X.509 and SSL

A look into the complex world of X.509 and SSL
http://www.phildev.net/ssl/

USC Linux Users Group
4/26/07

Phil Dibowitz
http://www.phildev.net/

The Outline

- Introduction of concepts
- X.509
- SSL
- End-User Notes
  (from personal users to sysadmins)
- Certificate Authority Notes
- Comparisons to PGP

Introduction: Basics

Lets start with the basics:

X.509 != SSL
SSL != PKI
PKI != RSA

and so on.

Introduction: PKI

- Public Key Infrastructure
- No need for shared secret
- Examples: RSA/DSA/PGP
- Allows separation of privilege / liability limitation
- Can safely distribute public key
- Distributing your public key means anyone can encrypt to you and verify your work

Introduction: PKI Operations

- Signing
  Use private key to “sign” data
- Verification
  Use public key to verify “signature”

- Encryption
  Use public key to encrypt data
- Decryption
  Use private key to decrypt data

Introduction: X.509

- X.509 is *one* of many standards for PKI
- Determines a format for certificates, keys, revocations, and others pieces
- Derived form X.500
- PKIX = Public Key Infrastructure X.509 Working Group

Introduction: SSL

- A protocol for establishing secure communication
- Built on X.509 to build encrypted “tunnels”
- Slowly being deprecated by TLS
- TLSv1 and SSLv3 are roughly the same
- Used every time you see the 'lock' in your browser
- Usually using RSA

Introduction: Putting It all together

- SSL sits on X.509
- X.509 sits on PKI

- Applications can then be wrapped in SSL for security

X.509

- Keypairs
- Certificate Signing Requests (CSR) (PKCS#10)
- Certificates (CRT) (PKCS#7)
- Certificate Authorities (CA)
- Extensions
- Certificate Revocation List (CRL)
- PKCS#12 – private and public key in one object

- Note: single “trusted” authority, unlike PGP web-of-trust

X.509: Keypairs

- X.509 is a PKI standard
- That means keypairs
- All PKI standards start with a public and private key, aka keypair
- X.509 is usually RSA. Sometimes DSA.

X.509: CSR

- Certificate Signing Request
- Generally the form the public key is generated in when using most X.509 tools
- Unsigned Certificate: public key embedded in metadata
- Signed by private key
- Additional verification usually required by CA in order to sign
- Sent to CA to request certificate

X.509: CSR Example 1

```
Certificate Request:
    Data:
        Version: 0 (0x0)
        Subject: C=US, ST=California, L=Los Angeles, O=Insanity Palace of Metallica,
    CN=mail.ipom.com/emailAddress=phil@ipom.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:b9:ac:cc:61:1e:29:b4:73:b8:cf:a3:cb:a0:eb:
                    85:e5:60:52:33:fc:94:93:6d:7c:11:c4:d8:5b:ae:
                    51:37:c3:53:ec:d9:31:cf:a8:55:a2:5c:5c:3f:21:
                    4b:19:94:d3:79:94:6a:e5:d9:29:c9:3d:d2:71:f6:
                    a8:aa:38:ff:dd:df:21:59:3b:60:ab:1f:3d:f3:c8:
                    b7:26:d7:be:a7:86:f1:db:1c:01:6c:23:41:a7:86:
                    d4:78:96:b5:b7:af:ea:d9:b5:5e:30:37:cc:dd:1a:
                    eb:2c:4a:4e:48:2d:16:94:cd:c8:a3:d5:97:77:13:
                    f8:17:f1:17:67:44:75:5c:a7
                Exponent: 65537 (0x10001)
        Attributes:
            a0:00
    Signature Algorithm: md5WithRSAEncryption
        54:78:be:74:b1:d7:87:bc:aa:6c:ba:44:94:0e:7f:6e:af:1f:
        fc:8e:d0:78:7f:aa:a5:45:ad:6c:bf:c0:d9:5d:21:7d:2a:7b:
        4c:2e:1c:e2:57:d2:50:a2:6d:7c:5a:a8:2f:72:98:99:f7:92:
        83:5f:35:7e:ce:fa:c1:cf:8a:31:99:ad:eb:a8:47:ff:21:d0:
        0a:54:b1:1a:5e:db:7e:30:b1:e6:b6:d0:2d:f4:c9:46:ae:81:
        a3:46:72:3f:e2:6e:09:2c:c6:6f:dd:01:35:d0:c4:13:39:88:
        82:7b:fb:c7:96:7c:b2:2a:bd:6a:03:7b:34:71:76:95:9d:d7:
        a5:39
```

X.509: CSR Example 2

```
Certificate Request:
    Data:
        Version: 0 (0x0)
        Subject: C=US, ST=California, L=West Hollywood, O=Ticketmaster, OU=Websys,
    CN=metallica.office.tmcs/emailAddress=phil@ticketmaster.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:be:a0:5e:35:99:1c:d3:49:ba:fb:2f:87:6f:d8:
                    ed:e4:61:f2:ae:6e:87:d0:e2:c0:fd:c1:0f:ed:d7:
                    84:04:b5:c5:66:cd:6b:f0:27:a2:cb:aa:3b:d7:ad:
                    fa:f4:72:10:08:84:88:19:24:d0:b0:0b:a0:71:6d:
                    23:5e:53:4f:1b:43:07:98:4d:d1:ea:00:d1:e2:29:
                    ea:be:a9:c5:3e:78:f3:5e:30:1b:6c:98:16:60:ba:
                    61:57:63:5e:6a:b5:99:17:1c:ae:a2:86:fb:5b:8b:
                    24:46:59:3f:e9:84:06:e2:91:b9:2f:9f:98:04:01:
                    db:38:2f:5b:1f:85:c1:20:eb
                Exponent: 65537 (0x10001)
        Attributes:
            a0:00
    Signature Algorithm: sha1WithRSAEncryption
        ac:97:ca:4b:3c:c5:f0:32:c8:29:ea:2b:62:e7:39:b5:83:64:
        14:fe:6e:36:d0:7d:f0:73:34:7a:10:0e:d3:21:92:c1:b0:12:
        49:54:c7:f1:4f:c1:d3:51:4f:08:c3:bc:26:15:7d:df:44:44:
        a4:0a:82:e3:0e:9f:7f:fb:4b:9c:3f:a6:bf:59:a9:d6:ef:2e:
        52:dd:be:4d:a1:0d:7a:e7:88:ea:36:da:6d:fe:48:ac:99:51:
        54:8d:04:6c:e5:59:ab:e7:1b:ef:de:66:15:88:b2:4b:94:e3:
        ec:7b:ec:5f:85:6f:17:61:df:b4:3d:9f:b0:8c:78:08:43:2a:
        3f:3d
```

X.509: CRT

- Certificate
- A CSR signed by a Certificate Authority
- Often contains additional metadata, usually in the form of additional extensions
- CA uses its name to tie the public key to a subject.
- It can alter parts of the certificate before signing.

X.509: CRT Example 1

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            e5:ed:04:03:1b:0b:a7:9d
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=US, ST=California, O=PhilNet, CN=PhilNet CA v1
        Validity
            Not Before: Apr 22 09:18:53 2007 GMT
            Not After : Apr 21 09:18:53 2008 GMT
        Subject: C=US, ST=California, O=Insanity Palace of Metallica, CN=mail.ipom.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:b9:ac:cc:61:1e:29:b4:73:b8:cf:a3:cb:a0:eb:
                    ...
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Subject Key Identifier:
                92:CE:18:FD:03:AB:33:44:D2:D4:61:C0:A1:56:71:31:57:55:4F:06
            X509v3 Authority Key Identifier:
                keyid:FF:1A:C0:C5:42:8D:4C:67:97:2F:2A:34:E1:10:6C:80:5B:8B:F4:B7

            X509v3 Subject Alternative Name:
                email:phil@ipom.com
            X509v3 CRL Distribution Points:
                URI:http://www.phildev.net/philnet.crl

    Signature Algorithm: sha1WithRSAEncryption
        b4:5a:d7:43:fa:34:60:53:27:94:e0:e0:bc:34:12:e3:72:2d:
        ...
```

X.509: CRT Example 2

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            8e:a1:f1:dd:ec:fd:e9:b9:b0:9c:8e:e4:09:e7:ee:96
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=US, ST=California, L=West Hollywood, O=Ticketmaster, OU=Systems Engineering, CN=Ticketmaster Phil
     Test CA v1/emailAddress=phil@ticketmaster.com
        Validity
            Not Before: May 22 23:39:01 2006 GMT
            Not After : Jan 15 19:28:49 2008 GMT
        Subject: C=US, ST=California, L=West Hollywood, O=Ticketmaster, OU=Websys,
     CN=metallica.office.tmcs/emailAddress=phil@ticketmaster.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:be:a0:5e:35:99:1c:d3:49:ba:fb:2f:87:6f:d8:
                    ...
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Key Usage: critical
                Key Encipherment, Data Encipherment, Key Agreement
            Netscape Cert Type:
                SSL Server
            X509v3 Authority Key Identifier:
                keyid:A1:BD:91:24:39:47:DA:FB:CB:D0:E6:16:C5:C5:6E:74:B6:B9:D0:8E

            X509v3 Subject Key Identifier:
                0A:AC:A0:1A:80:3C:F2:CA:80:DF:8A:7A:8D:5D:2D:0C:C5:13:CA:AA
            X509v3 CRL Distribution Points:
                URI:http://www.ticketmaster.com/crl/Ticketmaster%20Phil%20Test%20CA%20v1.crl

    Signature Algorithm: sha1WithRSAEncryption
        68:bd:d3:7f:0b:20:e7:da:d8:15:6d:13:b0:50:a1:60:66:a6:
        ....
```

X.509: CRT Example 3

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            68:b7:5c:2c:ba:ba:d9:00:91:00:dd:b5:5d:eb:c7:2d
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=US, O=RSA Data Security, Inc., OU=Secure Server Certification Authority
        Validity
            Not Before: May 25 00:00:00 2006 GMT
            Not After : May 24 23:59:59 2009 GMT
        Subject: C=US, ST=California, L=West Hollywood, O=Ticketmaster, OU=Web Systems, OU=Terms of use at
    www.verisign.com/rpa (c)05, CN=www.ticketmaster.com
        Subject Public Key Info:
            ...
        X509v3 extensions:
            X509v3 Basic Constraints:
            CA:FALSE
            X509v3 Key Usage:
            Digital Signature, Key Encipherment
            X509v3 CRL Distribution Points:
            URI:http://SVRSecure-crl.verisign.com/SVRSecure.crl
            X509v3 Certificate Policies:
            Policy: 2.16.840.1.113733.1.7.23.3
              CPS: https://www.verisign.com/rpa
            X509v3 Extended Key Usage:
            TLS Web Server Authentication, TLS Web Client Authentication
            Authority Information Access:
            OCSP - URI:http://ocsp.verisign.com
            1.3.6.1.5.5.7.1.12:
                0_.].[0Y0W0U..image/gif0!0.0...+.............k...j.H.,{..0%.#http://logo.verisign.com/vslogo.gif
    Signature Algorithm: sha1WithRSAEncryption
        3d:bb:44:3d:a5:11:84:08:46:bf:5b:0c:b7:12:df:a5:21:ad:
        ...
```

X.509: CA

- Certificate Authority
- A central party all people “trust”
  - verify identity
  - verify correct key
- End user must have CA public certificate to verify certificates it signed
- Just a certificate that signs other certificates
- “Root CA” - CA signed by itself
- “Intermediate CA” - Any CA signed by another CA

X.509: CA Example 1

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            e5:ed:04:03:1b:0b:a7:9c
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=US, ST=California, O=PhilNet, CN=PhilNet CA v1
        Validity
            Not Before: Apr 22 09:06:18 2007 GMT
            Not After : Apr 21 09:06:18 2010 GMT
        Subject: C=US, ST=California, O=PhilNet, CN=PhilNet CA v1
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:bb:41:14:f8:91:06:6d:d7:54:3f:f3:b2:8c:10:
                    ...
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                FF:1A:C0:C5:42:8D:4C:67:97:2F:2A:34:E1:10:6C:80:5B:8B:F4:B7
            X509v3 Authority Key Identifier:
                keyid:FF:1A:C0:C5:42:8D:4C:67:97:2F:2A:34:E1:10:6C:80:5B:8B:F4:B7
                DirName:/C=US/ST=California/O=PhilNet/CN=PhilNet CA v1
                serial:E5:ED:04:03:1B:0B:A7:9C

            X509v3 Basic Constraints:
                CA:TRUE
            X509v3 CRL Distribution Points:
                URI:http://www.phildev.net/philnet.crl

            X509v3 Subject Alternative Name:
                email:phil@ipom.com
    Signature Algorithm: sha1WithRSAEncryption
        6d:d2:54:7f:81:7a:78:48:ce:ee:df:0c:6a:e8:26:5d:e0:92:
        ...
```

X.509: CA Example 2

```
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number:
            70:ba:e4:1d:10:d9:29:34:b6:38:ca:7b:03:cc:ba:bf
        Signature Algorithm: md2WithRSAEncryption
        Issuer: C=US, O=VeriSign, Inc., OU=Class 3 Public Primary Certification
    Authority
        Validity
            Not Before: Jan 29 00:00:00 1996 GMT
            Not After : Aug  1 23:59:59 2028 GMT
        Subject: C=US, O=VeriSign, Inc., OU=Class 3 Public Primary Certification
    Authority
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:c9:5c:59:9e:f2:1b:8a:01:14:b4:10:df:04:40:
                    ...
                Exponent: 65537 (0x10001)
    Signature Algorithm: md2WithRSAEncryption
        bb:4c:12:2b:cf:2c:26:00:4f:14:13:dd:a6:fb:fc:0a:11:84:
       ...
```

X.509: CA Example 3

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            75:33:7d:9a:b0:e1:23:3b:ae:2d:7d:e4:46:91:62:d4
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=US, O=VeriSign, Inc., OU=Class 3 Public Primary Certification Authority
        Validity
            Not Before: Jan 19 00:00:00 2005 GMT
            Not After : Jan 18 23:59:59 2015 GMT
        Subject: C=US, O=VeriSign, Inc., OU=VeriSign Trust Network, OU=Terms of use at https://www.verisign.com/rpa (c)05,
     CN=VeriSign Class 3 Secure Server CA
        Subject Public Key Info:
            ...
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:TRUE, pathlen:0
            X509v3 Certificate Policies:
                Policy: 2.16.840.1.113733.1.7.23.3
                  CPS: https://www.verisign.com/rpa
            X509v3 CRL Distribution Points:
                URI:http://crl.verisign.com/pca3.crl
            X509v3 Key Usage: critical
                Certificate Sign, CRL Sign
            Netscape Cert Type:
                SSL CA, S/MIME CA
            X509v3 Subject Alternative Name:
                DirName:/CN=Class3CA2048-1-45
            X509v3 Subject Key Identifier:
                6F:EC:AF:A0:DD:8A:A4:EF:F5:2A:10:67:2D:3F:55:82:BC:D7:EF:25
            X509v3 Authority Key Identifier:
                DirName:/C=US/O=VeriSign, Inc./OU=Class 3 Public Primary Certification Authority
                serial:70:BA:E4:1D:10:D9:29:34:B6:38:CA:7B:03:CC:BA:BF
    Signature Algorithm: sha1WithRSAEncryption
        c3:7e:08:46:5d:91:36:cf:67:dc:d7:a7:af:af:b8:22:c3:8b:
        ...
```

X.509: Extensions

- Subject Key Identifier (SKID) – a hash of the public key
- Authority Key Identifier (AKID)
  - A hash of the Issuer's public key (CA's SKID) and/or
  - The issuer and serial number of the CA
- CRL Distribution Point – where do I find revocation information?
- Extensions that define key usage limitations: Basic Constraints, X509v3 Key Usage, Netscape Cert Type

X.509 Extensions

- Subject Alternative Name – additional hostnames, email address, IP Addresses and more to associate with the cert
- Critical Extensions: Fail if you don't understand it.

X.509 CRLs

- A list of certificates revoked by a CA
- Contains expiration date
- Signed by CA's private key
- Shouldn't be SSL protected (circular problem)

X.509 CRL Example

```
Certificate Revocation List (CRL):
        Version 2 (0x1)
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: /C=US/ST=California/L=Hawthorne/O=PhilNet/CN=PhilNet CA
        Last Update: Mar  3 23:16:49 2007 GMT
        Next Update: Mar 10 23:16:49 2007 GMT
        CRL extensions:
            X509v3 CRL Number:
                1
Revoked Certificates:
    Serial Number: E5FE840E9495462E
        Revocation Date: Sep 14 01:19:48 2006 GMT
    Signature Algorithm: sha1WithRSAEncryption
        3f:f2:76:52:dd:30:4e:a6:6b:a0:b6:4d:99:af:b5:fa:e3:5e:
        f4:d0:c1:36:d9:76:cf:88:19:5b:79:e3:69:a4:f6:4c:8a:be:
        2e:82:af:2f:7b:20:8e:8c:7c:01:9d:59:ea:17:6e:63:c1:53:
        85:94:da:40:0c:b6:9c:ab:13:18:04:b9:12:f6:d8:57:7f:03:
        cd:9a:3d:36:f7:11:f6:01:f6:59:95:1a:77:e6:5e:9d:dd:10:
        6f:04:a9:7b:b4:7b:5c:31:f3:4d:37:a6:c4:6f:45:11:b7:45:
        80:d4:42:ee:4b:96:fb:c5:17:78:f8:e1:89:af:a9:05:cd:22:
        0a:63
```

SSL

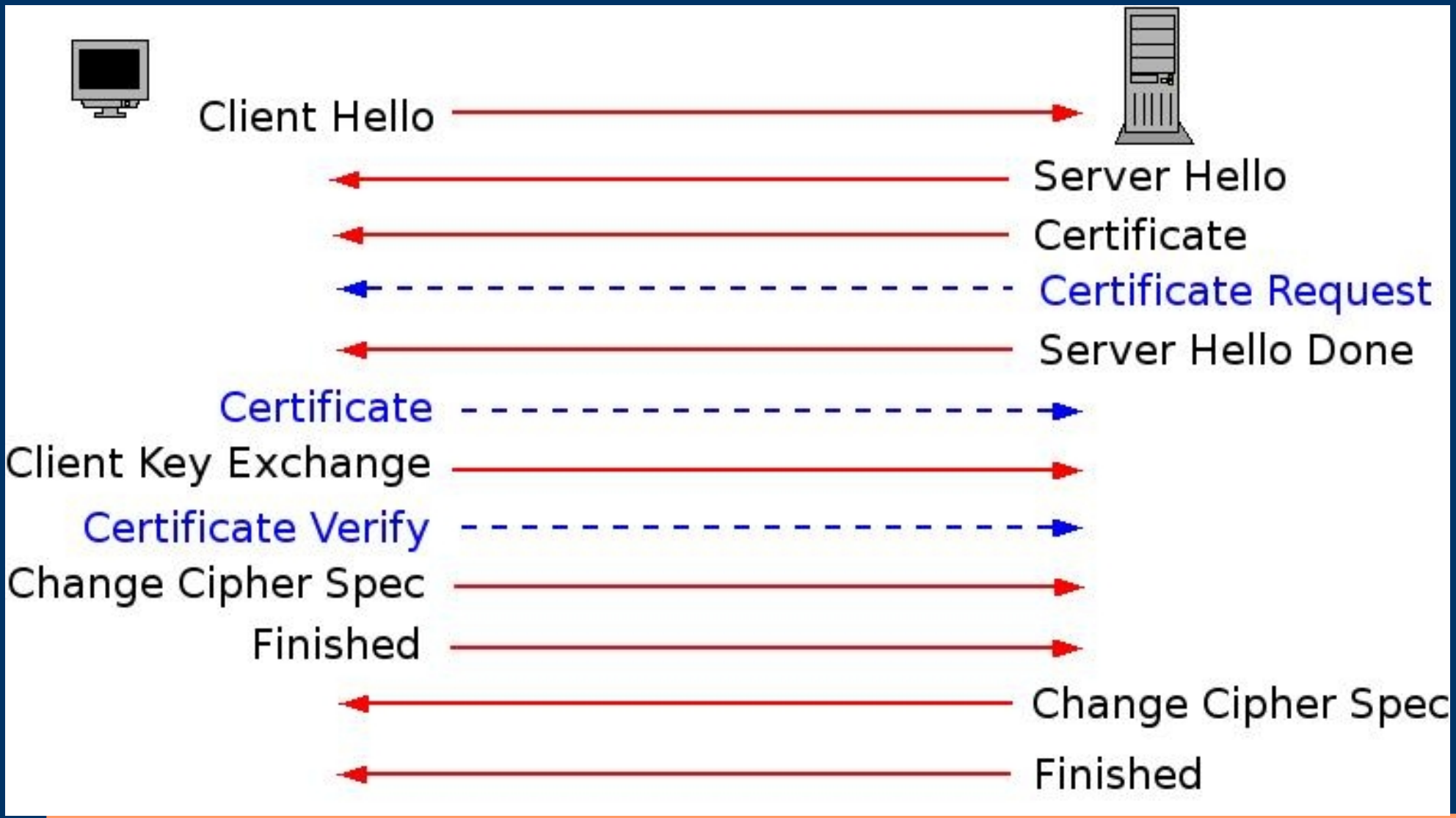
- Secure Sockets Layer
- Being deprecated by TLS
- SSLv3 is roughly equivalent to TLSv1
- TLS = Transport Layer Security
- SSL/TLS is a way to verify the part you connect to and create an encrypted tunnel for security communication
- Also allows for client authentication

SSL: Negotiation

- Client connects to server
- Server provides certificate (and optionally, certificate path)
- Client verifies certificate using own trusted roots
- Optionally: client provides certificate and server verifies
- Client uses server public key to encrypt symmetric key and send it to server
- This new key used to encrypt session

SSL: Negotiation

- OK, maybe there's a few more steps...



SSL: Negotiation: CAs

- Client (and server if using client auth) **MUST** already have a copy of the root CA **and** trust it to verify cert.
- Intermediate certs **MAY** be passed with certificate
- SSL Server Authentication: intermediate certs are **not** usually passed, but may be (i.e. new verisign CA cert)
- SSL Client Authentication: intermediate certs **are** usually passed

SSL: Certificate Verification

- Is the certificate valid (not expired?)
- Is the certificate signed by a CA we trust (or does it at least lead up to one we trust)?
- Is that signature (and all signatures in the chain) valid?
- Optionally: is the certificate revoked? (involves retrieving and verifying CRL)

SSL: Wait... symmetric key?

- Computationally easier and quicker
- Different key each time adds security
- In many cases client doesn't have a key, so a key would have to be generated anway

End-User Notes

- OpenSSL is devided into several subcommands:
  - **req** – PKCS#10 / CSRs
  - **x509** – certificates
  - **crl** – revocation lists
- Most options are the same across subcommands:
  - **-inform** – Is the file passed to '-in' in PEM or DER format
  - **-in** – read this file
  - **-noout** – don't print the actual cert/crl/csr
  - **-text** – print out the text of what's in the object

End-User Notes: Commands

- Create keypair:

  *openssl req -newkey rsa:1024 -keyout server.key *
  -out server.csr
- This creates a private key, and a CSR with a public key in it
- If you already have a private key key:

  openssl req -key server.key -out server.csr
- To view your key:

  openssl rsa -noout -text -in server.key
- To view your csr:

  openssl req -noout -text -in server.csr
- To view your crt:

  openssl x509 -noout -text -in server.crt

End-User Notes: Commands

- Verify a certificate against a CA
  openssl verify -CAfile ca.crt server.crt
- What if there's a CA-chain?
  cat root-ca.crt sub-ca.crt > bundle.crt
  openssl verify -CAfile bundle.crt server.crt
- Get subject “hash”
  openssl x509 -noout -hash -in server.crt

End-User Notes: HTTPS

- For Apache HTTPS Server
  - **SSLCertificateFile** – **Server's** certificate
  - **SSLCertificateKeyFile** – Matching private key
  - **SSLCertificateChainFile** – Intermediate CAs
  - **SSLCACertificateFile** – Do **NOT** put intermediate CAs here. Verisign recommends here, but here == bad.

End-User Notes: HTTPS

- For Apache accepting HTTPS Clients
  - **SSLCACertificateFile** - Valid CAs client certs may be signed with. Like the CAs in your browser.
  - **SSLCARevocationFile** – CRL bundle to use when validating client certificates
  - **SSLCADNRequestFile** – CAs names to send to the client as allowable CAs (so a client may select one if they have multiple)
  - **SSLVerifyClient** – Set to none/optional/required
  - **SSLVerifyDepth** – How deep down the rabbit hole are you willing to go?

End-User Notes: Private Keys

- Keep your private key **private**!
- Where possible, keep it encrypted
- If not encrypted, mode 400
- If your company relies on PKI, consider a keystore (i.e. Ingrian)
- Report key compromises to the CA **immediately** so the cert may be revoked

CA Notes

- Being a CA is a lot of responsibility
- Verification, policy, etc.

- There are two ways to build an OpenSSL CA
  - The standard way
  - The PKIX way

CA Notes: PKIX

- PKIX says that email address should **not** be in subject
- PKIX says that email address **should** be in SubjectAltName

- PKIX says lots of things, but this one is tricky with OpenSSL, unfortunately.

- We'll violate one PKIX rule: critical extensions.

CA Notes: Preparation

- This is the same for both methods:
  - *mkdir -p CA/{certsdb,certreqs,crl,private}*
  - *chmod 700 CA/private*
  - *touch CA/index.txt*
- In your new CA directory, make a copy of your system openssl.cnf
- Modify your openssl.cnf...

CA Notes: Preparation

- Set paths...
  - dir = <path_to_CA>
  - certs = $dir/certsdb
  - new_certs_dir = $certs
  - database = $dir/index.txt
  - certificate = $dir/cacert.pem
  - serial = $dir/serial
  - crldir = $dir/crl
  - crlnumber = $dir/crlnumber
  - crl = $crldir/crl.pem
  - private_key = $dir/private/cakey.pem
- Many systems use various names for these directories, so we must make sure we adjust to what we made

CA Notes: Preparation

- Set various options...
  - Section for extensions
    - x509_extensions = usr_cert
  - Expirations
    - default_dates = 365
    - default_crl_days = 30
  - Honor Extensions In Request
    - copy_extensions = copy
  - Set a policy
    - policy = policy_match
    - More on policies in a bit...

CA Notes: Preparation

- Set Extensions
  - Recall, we set 'usr_cert' as the place x509 extensions are stored
  - So, under “[usr_cert]”:
    - basicConstraints=CA:false
    - subjectKeyIdentifier=hash
    - authorityKeyIdentifier=keyid,issuer
    - crlDistributionPoints=URI:http://example.com/ca.crl
  - We'll also define an additional extension section for signing CAs (probably only ourself)
  - Under “[v3_ca]”
    - basicConstraints=CA:true
    - subjectKeyIdentifier=hash
    - authorityKeyIdentifier=keyid:always,issuer:always
    - crlDistributionPoints=URI:http://example.com/ca.crl

CA Notes: The standard way

- Create keypair:
  - *openssl req -new -keyout private/cakey.pem -out \ careq.pem -config ./openssl.cnf*
- Self-sign:
  - *openssl ca -create_serial -out cacert.pem -days 365 \ -keyfile private/cakey.pem -selfsign -extensions \ v3_ca -config ./openssl.cnf -infiles careq.pem*

- *Note*
  - *-create_serial* and *-selfsign*
  - Possible in one step with 'req', but then you can't use -create_serial

CA Notes: Let's have a look...

```
$ openssl req -noout -text -in careq.pem
Certificate Request:
    Data:
        Version: 0 (0x0)
        Subject: C=US, ST=California, L=Hawthorne, O=PhilNet, CN=test/emailAddress=phil@ipom.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:f9:c6:90:7f:07:14:ad:15:a9:be:ee:f0:27:d5:
                    ef:d3:cf:16:ec:72:32:1e:86:7f:90:8f:85:50:e4:
                    0b:cd:f1:8a:00:dc:a3:f1:e2:bb:8d:44:75:bf:a8:
                    bf:72:55:f6:3c:06:a9:e9:d5:6a:38:7f:6b:01:31:
                    cc:15:fb:1b:2b:d3:9e:f7:e1:0d:53:cb:7f:42:fc:
                    ba:d4:af:a8:ac:6e:8c:59:c2:dd:d9:e5:27:b0:3a:
                    c2:ab:25:26:1e:77:36:be:32:ea:e6:94:30:0b:96:
                    7b:36:92:8c:5c:ee:ad:69:52:8f:23:83:42:34:8a:
                    66:da:f8:44:da:13:1a:f7:8f
                Exponent: 65537 (0x10001)
        Attributes:
            a0:00
    Signature Algorithm: sha1WithRSAEncryption
        6c:a2:2b:b9:5a:5b:ca:90:f1:c5:a5:16:87:ca:a2:90:8b:74:
        07:ad:db:6e:6f:53:2b:97:24:a7:56:95:b4:e6:5f:2e:88:8b:
        ba:0e:ac:00:99:3d:16:18:a4:8c:41:f2:2c:69:48:a8:38:56:
        37:a0:ed:91:bd:53:79:ef:13:10:57:ba:bf:89:48:52:1d:93:
        72:18:c1:ce:f8:e7:da:d8:b0:3e:a0:8f:f8:d3:4a:6c:f8:72:
        62:0b:53:07:d5:f5:90:5e:dc:d3:94:87:34:9d:e4:e5:b9:fc:
        f3:f2:c6:dd:5c:58:9b:5c:b4:33:b2:f5:5a:57:42:9f:89:69:
        d0:71
```

CA Notes: Let's have a look

```
$ openssl x509 -noout -text -in cacert.pem
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            b9:bf:e9:b1:55:b9:ad:68
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=US, ST=California, L=Hawthorne, O=PhilNet, CN=test/emailAddress=phil@ipom.com
        Validity
            Not Before: Apr 22 02:26:20 2007 GMT
            Not After : Apr 21 02:26:20 2008 GMT
        Subject: C=US, ST=California, L=Hawthorne, O=PhilNet, CN=test/emailAddress=phil@ipom.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                  ...
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:TRUE
            X509v3 Subject Key Identifier:
                8A:D1:CD:4F:23:F6:86:81:57:04:F8:62:50:5E:1D:47:8B:02:69:CF
            X509v3 Authority Key Identifier:
                keyid:8A:D1:CD:4F:23:F6:86:81:57:04:F8:62:50:5E:1D:47:8B:02:69:CF
                DirName:/C=US/ST=California/L=Hawthorne/O=PhilNet/CN=test/emailAddress=phil@ipom.com
                serial:B9:BF:E9:B1:55:B9:AD:68

            X509v3 CRL Distribution Points:
                URI:http://alt.home.pv/philnet_ca.crl

    Signature Algorithm: sha1WithRSAEncryption
        ...
```

CA Notes

- Perfectly good CA
- It can sign certs, generate CRLs, etc.
- But... it's not PKIX compliant!

- Looking at man page...
  - Generate proper req with SubjectAltName=email:move ?
  - We also may play with copy_extensions...

CA Notes: PKIX

- Under “[req]” set
  - req_extensions = v3_req
- Under “[v3_req]” do
  - “SubjectAltName=email;move”

- This says make SubjectAltName in req
- And try it again...
- Looks good...

CA Notes: PKIX

- Looks good
  - Issuer: C=US, ST=California, L=Hawthorne, O=PhilNet, CN=test
  - Subject: C=US, ST=California, L=Hawthorne, O=PhilNet, CN=test
  - X509v3 Subject Alternative Name:
    email:phil@ipom.com
- Even CSR looks good:
  - Subject: C=US, ST=California, L=Hawthorne, O=PhilNet, CN=test
  - Requested Extensions:
    X509v3 Subject Alternative Name:
    email:phil@ipom.com

CA Notes: PKIX

- Important Notes
  - If you request the extension in x509_extensions, but don't set a value in req_extensions, SAN will be blank (not in Subject to be copied from).
  - req_extensions can't have AKID, crlDP, or other extensions only a CA can set. These must be in x509_extensions
  - With “subjectAltName = email:move” under “usr_cert” any certificate with SAN will LOOSE it upon signing (moving 'null' from Subject to SAN)!!
  - Same thing for v3_ca and signing future subordinate CAs!!
  - Without them, non-PKIX CSRs won't become PKIX upon signing!!
  - Suck... need multiple extension profiles

CA Notes: PKIX

- Extension Profiles Needed
  - usr_cert
  - usr_cert_has_san
  - v3_ca
  - v3_ca_has_san
- “has_san” ones don't do “subjectAltNames=email:move”

CA Notes: Policies

- Policies determine criteria for a CSR before signing it
- For each piece of the Subject, you can specify *optional*, *supplied*, or *match*
  - **Optional** – we don't care if it is there
  - **Supplied** – It has to be there, we don't care what it is
  - **Match** – It must match the CAs exactly

CA Notes: Policies

```
[ policy_match ]
countryName             = match
stateOrProvinceName     = match
localityName            = supplied
organizationName        = match
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional
```

CA Notes: Policies

```
[ policy_anything ]
countryName             = optional
stateOrProvinceName     = optional
localityName            = optional
organizationName        = optional
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional
```

CA Notes: Signing Certs

- Signing certificates
  - Put the CSR in certreqs/<name>.csr
  - *openssl ca -config ./openssl.cnf -infiles \ certreqs/<name>.csr*
  - Cert is in certsdb/<serial>.pem
- **Read** and **Verify** the information presented.
- Default extensions are in “usr_cert”, default policy is “policy_match”
- Override with “-extensions v3_ca” or “-policy policy_anything”

CA Notes: Revoking Certs

- To revoke a cert, first choose a reason:
  - unspecified
  - keyCompromise
  - CACompromise
  - affiliationChanged
  - superseded
  - cessationOfOperation
  - certificateHold
- Find the cert in certsdb and revoke
  - *openssl ca -config openssl.cnf -crl_reason \ superseded -revoke certsdb/<serial>.pem*
- Don't forget to generate CRL!

CA Notes: Generating CRLs

- Easy as cake:
  - *openssl ca -config openssl.cnf -gencrl -out \ crl.pem*
- Get it somewhere where people can find it. i.e. crlDP

Modifying Your CA

- Modifying your CA **is** possible
- Sometimes called 'reconstituting' the CA
- Usually done for a new crlDP or signingPolicy

- Simply re-(self-)signing the CA Certificate
- Old certs should still validate, but won't have new info
- **BE CAREFUL WITH THIS!**

X.509 Compared to PGP

- PGP
  - Web of trust
  - No central authority
  - **You** decide who you trust
  - Money is rarely involved
- X.509
  - Central Authority
  - You can only choose authorities:
    - But only sorta – you kinda **have** to trust the ones most people trust (Verisign, Thawte)
  - In some cases you can choose specific people/entites/certs – but the system isn't designed for it
  - Money is usually involved
  - Less work for users

Reading and References

- RFC 3280 – X.509 Certificates and CRLs
  ftp://ftp.rfc-editor.org/in-notes/rfc3280.txt
- RFC 4346 – TLSv1 Spec
  ftp://ftp.rfc-editor.org/in-notes/rfc4346.txt
- Netscape SSLv3 spec
  http://wp.netscape.com/eng/ssl3/3-SPEC.HTM
- RFC4158 – Certificate Path Building
  ftp://ftp.rfc-editor.org/in-notes/rfc4158.txt

- And of course...
  http://www.phildev.net/ssl/

The End!

- Thanks for your time
- Questions?