
A NEW PLATFORM FOR A NEW ERA

# Apache Tomcat and SSL

Mark Thomas, Staff Engineer, Pivotal

9 April 2014

Pivotal

# Agenda

- Introductions
- Cryptography Basics
- SSL
- Configuring Tomcat for SSL
  - Java connectors (BIO, NIO)
  - APR/native connector
  - Reverse proxy
- Questions

Pivotal

# Introductions

# Introductions

- markt@apache.org
  - Apache Tomcat committer since December 2003
  - Apache Tomcat PMC member from the beginning
- Tomcat 8 release manager
- Member of the Apache Tomcat security team
- Apache Commons PMC member
- Member of the Apache Infrastructure team

Pivotal

# Introductions

- Staff Engineer at Pivotal
- Primary role is to work on Apache Tomcat
- Pivotal tc Server
  - Based on Tomcat
  - Keep tc Server updated as new Tomcat versions are released
- 3rd line support for Tomcat and tc Server
  - tomcat@gopivotal.com
- Lead the Pivotal security team

Pivotal

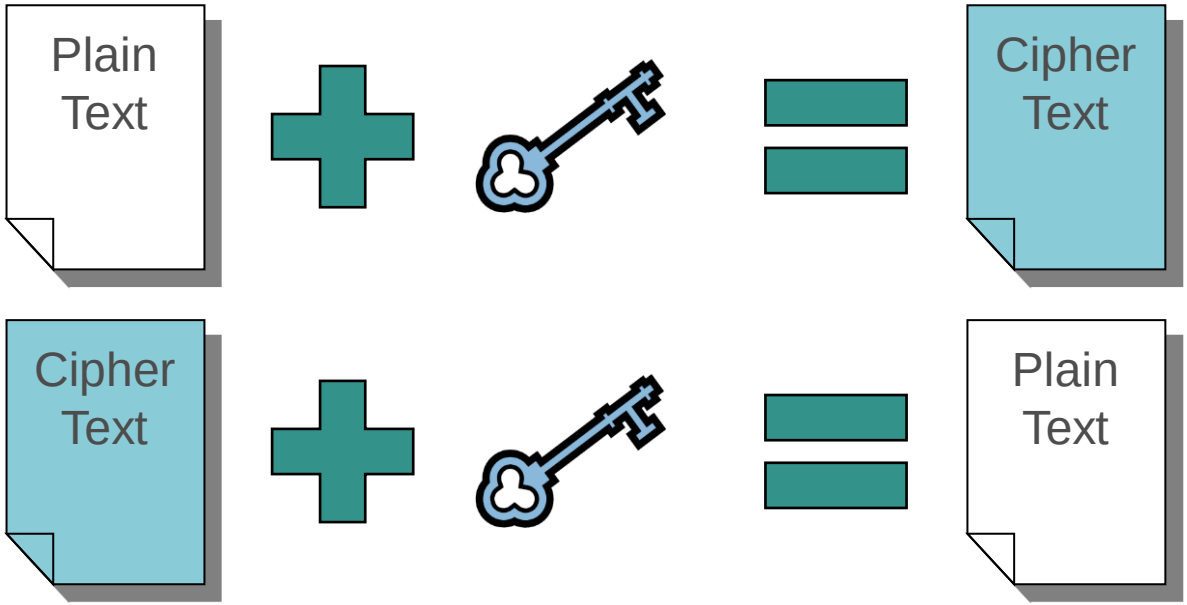
# Why This Presentation?

- Lots of questions about SSL on the Tomcat mailing lists
- It is clear from the questions many folks don’t understand how SSL works
- Debugging something you don’t understand is much harder than debugging something you do understand

# Cryptography Basics

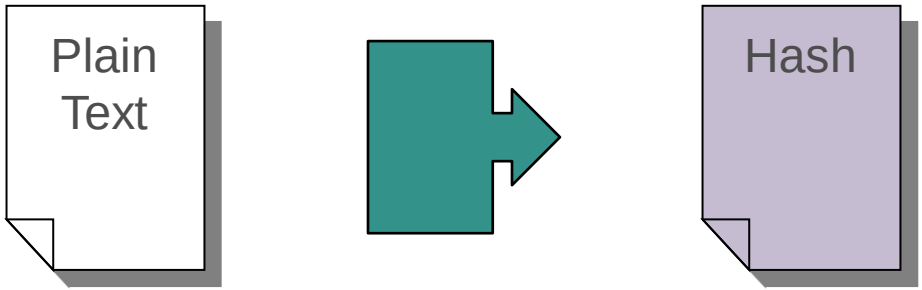
# Cryptography Basics: Symmetric Encryption

- Use the same key to encrypt and decrypt



# Cryptography Basics: Hash Functions

- Generate a fingerprint (hash) for the given input
- A small change in the input results in a large change in the hash
- Very difficult to generate an input for a given hash




Pivotal

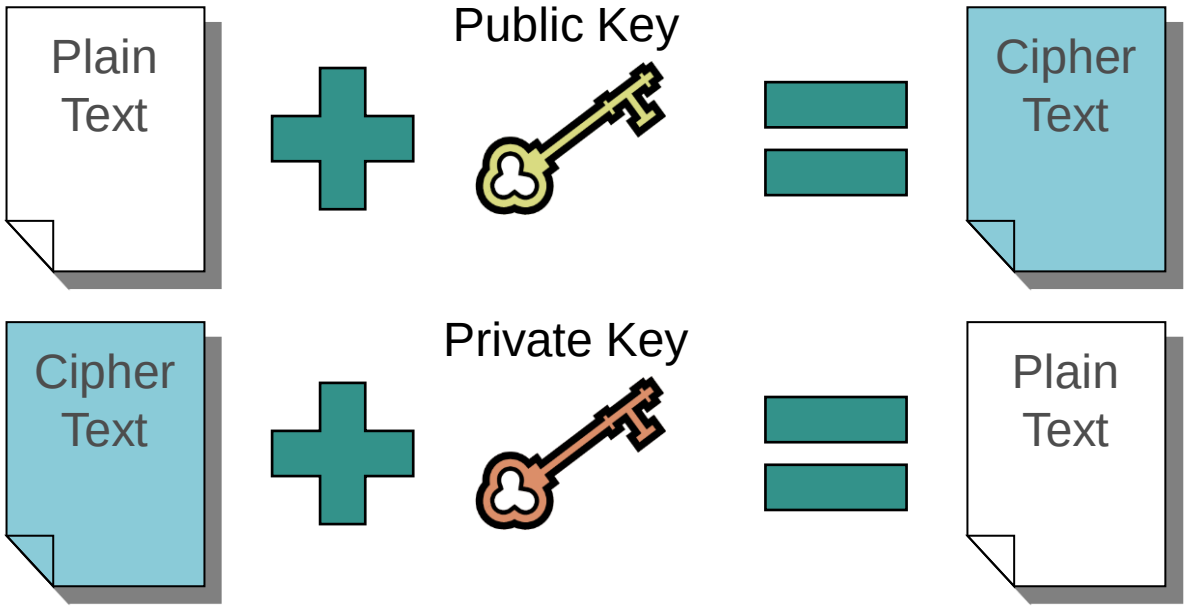
# Cryptography Basics: Asymmetric Encryption

- Pair of keys, A and B
  - If key A is used to encrypt, key B must be used to decrypt
  - If key B is used to encrypt, key A must be used to decrypt
- Very difficult to determine one key from the other
- One key is used as the “Public Key”
  - This key is made widely available to the general public
- One key is used as the “Private Key”
  - This key must be protected

Pivotal

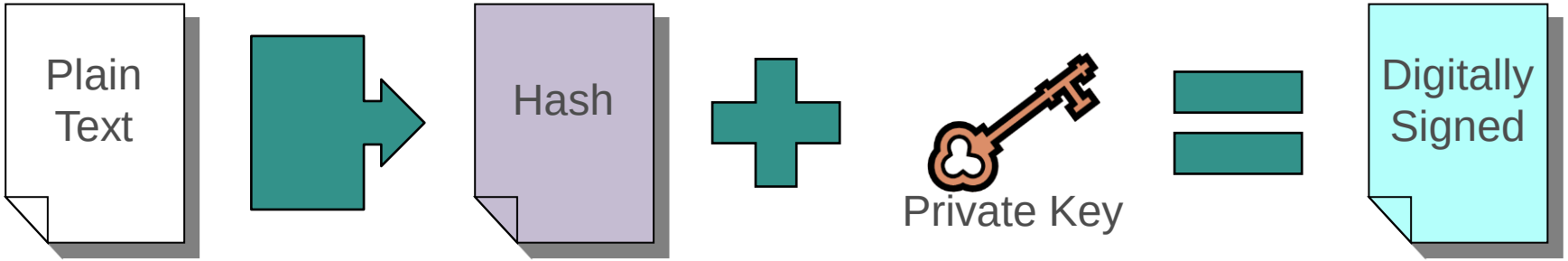
# Cryptography Basics: Asymmetric Encryption

- Use different keys to encrypt and decrypt



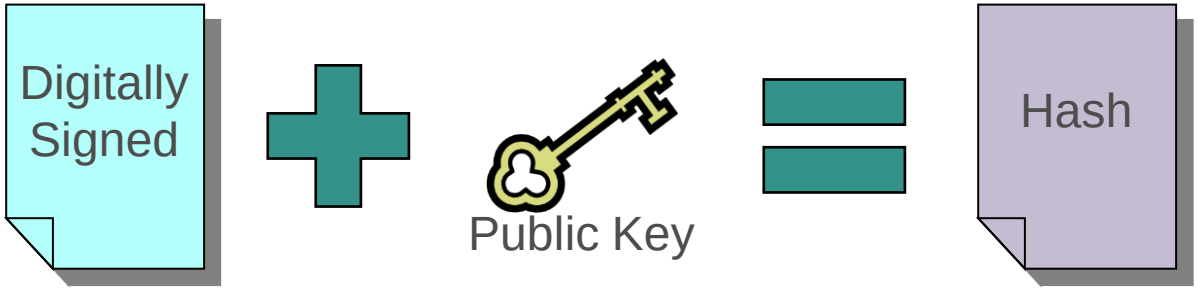
# Cryptography Basics: Digital Signatures

- Proves that a document was sent by a particular entity



# Cryptography Basics: Digital Signatures

- Validating a digital signature

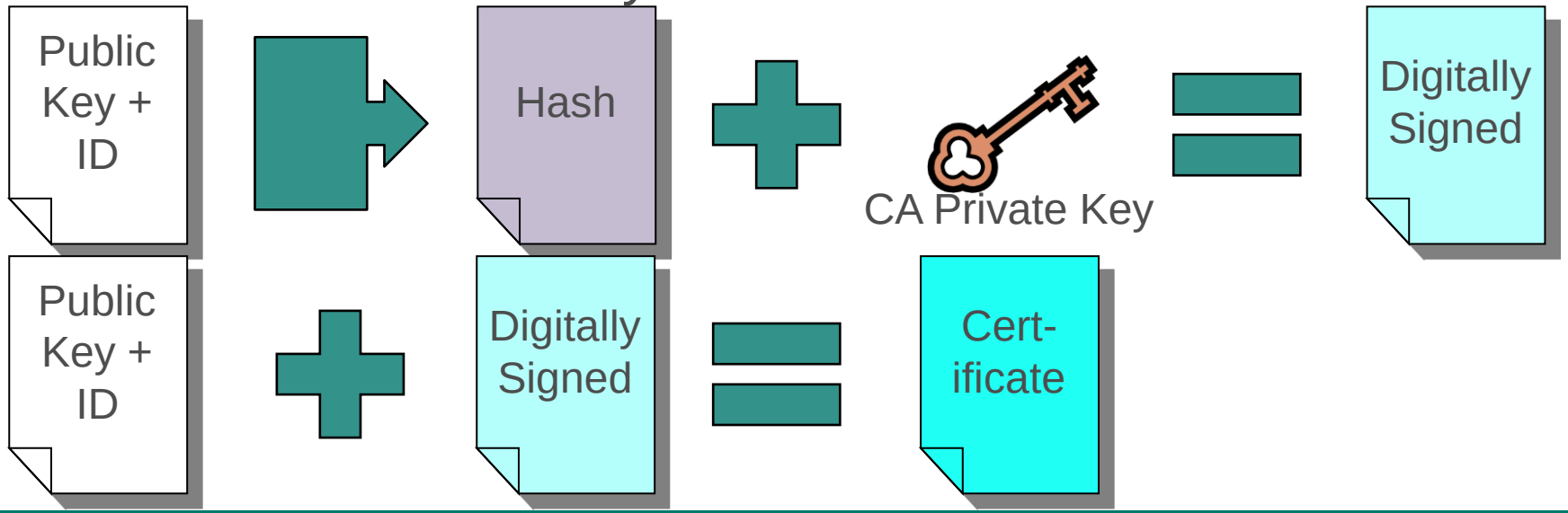


# Cryptography Basics: Digital Signatures

- If the hashes match then:
  - The public key decrypted the digital signature
  - Therefore, the associated private key must have created the digital signature
  - Therefore, the recipient can be certain that the owner of the public/private key pair sent the document
- Determining who is the owner of the public/private key pair is the next problem

# Cryptography Basics: Certificates

- Certificates link a public key with an identity
- Certificates are issued by certificate authorities



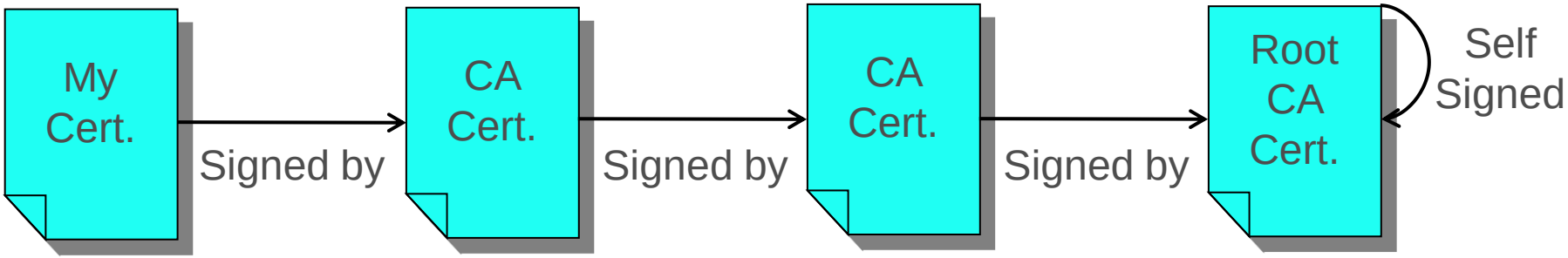
# Cryptography Basics: Certificates

- To validate the certificate authority's signature, you need to be able to link their public key to their identity
- You do this with a certificate
- This builds a trust chain
- At the top of the chain is the root certificate from the root certificate authority
- There are multiple root certificate authorities

Pivotal

# Cryptography Basics: Root Certificates

- Root certificates are self-signed
- Some other mechanism is required to trust root certificates
  - Usually installed by the operating system
  - You can manually validate them by checking them against the published versions on the CA's web site




Pivotal

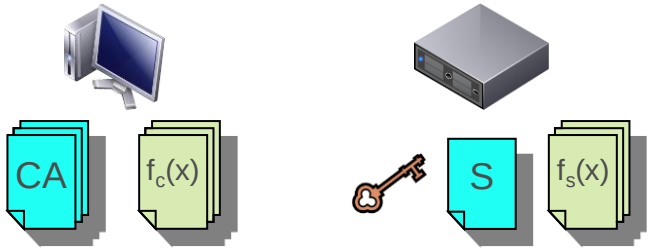
# SSL

# SSL

- SSL connections are initiated by a handshake
- Handshake
  - Mandatory steps
  - Optional steps
- This presentation considers the common case

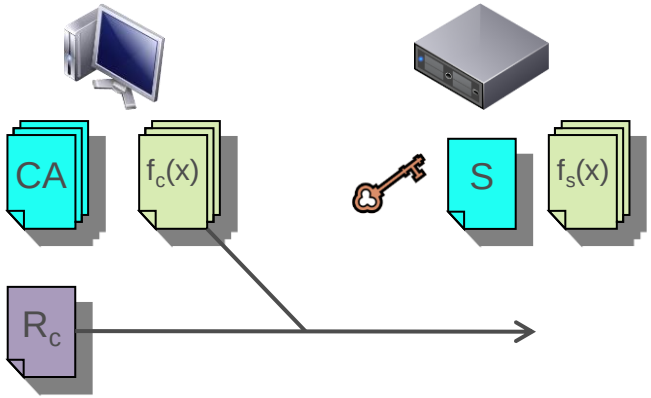
# SSL: Handshake Starting Point

- Server
  - Private key
  - Certificate
    - Public key
    - Identity (domain name)
  - List of supported algorithms
- Client
  - List of trusted (Root) Certificate Authorities
  - List of supported algorithms



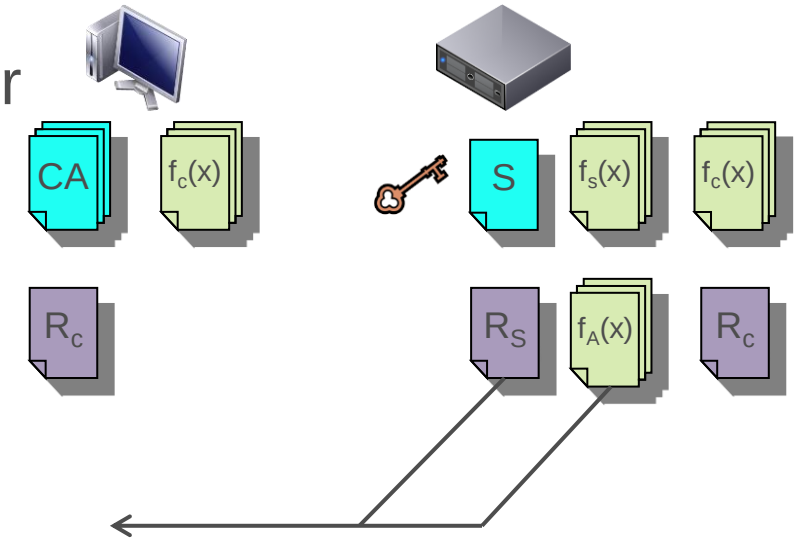
# SSL: Handshake Step 1: ClientHello

- Client generates a random number
- Client sends message to server
  - Client's random number
  - List of supported algorithms



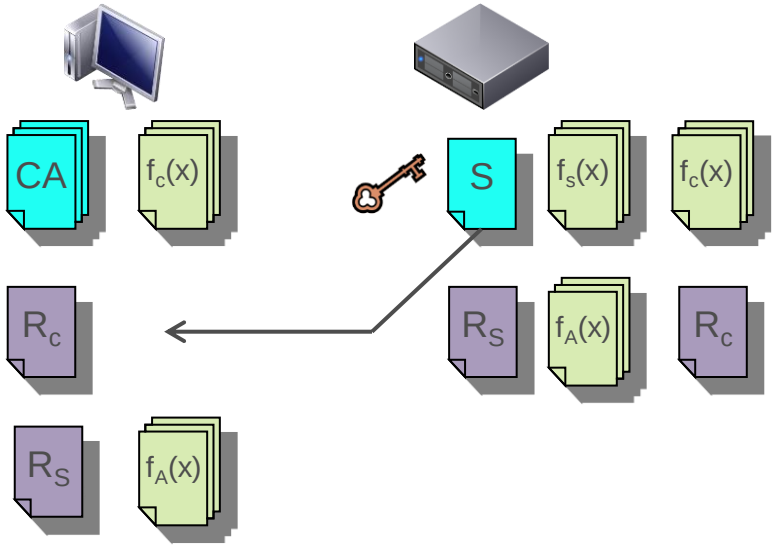
# SSL: Handshake Step 2: ServerHello

- Server generates a random number
- Server compares algorithms
  - Selects appropriate algorithms
- Server sends message to client
  - Server's random number
  - Selected algorithms



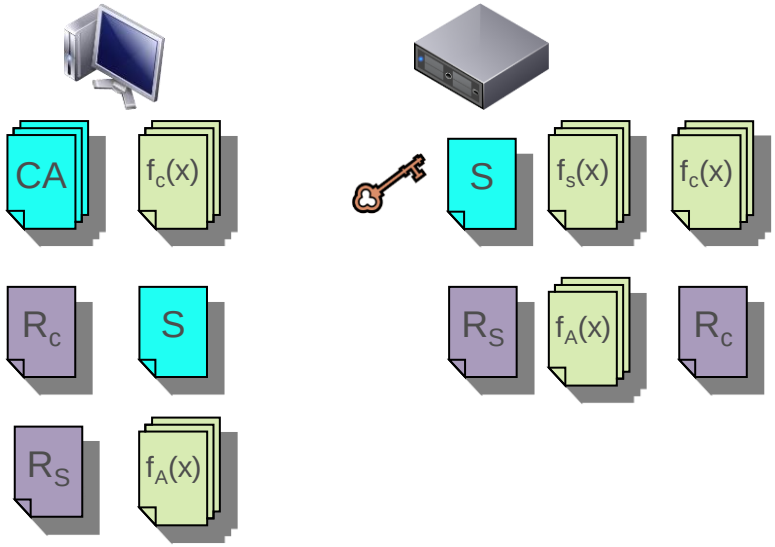
# SSL: Handshake Step 3: Certificate

- Server sends message to client
  - Server's certificate
- Client validates server certificate



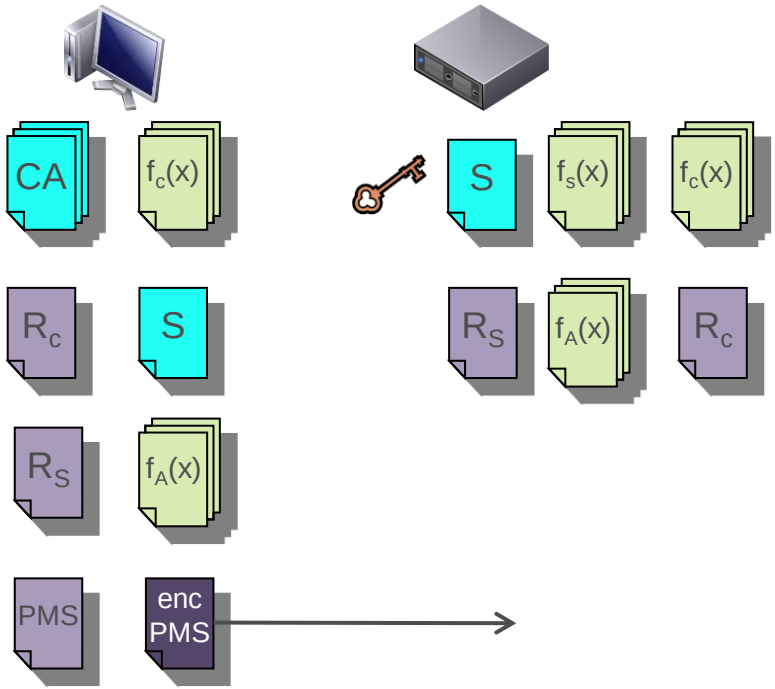
# SSL: Handshake Step 6: ServerHelloDone

- Server sends message to client
  - No content



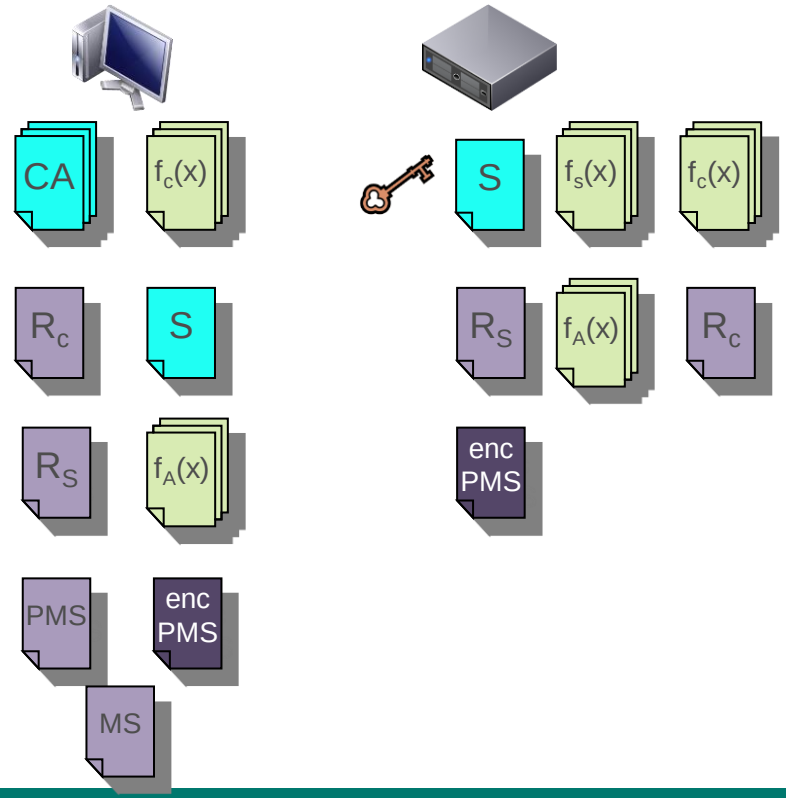
# SSL: Handshake Step 8: Client Key Exchange

- Client generates pre-master secret
- Client encrypts PMS with server's public key
- Client sends message to server
  - Encrypted PMS



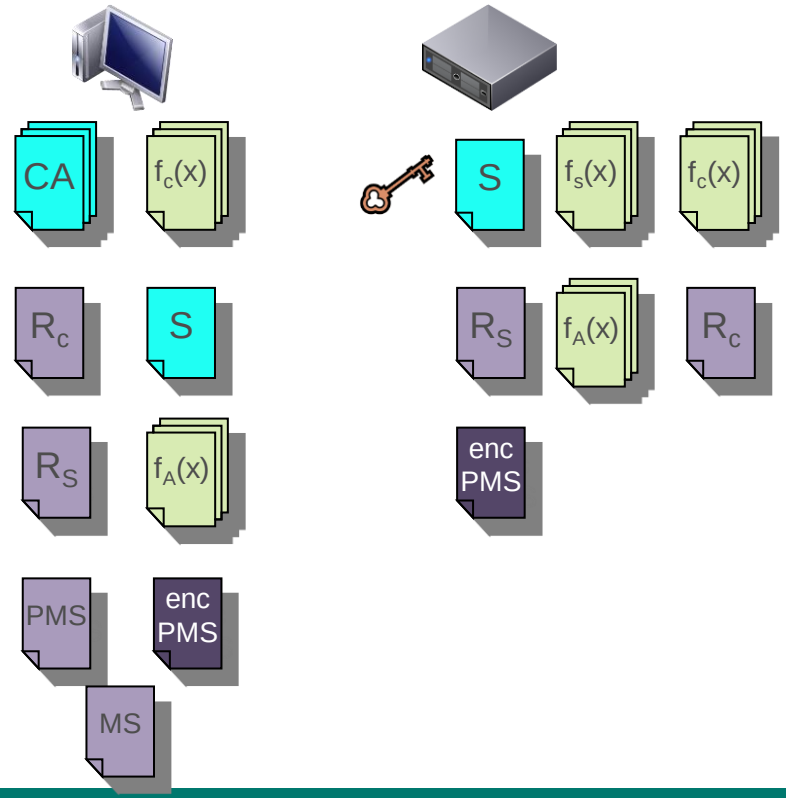
# SSL: Handshake Step 10: ChangeCipherSpec

- Client creates master secret
  - $R_c + R_s + PMS$
- Client switches to encrypted mode
  - Algorithm agreed in step 2
  - Symmetric encryption with MS
- Client sends message to server
  - No content



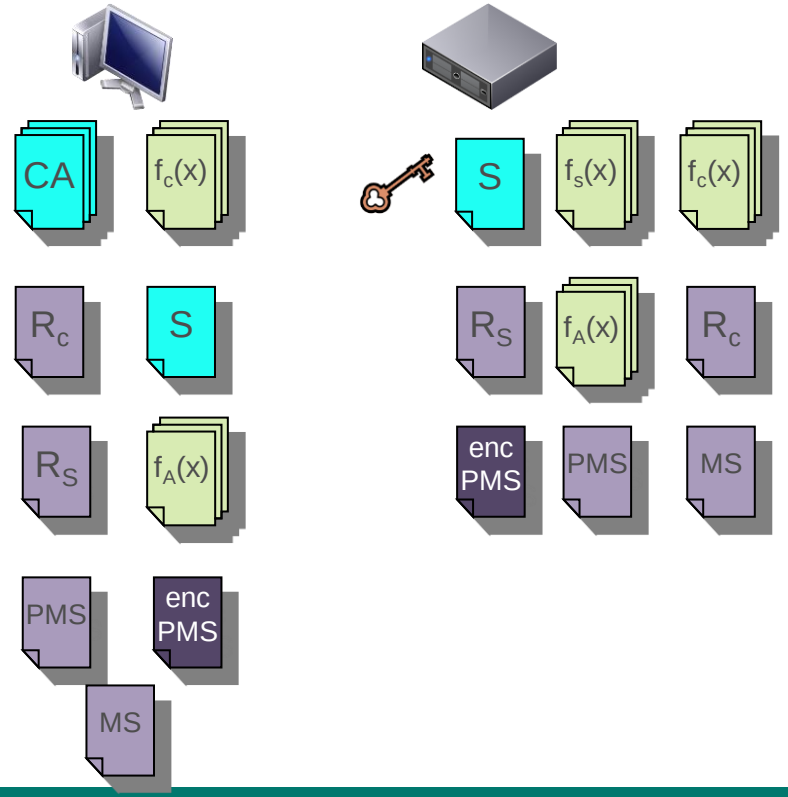
# SSL: Handshake Step 11: Finished

- Client has completed SSL handshake
- Client sends message to server
  - No content



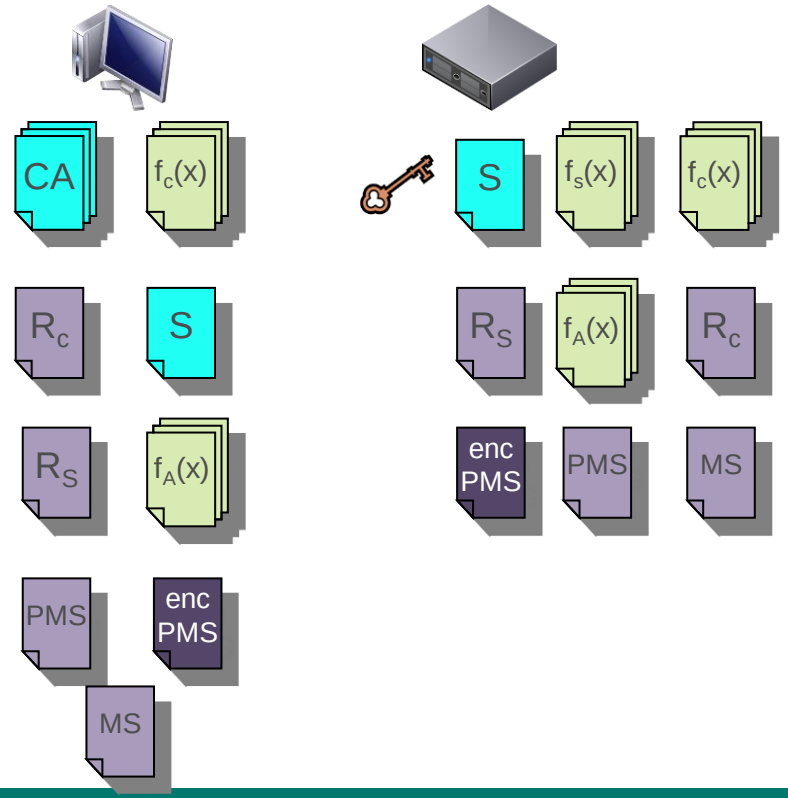
# SSL: Handshake Step 12: ChangeCipherSpec

- Server decrypts PMS
- Server creates master secret
  - $R_c + R_s + PMS$
  - Server switches to encrypted mode
  - Algorithm agreed in step 2
  - Symmetric encryption with MS
- Server sends message to client
  - No content



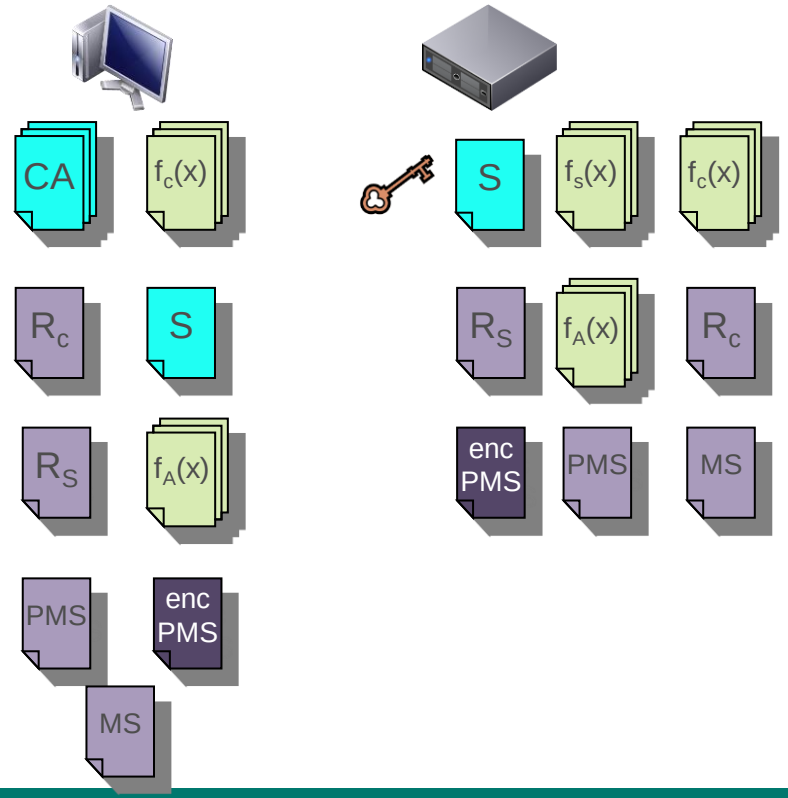
# SSL: Handshake Step 13: Finished

- Server has completed SSL handshake
- Server sends message to client
  - No content



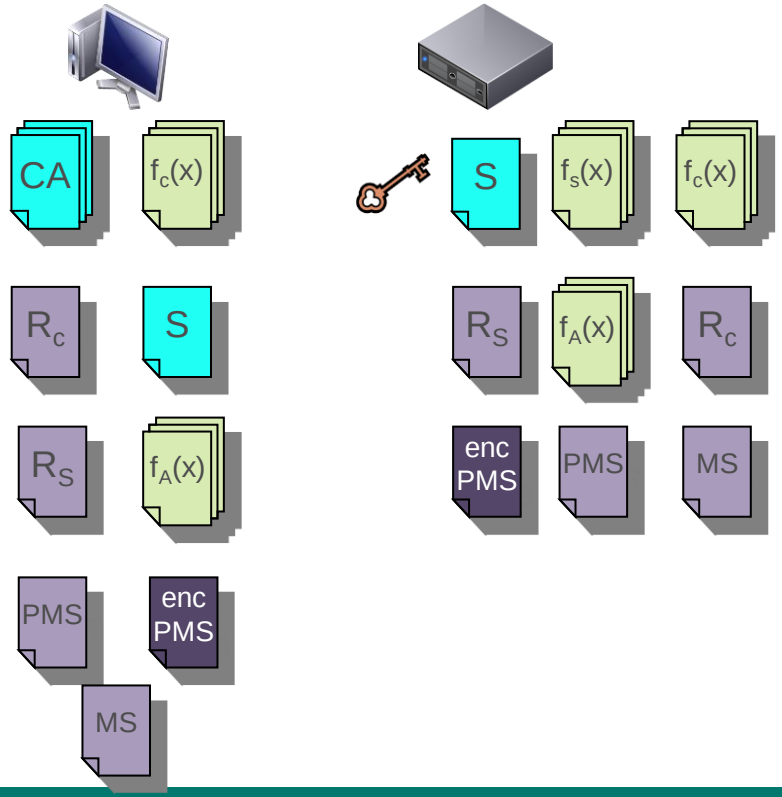
# SSL: Encrypted communication

- Algorithm agreed in step 2
- Symmetric
- Use Master Secret as key



# SSL: Extensons

- Client certificate authentication
  - Client authenticates to server with certificate
- Server Name Indication
  - Client tells server which host it wants to connect to and server sends appropriate certificate (virtual hosting)
- Renegotiation



# SSL Config for Tomcat

# Requirements

- A public/private key-pair
- A certificate
  - Public key
  - Identity (domain name e.g. www.apache.org)
- A Certificate Authority (CA) to generate the certificate
- The certificates for each CA in the trust chain
  - Root CA plus any intermediate CAs

# Formats

- Java keystore
  - Keys and certificates
  - Only used by Java
  - Generally easier insert than extract information
  - OpenSSL does not understand this format
- PKCS #12
  - Keys and certificates
  - OpenSSL does understand this format

# Formats

- DER
  - Certificates
  - Binary encoding
  - OpenSSL does understand this format
- PEM
  - Certificates
  - ASCII encoding
  - OpenSSL does understand this format

Pivotal

# Tools

- Apache Tomcat 8.0.x
  - Latest source as at time of presentation
  - Works equally well with any 6.0.x, 7.0.x or 8.0.x release
- OpenSSL 1.0.1f
  - OSX
  - Works on other platforms – adjust paths as necessary

# Configuration

- Initial set up

```
$ cd
$ mkdir demo
$ cd demo
$ mkdir certs newcerts private requests
$ echo 1000 > serial
$ touch index.txt
$ cp /opt/local/etc/openssl/openssl.cnf .
```

# Configuration

- Modify copy of openssl.cnf

```
$ vi openssl.cnf


dir = .

default_bits = 2048

countryName_default = US
```

# Configuration

- Create your own root certificate authority

```
$ openssl req -new -x509 -days 3650 -extensions v3_ca \
          -keyout private/cakey.pem -out cacert.pem \
          -config ./openssl.cnf
```

# Configuration

- Create and sign host certificate request

```
$ openssl req -new -nodes \
          -out requests/localhost-req.pem \
          -keyout private/localhost-key.pem \
          -config ./openssl.cnf


$ openssl ca -days 730 -config ./openssl.cnf \
          -out certs/localhost-cert.pem \
          -infiles requests/localhost-req.pem
```

Pivotal

# Configuration

- Convert the host key and certificate to PKCS #12

```
$ openssl pkcs12 -export -out private/localhost.p12 \
          -inkey private/localhost-key.pem \
          -in certs/localhost-cert.pem \
          -certfile cacert.pem
```

# Configuration

- Configure Tomcat for SSL using the PKCS #12 file

```
<Connector port="8443"
    protocol="org.apache.coyote.http11.Http11NioProtocol"
    SSLEnabled="true" scheme="https" secure="true"
    clientAuth="false" sslProtocol="TLS"
    keystoreType="pkcs12"
    keystoreFile="${catalina.base}/conf/localhost.p12"
    keyPass="changeit"
    />
```

# Configuration

- Similarly using BIO

```
<Connector port="8443"
    protocol="org.apache.coyote.http11.Http11Protocol"
    maxThreads="150"
    SSLEnabled="true" scheme="https" secure="true"
    clientAuth="false" sslProtocol="TLS"
    keystoreType="pkcs12"
    keystoreFile="${catalina.base}/conf/localhost.p12"
    keyPass="changeit"
    />
```

Pivotal

# Configuration

- Configuration for APR/native is significantly different

```
<Connector port="8443"
    protocol="org.apache.coyote.http11.Http11AprProtocol"
    maxThreads="150"
    SSLEnabled="true" scheme="https" secure="true"
    clientAuth="false" sslProtocol="TLS"
    SSLCertificateFile="${catalina.base}/conf/localhost-cert.pem"
    SSLCertificateKeyFile="${catalina.base}/conf/localhost-key.pem"
    SSLCertificateChainFile="${catalina.base}/conf/cacert.pem"
    />
```

Pivotal

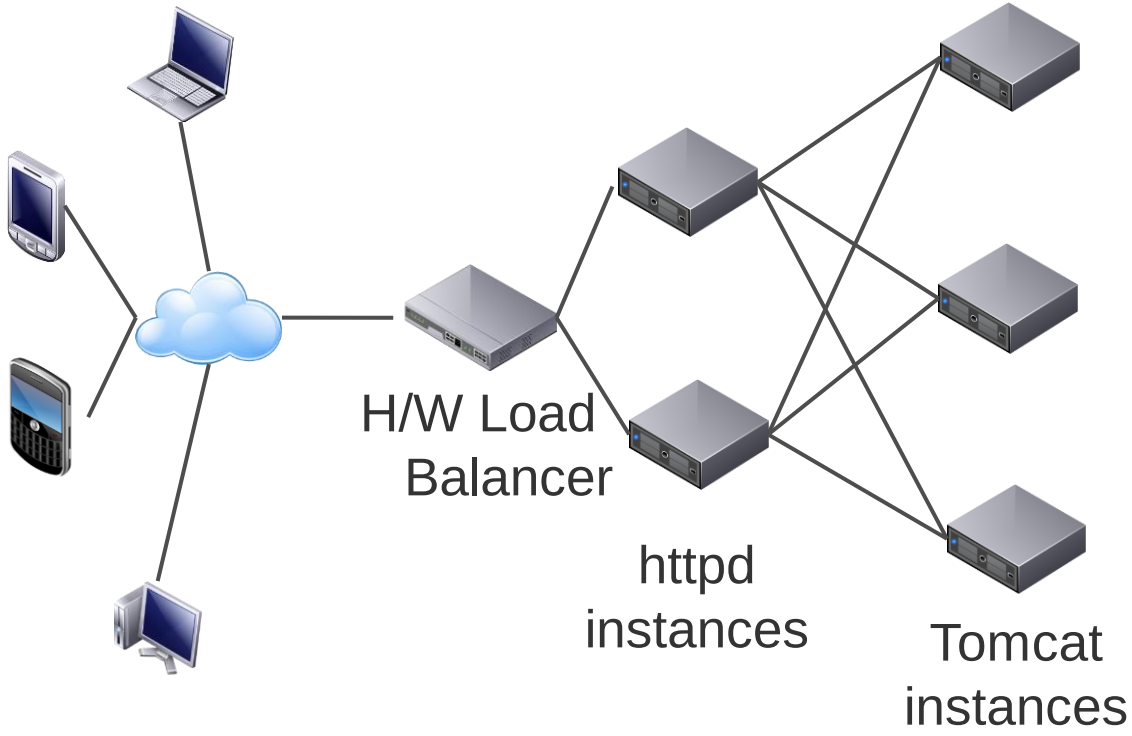
# Configuration

- There are other options
- Convert *.pem files to Java KeyStore
  - Historically painful
  - Better now but still requires you to create the *.p12 file
  - Since Tomcat can use the *.p12 file why bother with a keystore?
- Easy to move between separate *.pem files and a single .p12 file

Pivotal

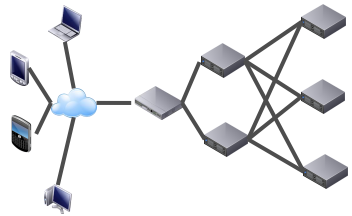
# SSL & Reverse Proxies

# What Is A Reverse Proxy?



# Design Considerations

- How will Tomcat differentiate between clients using http and https?
- Does the proxy <-> Tomcat traffic need to be encrypted?



Pivotal

# Why Does Tomcat Need SSL Information?

- To enforce transport guarantees specified in web.xml
- To determine if session was created over a secure connection
  - In which case session cookie needs to be marked as secure
- To correctly construct links, redirects etc. with http or https
- To obtain the identity of the authenticated user
  - When user client certificate authentication

Pivotal

# Protocol Choices

- AJP
  - Proxy implementations includes client <-> proxy SSL information automatically
  - Does not support encryption
- HTTP
  - Proxy implementations do not include client <-> proxy SSL information automatically
  - Supports encryption (proxy using https)

# Recommended Protocol

- If you do not need to encrypt proxy <-> Tomcat traffic
  - AJP
- If you do need to encrypt proxy <-> Tomcat traffic
  - HTTPS

- But if you use HTTPS, how do you get the SSL information?

# SSLValve

- In httpd:

```
<IfModule ssl_module>
  RequestHeader set SSL_CLIENT_CERT "%{SSL_CLIENT_CERT}s"
  RequestHeader set SSL_CIPHER "%{SSL_CIPHER}s"
  RequestHeader set SSL_SESSION_ID "%{SSL_SESSION_ID}s"
  RequestHeader set SSL_CIPHER_USEKEYSIZE "%{SSL_CIPHER_USEKEYSIZE}s"
</IfModule>
```

- In Tomcat:

```
<Host … >
  <Valve className="org.apache.catalina.valves.SSLValve"
…
</Host>
```

Pivotal

# An Alternative Solution

- Create two HTTP connectors in Tomcat
- Configure the first with
  - SSLEnabled="false" scheme="http" secure="false" proxyPort="80"
- Configure the second with
  - SSLEnabled="false" scheme="https" secure="true“ proxyPort="443"
- Proxy HTTP traffic to the first connector over HTTP
- Proxy HTTPS traffic to the second connector over HTTP

Pivotal

# Questions

Pivotal


A NEW PLATFORM FOR A NEW ERA