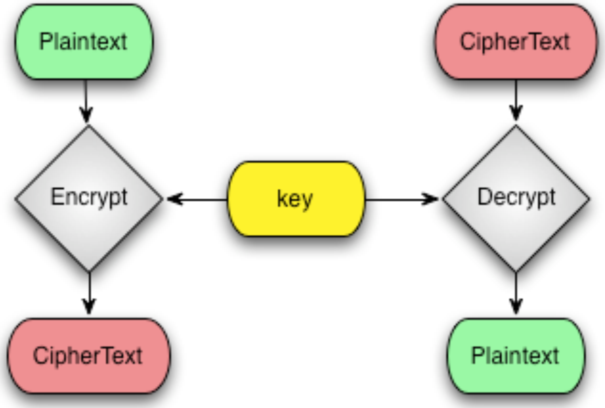
PKI, X.509 CERTIFICATES AND HTTPS WEBSERVERS

Public Key algorithms, digital certificates and PKI

PRELIMINARIES

Symmetric/Asymmetric cryptography



symmetric

The encryption and decryption keys are the same or can be directly derived from each other. Both keys are kept secret.
Examples: 3DES, AES, Blowfish, RC4

asymmetric

Encryption/decryption keys are different and it is computationally unfeasible to derive them from each other.
The encryption key be distributed, the other has to be kept secret.
For this reason it is also called Public Key cryptography.
Examples: RSA, Diffie-Hellman, ElGamal

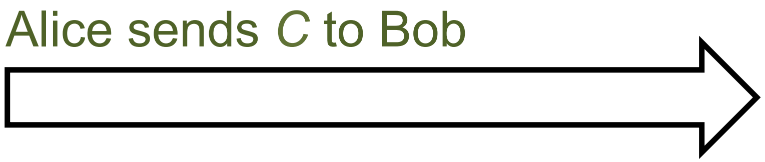
Public Key cryptography: encryption/decryption

Alice

Bob

Alice wants to send a message *M* encrypted for Bob

Gets Bob's public key B_{pub}
(Somehow) verifies B_{pub} authenticity
Encrypts *M* with $B_{pub} \rightarrow C = F(B_{pub}, M)$



Alice sends *C* to Bob

Decrypts C with Bob's private key B_{priv}
$M = F(B_{priv}, C)$

Note:

1) Only Bob can decrypt *C*
2) Nobody "can" derive B_{priv} from B_{pub}
3) This procedure can be inverted to implement a digital signature

Public Key cryptography: digital signature

Alice

Alice wants to sign a message *M* so that Bob can verify its authenticity

Bob

Gets his own private key A_{priv}
Computes a hash of the message *H(M)*
Signs H(M) with A_{priv}→ $S = F(A_{priv}, H(M))$

Alice sends *(H(M), S)* to Bob

Computes a hash of the message *H(M)*
Verify the signature by verifying the following:
$H(M) = F(A_{pub,}\ H(M))$?

Note:

1) Only Alice can sign *M*
2) Nobody can modify M and compute a valid signature S without knowing A_{priv}
3) Alice can include a nonce (given by Bob) in the signature to avoid a third entity to reuse the same signature for the same message *M*

RSA: key generation

1. Extract two “big” prime numbers ***p*** e ***q*** **(random, secret)**
2. Compute the RSA modulus: $\boldsymbol{N = p \times q}$
3. Compute $\boldsymbol{\Phi(N) = (p - 1)(q - 1)}$ (Eulero’s function)
4. Randomly generates the the number **e:** $1 < \mathbf{e} < \boldsymbol{\Phi(N)}$ relatively prime to $\boldsymbol{\Phi(N)}$
5. Compute the number $\boldsymbol{d}$**:** $\boldsymbol{e \times d = 1 \bmod \Phi(N)}$, or in other words **e** is the inverse of ***d*** in the group $\boldsymbol{\Phi(N)}$

PUBLIC KEY: *(N, e)*

PRIVATE KEY: *d*

Must be kept secret: ***p*, *q*, *Φ(N)*, *d***

Note:

1) *to derive d from e an attacker should compute* e^{-1} *in* $\Phi(N)$

2) $\Phi(N)$ *is the number of integers less than or equal to n that are relatively prime to N*

2.1) *to compute* $\Phi(N)$ *an attacker should know p and q (otherwise it’s unfeasible)*

3) *it is computationally unfeasible to factorize the product of two “big enough” prime numbers*

RSA tranformations

RSA transformation is simply a modular exponentiation with respectively the public private key

ENCRYPTION

DECRYPTION

$$X \Rightarrow X^{e} \bmod N \Rightarrow Y \Rightarrow Y^{d} \bmod N \Rightarrow X$$

ciphertext

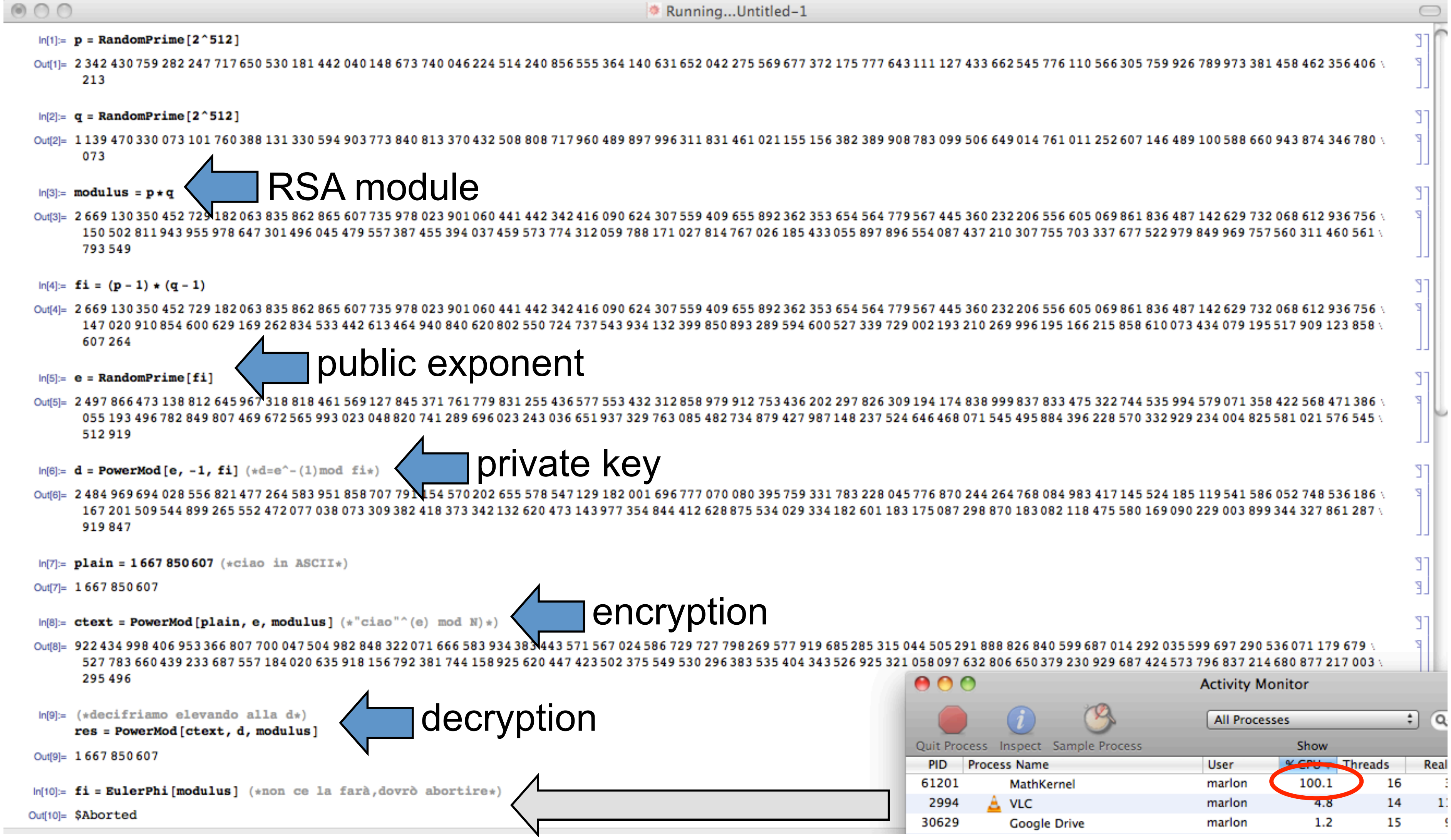
SIGNATURE

VERIFICATION

$$X \Rightarrow X^{d} \bmod N \Rightarrow Y \Rightarrow Y^{e} \bmod N \Rightarrow X$$

signed text

RSA with mathematica



Running...Untitled-1

```
In[1]:= p = RandomPrime[2^512]

Out[1]= 2 342 430 759 282 247 717 650 530 181 442 040 148 673 740 046 224 514 240 856 555 364 140 631 652 042 275 569 677 372 175 777 643 111 127 433 662 545 776 110 566 305 759 926 789 973 381 458 462 356 406 \
         213

In[2]:= q = RandomPrime[2^512]

Out[2]= 1 139 470 330 073 101 760 388 131 330 594 903 773 840 813 370 432 508 808 717 960 489 897 996 311 831 461 021 155 156 382 389 908 783 099 506 649 014 761 011 252 607 146 489 100 588 660 943 874 346 780 \
         073

In[3]:= modulus = p * q

Out[3]= 2 669 130 350 452 729 182 063 835 862 865 607 735 978 023 901 060 441 442 342 416 090 624 307 559 409 655 892 362 353 654 564 779 567 445 360 232 206 556 605 069 861 836 487 142 629 732 068 612 936 756 \
         150 502 811 943 955 978 647 301 496 045 479 557 387 455 394 037 459 573 774 312 059 788 171 027 814 767 026 185 433 055 897 896 554 087 437 210 307 755 703 337 677 522 979 849 969 757 560 311 460 561 \
         793 549

In[4]:= fi = (p - 1) * (q - 1)

Out[4]= 2 669 130 350 452 729 182 063 835 862 865 607 735 978 023 901 060 441 442 342 416 090 624 307 559 409 655 892 362 353 654 564 779 567 445 360 232 206 556 605 069 861 836 487 142 629 732 068 612 936 756 \
         147 020 910 854 600 629 169 262 834 533 442 613 464 940 840 620 802 550 724 737 543 934 132 399 850 893 289 594 600 527 339 729 002 193 210 269 996 195 166 215 858 610 073 434 079 195 517 909 123 858 \
         607 264

In[5]:= e = RandomPrime[fi]

Out[5]= 2 497 866 473 138 812 645 967 318 818 461 569 127 845 371 761 779 831 255 436 577 553 432 312 858 979 912 753 436 202 297 826 309 194 174 838 999 837 833 475 322 744 535 994 579 071 358 422 568 471 386 \
         055 193 496 782 849 807 469 672 565 993 023 048 820 741 289 696 023 243 036 651 937 329 763 085 482 734 879 427 987 148 237 524 646 468 071 545 495 884 396 228 570 332 929 234 004 825 581 021 576 545 \
         512 919

In[6]:= d = PowerMod[e, -1, fi] (*d=e^-(1)mod  fi*)

Out[6]= 2 484 969 694 028 556 821 477 264 583 951 858 707 791 154 570 202 655 578 547 129 182 001 696 777 070 080 395 759 331 783 228 045 776 870 244 264 768 084 983 417 145 524 185 119 541 586 052 748 536 186 \
         167 201 509 544 899 265 552 472 077 038 073 309 382 418 373 342 132 620 473 143 977 354 844 412 628 875 534 029 334 182 601 183 175 087 298 870 183 082 118 475 580 169 090 229 003 899 344 327 861 287 \
         919 847

In[7]:= plain = 1 667 850 607 (*ciao in ASCII*)

Out[7]= 1 667 850 607

In[8]:= ctext = PowerMod[plain, e, modulus] (*"ciao"^(e) mod N)*)

Out[8]= 922 434 998 406 953 366 807 700 047 504 982 848 322 071 666 583 934 383 443 571 567 024 586 729 727 798 269 577 919 685 285 315 044 505 291 888 826 840 599 687 014 292 035 599 697 290 536 071 179 679 \
         527 783 660 439 233 687 557 184 020 635 918 156 792 381 744 158 925 620 447 423 502 375 549 530 296 383 535 404 343 526 925 321 058 097 632 806 650 379 230 929 687 424 573 796 837 214 680 877 217 003 \
         295 496

In[9]:= (*decifriamo elevando alla d*)
        res = PowerMod[ctext, d, modulus]

Out[9]= 1 667 850 607

In[10]:= fi = EulerPhi[modulus] (*non ce la farà,dovrò abortire*)

Out[10]= $Aborted
```

RSA module

public exponent

private key

encryption

decryption

Activity Monitor

PID	Process Name	User	% CPU	Threads	Real
61201	MathKernel	marlon	100.1	16	
2994	VLC	marlon	4.8	14	
30629	Google Drive	marlon	1.2	15	

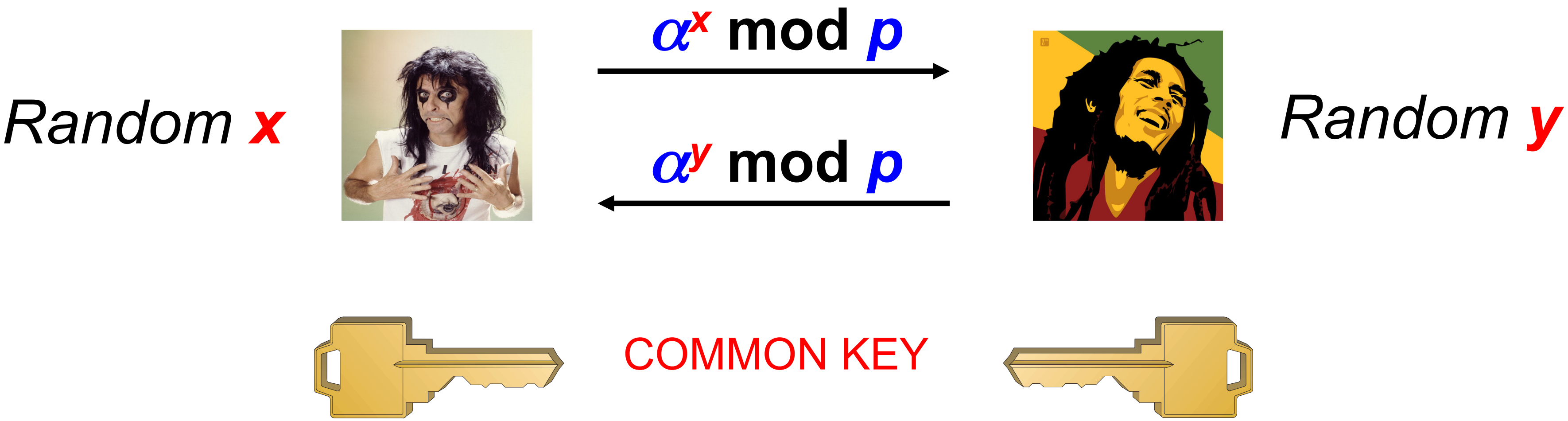
…not exactly the real algorithm, but the concepts are the same!

Diffie-Hellman Key exchange alogorithm

Public: α, p

Secret: x, y

GOAL: exchange a common secret that only Alice and Bob can derive



Random x

α^x mod p

α^y mod p

Random y

COMMON KEY

$$K = (\alpha^y)^x \bmod p \qquad K = (\alpha^x)^y \bmod p$$

Note:

1) Common secret number exchanged with an asymmetric algorithm

2) to compute K from (α^x mod p) and (α^y mod p) an attacker should be able to compute the discrete logarithm x = $\log_\alpha(\alpha^x \bmod p)$ and y = $\log_\alpha(\alpha^y \bmod p)$…

3) …which is computationally unfeasible for an attacker with “limited computational resources”

How does Alice obtain Bob’s public key?

- Everything’s perfect, you believe that nobody can break the public key algorithms if the numbers are “big enough”
- How are the public keys distributed?
  - In a network with n nodes, n(n-1)/2 keys have to be distributed!
  - What if my private key is lost or stolen? Should I need to notify all the remaining (n-1) nodes to revoke my public key?
  - Solution: centralized or opportunistic distribution! (obvious, the public key don’t have to be kept secret!)
- OK, the scalability issue is solved, but how can I be sure that a public key is authentic? How can Alice get the public key of Bob and be sure that it’s really his?
- SOLUTION:
  - A trusted third party that issues some kind of proof that a public key is really related to a given identity

Public Key Certificate

- A public key certificate is a data structure that binds a public key (and therefor the related private key) to the the identy of the legitimate owner → CERT_{ID}:{ID, Pub_{ID}}
- The binding between {ID, Pub_{ID}} is granted by a trusted certification authority that signs CERT_{ID}
- Provided that we have the CA's public key, we can verify the CA signature and therefor verify the public key authenticity

EXAMPLE:

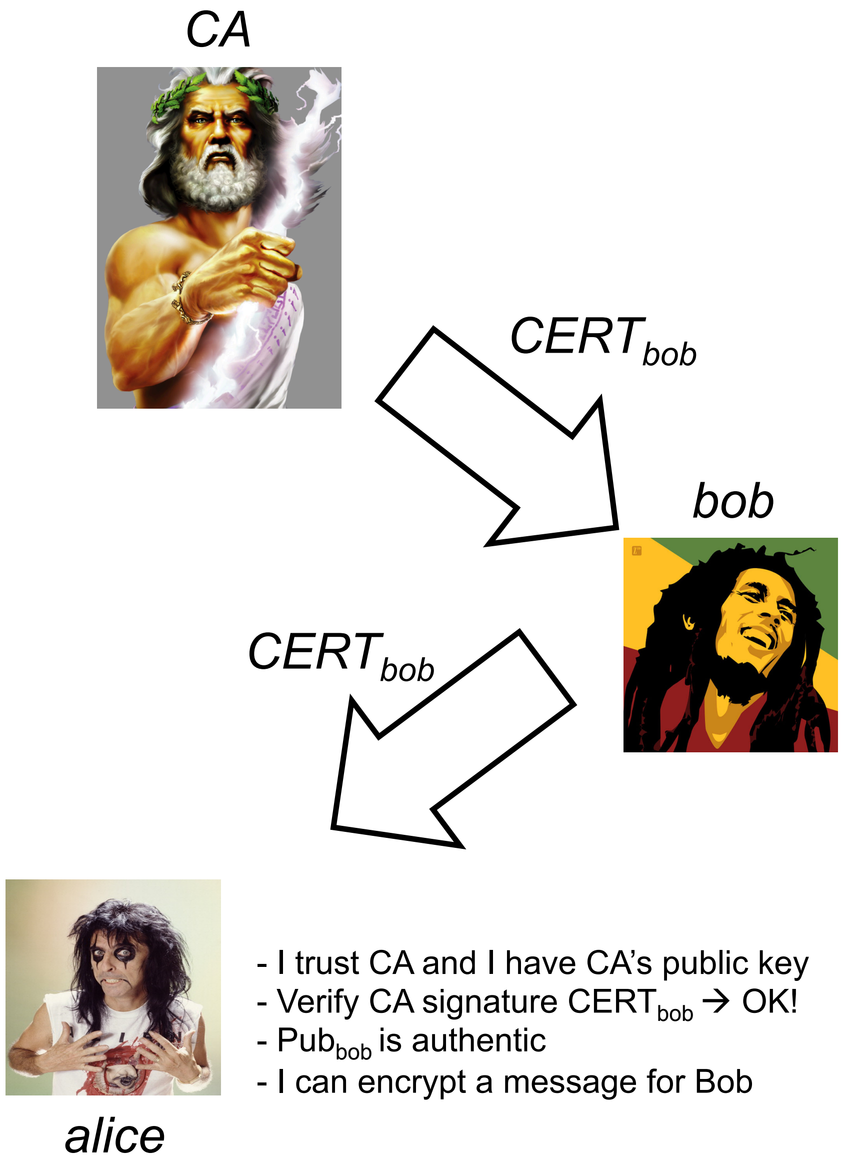
CA issues a public certificate for bob CERT_{bob}

CERT_{bob} contains:

1) Pub_{bob}
2) CA identity CA_{id}
3) CA signature of CERT_{bob}

Once I have the authentic Pub_{bob}, I just need to verify that the party I'm communicating with is actually Bob (i.e.: it has the private key)

To do so, I perform a simple challenge/response mechanism. I extract a nonce and challenge Bob to sign this random number. Since the public key is authentic, and Bob couldn't know the random number, only the real Bob can sign the nonce correctly (and I can verify it)

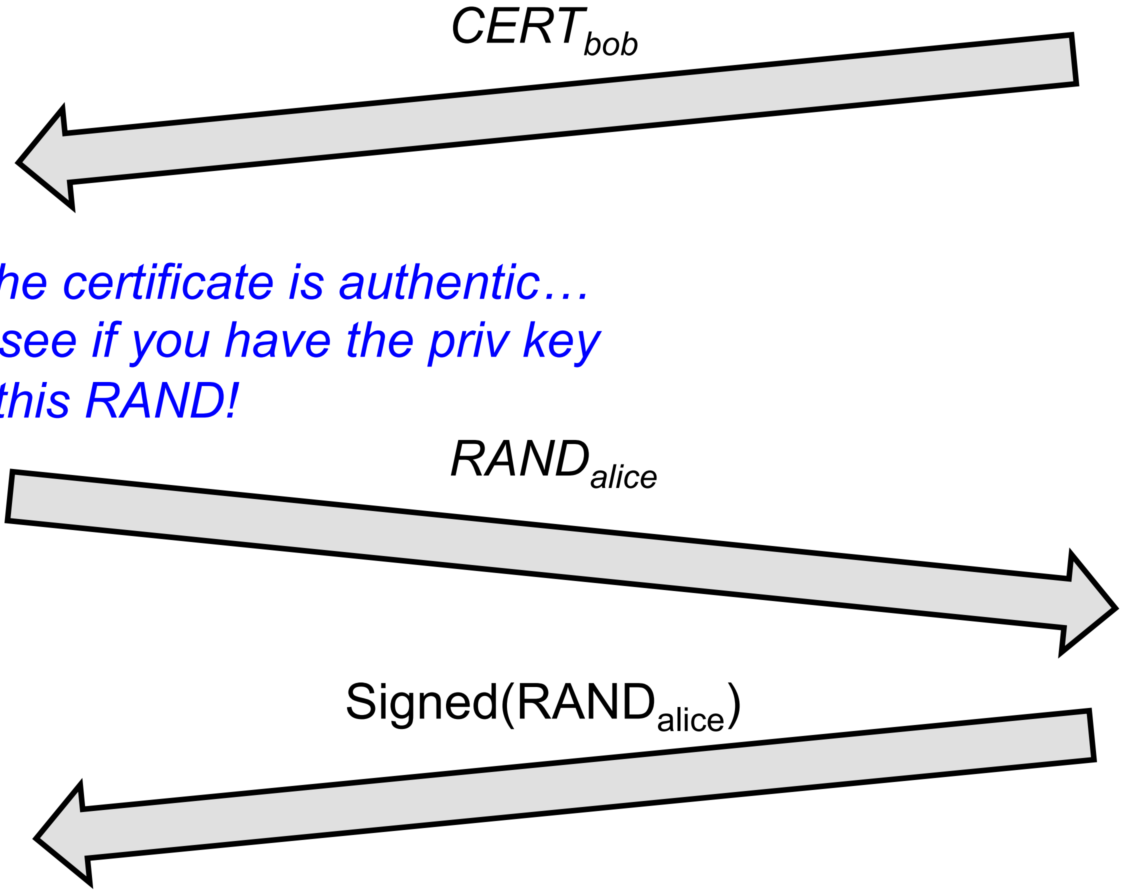


- I trust CA and I have CA's public key
- Verify CA signature CERT_{bob} → OK!
- Pub_{bob} is authentic
- I can encrypt a message for Bob

Challenge/Response concept

alice

bob



$CERT_{bob}$

OK, the certificate is authentic…
Let's see if you have the priv key
Sign this RAND!

$RAND_{alice}$

OK noprob!
$(RAND_{alice})^{priv}$

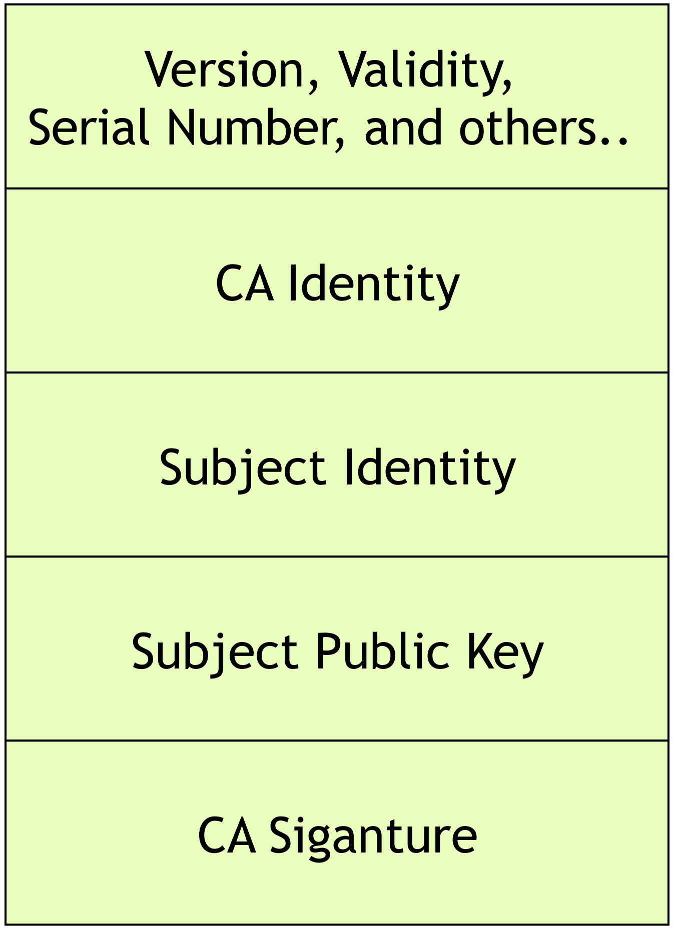
Signed($RAND_{alice}$)

OK, I have Bob's public key and $RAND_{alice,}$
I can verify Bob's signature

Public Key Infrastructure

- A PKI consists of the protocols, the policies and the cryptographic mechanism used to manage the management of public key certificate
  - Creation, distribution, revocation, etc…
- A PKI requires the definition of:
  - Certificate format
  - Relationship among CAs
  - Mechanisms and policies for issuing and revoking certificate
  - Storage services
- Typical certificate format: X.509

X.509 format (high level)



Version, Validity,
Serial Number, and others..

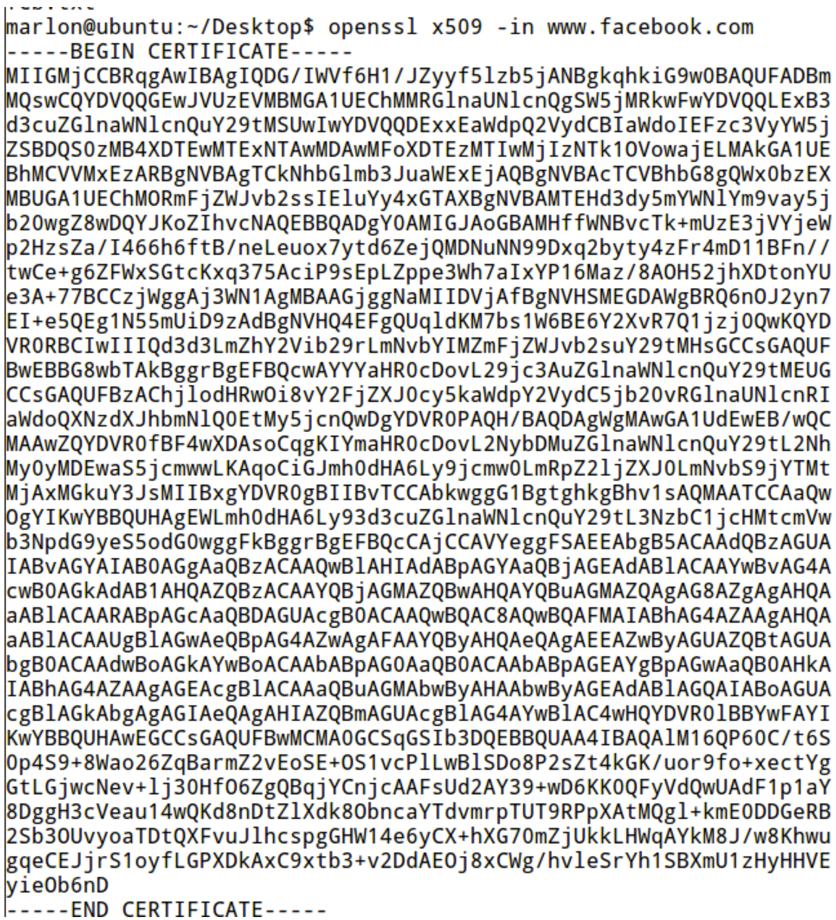
CA Identity

Subject Identity

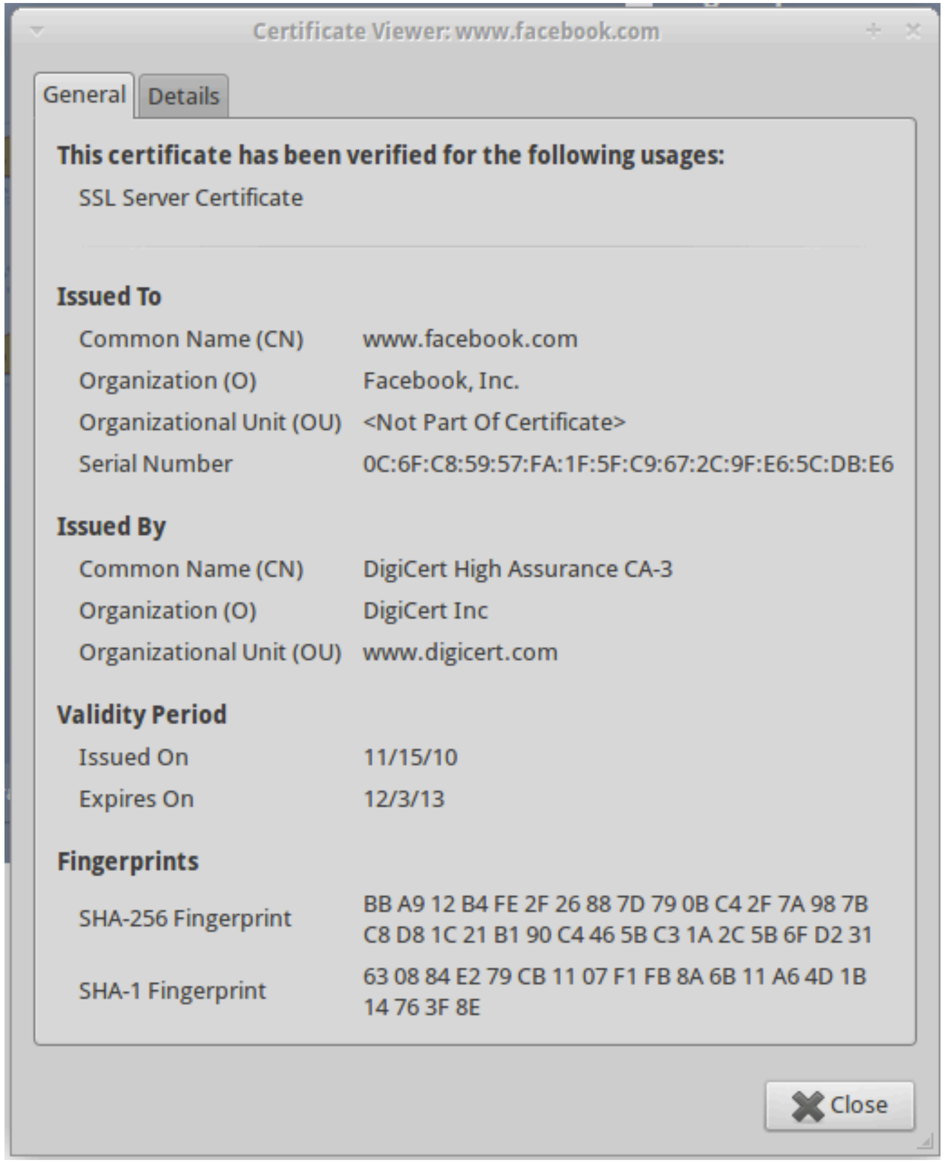
Subject Public Key

CA Siganture

X.509 certificate: real example



```
marlon@ubuntu:~/Desktop$ openssl x509 -in www.facebook.com
-----BEGIN CERTIFICATE-----
MIIGMjCCBRqgAwIBAgIQDG/IWVf6H1/JZyyf5lzb5jANBgkqhkiG9w0BAQUFADBm
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMRkwFwYDVQQLExB3
d3cuZGlnaWNlcnQuY29tMSUwIwYDVQQDExxEaWdpQ2VydCBIaWdoIEFzc3VyYW5j
ZSBDQS0zMB4XDTEwMTExNTAwMDAwMFoXDTEzMTIwMjIzNTk1OVowajELMAkGA1UE
BhMCVVMxEzARBgNVBAgTCkNhbGlmb3JuaWExEjAQBgNVBAcTCVBhbG8gQWx0bzEX
MBUGA1UEChMORmFjZWJvb2ssIEluYy4xGTAXBgNVBAMTEHd3dy5mYWNlYm9vay5j
b20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAMHffWNBvcTk+mUzE3jVYjeW
p2HzsZa/I466h6ftB/neLeuox7ytd6ZejQMDNuNN99Dxq2byty4zFr4mD11BFn//
twCe+g6ZFWxSGtcKxq375AciP9sEpLZppe3Wh7aIxYP16Maz/8AOH52jhXDtonYU
e3A+77BCCzjWggAj3WN1AgMBAAGjggNaMIIDVjAfBgNVHSMEGDAWgBRQ6nOJ2yn7
EI+e5QEg1N55mUiD9zAdBgNVHQ4EFgQUqldKM7bs1W6BE6Y2XvR7Q1jzj0QwKQYD
VR0RBCIwIIIQd3d3LmZhY2Vib29rLmNvbYIMZmFjZWJvb2suY29tMHsGCCsGAQUF
BwEBBG8wbTAkBggrBgEFBQcwAYYYaHR0cDovL29jc3AuZGlnaWNlcnQuY29tMEUG
CCsGAQUFBzAChjlodHRwOi8vY2FjZXJ0cy5kaWdpY2VydC5jb20vRGlnaUNlcnRI
aWdoQXNzdXJhbmNlQ0EtMy5jcnQwDgYDVR0PAQH/BAQDAgWgMAwGA1UdEwEB/wQC
MAAwZQYDVR0fBF4wXDAsoCqgKIYmaHR0cDovL2NybDMuZGlnaWNlcnQuY29tL2Nh
My0yMDEwaS5jcmwwLKAqoCiGJmh0dHA6Ly9jcmw0LmRpZ2ljZXJ0LmNvbS9jYTMt
MjAxMGkuY3JsMIIBxgYDVR0gBIIBvTCCAbkwggG1BgtghkgBhv1sAQMAATCCAaQw
OgYIKwYBBQUHAgEWLmh0dHA6Ly93d3cuZGlnaWNlcnQuY29tL3NzbC1jcHMtcmVw
b3NpdG9yeS5odG0wggFkBggrBgEFBQcCAjCCAVYeggFSAEEAbgB5ACAAdQBzAGUA
IABvAGYAIAB0AGgAaQBzACAAQwBlAHIAdABpAGYAaQBjAGEAdABlACAAYwBvAG4A
cwB0AGkAdAB1AHQAZQBzACAAYQBjAGMAZQBwAHQAYQBuAGMAZQAgAG8AZgAgAHQA
aABlACAARABpAGcAaQBDAGUAcgB0ACAAQwBQAC8AQwBQAFMAIABhAG4AZAAgAHQA
aABlACAAUgBlAGwAeQBpAG4AZwAgAFAAYQByAHQAeQAgAEEAZwByAGUAZQBtAGUA
bgB0ACAAdwBoAGkAYwBoACAAbABpAG0AaQB0ACAAbABpAGEAYgBpAGwAaQB0AHkA
IABhAG4AZAAgAGEAcgBlACAAaQBuAGMAbwByAHAAbwByAGEAdABlAGQAIABoAGUA
cgBlAGkAbgAgAGIAeQAgAHIAZQBmAGUAcgBlAG4AYwBlAC4wHQYDVR0lBBYwFAYI
KwYBBQUHAwEGCCsGAQUFBwMCMA0GCSqGSIb3DQEBBQUAA4IBAQAlM16QP60C/t6S
0p4S9+8Wao26ZqBarmZ2vEoSE+OS1vcPlLwBlSDo8P2sZt4kGK/uor9fo+xectYg
GtLGjwcNev+lj30Hf06ZgQBqjYCnjcAAFsUd2AY39+wD6KK0QFyVdQwUAdF1p1aY
8DggH3cVeau14wQKd8nDtZlXdk8ObncaYTdvmrpTUT9RPpXAtMQgl+kmE0DDGeRB
2Sb3OUvyoaTDtQXFvuJlhcspgGHW14e6yCX+hXG70mZjUkkLHWqAYkM8J/w8Khwu
gqeCEJjrS1oyfLGPXDkAxC9xtb3+v2DdAE0j8xCWg/hvleSrYh1SBXmU1zHyHHVE
yieOb6nD
-----END CERTIFICATE-----
```



Certificate Viewer: www.facebook.com

This certificate has been verified for the following usages:

SSL Server Certificate

Issued To

Common Name (CN)	www.facebook.com
Organization (O)	Facebook, Inc.
Organizational Unit (OU)	<Not Part Of Certificate>
Serial Number	0C:6F:C8:59:57:FA:1F:5F:C9:67:2C:9F:E6:5C:DB:E6

Issued By

Common Name (CN)	DigiCert High Assurance CA-3
Organization (O)	DigiCert Inc
Organizational Unit (OU)	www.digicert.com

Validity Period

Issued On	11/15/10
Expires On	12/3/13

Fingerprints

SHA-256 Fingerprint	BB A9 12 B4 FE 2F 26 88 7D 79 0B C4 2F 7A 98 7B C8 D8 1C 21 B1 90 C4 46 5B C3 1A 2C 5B 6F D2 31
SHA-1 Fingerprint	63 08 84 E2 79 CB 11 07 F1 FB 8A 6B 11 A6 4D 1B 14 76 3F 8E

X.509 certificate: real example

```
Version: 3 (0x2)
Serial Number:
    0c:6f:c8:59:57:fa:1f:5f:c9:67:2c:9f:e6:5c:db:e6
Signature Algorithm: sha1WithRSAEncryption
Issuer: C=US, O=DigiCert Inc, OU=www.digicert.com, CN=DigiCert High Assurance CA-3
Validity
    Not Before: Nov 15 00:00:00 2010 GMT
    Not After : Dec  2 23:59:59 2013 GMT
Subject: C=US, ST=California, L=Palo Alto, O=Facebook, Inc., CN=www.facebook.com
Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
    RSA Public Key: (1024 bit)
        Modulus (1024 bit):
            00:c1:df:7d:63:41:bd:c4:e4:fa:65:33:13:78:d5: (... cut…) 0b:38:d6:82:00:23:dd:63:75
        Exponent: 65537 (0x10001)

X509v3 extensions: (cut)
    X509v3 Subject Key Identifier:
        AA:57:4A:33:B6:EC:D5:6E:81:13:A6:36:5E:F4:7B:43:58:F3:8F:44
    X509v3 Subject Alternative Name:
        DNS:www.facebook.com, DNS:facebook.com
    X509v3 Key Usage: critical
        Digital Signature, Key Encipherment
    X509v3 Basic Constraints: critical
        CA:FALSE
    X509v3 Extended Key Usage:
        TLS Web Server Authentication, TLS Web Client Authentication

Signature Algorithm: sha1WithRSAEncryption
25:33:5e:90:3f:ad:02:fe:de:92:d2:9e:12:f7:ef:16:6a:8d: (... cut…) 8e:6f:a9:c3
```

Certificate Signing Request

- A certificate signing request (also CSR or certification request) is a message sent from an applicant to a certificate authority in order to apply for a digital identity certificate
- The most common format for CSRs is the PKCS#10 specification
- Operations:
  - the applicant first generates a key pair, keeping the private key secret
  - the applicant generates a CSR contains information identifying herself (X.509 subject field), optional X.509 extensions (e.g. key usage: RSA authentication for web servers) and the public key chosen by the applicant
  - The CSR may be accompanied by other credentials or proofs of identity required by the certificate authority, and the certificate authority may contact the applicant for further information

X509v3 extensions

- An X.509 v3 certificate contains an extension field that permits any number of additional fields to be added to the certificate

- Certificate extensions provide a way of adding information such as alternative subject names and usage restrictions to certificates

Some standard extensions

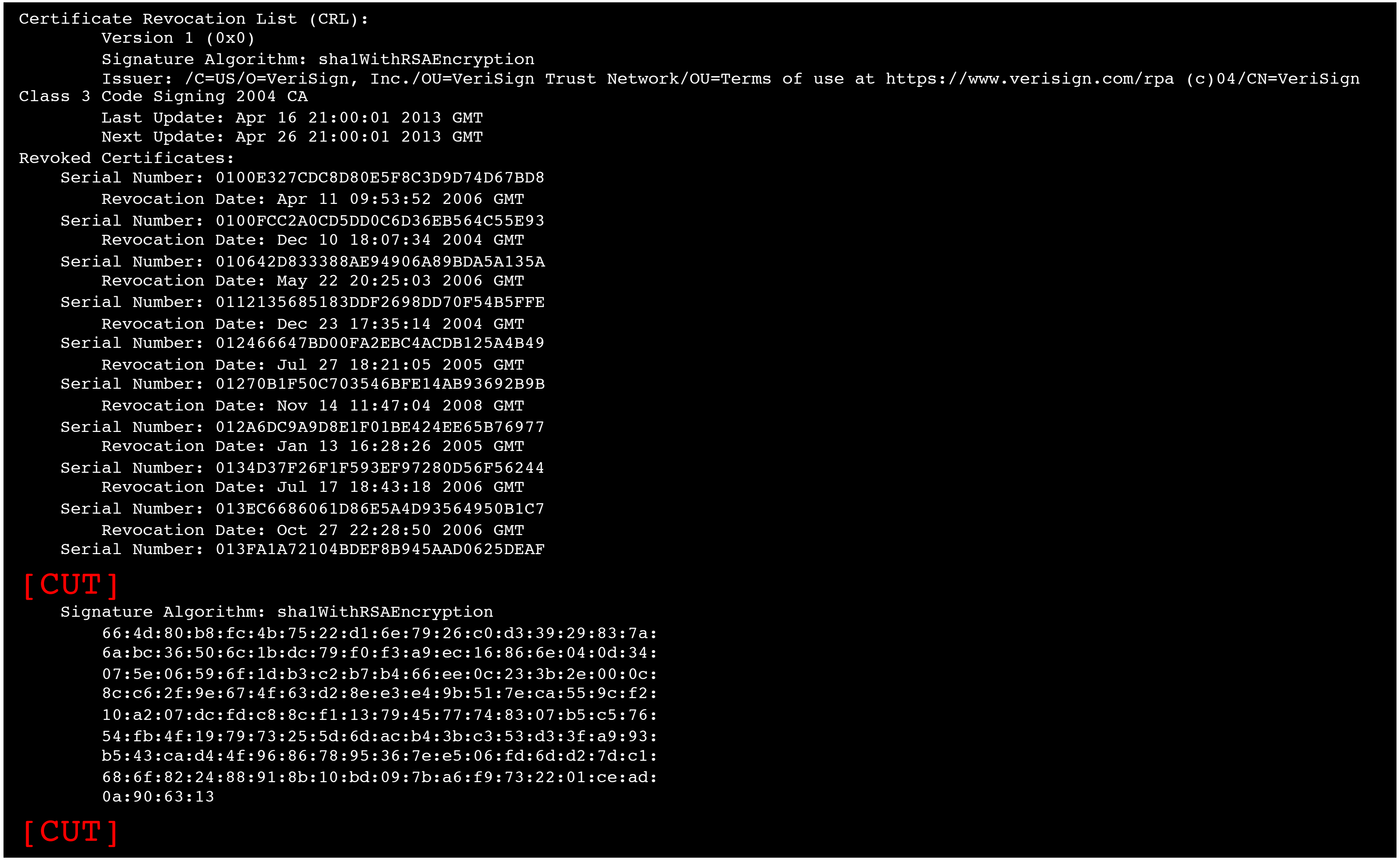
- **Authority Key Identifier**
  - The authority key identifier extension provides a means of identifying the public key corresponding to the private key used to sign a certificate
- **Subject Key Identifier**
  - The subject key identifier extension provides a means of identifying certificates that contain a particular public key
- **Key Usage**
  - The key usage extension defines the purpose (e.g., encipherment, signature, certificate signing) of the key contained in the certificate.
  - digitalSignature, nonRepudiation, contentCommitment, keyEncipherment , dataEncipherment, keyAgreement, keyCertSign, cRLSign, encipherOnly, decipherOnly
- **Subject Alternative Name**
  - The subject alternative name extension allows identities to be bound to the subject of the certificate. These identities may be included in addition to or in place of the identity in the subject field of the certificate
- **Extended Key Usage**
  - This extension indicates one or more purposes for which the certified public key may be used, in addition to or in place of the basic purposes indicated in the key usage extension.
  - TLS WWW server authentication, TLS WWW client authentication, Signing of downloadable executable code, Email protection, Timestamping

See http://tools.ietf.org/html/rfc5280 for the complete list

Certificate Revocation List

- Various circumstances may cause a certificate to become invalid prior to the expiration of the validity period
  - change of name, change of association between subject and CA (e.g., an employee terminates employment with an organization), and compromise or suspected compromise of the corresponding private key.
- Under such circumstances, the CA needs to revoke the certificate
- CA periodically issuing a signed data structure called a certificate revocation list (CRL)
- A CRL is a time-stamped list identifying revoked certificates that is signed by a CA or CRL issuer and made freely available in a public repository.
- When a certificate-using system uses a certificate that system not only checks the certificate signature and validity but also acquires a suitably recent CRL and checks that the certificate serial number is not on that CRL.
- Advantage: CRLs may be distributed by exactly the same means as certificates themselves, namely, via untrusted servers and untrusted communications.
- One limitation: time granularity of revocation is limited to the CRL issue period.

CRL example



```
Certificate Revocation List (CRL):
        Version 1 (0x0)
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: /C=US/O=VeriSign, Inc./OU=VeriSign Trust Network/OU=Terms of use at https://www.verisign.com/rpa (c)04/CN=VeriSign
Class 3 Code Signing 2004 CA
        Last Update: Apr 16 21:00:01 2013 GMT
        Next Update: Apr 26 21:00:01 2013 GMT
Revoked Certificates:
    Serial Number: 0100E327CDC8D80E5F8C3D9D74D67BD8
        Revocation Date: Apr 11 09:53:52 2006 GMT
    Serial Number: 0100FCC2A0CD5DD0C6D36EB564C55E93
        Revocation Date: Dec 10 18:07:34 2004 GMT
    Serial Number: 010642D833388AE94906A89BDA5A135A
        Revocation Date: May 22 20:25:03 2006 GMT
    Serial Number: 0112135685183DDF2698DD70F54B5FFE
        Revocation Date: Dec 23 17:35:14 2004 GMT
    Serial Number: 012466647BD00FA2EBC4ACDB125A4B49
        Revocation Date: Jul 27 18:21:05 2005 GMT
    Serial Number: 01270B1F50C703546BFE14AB93692B9B
        Revocation Date: Nov 14 11:47:04 2008 GMT
    Serial Number: 012A6DC9A9D8E1F01BE424EE65B76977
        Revocation Date: Jan 13 16:28:26 2005 GMT
    Serial Number: 0134D37F26F1F593EF97280D56F56244
        Revocation Date: Jul 17 18:43:18 2006 GMT
    Serial Number: 013EC6686061D86E5A4D93564950B1C7
        Revocation Date: Oct 27 22:28:50 2006 GMT
    Serial Number: 013FA1A72104BDEF8B945AAD0625DEAF
[CUT]
    Signature Algorithm: sha1WithRSAEncryption
        66:4d:80:b8:fc:4b:75:22:d1:6e:79:26:c0:d3:39:29:83:7a:
        6a:bc:36:50:6c:1b:dc:79:f0:f3:a9:ec:16:86:6e:04:0d:34:
        07:5e:06:59:6f:1d:b3:c2:b7:b4:66:ee:0c:23:3b:2e:00:0c:
        8c:c6:2f:9e:67:4f:63:d2:8e:e3:e4:9b:51:7e:ca:55:9c:f2:
        10:a2:07:dc:fd:c8:8c:f1:13:79:45:77:74:83:07:b5:c5:76:
        54:fb:4f:19:79:73:25:5d:6d:ac:b4:3b:c3:53:d3:3f:a9:93:
        b5:43:ca:d4:4f:96:86:78:95:36:7e:e5:06:fd:6d:d2:7d:c1:
        68:6f:82:24:88:91:8b:10:bd:09:7b:a6:f9:73:22:01:ce:ad:
        0a:90:63:13
[CUT]
```

Let’s build our own certification authority

OPENSSL X509 TUTORIAL

OpenSSL

- OpenSSL is a cryptography toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) network protocols and related cryptography standards required by them
  - www.openssl.org
- Main component
  - Cryptography library: `libcrypto`
  - SSL/TLS protocol library: `libssl`
  - `openssl` program
- The openssl program is a command line tool for using the various cryptography functions of OpenSSL's crypto library from the shell. It can be used for
  - Creation and management of private keys, public keys and parameters
  - Public key cryptographic operations
  - Creation of X.509 certificates, CSRs and CRLs
  - Calculation of Message Digests
  - Encryption and Decryption with Ciphers
  - SSL/TLS Client and Server Tests
  - Handling of S/MIME signed or encrypted mail
  - Time Stamp requests, generation and verification

Create a CA and sign certificate request with openssl

- Typical workflow
  1. Generate the RSA key pair for our CA
  2. Create a self-signed certificate for our CA
  3. Generate the RSA key pair for the web server
  4. Generate a CSR for the web server
  5. Sign the CSR with the CA private key

- Very simple Lab-pki
  - Create the CA and issue the certificates (single level certification ROOT_CA→certificate) with openssl from the host machine
  - Create a netikit lab (Lab9-pki) with just one VM (with a TAP 10.0.0.1,10.0.0.2) that will be our test web server
  - Setup Apache2 for a HTTPS website

Create the CA keys

Prepare our CA folder and the serial number file

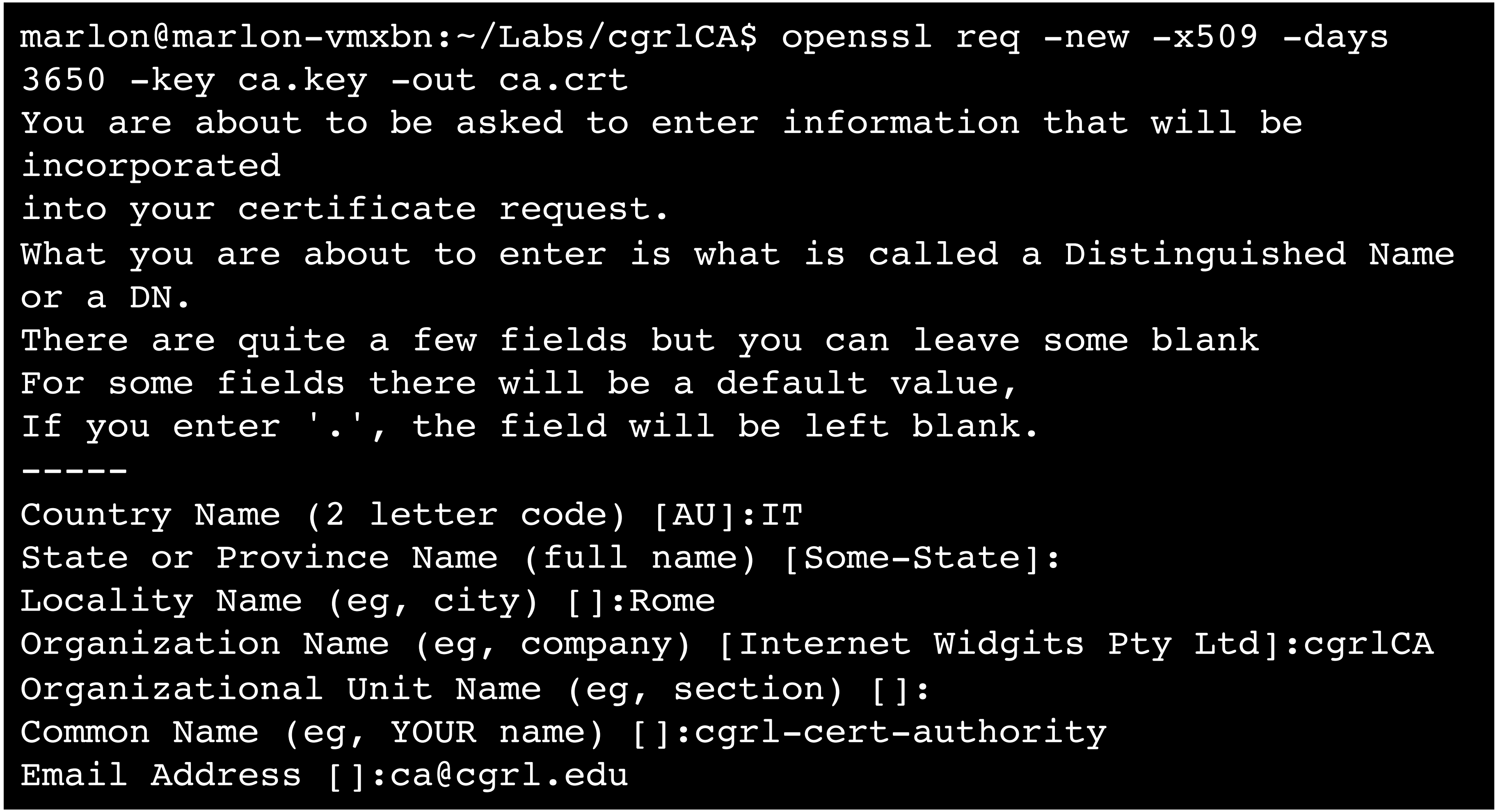
```
marlon@marlon-vmxbn:~/Labs$ mkdir cgrlCA
marlon@marlon-vmxbn:~/Labs$ cd cgrlCA/
marlon@marlon-vmxbn:~/Labs/cgrlCA$ echo -e "01\n" > serial
```

Create the CA key pair

```
marlon@marlon-vmxbn:~/Labs/cgrlCA$ openssl genrsa -out ca.key 2048
Generating RSA private key, 2048 bit long modulus
..................................+++
........+++
e is 65537 (0x10001)
```

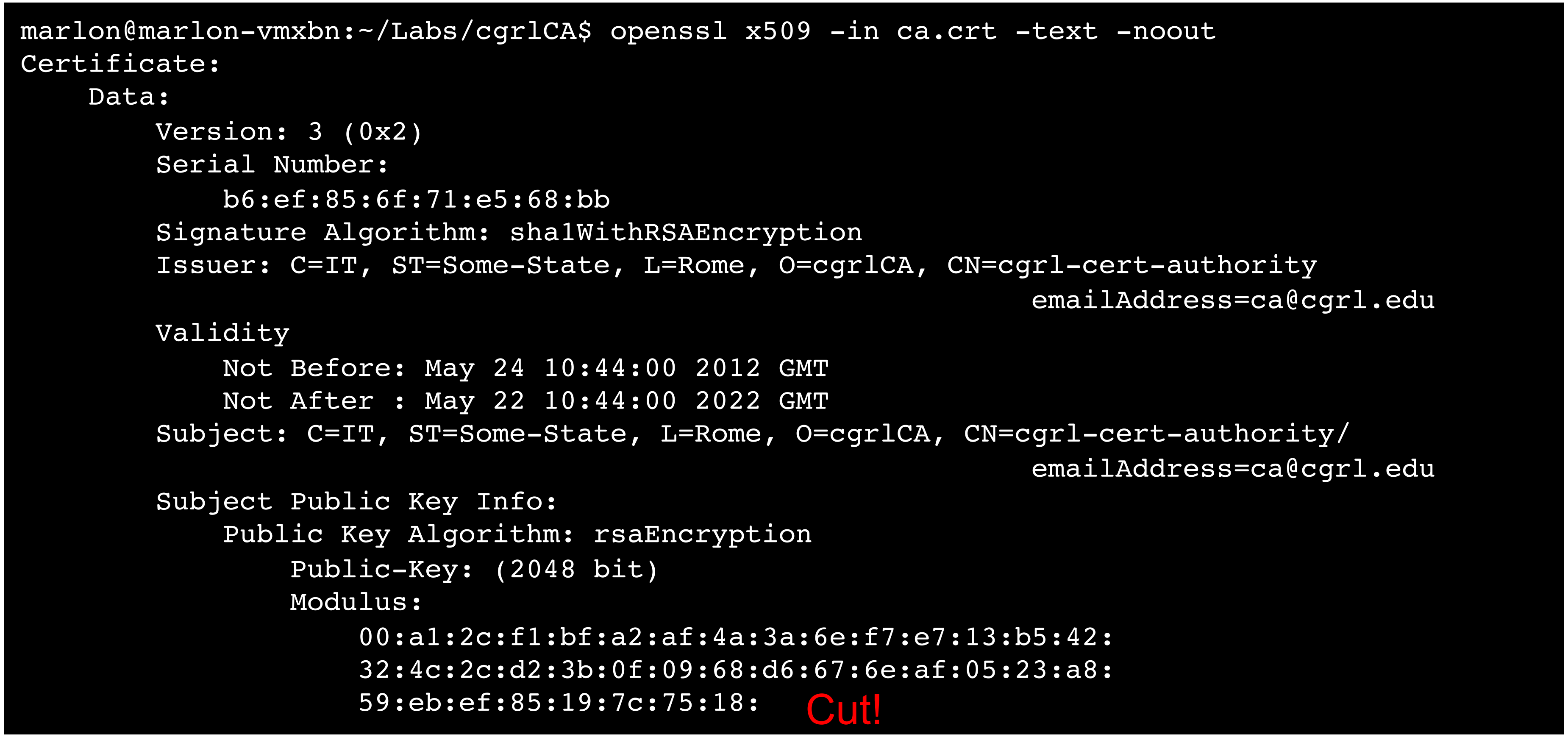
Generate the CA self signed certificate

This command will create a self signed certificate, i.e. a certificate where the issuer and the subject are the same entities



```
marlon@marlon-vmxbn:~/Labs/cgrlCA$ openssl req -new -x509 -days
3650 -key ca.key -out ca.crt
You are about to be asked to enter information that will be
incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name
or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:IT
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:Rome
Organization Name (eg, company) [Internet Widgits Pty Ltd]:cgrlCA
Organizational Unit Name (eg, section) []:
Common Name (eg, YOUR name) []:cgrl-cert-authority
Email Address []:ca@cgrl.edu
```

Let’s take a look at our first certificate



```
marlon@marlon-vmxbn:~/Labs/cgrlCA$ openssl x509 -in ca.crt -text -noout
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            b6:ef:85:6f:71:e5:68:bb
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=IT, ST=Some-State, L=Rome, O=cgrlCA, CN=cgrl-cert-authority
                                                            emailAddress=ca@cgrl.edu
        Validity
            Not Before: May 24 10:44:00 2012 GMT
            Not After : May 22 10:44:00 2022 GMT
        Subject: C=IT, ST=Some-State, L=Rome, O=cgrlCA, CN=cgrl-cert-authority/
                                                            emailAddress=ca@cgrl.edu
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:a1:2c:f1:bf:a2:af:4a:3a:6e:f7:e7:13:b5:42:
                    32:4c:2c:d2:3b:0f:09:68:d6:67:6e:af:05:23:a8:
                    59:eb:ef:85:19:7c:75:18:
```

Cut!

Let’s make the web server keys and CSR

Create the subject’s (i.e. our web server) key pair

```
marlon@marlon-vmxbn:~/Labs/cgrlCA$ openssl genrsa -out server.key 1024
Generating RSA private key, 1024 bit long modulus
.++++++
......................................................................++++++
e is 65537 (0x10001)
```

Create the subject’s CSR. This certificate will be signed with the CA’s private key

```
marlon@marlon-vmxbn:~/Labs/cgrlCA$ openssl req -new -key server.key -out
server.csr

Country Name (2 letter code) [AU]:IT
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:Rome
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (eg, YOUR name) []:testssl.cgrl.edu
Email Address []:testssl@cgrl.edu
```

This has to be
The web site FQDN

CSR signing

This command will sign the CSR with the CA’s private key

```
marlon@marlon-vmxbn:~/Labs/cgrlCA$ openssl x509 -req -in server.csr -out
server.crt -sha1 -CA ca.crt -CAkey ca.key -CAserial serial -days 3650
Signature ok
subject=/C=IT/ST=Some-State/L=Rome/O=Internet Widgits Pty Ltd/
CN=testssl.cgrl.edu/emailAddress=testssl@cgrl.edu
Getting CA Private Key
```

Dump the signed certificate

```
marlon@marlon-vmxbn:~/Labs/cgrlCA$ openssl x509 -in server.crt -text -noout
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number: 3 (0x3)
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=IT, ST=Some-State, L=Rome, O=cgrlCA, CN=cgrl-cert-authority/
                                                       emailAddress=ca@cgrl.edu
        Validity
            Not Before: May 24 10:50:25 2012 GMT
            Not After : May 22 10:50:25 2022 GMT
        Subject: C=IT, ST=Some-State, L=Rome, O=Internet Widgits Pty Ltd,
                                CN=testssl.cgrl.edu/emailAddress=testssl@cgrl.edu
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
```

Adding X509v3 extensions

When you sign a certificate set the following two options:
`-extfile [file_name]`
`-extensions [section_name]`

In opnessl configuration file (in /etc/ssl/openssl.conf) we already have 4 standard section defined:
`usr_cert, v3_req, v3_ca, crl_ext`

In addition, you can define extra sections

```
[ section_name ]
Option1=valye
OptionN=value
```

See https://www.openssl.org/docs/apps/x509v3_config.html for extensions

```
marlon@marlon-vmxbn:~/Labs/CA$ openssl x509 -req -in server.csr -out
server.crt -sha1 -CA ca.crt -CAkey ca.key -CAserial serial -days 3650 -
extfile /etc/ssl/openssl.conf -extensions usr_cert
Signature ok
subject=/C=IT/ST=Some-State/L=Rome/O=Internet Widgits Pty Ltd/
CN=testssl.cgrl.edu/emailAddress=testssl@cgrl.edu
Getting CA Private Key
```

How to protect our web server

HTTPS SERVER WITH APACHE2

Let’s configure Apache2

We are going to create a virtual host for the website “testssl.cgrl.edu” in the netkit lab “Lab9-pki”

Configuration file, keys and certificate already in server:root/
Webserver media file and index.html in server:/var/www/testssl

Set-up everything properly before enabling the new site

- Configuration file testssl.cgrl.edu goes into /etc/apache2/site-available
- Keys and Certificate in the proper directory (see the conf file)

Run the following commands:

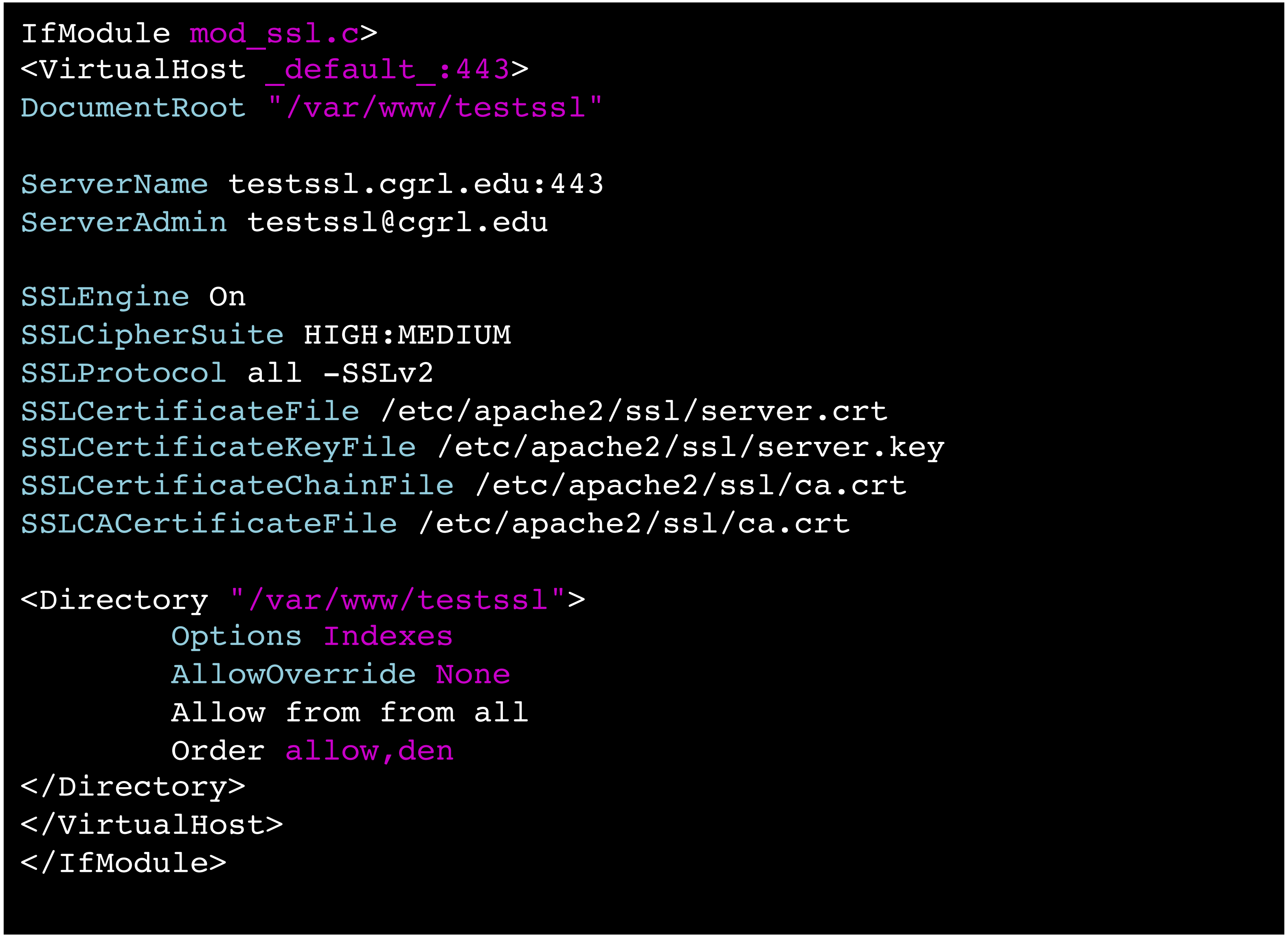
```
server# a2ensite testssl.cgrl.edu

server# a2enmod ssl

server# /etc/init.d/apache2 start
```

Enable our HTTPS web site

Enable Apache2 SSL module

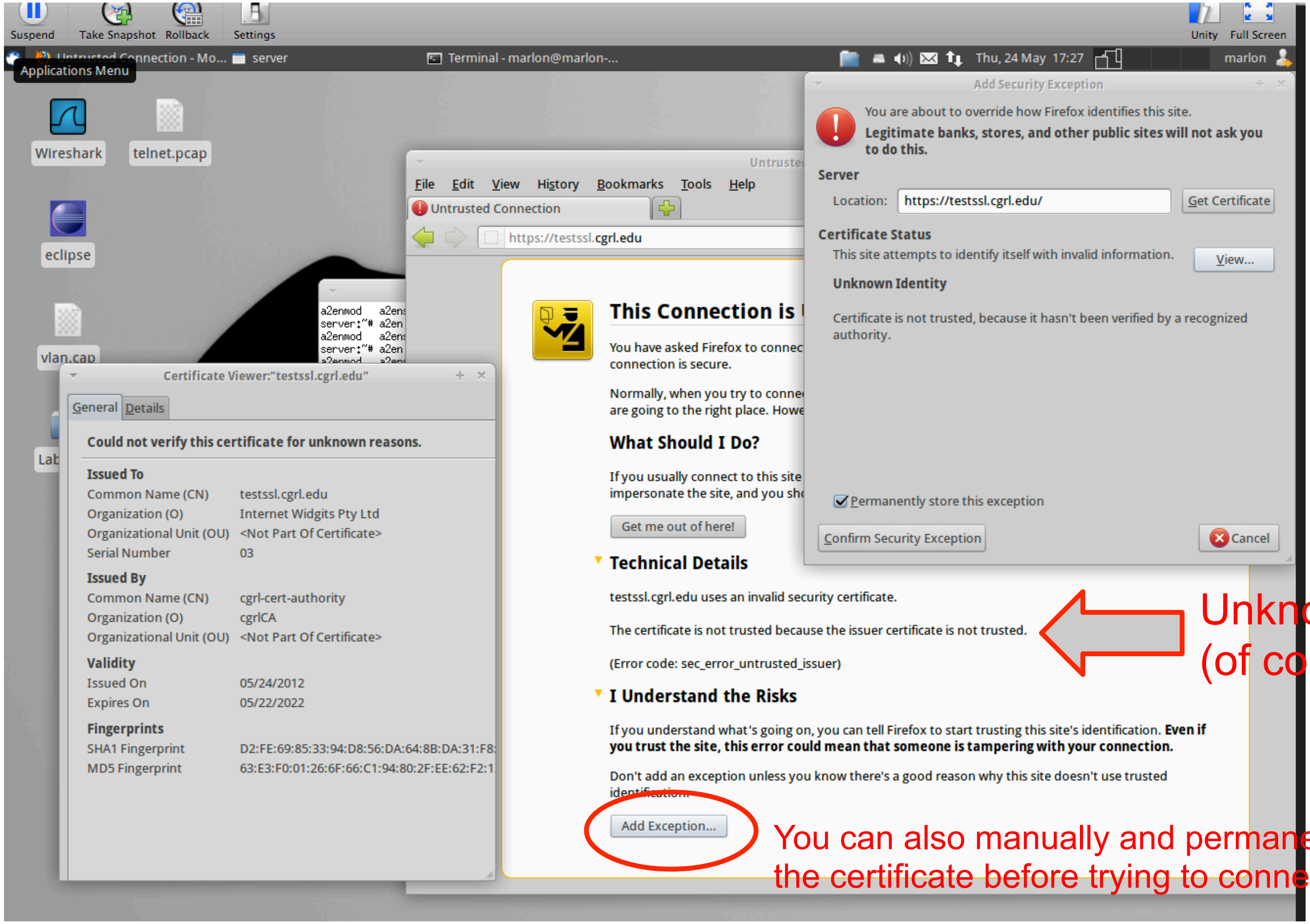
Start Apache2
(or “restart” if already up)

testssl.cgrl.edu config file



```
IfModule mod_ssl.c>
<VirtualHost _default_:443>
DocumentRoot "/var/www/testssl"

ServerName testssl.cgrl.edu:443
ServerAdmin testssl@cgrl.edu

SSLEngine On
SSLCipherSuite HIGH:MEDIUM
SSLProtocol all -SSLv2
SSLCertificateFile /etc/apache2/ssl/server.crt
SSLCertificateKeyFile /etc/apache2/ssl/server.key
SSLCertificateChainFile /etc/apache2/ssl/ca.crt
SSLCACertificateFile /etc/apache2/ssl/ca.crt

<Directory "/var/www/testssl">
        Options Indexes
        AllowOverride None
        Allow from all
        Order allow,den
</Directory>
</VirtualHost>
</IfModule>
```

Connect to the server



```
a2enmod    a2ens
server:~# a2en
a2enmod    a2ens
server:~# a2en
```

Certificate Viewer:"testssl.cgrl.edu"

Could not verify this certificate for unknown reasons.

Issued To

Common Name (CN) testssl.cgrl.edu
Organization (O) Internet Widgits Pty Ltd
Organizational Unit (OU) <Not Part Of Certificate>
Serial Number 03

Issued By

Common Name (CN) cgrl-cert-authority
Organization (O) cgrlCA
Organizational Unit (OU) <Not Part Of Certificate>

Validity

Issued On 05/24/2012
Expires On 05/22/2022

Fingerprints

SHA1 Fingerprint D2:FE:69:85:33:94:D8:56:DA:64:8B:DA:31:F8:
MD5 Fingerprint 63:E3:F0:01:26:6F:66:C1:94:80:2F:EE:62:F2:1

This Connection is

You have asked Firefox to connec
connection is secure.

Normally, when you try to conne
are going to the right place. Howe

What Should I Do?

If you usually connect to this site
impersonate the site, and you sho

Technical Details

testssl.cgrl.edu uses an invalid security certificate.

The certificate is not trusted because the issuer certificate is not trusted.

(Error code: sec_error_untrusted_issuer)

I Understand the Risks

If you understand what's going on, you can tell Firefox to start trusting this site's identification. **Even if you trust the site, this error could mean that someone is tampering with your connection.**

Don't add an exception unless you know there's a good reason why this site doesn't use trusted identification.

Add Security Exception

You are about to override how Firefox identifies this site.
Legitimate banks, stores, and other public sites will not ask you to do this.

Server

Location: https://testssl.cgrl.edu/

Certificate Status

This site attempts to identify itself with invalid information.

Unknown Identity

Certificate is not trusted, because it hasn't been verified by a recognized authority.

☑ Permanently store this exception

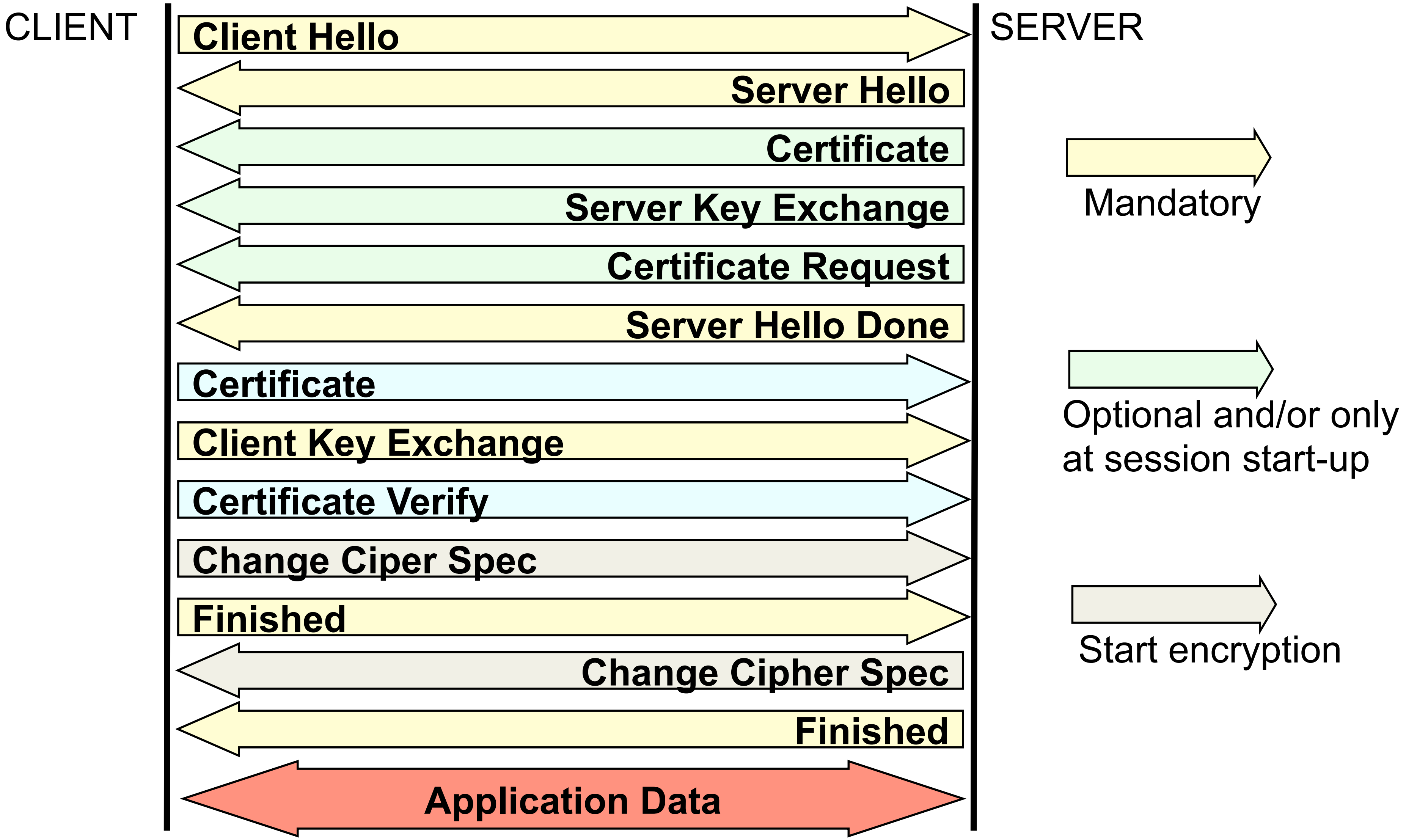
Unknown CA
(of course…)

You can also manually and permanently add the certificate before trying to connect

Note: append the following line to the file `/etc/hosts` on the host machine

```
testssl.cgrl.edu 10.0.0.2
```

TLS handshake



CLIENT

SERVER

Client Hello

Server Hello

Certificate

Server Key Exchange

Certificate Request

Server Hello Done

Certificate

Client Key Exchange

Certificate Verify

Change Ciper Spec

Finished

Change Cipher Spec

Finished

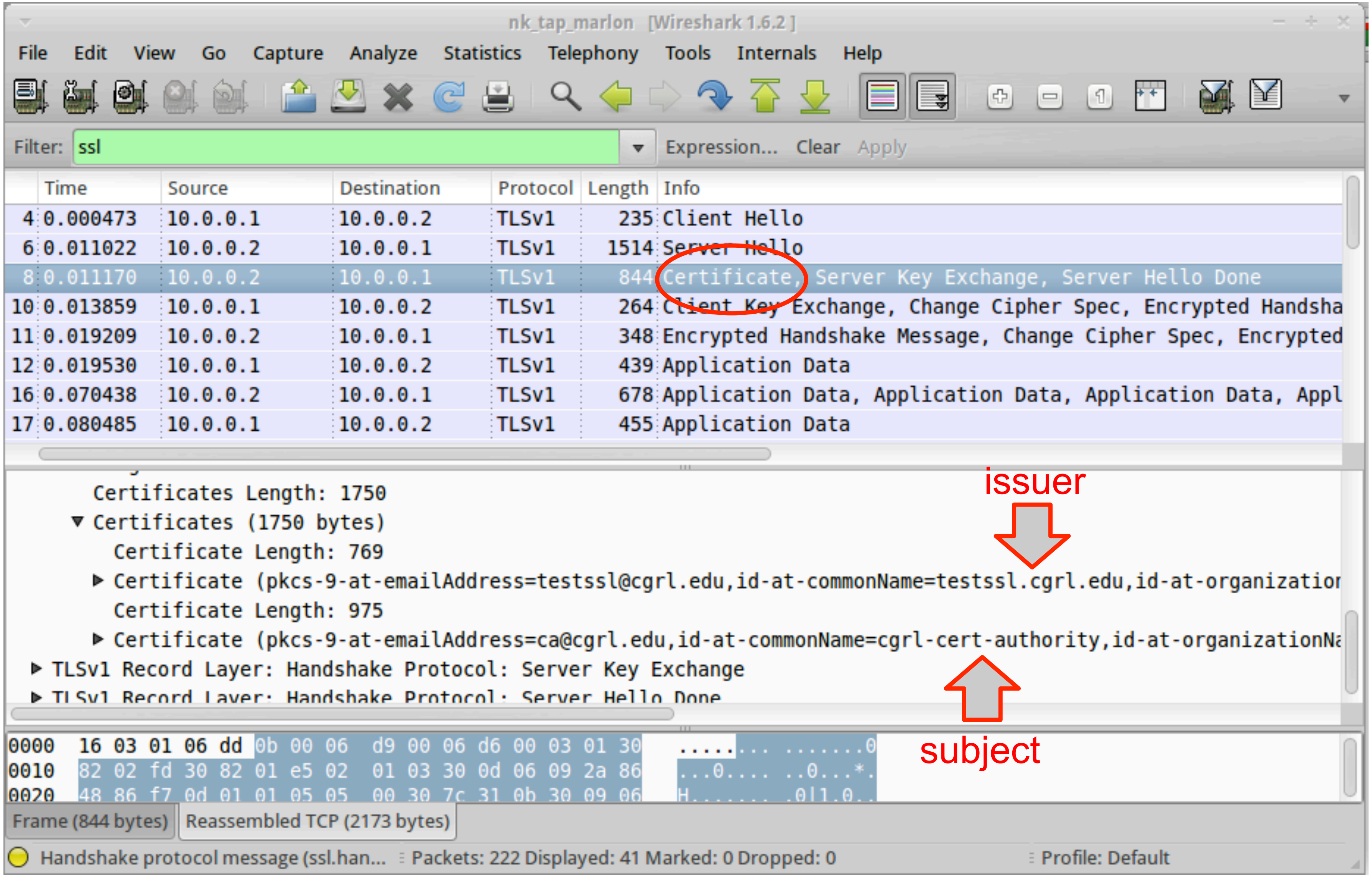
Application Data

Mandatory

Optional and/or only at session start-up

Start encryption

TLSv1 trace with our certificate



nk_tap_marlon [Wireshark 1.6.2]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply
issuer
subject

Filter: ssl

```
    Time      Source    Destination Protocol Length Info
 4 0.000473  10.0.0.1  10.0.0.2    TLSv1      235 Client Hello
 6 0.011022  10.0.0.2  10.0.0.1    TLSv1     1514 Server Hello
 8 0.011170  10.0.0.2  10.0.0.1    TLSv1      844 Certificate, Server Key Exchange, Server Hello Done
10 0.013859  10.0.0.1  10.0.0.2    TLSv1      264 Client Key Exchange, Change Cipher Spec, Encrypted Handsha
11 0.019209  10.0.0.2  10.0.0.1    TLSv1      348 Encrypted Handshake Message, Change Cipher Spec, Encrypted
12 0.019530  10.0.0.1  10.0.0.2    TLSv1      439 Application Data
16 0.070438  10.0.0.2  10.0.0.1    TLSv1      678 Application Data, Application Data, Application Data, Appl
17 0.080485  10.0.0.1  10.0.0.2    TLSv1      455 Application Data
```

```
      Certificates Length: 1750
    ▼ Certificates (1750 bytes)
        Certificate Length: 769
      ▶ Certificate (pkcs-9-at-emailAddress=testssl@cgrl.edu,id-at-commonName=testssl.cgrl.edu,id-at-organizatior
        Certificate Length: 975
      ▶ Certificate (pkcs-9-at-emailAddress=ca@cgrl.edu,id-at-commonName=cgrl-cert-authority,id-at-organizationNa
▶ TLSv1 Record Layer: Handshake Protocol: Server Key Exchange
▶ TLSv1 Record Layer: Handshake Protocol: Server Hello Done
```

```
0000  16 03 01 06 dd 0b 00 06  d9 00 06 d6 00 03 01 30   ........ .......0
0010  82 02 fd 30 82 01 e5 02  01 03 30 0d 06 09 2a 86   ...0.... ..0...*.
0020  48 86 f7 0d 01 01 05 05  00 30 7c 31 0b 30 09 06   H....... .0|1.0..
```

Frame (844 bytes) | Reassembled TCP (2173 bytes)

Handshake protocol message (ssl.han... Packets: 222 Displayed: 41 Marked: 0 Dropped: 0 Profile: Default

HTTP plaintext auth over TLS

- Safest way to authenticate via HTTP, better then digest auth
- You first create a secure channel with the authenticated web server
- You send authentication credential in clear (from the HTTP point of view) but inside the secure (encrypted/authenticated) TLS channel
- The test website already has the following password protected directory

```
<Directory "/var/www/testssl/secret">
    AuthType Basic
    AuthName "Username and Password Required"
    AuthUserFile /etc/apache2/.htpasswd
    Require valid-user
</Directory>
```

To try it you need to grant access to a new user, for example: uid “007” password “jamesbond”

```
server# htpasswd -c -m /etc/apache2/.htpasswd
007
New password:
```

Client authentication via X509 certificate

- The client may authenticate itself with a X509 certificate
- To do so we need to
  1. Configure the web server to force SSL client authentication

```
<Directory “/var/www/testssl/cert-required”>
    SSLVerifyClient require
    SSLVerifyDepth 1
</Directory>
```

  2. Create a client certificate and configure the web browser to use it (exported it in PCKS 12 format. NOTE: to use it with firefox you need to enable SSL renegotiation. With (my) chrome (v. 15.0.874.106 (Developer Build 107270 Linux) Ubuntu 11.10) it’s already OK)

```
server# openssl genrsa -out client.key 1024
server# openssl req -new -key client.key -out client.csr
server# openssl x509 -req -in client.csr -out client.crt -sha1 -CA
ca.crt -CAkey ca.key -CAserial serial -days 3650
server# openssl pkcs12 -export -in client.crt -inkey client.key -
out client.p12
```