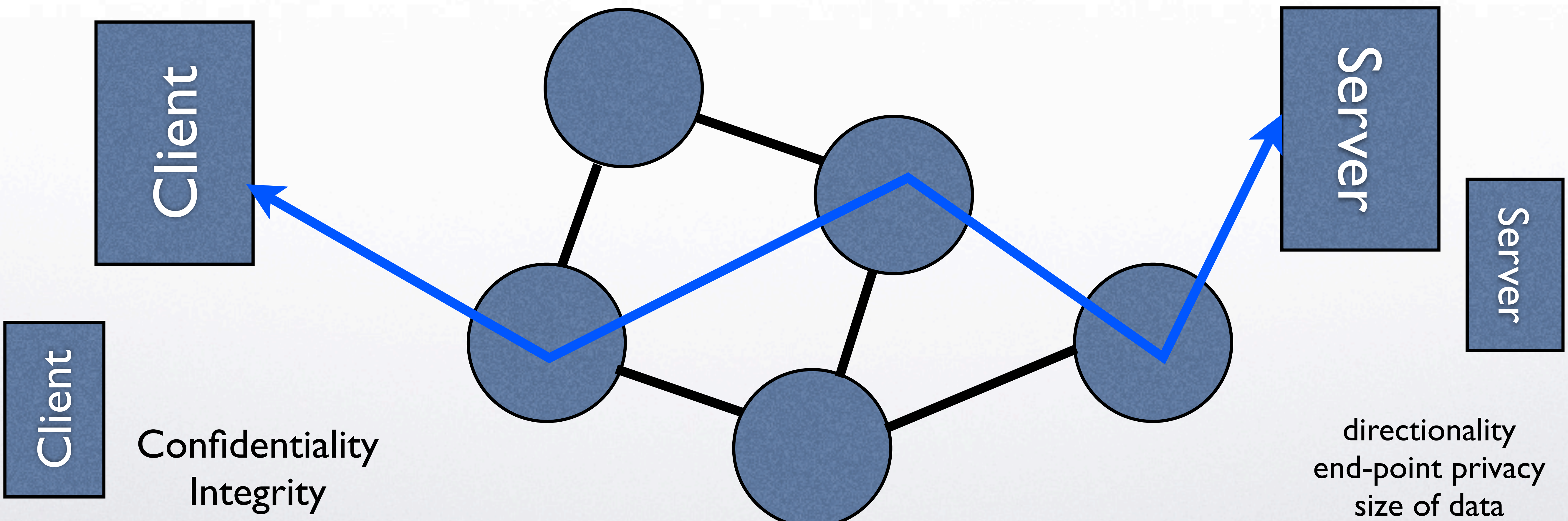
SSL/TLS/X.509

Aggelos Kiayias

Objective

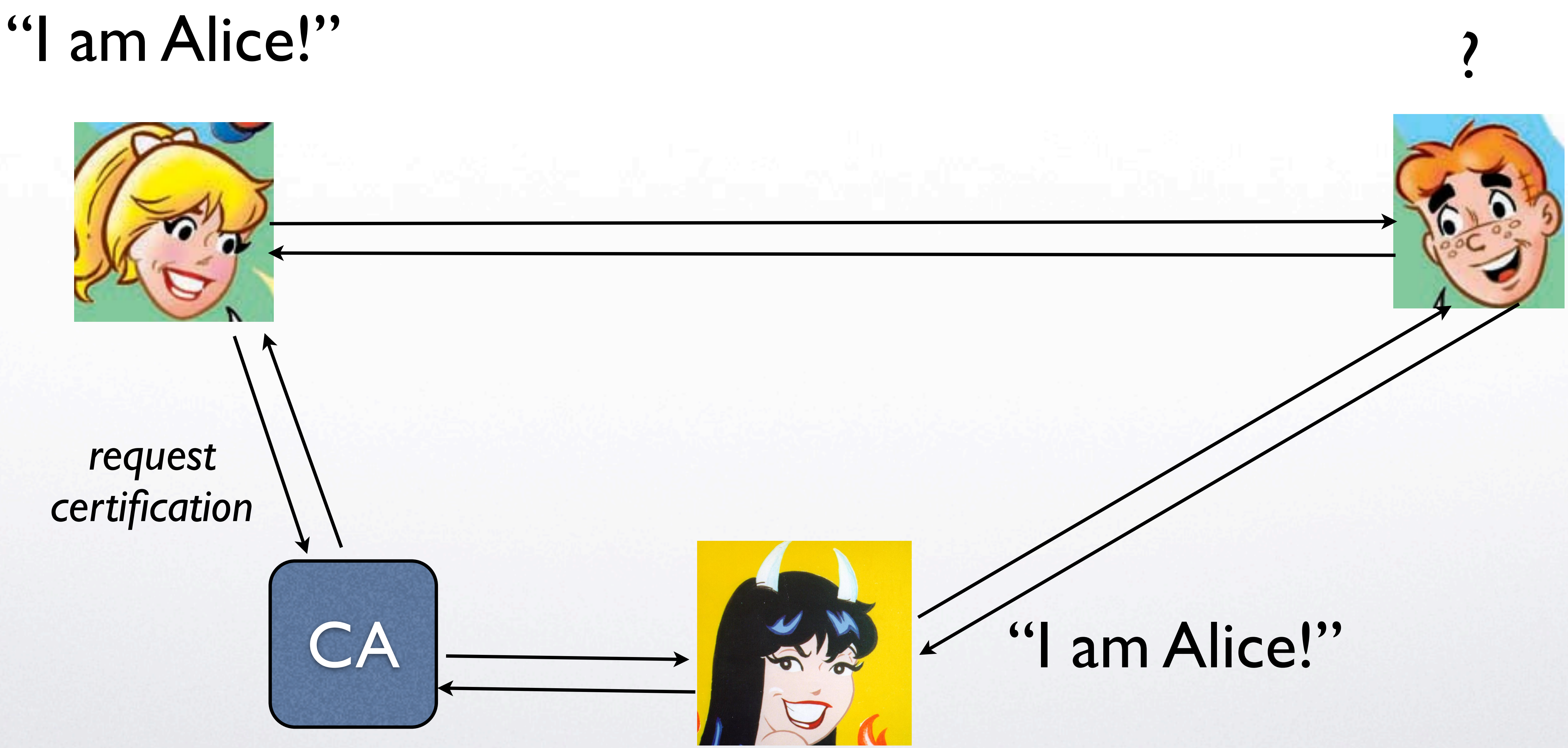
- Build a point to point secure channel



Confidentiality
Integrity
YES

directionality
end-point privacy
size of data
NO

Identification Problem



SSL/TLS

- Secure Sockets Layer: Developed by Netscape.
- SSL Version 3 released in 1996.
- Substituted by TLS in 1999: (Transport Layer Security). Standardized by IETF. published as RFC 2246.
- On top of TCP/IP, below application protocols.

TLS

- The Record protocol
  - encapsulates higher level protocols
- The Handshake protocol
  - enables authentication and session key establishment.

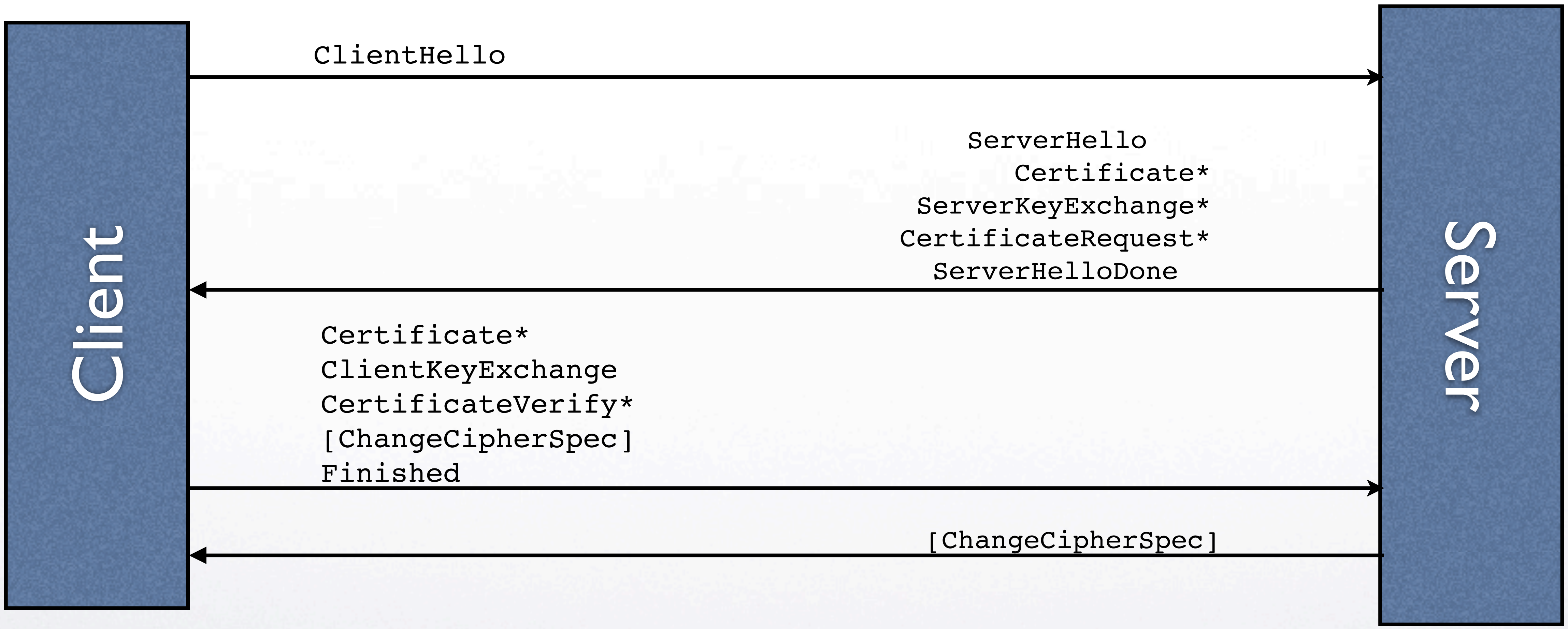
TLS Connections Parameters

- Connection end (client, server)
- Bulk encryption algorithm
- MAC algorithm
- Compression algorithm
- Master secret (48 bytes)
- Client random (32 byte)
- Server random (32 byte)

TLS Handshake

- Identities can be authenticated using public-key means. Authentication is optional but usually required at least for server side.
- Session-key is generated by public-key techniques.
- Special protocol modifications allow robustness against modification and man-in-the-middle attacks.

TLS Handshake



* = optional or situation-dependent

TLS Handshake, II

- The `ClientHello` message:
  - Random 28 byte string.
  - Client Timestamp
  - SessionID array (can be empty / it is used for resuming a previous session)
  - Ciphersuite list.
  - Compression list.

```
struct {
    ProtocolVersion client_version;
    Random random;
    SessionID session_id;
    CipherSuite cipher_suites<2..2^16-1>;
    CompressionMethod compression_methods<1..2^8-1>;
      } ClientHello;
```

Example - ClientHello

```
ClientVersion 3,1
ClientRandom[32]
SessionID: None (new session)
Suggested Cipher Suites:
   TLS_RSA_WITH_3DES_EDE_CBC_SHA
   TLS_RSA_WITH_DES_CBC_SHA
Suggested Compression Algorithm: NONE
```

RSA for key exchange
3DES symmetric cipher in EDE mode
CBC encryption mode
SHA = hash algorithm

as above but DES is used.
this is not as secure
(will only use a 56 bit key as opposed to 168 bits for 3DES)

format:

TLS_[handshake specs]_WITH_[record specs]

TLS Handshake III

- The `ServerHello` message:

taking client's suggestions into account

highest version
strongest protocol

```
Structure of this message:
      struct {
          ProtocolVersion server_version;
          Random random;
          SessionID session_id;
          CipherSuite cipher_suite;
          CompressionMethod compression_method;
      } ServerHello;
```

of the same format as of client's

one choice

will either match
client's suggestion (resume)
indicate a new session_id
or will be NULL

Example - ServerHello

```
Version 3,1
ServerRandom[32]
SessionID:
bd608869f0c629767ea7e3ebf7a63bdcffb0ef58b1b941e6b0c044acb6820a77
Use Cipher Suite:
TLS_RSA_WITH_3DES_EDE_CBC_SHA
Compression Algorithm: NONE
```

TLS Handshake IV

- The `Certificate` message: the server will provide its certificate to authenticate its name on the public-key it provides to the client.

```
opaque ASN.1Cert<1..2^24-1>;

struct {
    ASN.1Cert certificate_list<0..2^24-1>;
} Certificate;
```

chain of certificates

```
The end entity certificate's public key (and associated
      restrictions) MUST be compatible with the selected key exchange
      algorithm.
```

server [certificate]

Key Exchange Alg.	Required Certificate Key Type
RSA RSA_PSK	RSA public key; the certificate MUST allow the key to be used for encryption (the keyEncipherment bit MUST be set if the key usage extension is present).
DHE_RSA ECDHE_RSA	RSA public key; the certificate MUST allow the key to be used for signing (the digitalSignature bit MUST be set if the key usage extension is present) with the signature scheme and hash algorithm that will be employed in the server key exchange message. Note: ECDHE_RSA is defined in [TLSECC].
DHE_DSS	DSA public key; the certificate MUST allow the key to be used for signing with the hash algorithm that will be employed in the server key exchange message.
DH_DSS DH_RSA	Diffie-Hellman public key; the keyAgreement bit MUST be set if the key usage extension is present.

[serverkeyexchange]

- `ServerKeyExchange :` will contain public key information in case the information in the `Certificate` message is not sufficient (or this message has not been provided at all).

[serverkeyexchange]

required when TLS follows: `DHE_DSS` `DHE_RSA` `DH_anon`

illegal when TLS follows: `RSA` `DH_DSS` `DH_RSA`

In this case it will contain $\langle p, g, g^x \bmod p \rangle$

signed with DSS or RSA signing algorithm with the public-key that was provided in the [certificate] except for `DH_anon` where no signature is given

in these cases the [certificate] has enough info for the key-exchange to complete

[certificaterequest]

- `CertificateRequest :` will prompt the client to send a certificate to authenticate itself (typically not used).

TLS Handshake, V

- The `ClientKeyExchange` message

In case of RSA, client encrypts a random 46 byte string with the public-key of the server

In case of DH, client prepares his public-key according to the exchange.

```
struct {
           select (KeyExchangeAlgorithm) {
               case rsa: EncryptedPreMasterSecret;
               case diffie_hellman: ClientDiffieHellmanPublic;
           } exchange_keys;
       } ClientKeyExchange;
```

TLS Handshake, VI

Client Authentication

- `CertificateRequest` : server will use this message to request a certificate-based authentication from the client.
- `Certificate` : response to a `CertificateRequest` message. This will be sent before `ClientKeyExchange`
- `CertificateVerify` : if client's certificate has signing capability, this message will be a digital signature of all handshake messages so far. This will be sent after `ClientKeyExchange`

TLS Handshake VII

The Master-Secret (48 bytes):

```
master_secret = PRF(pre_master_secret, "master secret",
                          ClientHello.random + ServerHello.random)
```

Computed depending on key exchange method

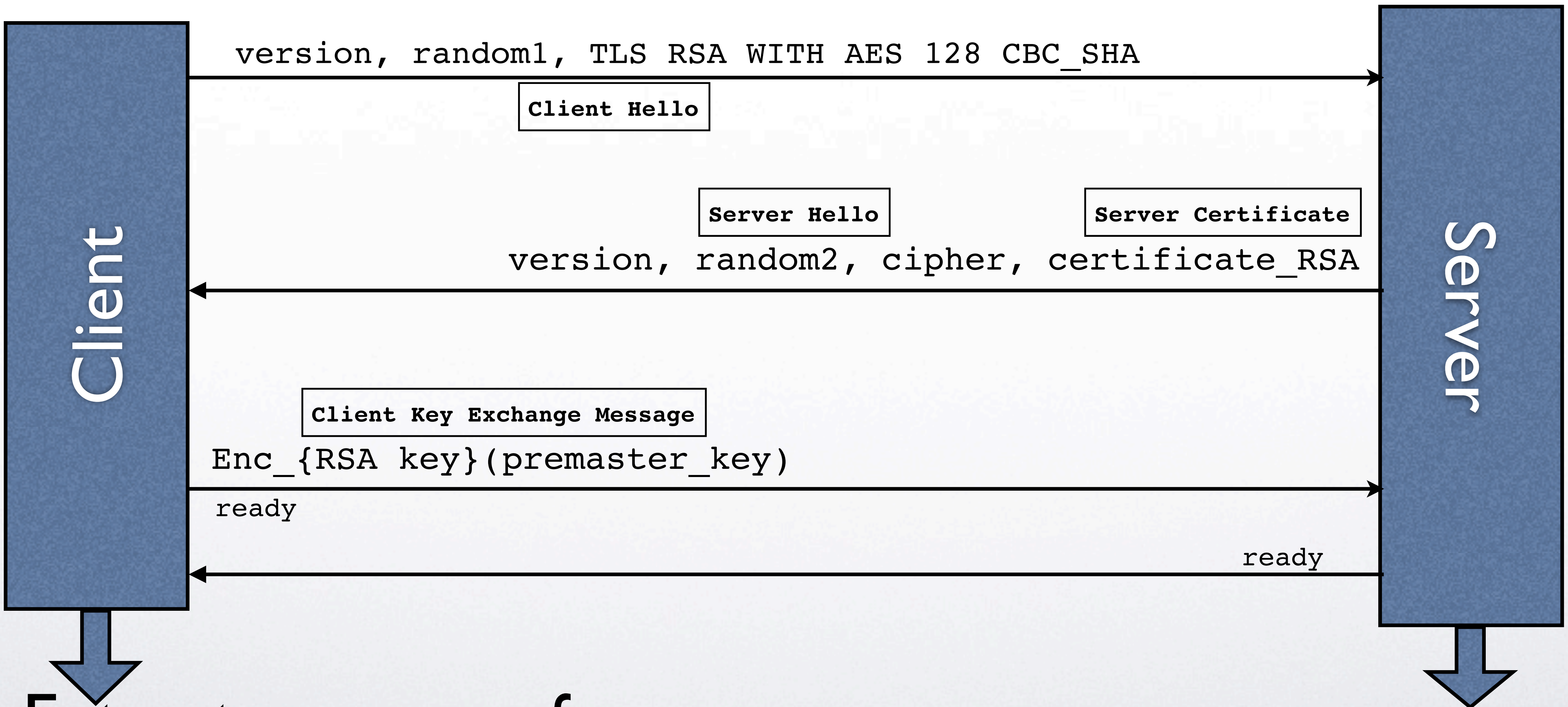
The `Finished` message:

```
PRF(master_secret, finished_label, MD5(handshake_messages) +
           SHA-1(handshake_messages)) [0..11];
```

Pseudorandom Function based on SHA-1 xor MD5

TLS Handshake, VIII

EXAMPLE



Extract `master_key` from `random1+random2+premaster_key`

Summary :TLS Authentication

- Anonymous (neither client nor server provide certificates) - rare.
- Server-side - common.
- Both client and server - rare.

Alert sub-protocol

- RFC 2246.
- When things go wrong an alert is generated and either session ends or the recipient is given the opportunity to continue.

bad_certificate

certificate_expired

handshake_failure

unsupported_certificate

unknown_ca

Record Protocol

- takes messages to be transmitted and
  - fragments data into manageable blocks.
  - (optionally) compresses data.
  - applies a MAC.
  - encrypts.
  - transmits while maintaining proper order.

Key derived from `master_key` and prf

Certificates

- Solving the authentication problem:

?

version, random2, cipher, SignCA(PK), PK

Server

How do you know that a certain public key you receive is really coming from the source?

What is a certificate?

What is a certificate?

- A digital signature on a public-key + other info.

What is a certificate?

- A digital signature on a public-key + other info.
- What should be contained into the ‘info’?

What is a certificate?

- A digital signature on a public-key + other info.
- What should be contained into the ‘info’?
- Who is the signer?

What is a certificate?

- A digital signature on a public-key + other info.
- What should be contained into the 'info'?
- Who is the signer?
- How do you verify the signature?

What is a certificate?

- A digital signature on a public-key + other info.
- What should be contained into the ‘info’?
- Who is the signer?
- How do you verify the signature?
- How do you know that you have the correct signer’s public-key?

Certification

- Hierarchical certification structure.
    - *<u>Root certification authorities</u>* sign public-keys of lower-level authorities and so on till a public-key used for a session-key exchange is created.
- The web-of-trust (PGP).
    - "I sign your key you sign mine."

X.509 Certificates

- Internet standard since 1988.
- Hierarchical.

http://www.ietf.org/rfc/rfc3280.txt

Structure of X.509 Certificates

Version
Serial Number
Algorithm / Parameters
Issuer
Period of Validity: not before date not after date
Subject
Algorithm/ Parameters/ Key
x509v3 extensions
...
Signature

X.509
does not
specify
cryptographic
algorithms

Example

```
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number:
      13:86:35:4d:1d:3f:06:f2:c1:f9:65:05:d5:90:1c:62
    Signature Algorithm: sha1WithRSAEncryption
    Issuer: C=US, O=VISA, OU=Visa International Service Association, CN=Visa eCommerce Root
    Validity
      Not Before: Jun 26 02:18:36 2002 GMT
      Not After : Jun 24 00:16:12 2022 GMT
    Subject: C=US, O=VISA, OU=Visa International Service Association, CN=Visa eCommerce Root
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      RSA Public Key: (2048 bit)
        Modulus (2048 bit):
          00:af:57:de:56:1e:6e:a1:da:60:b1:94:27:cb:17:
          db:07:3f:80:85:4f:c8:9c:b6:d0:f4:6f:4f:cf:99:
          d8:e1:db:c2:48:5c:3a:ac:39:33:c7:1f:6a:8b:26:
          3d:2b:35:f5:48:b1:91:c1:02:4e:04:96:91:7b:b0:
          33:f0:b1:14:4e:11:6f:b5:40:af:1b:45:a5:4a:ef:
          7e:b6:ac:f2:a0:1f:58:3f:12:46:60:3c:8d:a1:e0:
          7d:cf:57:3e:33:1e:fb:47:f1:aa:15:97:07:55:66:
          a5:b5:2d:2e:d8:80:59:b2:a7:0d:b7:46:ec:21:63:
```

Hierarchical name spec

C=Country, L=Location, O=Organization,
OU=Organization Unit, CN=Common Name

A self-signed root certificate

Example continued

```
        4a:ea:db:df:72:38:8c:f3:96:bd:f1:17:bc:d2:ba:
        3b:45:5a:c6:a7:f6:c6:17:8b:01:9d:fc:19:a8:2a:
        83:16:b8:3a:48:fe:4e:3e:a0:ab:06:19:e9:53:f3:
        80:13:07:ed:2d:bf:3f:0a:3c:55:20:39:2c:2c:00:
        69:74:95:4a:bc:20:b2:a9:79:e5:18:89:91:a8:dc:
        1c:4d:ef:bb:7e:37:0b:5d:fe:39:a5:88:52:8c:00:
        6c:ec:18:7c:41:bd:f6:8b:75:77:ba:60:9d:84:e7:
        fe:2d
      Exponent: 65537 (0x10001)
  X509v3 extensions:
    X509v3 Basic Constraints: critical
      CA:TRUE
    X509v3 Key Usage: critical
      Certificate Sign, CRL Sign
    X509v3 Subject Key Identifier:
      15:38:83:0F:3F:2C:3F:70:33:1E:CD:46:FE:07:8C:20:E0:D7:C3:B7
Signature Algorithm: sha1WithRSAEncryption
  5f:f1:41:7d:7c:5c:08:b9:2b:e0:d5:92:47:fa:67:5c:a5:13:
  c3:03:21:9b:2b:4c:89:46:cf:59:4d:c9:fe:a5:40:b6:63:cd:
  ..............
```

note:
public-exponent,
fixed / small

Certificate Encoding

- Certificates are binary files.
- They can be Base64 encoded to be stored/tansported as text files.
- This encoding is called PEM from (privacy enhanced mail)


```
-----BEGIN CERTIFICATE-----
MIIFXzCCBEegAwIBAgIHK5Bmkq9qPzANBgkqhkiG9w0BAQUFADCByjELMAkGA1UE
BhMCVVMxEDAOBgNVBAgTB0FyaXpvbmExEzARBgNVBAcTClNjb3R0c2RhbGUxGjAY
BgNVBAoTEUdvRGFkZHkuY29tLCBJbmMuMTMwMQYDVQQLEypodHRwOi8vY2VydGlm
aWNhdGVzLmdvZGFkZHkuY29tL3JlcG9zaXRvcnkxMDAuBgNVBAMTJ0dvIERhZGR5
IFNlY3VyZSBDZXJ0aWZpY2F0aW9uIEF1dGhvcml0eTERMA8GA1UEBRMIMDc5Njky
ODcwHhcNMTEwMTEwMTkyMzIyWhcNMTIwMjI1MTMyMTEyWjBZMRkwFwYDVQQKDBAq
LmVuZ3IudWNvbm4uZWR1MSEwHwYDVQQLDBhEb21haW4gQ29udHJvbCBWYWxpZGF0
ZWQxGTAXBgNVBAMMECouZW5nci51Y29ubi5lZHUwggEiMA0GCSqGSIb3DQEBAQUA
A4IBDwAwggEKAoIBAQCOgAbFE4TOs2KA/WM9rZ5Q+3G6teSFdjmxZynMeCch+/e/
bSudinQ0oJShqXkyiwrGa1brr3pbQkGLn7AUdli+479IMOq2tFpvZ9SgCVsp2Jk1
/6MqBleBTKfVGfVEz6+YzkwcoiEe7cY+deOeIG1ZFVlnI3kw19V42gk+ZU25oVgL
uiJB2RRD3djpaV1txMVyCnzHa0u2eEHhmh87faIA6EckEW/cWRoVuBZF5Oojsr5c
iXgLIwImQn+Cmo7NQVGV4pKqJVnHPyFX8w+CcLJ8GonVxQeKlehO2DycpDoi2SpP
/hAZ1Vh9mEfftpCikXPwF1unoZAmaN9mWUtcpyMnAgMBAAGjggG4MIIBtDAPBgNV
HRMBAf8EBTADAQEAMB0GA1UdJQQWMBQGCCsGAQUFBwMBBggrBgEFBQcDAjAOBgNV
HQ8BAf8EBAMCBaAwMwYDVR0fBCwwKjAooCagJIYiaHR0cDovL2NybC5nb2RhZGR5
LmNvbS9nZHMxLTI4LmNybDBNBgNVHSAERjBEMEIGC2CGSAGG/W0BBxcBMDMwMQYI
KwYBBQUHAgEWJWh0dHBzOi8vY2VydHMuZ29kYWRkeS5jb20vcmVwb3NpdG9yeS8w
gYAGCCsGAQUFBwEBBHQwcjAkBggrBgEFBQcwAYYYaHR0cDovL29jc3AuZ29kYWRk
eS5jb20vMEoGCCsGAQUFBzAChj5odHRwOi8vY2VydGlmaWNhdGVzLmdvZGFkZHku
Y29tL3JlcG9zaXRvcnkvZ2RfaW50ZXJtZWRpYXRlLmNydDAfBgNVHSMEGDAWgBT9
rGEyk2xF1uLuhV+auud2mWjM5zArBgNVHREEJDAighAqLmVuZ3IudWNvbm4uZWR1
gg5lbmdyLnVjb25uLmVkdTAdBgNVHQ4EFgQU9Z+3hgNCeZLUiGGdUr4pyrMtS4cw
DQYJKoZIhvcNAQEFBQADggEBAGJzGVObEDODThT3di8+JC8hKde1DxOHNzmtv3I3
/eDGR5vO8squzmWh9fn5FZFbHd/UfFPddO6ykR7ffPLGTNCUYbviVjfA+hTpPnPA
3D5u791VfPUJBZf+ccAGr9E1BY+kYGaGqJgHmHcdG7XU0d7bU0XW82Q8wbHQj7H4
jvKSrir4bdGbmZ5VMAjsRL7xMwVypbh2KlOq/wU6AUDs3Rodawa6BnqyCHEpjn0I
i2NUiTzDRKL2OvRCPNs7aY1s4L0oN+HU6yeDkriOHR1NBt3lBGLW+bjZalT6GjPl
xz4fZtag71g7a9mgSAwDE9YlCP2e9WSdlf0THZQ5y3aV868=
-----END CERTIFICATE-----
```

Binary Encoding

- The elements of certificate are encoded using DER (distinguished encoding rules)

Items are stored in TLV format : triplets <Type,Length,Value>

```
30 23            ; SEQUENCE (23 Bytes)
|  |  31 0f            ; SET (f Bytes)
|  |  |  30 0d            ; SEQUENCE (d Bytes)
|  |  |     06 03         ; OBJECT_ID (3 Bytes)
|  |  |     |  55 04 03
|  |  |     |     ; 2.5.4.3 Common Name (CN)
|  |  |     13 06         ; PRINTABLE_STRING (6 Bytes)
|  |  |        54 65 73 74 43 4e                   ; TestCN
|  |  |           ; "TestCN"
|  |  31 10            ; SET (10 Bytes)
|  |     30 0e            ; SEQUENCE (e Bytes)
|  |        06 03         ; OBJECT_ID (3 Bytes)
|  |        |  55 04 0a
|  |        |     ; 2.5.4.10 Organization (O)
|  |        13 07         ; PRINTABLE_STRING (7 Bytes)
|  |           54 65 73 74 4f 72 67                ; TestOrg
|  |              ; "TestOrg"
```

example from http://msdn.microsoft.com

Parsing DER encoding

```
openssl asn1parse -in certforstar.engr.uconn.edu.pem
    0:d=0  hl=4 l=1375 cons: SEQUENCE
    4:d=1  hl=4 l=1095 cons: SEQUENCE
    8:d=2  hl=2 l=   3 cons: cont [ 0 ]
   10:d=3  hl=2 l=   1 prim: INTEGER           :02
   13:d=2  hl=2 l=   7 prim: INTEGER           :2B906692AF6A3F
   22:d=2  hl=2 l=  13 cons: SEQUENCE
   24:d=3  hl=2 l=   9 prim: OBJECT            :sha1WithRSAEncryption
   35:d=3  hl=2 l=   0 prim: NULL
   37:d=2  hl=3 l= 202 cons: SEQUENCE
   40:d=3  hl=2 l=  11 cons: SET
   42:d=4  hl=2 l=   9 cons: SEQUENCE
   44:d=5  hl=2 l=   3 prim: OBJECT            :countryName
   49:d=5  hl=2 l=   2 prim: PRINTABLESTRING   :US
   53:d=3  hl=2 l=  16 cons: SET
   55:d=4  hl=2 l=  14 cons: SEQUENCE
   57:d=5  hl=2 l=   3 prim: OBJECT            :stateOrProvinceName
   62:d=5  hl=2 l=   7 prim: PRINTABLESTRING   :Arizona
   71:d=3  hl=2 l=  19 cons: SET
   73:d=4  hl=2 l=  17 cons: SEQUENCE
   75:d=5  hl=2 l=   3 prim: OBJECT            :localityName
   80:d=5  hl=2 l=  10 prim: PRINTABLESTRING   :Scottsdale
   92:d=3  hl=2 l=  26 cons: SET
   94:d=4  hl=2 l=  24 cons: SEQUENCE
   96:d=5  hl=2 l=   3 prim: OBJECT            :organizationName
  101:d=5  hl=2 l=  17 prim: PRINTABLESTRING   :GoDaddy.com, Inc.
  120:d=3  hl=2 l=  51 cons: SET
  122:d=4  hl=2 l=  49 cons: SEQUENCE
  124:d=5  hl=2 l=   3 prim: OBJECT            :organizationalUnitName
  129:d=5  hl=2 l=  42 prim: PRINTABLESTRING   :http://certificates.godaddy.com/repository
  173:d=3  hl=2 l=  48 cons: SET
  175:d=4  hl=2 l=  46 cons: SEQUENCE
.........
```

FORMAT of each line

offset:
d=**depth**
hl=**header length**
l=**content length**
... followed by type of encoding
primitive - the contents represent the final payload.
constructed - the contents are other TLV values.

Hash and Sign Paradigm

- Sign is a cryptographic operation that operates on fixed length inputs.
- Hashing allows the signing of arbitrary input lengths.
  - What must be the required properties of the hash function?

Available Signature Algorithms

- Common choices:
  - sha1withRSA
  - sha1withDSA
  - (previously md5withRSA etc. md5 is now totally broken)

HTTP Secure

- Combination of HTTP protocol and TLS.
- Integration with browser :
  - CA public-keys come within the browser.
  - Browser issues warning if certificate is invalid or not trusted. Key points : (1) CommonName in certificate should match web-site DNS name. (2) Wildcards are allowed.
  - Typically only server side authentication is performed.

Colliding Certificates

- Based on collision attacks against hash functions the feasibility of producing MD5-based colliding certificates was demonstrated.
- What does this mean for SSL/TLS?

Arjen Lenstra and Xiaoyun Wang and Benne de Weger

Colliding X.509 Certificates

http://eprint.iacr.org/2005/067

Colliding Certificates

- In 2008 directed collisions were produced for SSL certificates => CA hacks feasible.

http://www.win.tue.nl/hashclash/rogue-ca/

Revoking Certificates

- What happens when a certified public-key is compromised?
- E.g., secret key is leaked and published online?
- What if a previously trusted entity is proved to be untrustworthy?

Microsoft Security Bulletin MS01-017

Erroneous VeriSign-Issued Digital Certificates Pose Spoofing Hazard
Originally posted: March 22, 2001
Updated: June 23, 2003
Summary
Who should read this bulletin:
All customers using Microsoft® products.
Impact of vulnerability:
Attacker could digitally sign code using the name "Microsoft Corporation".
Recommendation:
All customers should install the update discussed below.

http://www.microsoft.com/technet/security/bulletin/MS01-017.mspx

Microsoft Security Bulletin MS01-017

- **Technical description:**
- In mid-March 2001, VeriSign, Inc., advised Microsoft that on January 29 and 30, 2001, it issued two VeriSign Class 3 code-signing digital certificates to an individual who fraudulently claimed to be a Microsoft employee. The common name assigned to both certificates is "Microsoft Corporation". The ability to sign executable content using keys that purport to belong to Microsoft would clearly be advantageous to an attacker who wished to convince users to allow the content to run.

- The certificates could be used to sign programs, ActiveX controls, Office macros, and other executable content. Of these, signed ActiveX controls and Office macros would pose the greatest risk, because the attack scenarios involving them would be the most straightforward. Both ActiveX controls and Word documents can be delivered via either web pages or HTML mails. ActiveX controls can be automatically invoked via script, and Word documents can be automatically opened via script unless the user has applied the Office Document Open Confirmation Tool.

Fraudulent Certificates

- The attacker could use the certified public-key and sign code as “Microsoft Corporation”
- How to revoke such certificates?

“man-in-the-middle”

In *anonymous mode* TLS Authentication is totally vulnerable to a man-in-the-middle attack.

Version Rollback Attacks

(a man in the middle attack)

- When a client and server rollback to a previous version of a protocol despite both being capable.
- Can be a problem when client and server negotiate a version in the clear.
- Related example : rollback to `DH_anon`

certificate verification flaws

that have been demonstrated due to bad implementation

- Check validity of certificate but don't check common name.
- Check common name but fail to verify the whole chain of certificates.
- Check everything but allow non-CA signing certificates to be used for issuing certificates.

CA hacks

- Comodo and DigiNotar in 2011
- "several dozens of SSL certificates" issued fraudulently.

Certificate Revocation Lists

- A certificate revocation list (CRL) contains the serial numbers of valid certificates that should not be accepted.
- A CRL must be issued and signed by the CA that originally issued the revoked certificate.
    - From where can you get a CRL?

try http://crl.verisign.com/

Usability aspects of Certificate Management

- From an internet discussion board:

subject: HELP

for some unknown reason, as of today, every file that I access,
every program that runs, every folder in explorer.exe that I open wants
 to connect to some address at crl.verisign.com 198.49.161.200

I have done 3 restores as far back as last week......no difference......

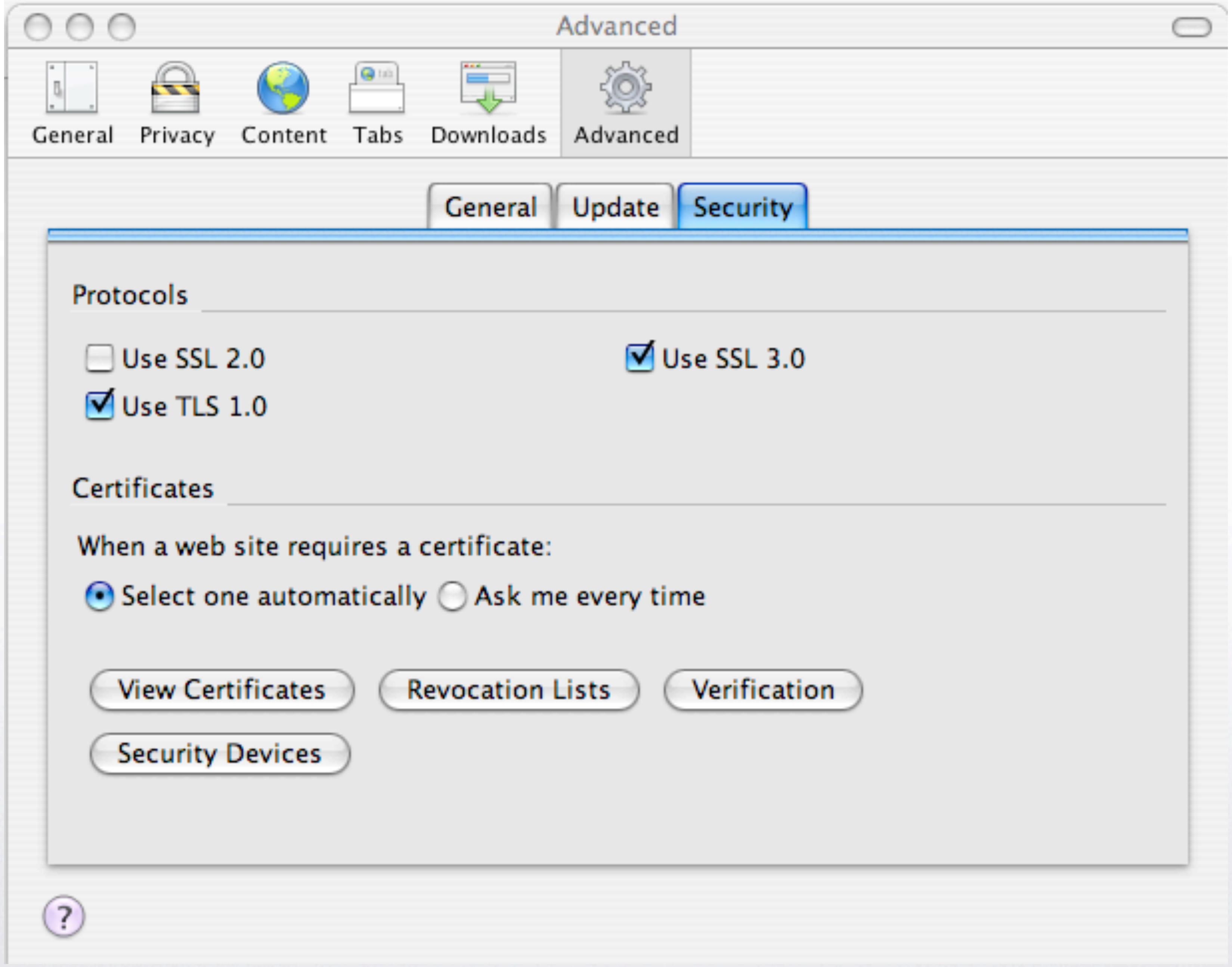
I have search the registry and the hard drives.............

I have ran a virus scanner and a trojan detector.........

Distributing CRLs

- Certificates include inside the extension fields the URL of a ***CRL distribution point***.
- Your system will be responsible to collect such CRLs.

A quick look at Firefox



OCSP

- Online Certificate Status Protocol.
  - Idea: obviate the need for CRLs by verifying certificates online with the CA
  - Has various advantages over CRLs.
  - but requires CA to be online.
  - privacy issues?

PKI

- Public-key Infrastructure
    - A structure that allows entities communicating over an insecure network to form authenticated and secure communication channels.
- Basic tool: digital certification of public-keys.
- Essential ingredients: certificate issuing criteria, trusted distribution, certificate revocation and expiration.

CCA Attacks

- In public-key encryption based key exchange protocols, the server acts as a decryption box.
- Even though it may not return the plaintext it may leak some information about it (e.g., through an error message).
- Such leakage has been exploited for attacks.

Bleichenbacher's PKCS #1 Attack

- Applicable to SSL v3.0
- A chosen ciphertext attack:
  - goal compute $c^{d} \bmod n$ for a given c (i.e., a decryption).
  - By exploiting only error-messages in SSL protocol!!
  - Many queries needed but possible (>300,000).