you could add it to the list.

X.509 Certificate Format

- The format in which digital certificates are encoded is called the X.509 certificate format.
- An X.509 certificate is a plain text file which includes a lot of information in a very specific syntax.
- The four most important things we can find in an X.509 certificate:
  - **Subject**: This is the 'name' of the user. It is encoded as a *distinguished name* (the format for distinguished names will be explained next)
  - **Subject's public key**: This includes not only the key itself, but information such as the algorithm used to generate the public key.
  - **Issuer's Subject**: CA's distinguished name.
  - **Digital signature**: The certificate includes a digital signature of all the information in the certificate. This digital signature is generated using the CA's private key. To verify the digital signature, we need the CA's public key (which can be found in the CA's certificate).

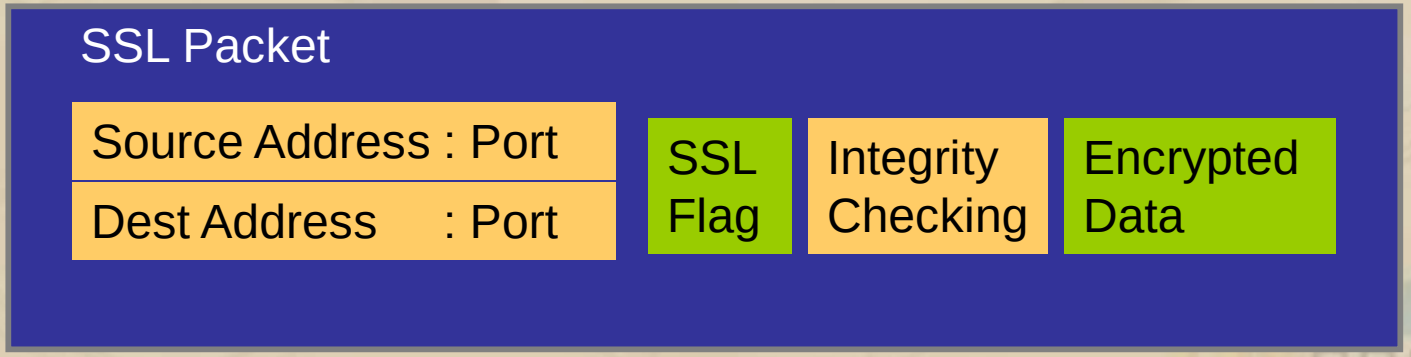
Certificates Expiring

- One of the biggest problems with certificates is that they are only valid for a certain amount of time.

- If a certificate expires, the connection will fail.

SSL Packet and Flow

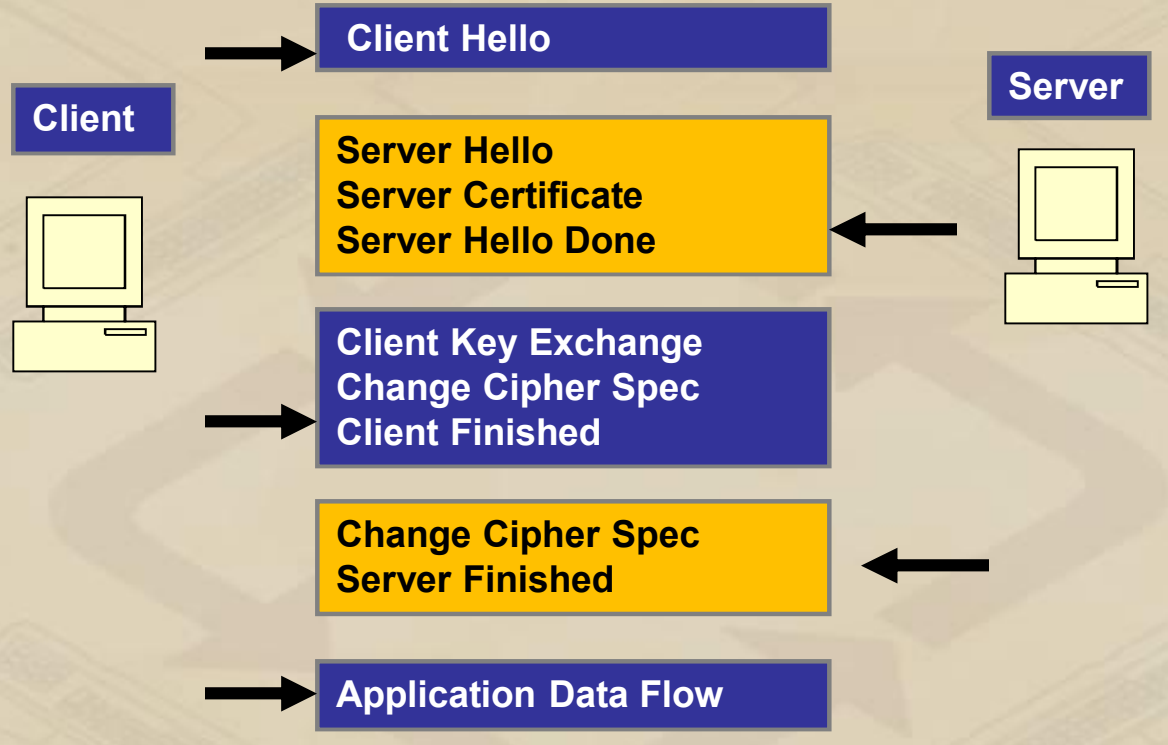
- A typical SSL packet may contain
  - SSL flags indicating the type of information transported - the type of *SSL message*.
  - Cryptographic integrity checking (typically using the MD5 or SHA-1 algorithm).
  - Encrypted data (typically using the RC2, RC4, DES, or Triple DES algorithms).
- An SSL connection begins with a handshake which uses asymmetric (public key) cryptography.
- The handshake is followed by a data transfer phase, also called the SSL *record protocol*, which uses symmetric cryptography.



SSL Handshake

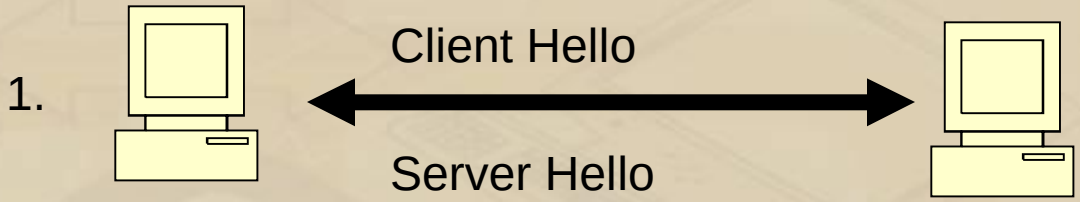
- An SSL connection begins with a handshake. As the name suggests, the handshake entails the initial setup. During the handshake, an exchange of information occurs that includes the following:
  - Authentication of the server.
  - Decision on how the data is to be encrypted.
  - Optionally, the authentication of the client.
- When the session begins, the client must know the public key of the server.
- No encryption is in use initially, so both parties (and any eavesdropper) can read this key, but the client can now transmit information to the server in a way that no one else could decode.

Packets in SSL Handshake Server Certificate

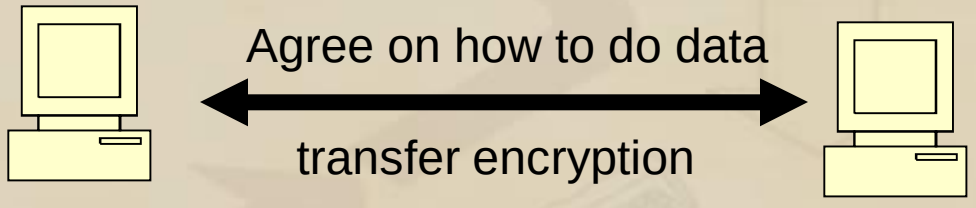


SSL Protocol Exchanges

1. The client connects to the server indicating that it wants to perform SSL. Contains sessionID. Server agrees ("server hello"). Handshake begins.

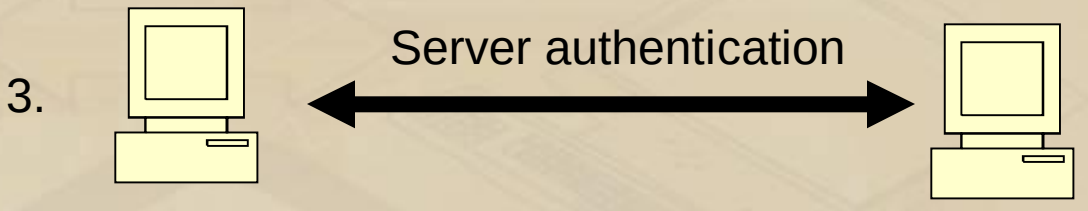


2. The client and server agree on a common symmetric algorithm to be used for the data transfer that follows the handshake. Both have a list of possible algorithms in the order of preference

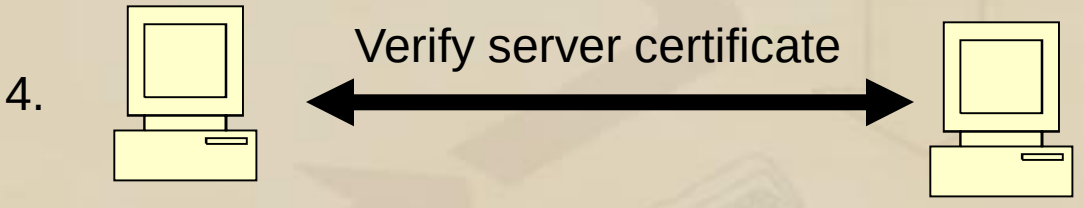


SSL Protocol Exchanges

3. The server provides its public key in its certificate to the client-this is also called *server authentication*, and is a required step of the SSL handshake.

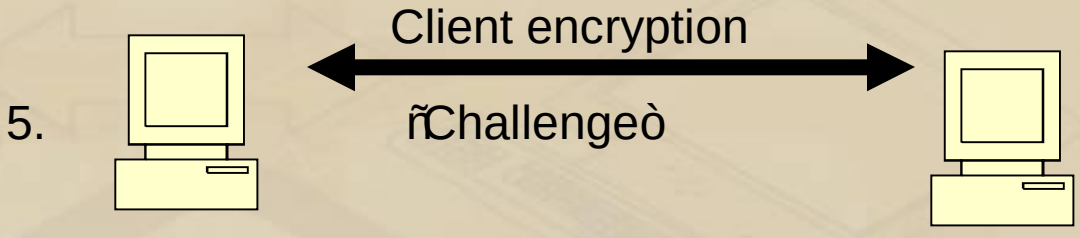


4. The client verifies the integrity of the server's certificate Depending on the client design, if this certificate is not available, the client may ask the end user to agree to either pursue the communication or to abort it.

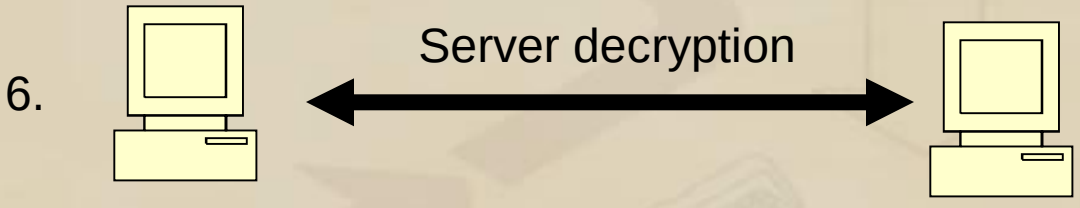


SSL Protocol Exchanges

5. The client now has the server's public key and uses it to encrypt a random number, which is then sent to the server.

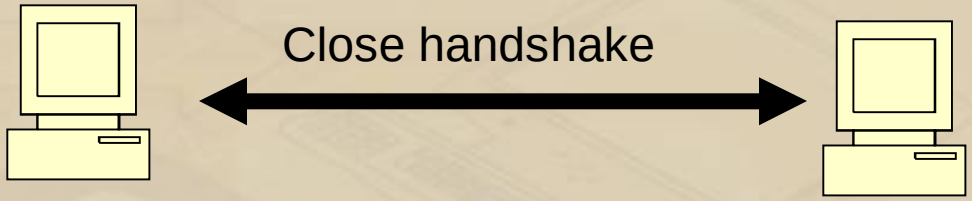


6. The server retrieves the value of this random number using its private key. The decryption of the secret random number using the server's private key can cost a great deal of computing resource during the SSL handshake.

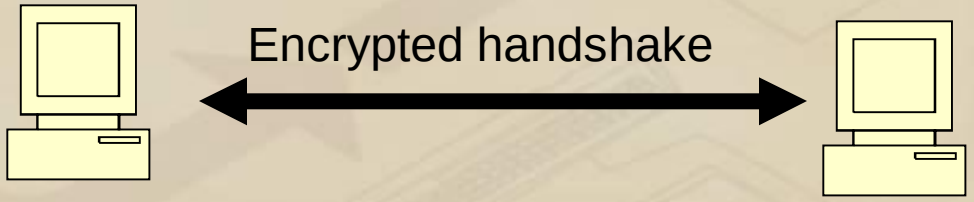


SSL Protocol Exchanges

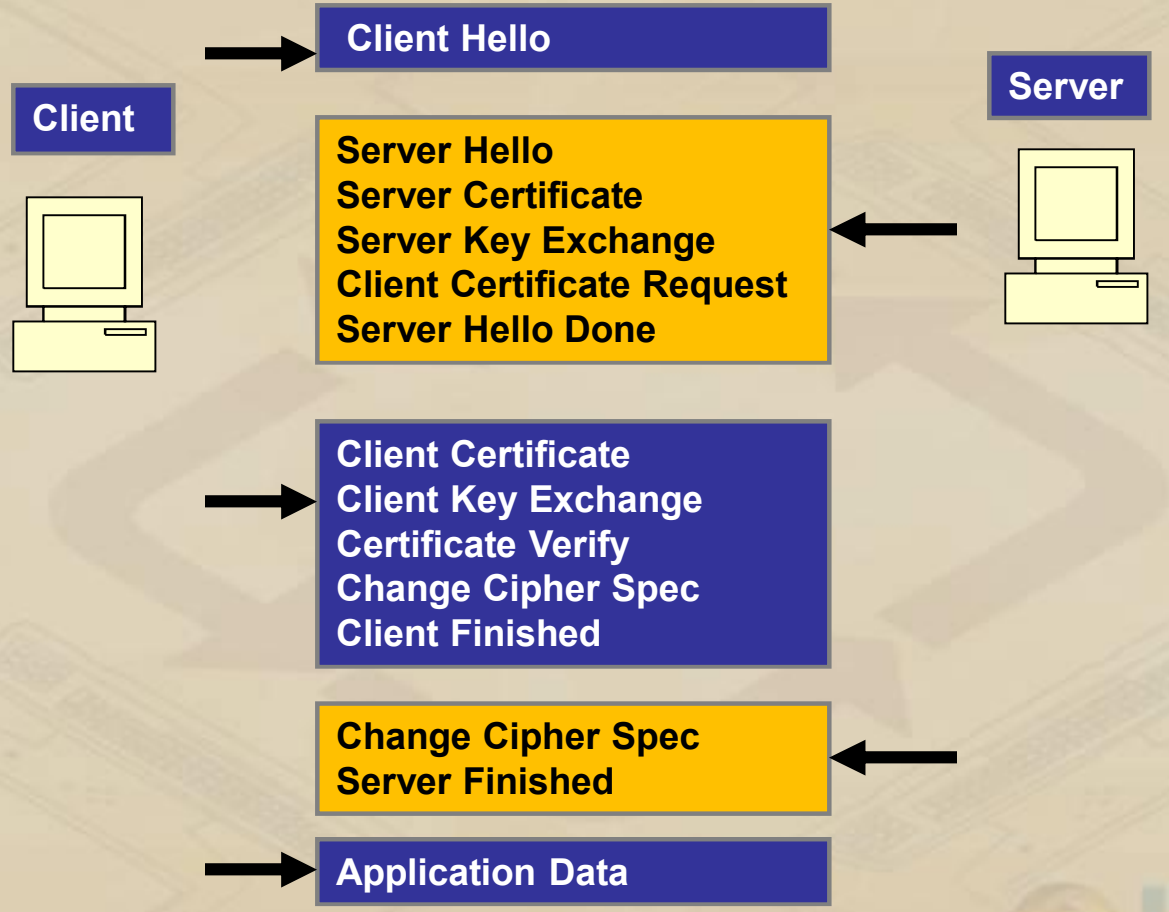
7. At this point only the client and server know this secret random number (the pre master secret), which can then be used to generate the keys to encrypt and decrypt the data, using the symmetric algorithm previously selected. The client and the server then close the handshake phase.

7. 

8. The data transfer phase (also called SSL record protocol) begins.

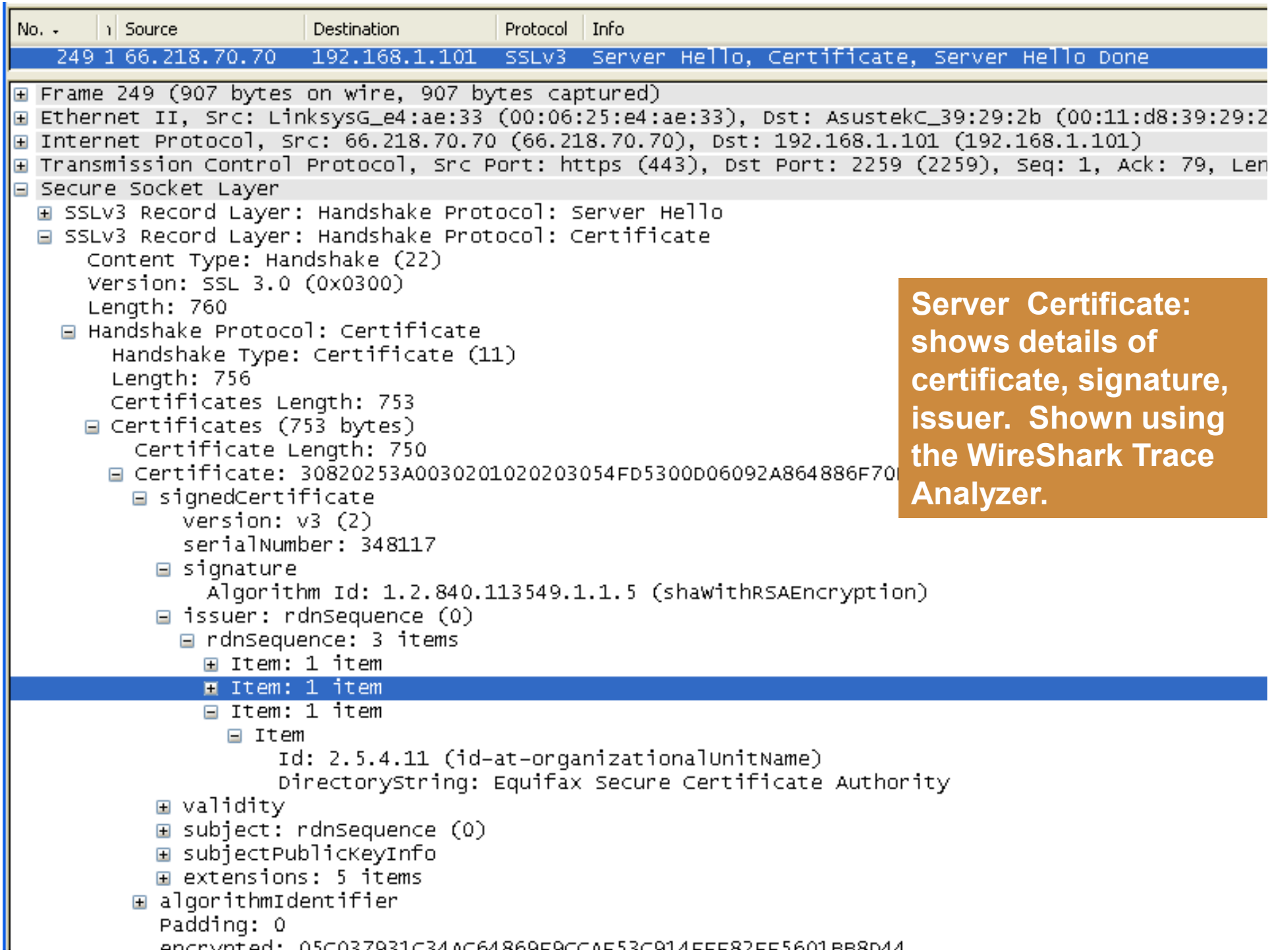
8. 

Packets in SSL Handshake Server and Client Certificates



SSL Hello Packet

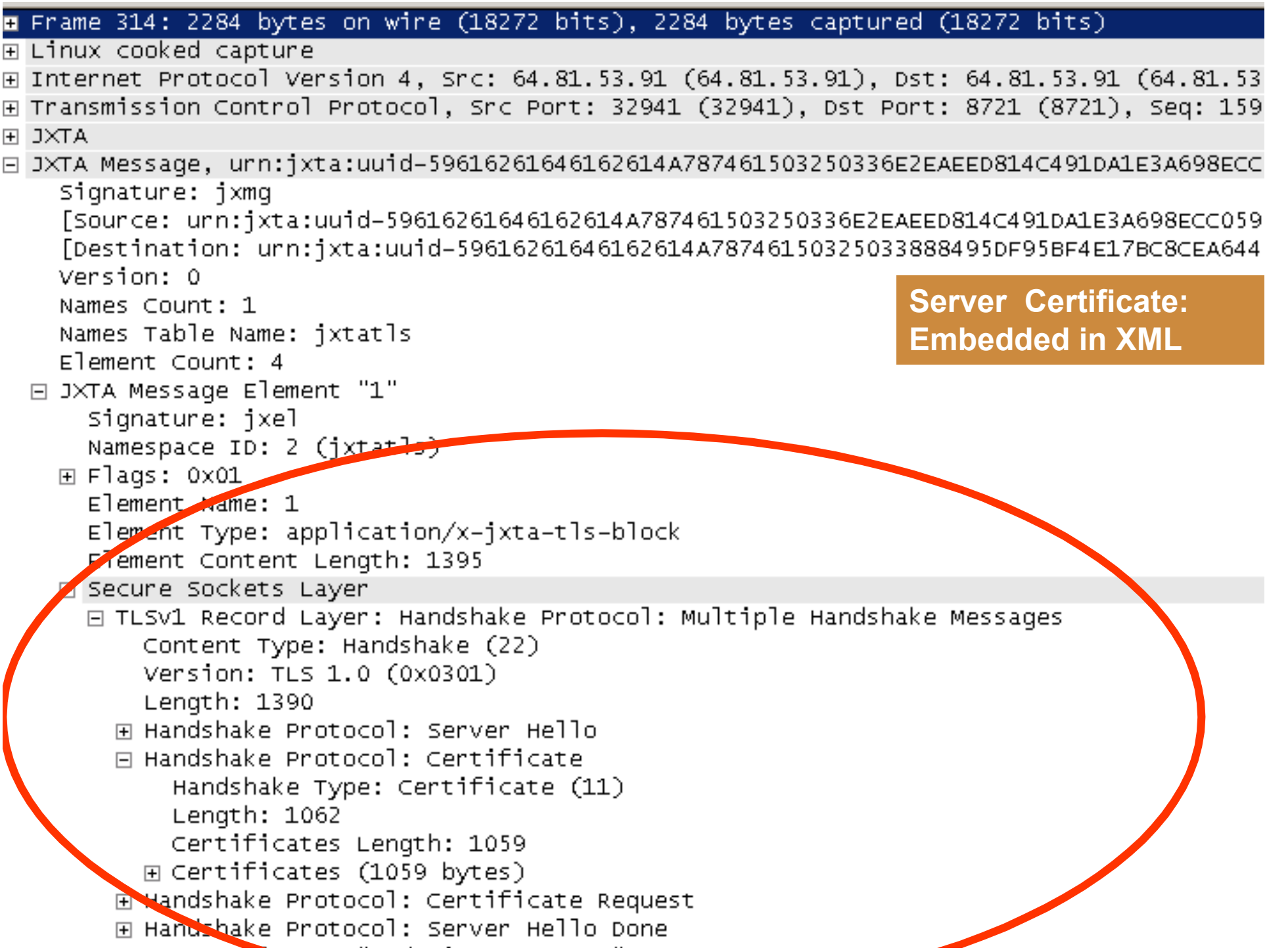

```
000000 16030000 35010000 31030049 8B12CB92 |................k ....5...1..I....|
       2FD3CA5B 5716BDEA 15FE6EA0 |..Y..L.$......>. "..>/...W.....n.|
       16BF30E0 B4462B00 000A0004 |...X............ .c0...0..F+.....|
       00060003 0100              |..........       ..........      |

SSLV3 Record Layer
Content type = Handshake Protocol (22)
Version = 3.0
Length = 53
010000310300498b12cb922203e83e2fd3ca5b5716bdea15fe6ea0cd6330e716bf30e0b4462b0000
Handshake Protocol : Client Hello(1)
Length : 49
Version :3.0
Random :
   gmt_unix_time : Saturday, 26 July 2008 05:00:11
   Random Bytes : 922203e83e2fd3ca5b5716bdea15fe6ea0cd6330e716bf30e0b4462b
Session ID Length : 0
Session ID :
Cipher Suites Length : 10
Cipher Suites : 5
Cipher Suite read in : 4
   Cipher Suite : TLS_RSA_WITH_RC4_128_MD5 (0x04)
Cipher Suite read in : 10
   Cipher Suite : TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x0A)
Cipher Suite read in : 9
   Cipher Suite : TLS_RSA_WITH_DES_CBC_SHA (0x09)
Cipher Suite read in : 6
   Cipher Suite : TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5 (0x06)
Cipher Suite read in : 3
   Cipher Suite : TLS_RSA_EXPORT_WITH_RC4_40_MD5 (0x03)
Compression Methods Length : 1
Compression Methods : 1
   Compression Method : null (0)
```

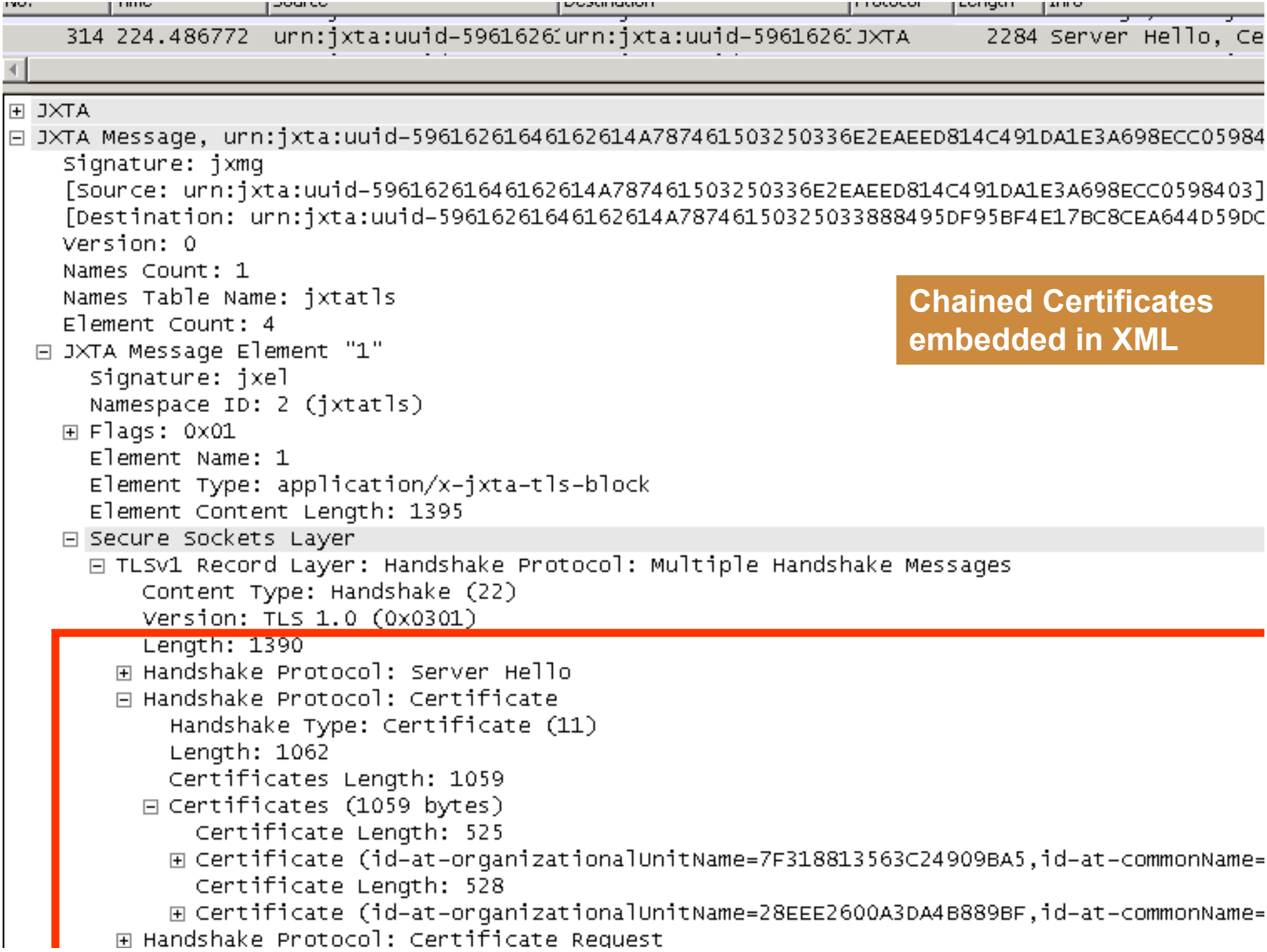

No. | Source | Destination | Protocol | Info
249 1 66.218.70.70 192.168.1.101 SSLv3 Server Hello, Certificate, Server Hello Done
Frame 249 (907 bytes on wire, 907 bytes captured)
Ethernet II, Src: LinksysG_e4:ae:33 (00:06:25:e4:ae:33), Dst: AsustekC_39:29:2b (00:11:d8:39:29:2
Internet Protocol, Src: 66.218.70.70 (66.218.70.70), Dst: 192.168.1.101 (192.168.1.101)
Transmission Control Protocol, Src Port: https (443), Dst Port: 2259 (2259), Seq: 1, Ack: 79, Len
Secure Socket Layer
SSLv3 Record Layer: Handshake Protocol: Server Hello
SSLv3 Record Layer: Handshake Protocol: Certificate
Content Type: Handshake (22)
Version: SSL 3.0 (0x0300)
Length: 760
Handshake Protocol: Certificate
Handshake Type: Certificate (11)
Length: 756
Certificates Length: 753
Certificates (753 bytes)
Certificate Length: 750
Certificate: 30820253A003020102020203054FD5300D06092A864886F70
signedCertificate
version: v3 (2)
serialNumber: 348117
signature
Algorithm Id: 1.2.840.113549.1.1.5 (shaWithRSAEncryption)
issuer: rdnSequence (0)
rdnSequence: 3 items
Item: 1 item
Item: 1 item
Item: 1 item
Item
Id: 2.5.4.11 (id-at-organizationalUnitName)
DirectoryString: Equifax Secure Certificate Authority
validity
subject: rdnSequence (0)
subjectPublicKeyInfo
extensions: 5 items
algorithmIdentifier
Padding: 0
encrypted: 05C037931C34AC64869F9CCAF53C914FFF82FF5601BB8D44
Server Certificate: shows details of certificate, signature, issuer. Shown using the WireShark Trace Analyzer.

Handshake Analysis

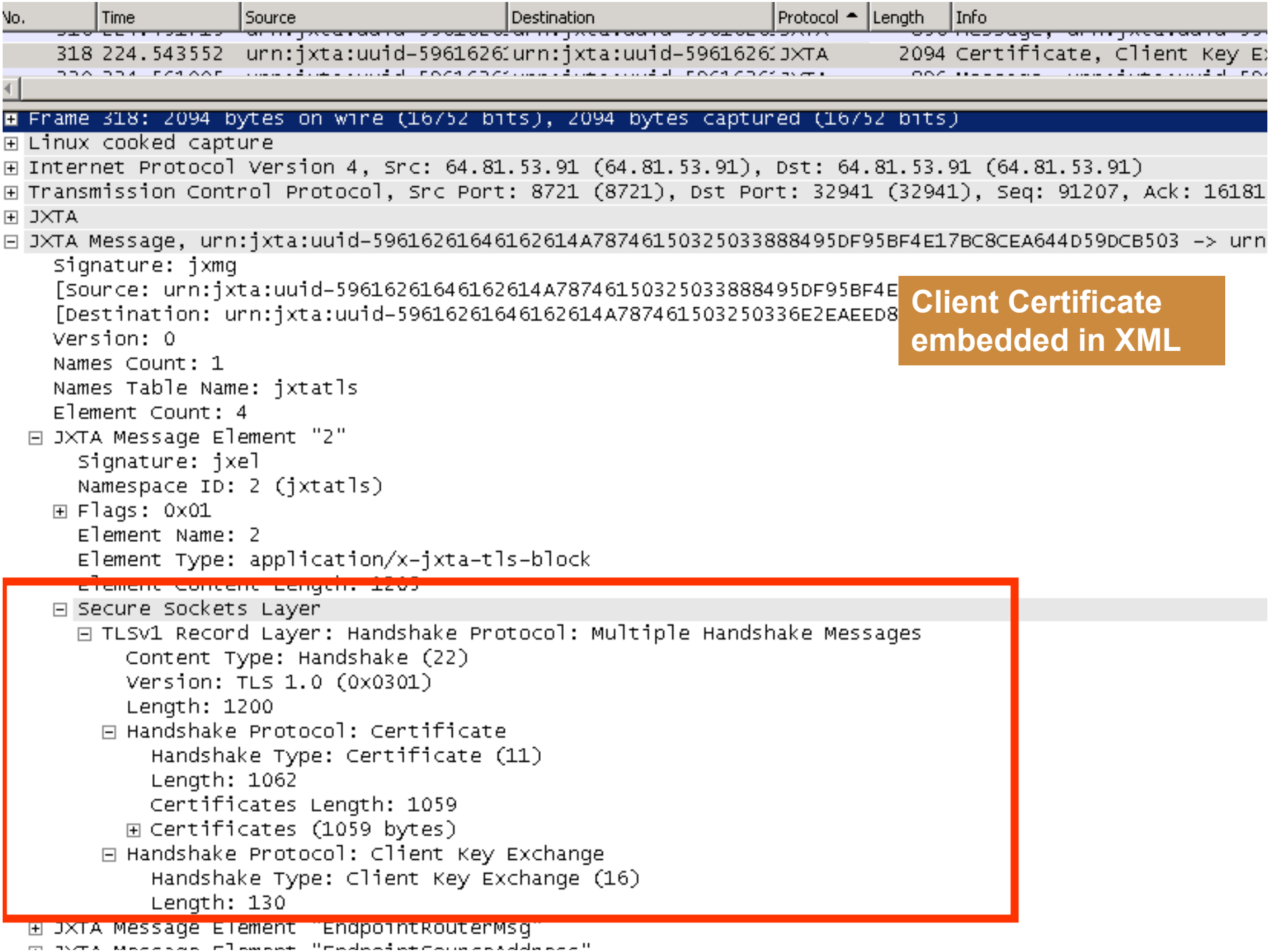
Handshake from client IP address:10.146.12.31 port: 4604 to server IP address 123.36.135.6 port: 23024 did not complete properly.

- The Client Hello was sent at 2010-01-13 12:06:43.993334.
- The next packet expected is the Server Hello.
- A Server Hello was sent at: 2010-01-13 12:06:43.995621.
- The Cipher Suite used is: TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x0A).
- The next packet expected is from the server and it is a Server Certificate.
- A Server Certificate was sent at: 2010-01-13 12:06:43.995621.
- This indicates that agreement was reached between the client and server on the Cipher Suite.
- The Server Certificate may be sent in multiple packets.
- The next packet expected is from the server and it is a Server Done packet.
- No Server Done packet was sent.
- The Server Certificate may not have been completely sent. There may be a problem at the server.


Frame 314: 2284 bytes on wire (18272 bits), 2284 bytes captured (18272 bits)
Linux cooked capture
Internet Protocol Version 4, Src: 64.81.53.91 (64.81.53.91), Dst: 64.81.53.91 (64.81.53
Transmission Control Protocol, Src Port: 32941 (32941), Dst Port: 8721 (8721), Seq: 159
JXTA
JXTA Message, urn:jxta:uuid-59616261646162614A787461503250336E2EAEED814C491DA1E3A698ECC
Signature: jxmg
[Source: urn:jxta:uuid-59616261646162614A787461503250336E2EAEED814C491DA1E3A698ECC059
[Destination: urn:jxta:uuid-59616261646162614A7874615032503388495DF95BF4E17BC8CEA644
Version: 0
Names Count: 1
Names Table Name: jxtatls
Element Count: 4
JXTA Message Element "1"
Signature: jxel
Namespace ID: 2 (jxtatls)
Flags: 0x01
Element Name: 1
Element Type: application/x-jxta-tls-block
Element Content Length: 1395
Secure Sockets Layer
TLSv1 Record Layer: Handshake Protocol: Multiple Handshake Messages
Content Type: Handshake (22)
Version: TLS 1.0 (0x0301)
Length: 1390
Handshake Protocol: Server Hello
Handshake Protocol: Certificate
Handshake Type: Certificate (11)
Length: 1062
Certificates Length: 1059
Certificates (1059 bytes)
Handshake Protocol: Certificate Request
Handshake Protocol: Server Hello Done
Server Certificate: Embedded in XML


314 224.486772 urn:jxta:uuid-5961626 urn:jxta:uuid-5961626 JXTA 2284 Server Hello, Ce
JXTA
JXTA Message, urn:jxta:uuid-596162616461626 14A787461503250336E2EAEED814C491DA1E3A698ECC05984
Signature: jxmg
[Source: urn:jxta:uuid-596162616461626 14A787461503250336E2EAEED814C491DA1E3A698ECC0598403]
[Destination: urn:jxta:uuid-596162616461626 14A787461503250338884 95DF95BF4E17BC8CEA644D59DC
Version: 0
Names Count: 1
Names Table Name: jxtatls
Element Count: 4
JXTA Message Element "1"
Signature: jxel
Namespace ID: 2 (jxtatls)
Flags: 0x01
Element Name: 1
Element Type: application/x-jxta-tls-block
Element Content Length: 1395
Secure Sockets Layer
TLSv1 Record Layer: Handshake Protocol: Multiple Handshake Messages
Content Type: Handshake (22)
Version: TLS 1.0 (0x0301)
Length: 1390
Handshake Protocol: Server Hello
Handshake Protocol: Certificate
Handshake Type: Certificate (11)
Length: 1062
Certificates Length: 1059
Certificates (1059 bytes)
Certificate Length: 525
Certificate (id-at-organizationalUnitName=7F318813563C24909BA5,id-at-commonName=
Certificate Length: 528
Certificate (id-at-organizationalUnitName=28EEE2600A3DA4B889BF,id-at-commonName=
Handshake Protocol: Certificate Request
Chained Certificates embedded in XML


314 224.486772 urn:jxta:uuid-5961626 urn:jxta:uuid-5961626 JXTA 2284 S
Handshake Type: Certificate (11)
Length: 1062
Certificates Length: 1059
Certificates (1059 bytes)
Certificate Length: 525
Certificate (id-at-organizationalUnitName=7F318813563C24909BA5,ic
signedCertificate
version: v3 (2)
serialNumber: 1
signature (shaWithRSAEncryption)
issuer: rdnSequence (0)
validity
notBefore: utcTime (0)
utcTime: 05-06-09 03:02:26 (UTC)
notAfter: utcTime (0)
utcTime: 15-06-09 03:02:26 (UTC)
subject: rdnSequence (0)
subjectPublicKeyInfo
algorithmIdentifier (shaWithRSAEncryption)
Padding: 0
encrypted: 62a1efe6d027ccd2c0d851000c47913f8a72659bc1bb7d45...
Certificate Length: 528
Certificate (id-at-organizationalUnitName=28EEE2600A3DA4B889BF,ic
signedCertificate
version: v3 (2)
serialNumber: 1
Server Certificate expiration dates
Format is:
YY-MM-DD


No.
Time
Source
Destination
Protocol
Length
Info
318 224.543552 urn:jxta:uuid-5961626 urn:jxta:uuid-5961626 JXTA 2094 Certificate, Client Key E
Frame 318: 2094 bytes on wire (16752 bits), 2094 bytes captured (16752 bits)
Linux cooked capture
Internet Protocol Version 4, Src: 64.81.53.91 (64.81.53.91), Dst: 64.81.53.91 (64.81.53.91)
Transmission Control Protocol, Src Port: 8721 (8721), Dst Port: 32941 (32941), Seq: 91207, Ack: 16181
JXTA
JXTA Message, urn:jxta:uuid-59616261646162614A787461503250338884 95DF95BF4E17BC8CEA644D59DCB503 -> urn
Signature: jxmg
[Source: urn:jxta:uuid-59616261646162614A787461503250338884 95DF95BF4E
[Destination: urn:jxta:uuid-59616261646162614A787461503250336E2EAEED8
Version: 0
Names Count: 1
Names Table Name: jxtatls
Element Count: 4
JXTA Message Element "2"
Signature: jxel
Namespace ID: 2 (jxtatls)
Flags: 0x01
Element Name: 2
Element Type: application/x-jxta-tls-block
Secure Sockets Layer
TLSv1 Record Layer: Handshake Protocol: Multiple Handshake Messages
Content Type: Handshake (22)
Version: TLS 1.0 (0x0301)
Length: 1200
Handshake Protocol: Certificate
Handshake Type: Certificate (11)
Length: 1062
Certificates Length: 1059
Certificates (1059 bytes)
Handshake Protocol: Client Key Exchange
Handshake Type: Client Key Exchange (16)
Length: 130
JXTA Message Element "EndpointRouterMsg"
Client Certificate embedded in XML

What are the problems?

- What are the measurement problems?
  - What metrics do we collect?
  - How?
  - Where?
- Can't do without secure protocols!
- Certificates are being used in more and more places.
- Problems:
  - Complexity
  - Certificates are everywhere!
  - Centralized monitoring (easy to say!)

Summary

Å It will only get worse!

Case Study

- Case Study #1
- SSL Application Concurrent Sessions

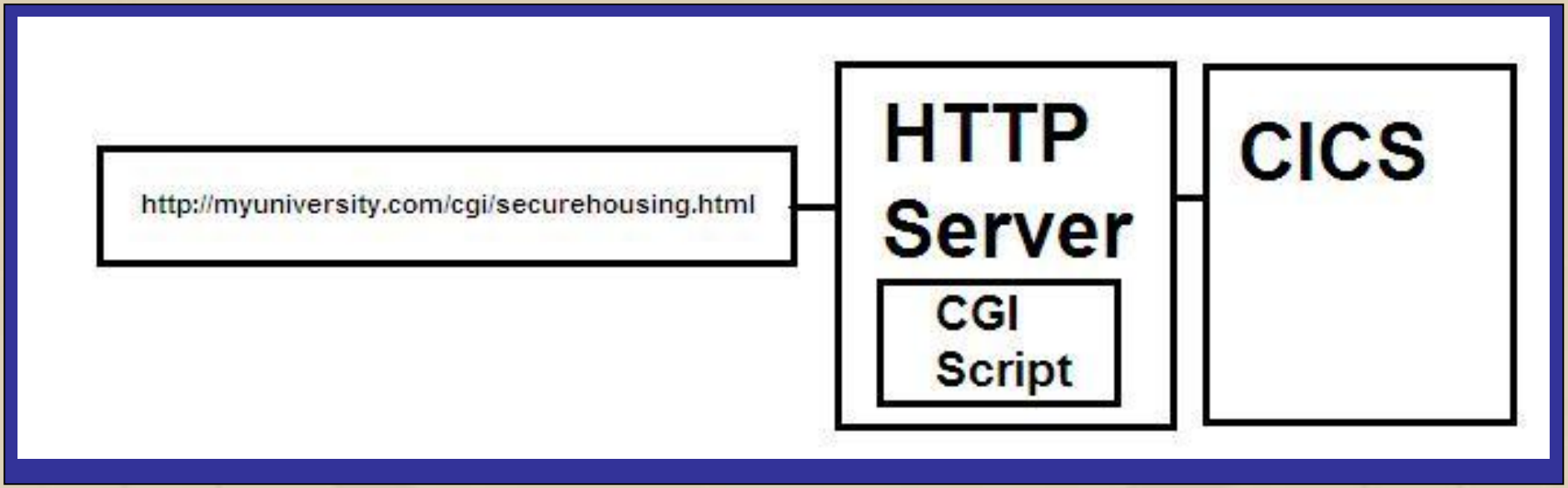
Problem Using SSL

- Large 4-year public university has a problem using a mainframe TCP application using Secure Sockets Layer (SSL).
- At times when students registered for a dormitory room using their CICS web-enabled application, the CICS region became ôhungö and had to be restarted.
- Students were also unable to use the application and many calls were received at the Help Desk.

Staging Use of Application

- Application allowed students to select residence hall, roommate, meal plan and other related matters.
- Application was available for one week out of the year.
- Tried to stage the use of the application by having honors students and Seniors sign up the first day, Juniors, the following day etc.
- Application worked fine until the last day when students with poor grades, freshmen, and all others were allowed in.
- Then, chaos erupted.

Application Architecture



- Student logs on to a web page such as: http://myuniversity.com/cgi/securehousing.html.
- Web page is actually a CGI script running under the HTTP Web server on the mainframe.
- Port used was port 443 for secure sockets.
- CGI script initiated a connection to CICS to get data and pass it back.

SSL Security Directives

We looked at the SSL security directives in the HTTP WebServer. They had coded the SSL Cipher Specs to go from highest strength to lowest. We found some documentation for another web server that led us to believe that this may make the SSL handshake longer.

```
SSLCipherSpec directive
# Specify the methods of encryption that an SSL connection will support. Each
# encoded cipher specification is tested in the order specified for
# compatibility with the requester. If the requester supports a method specified
# here, an SSL connection can be established. If not, the connection is refused.
SSLCipherSpec 39
SSLCipherSpec 27
SSLCipherSpec 21
SSLCipherSpec 23
SSLCipherSpec 26
SSLCipherSpec 22
SSLCipherSpec 24
SSLCipherSpec 3A
```

SSL Cipher Specs

```
#       Syntax:   SSLCipherSpec <code>#  where <code> is one of:  SSL V2:
#
#                 Code    Meaning                  Note     Strength
#                 ====    =============            ====     ========
#                  21     RC4 (128 bit)              *      (weaker)
#                  22     RC4 (40 bit)
#                  23     RC2 (128 bit)              *          |
#                  24     RC2 (40 bit)                          V
#                  26     DES (56 bit)               *
#                  27     Triple DES (192 bit)       *     (stronger)
#
#                 SSL V3:
#                 Code    Meaning                  Note     Strength
#                 ====    =============            ====     ========
#                  33     RC4 MD5 (128 bit)                 (weaker)
#                  34     RC4 MD5 (128 bit)          *
#                  35     RC4 SHA (128 bit)          *          |
#                  36     RC2 MD5 (40 bit)                      V
#                  39     DES SHA (56 bit)
#                  3A     Triple DES SHA (192 bit)   *     (stronger)
```

* Note: Not supported in versions available outside North America.

SSL Strength

- Set SSL Cipher Specs to go from lowest strength to highest instead of highest strength to lowest.
  - SSLCipherSpec 21
  - SSLCipherSpec 22
  - SSLCipherSpec 23
  - SSLCipherSpec 24
  - SSLCipherSpec 33
  - SSLCipherSpec 35 etc
- This fixed the problem a great deal.
- Should SSL be done in the Cryptography hardware?