X.509 and SSL

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

UUASC
07/05/07

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

Let's start with the basics:

X.509 \neq \text{SSL}
\text{SSL} \neq \text{PKI}
\text{PKI} \neq \text{RSA}

and so on.
Introduction: Really, Really, Really Quick History

- Encryption
  - Simple encoding (ROT13!)
    - Easy to crack
  - Complex Symmetric Ciphers
    - How do you negotiate a key?
  - Public-Key (2-key) Cryptography

- PKI
  - Enabled new things:
    - signing
    - Method for symmetric key transport
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: Outline

- 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
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):
            a8:aa:38:ff:dd:df:21:59:3b:60:ab:1f:3d:f3:c8:
            f8:17:f1:17:67:44:75:5c:a7
      Exponent: 65537 (0x10001)

Attributes:
   a0:00

Signature Algorithm: md5WithRSAEncryption
   a5:39
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:
        ea:be:a9:c5:3e:78:f3:5e:30:1b:6c:98:16:60:ba:
        db:38:2f:5b:1f:85:c1:20:eb
      Exponent: 65537 (0x10001)
  Attributes:
    a0:00
  Signature Algorithm: sha1WithRSAEncryption
    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):
        ...
      Exponent: 65537 (0x10001)
  X509v3 extensions:
    X509v3 Basic Constraints:
      CA:FALSE
    X509v3 Subject Key Identifier:
    X509v3 Authority Key Identifier:
    X509v3 Subject Alternative Name:
      email:phil@ipom.com
    X509v3 CRL Distribution Points:
      URL:http://www.phildev.net/philnet.crl
  Signature Algorithm: sha1WithRSAEncryption
X.509: CRT Example 2

Certificate:
Data:
Version: 3 (0x2)
Serial Number:
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:
  X509v3 Subject Key Identifier:
  X509v3 CRL Distribution Points:
    URI:http://www.ticketmaster.com/crl/Ticketmaster%20Phil%20Test%20CA%20v1.crl
Signature Algorithm: sha1WithRSAEncryption
  ....
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
...
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:
  X509v3 Authority Key Identifier:
    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:
    URL:http://www.phildev.net/philnet.crl
X509v3 Subject Alternative Name:
  email:phil@ipom.com
Signature Algorithm: sha1WithRSAEncryption
  ...
Certificate:
Data:
  Version: 1 (0x0)
  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
  ...
X.509: CA Example 3

Certificate:
Data:
  Version: 3 (0x2)
  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:
  X509v3 Authority Key Identifier:
    DirName:/C=US/O=VeriSign, Inc./OU=Class 3 Public Primary Certification Authority
  Signature Algorithm: sha1WithRSAEncryption
...
**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)
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
  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 anyway
End-User Notes

• OpenSSL is divided 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):
          0b:cd:f1:8a:00:dc:a3:f1:e2:bb:8d:44:75:bf:a8:
      Exponent: 65537 (0x10001)
    Attributes:
      a0:00
    Signature Algorithm: sha1WithRSAEncryption
      d0:71
CA Notes: Let's have a look

$ openssl x509 -noout -text -in cacert.pem

Certificate:
Data:
  Version: 3 (0x2)
  Serial Number:
  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:
  X509v3 Authority Key Identifier:
    DirName:/C=US/ST=California/L=Hawthorne/O=PhilNet/CN=test/emailAddress=phil@ipom.com
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 CSR 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
[ 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/entities/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
- RFC 4346 – TLSv1 Spec
- Netscape SSLv3 spec
  http://wp.netscape.com/eng/ssl3/3-SPEC.HTM
- RFC4158 – Certificate Path Building

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

- Thanks for your time
- Questions?