TomcatCon
Apache Tomcat and TLS

Mark Thomas

Introduction

Why This Presentation?

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

- I’ll use SSL and TLS interchangeably (as do the Tomcat docs)

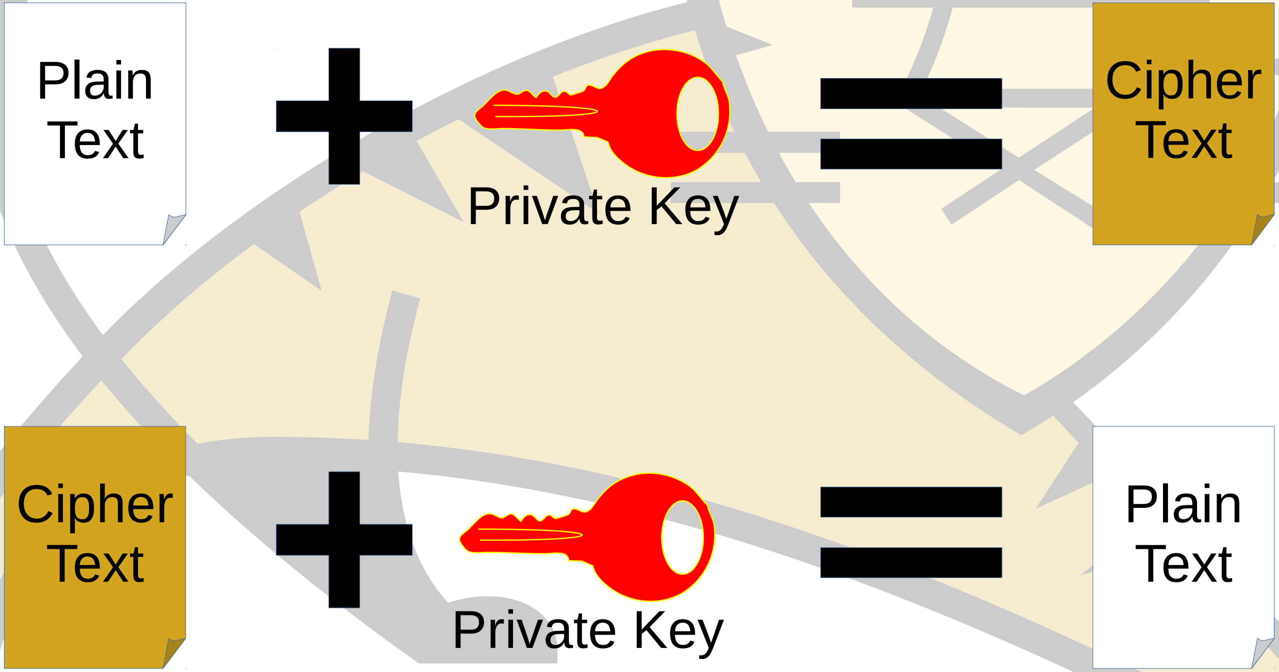
Agenda

- Cryptography basics
- TLS
- Configuring Tomcat for TLS
- Questions

Cryptography Basics

Cryptography Basics: Symmetric Encryption

- Use the same key to encrypt and decrypt



Plain Text + Private Key = Cipher Text

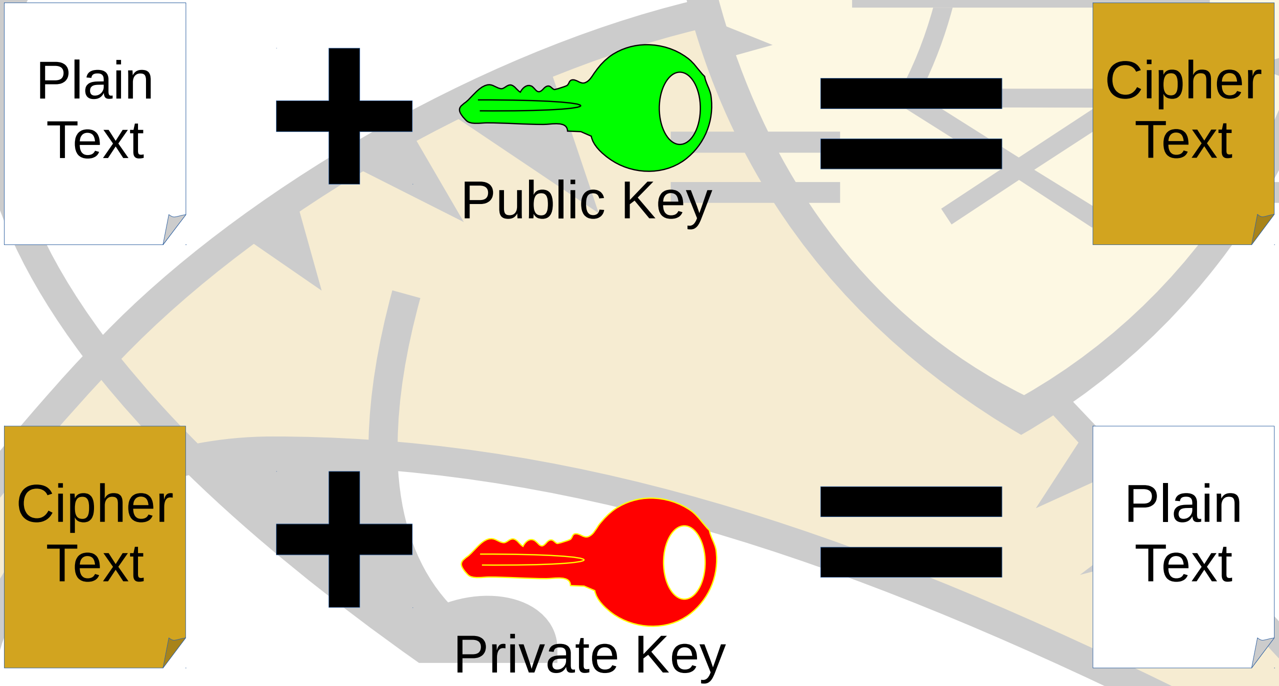
Cipher Text + Private Key = Plain Text

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

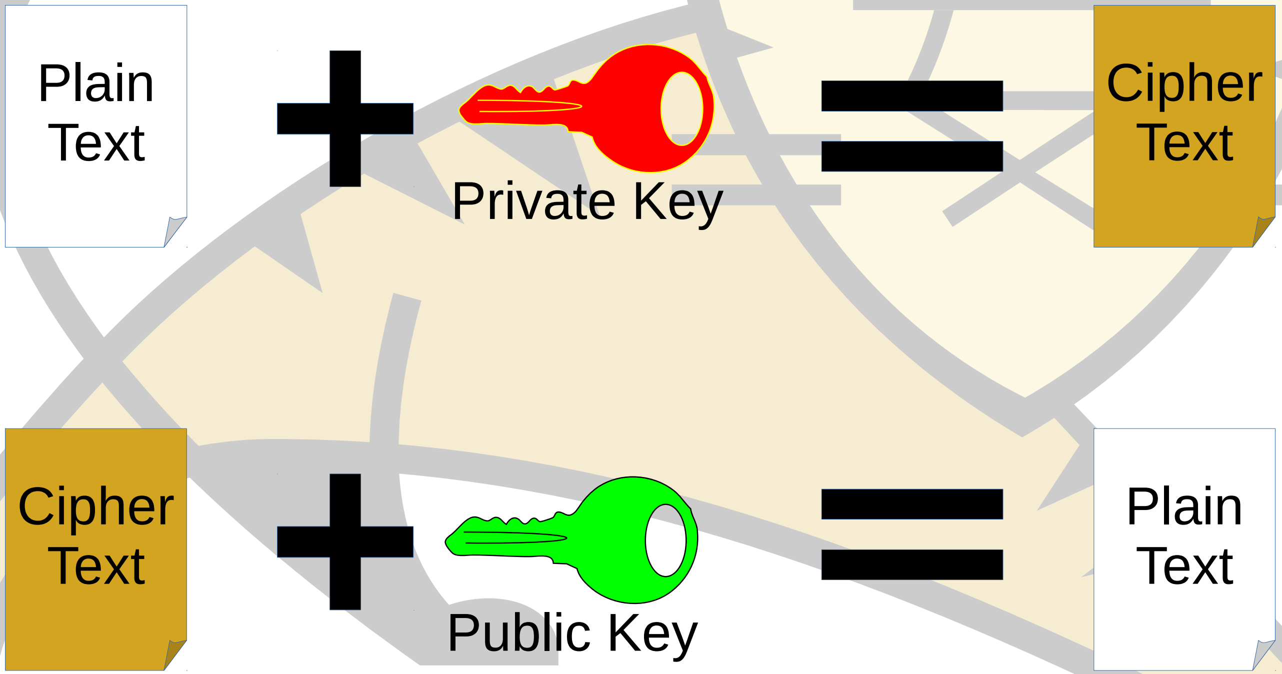
Cryptography Basics: Asymmetric Encryption

- Use different keys to encrypt and decrypt



Cryptography Basics: Asymmetric Encryption

- You can use the keys either way around

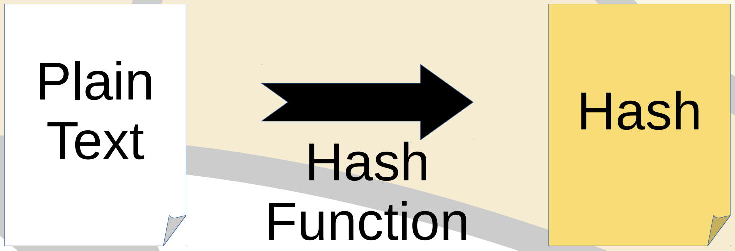


Plain Text + Private Key = Cipher Text

Cipher Text + Public Key = Plain Text

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



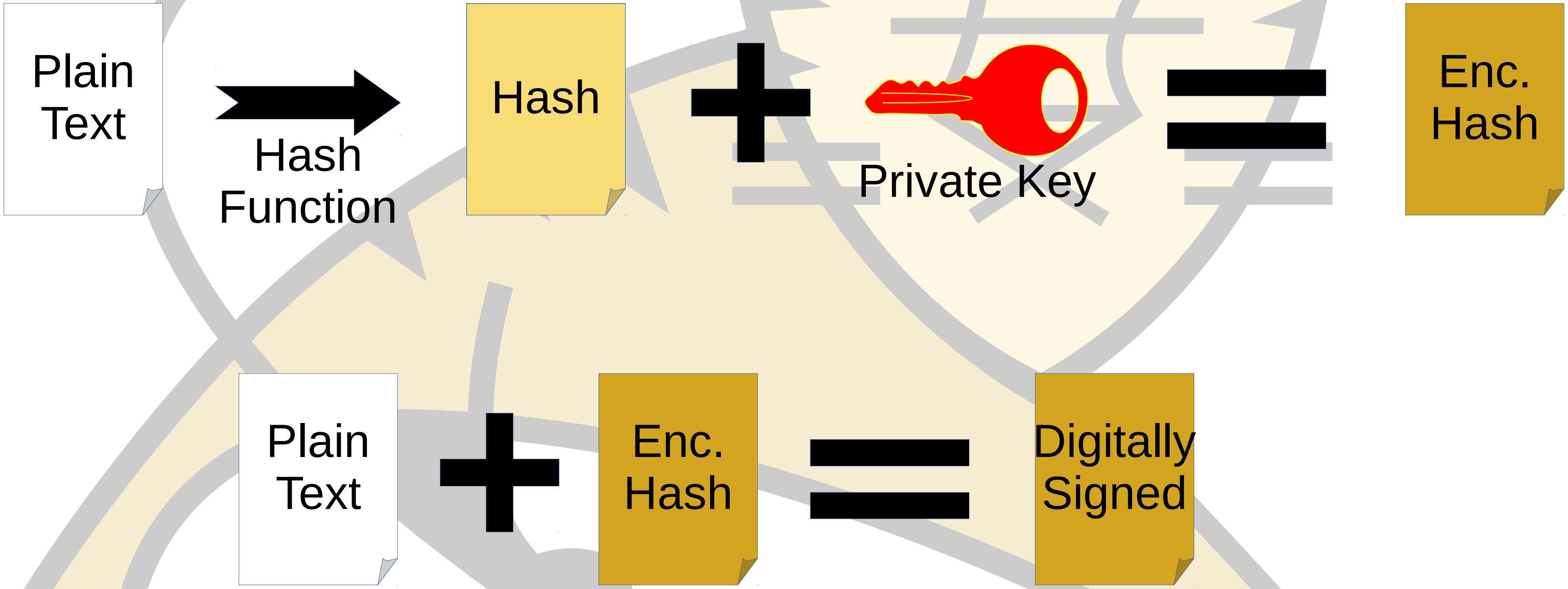
Plain Text

Hash Function

Hash

Cryptography Basics: Digital Signatures

- Proves a document was sent by a particular entity



Plain Text

Hash Function

Hash

Private Key

Enc. Hash

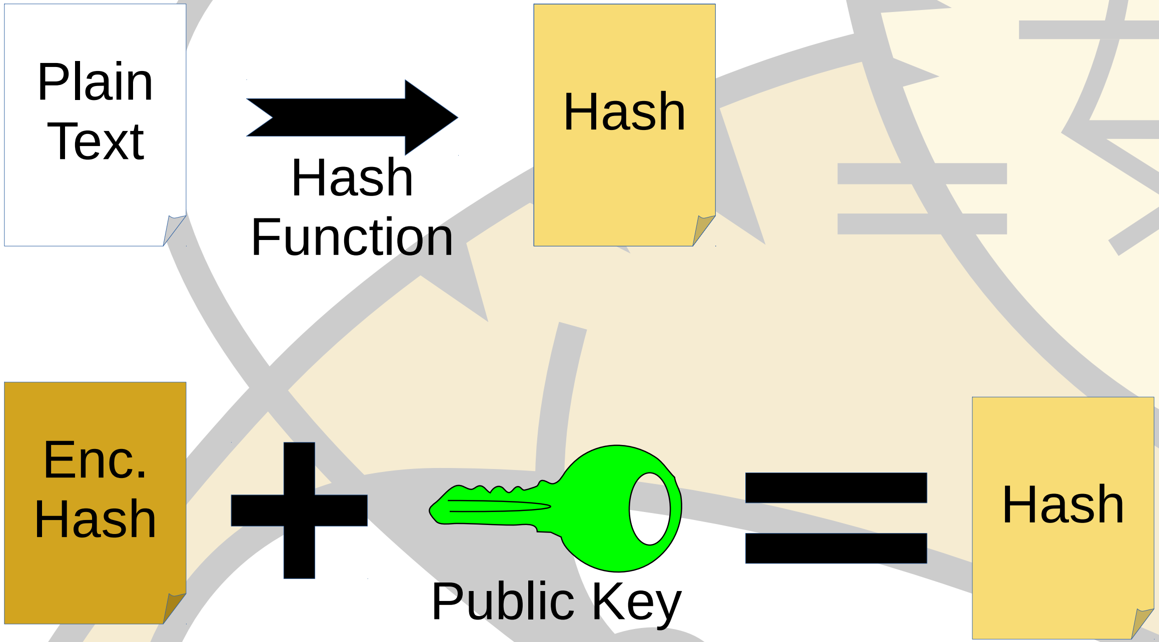
Plain Text

Enc. Hash

Digitally Signed

Cryptography Basics: Digital Signatures

- Validating a digital signature



Plain Text

Hash Function

Hash

Enc. Hash

Public Key

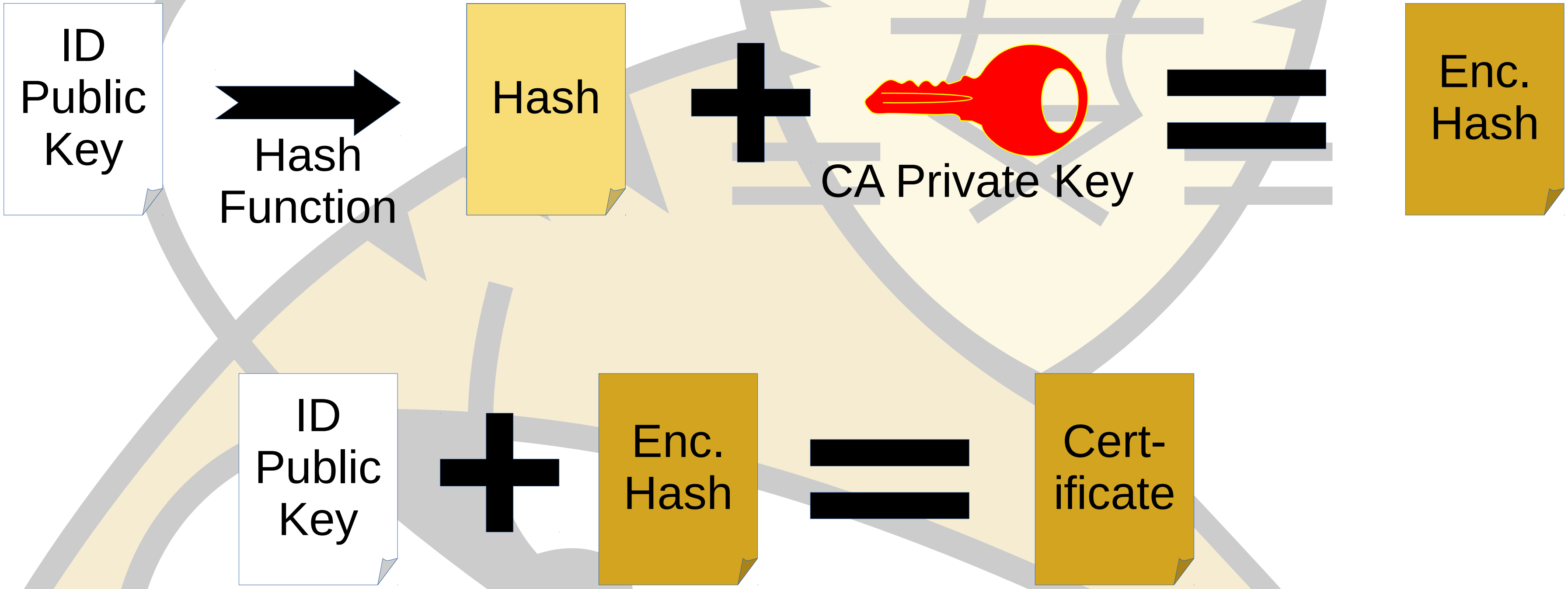
Hash

Cryptography Basics: Digital Signatures

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

Cryptography Basics: Certificates

- Proves a public key is associated with a given identity

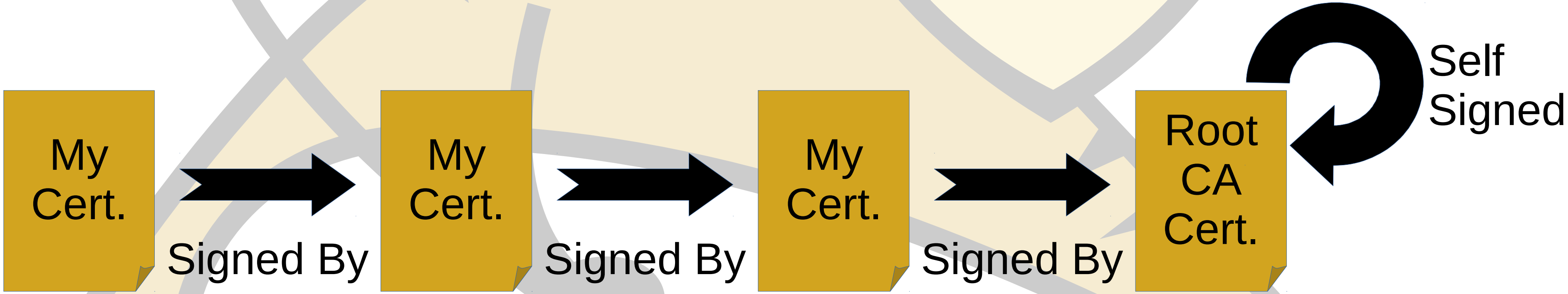


Cryptography Basics: Certificates

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

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

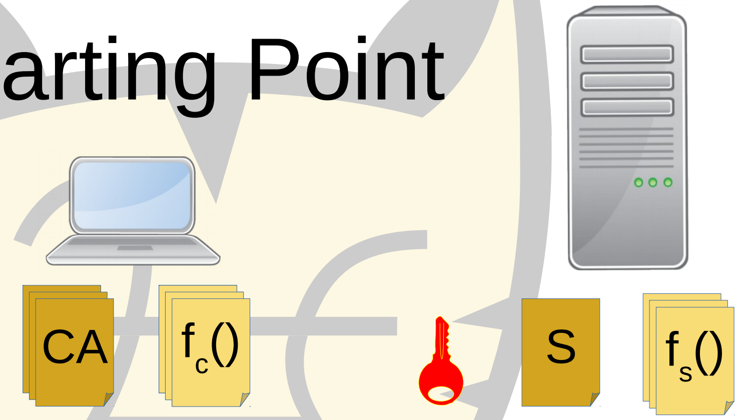


TLS

TLS

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

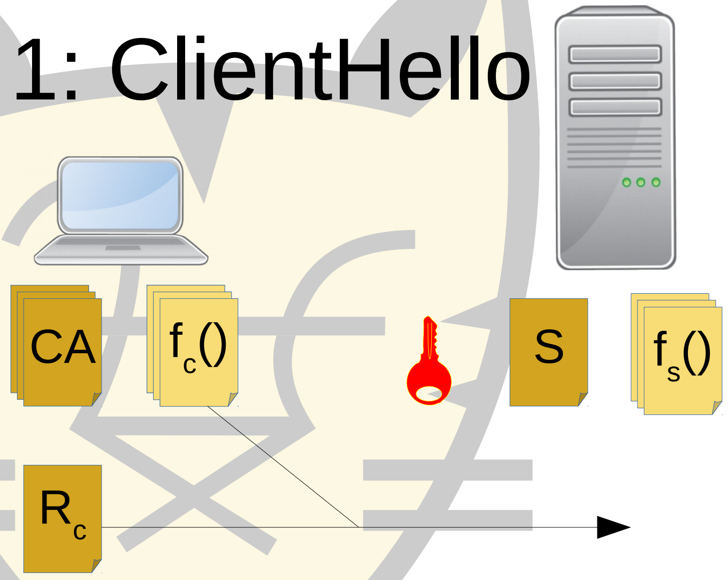
TLS: Handshake Starting Point



- Server
  - Private key
  - Certificate
    - Public Key
    - ID (domain name)
  - List of supported algorithms
- Client
  - List of trusted (Root) CAs
  - List of supported algorithms

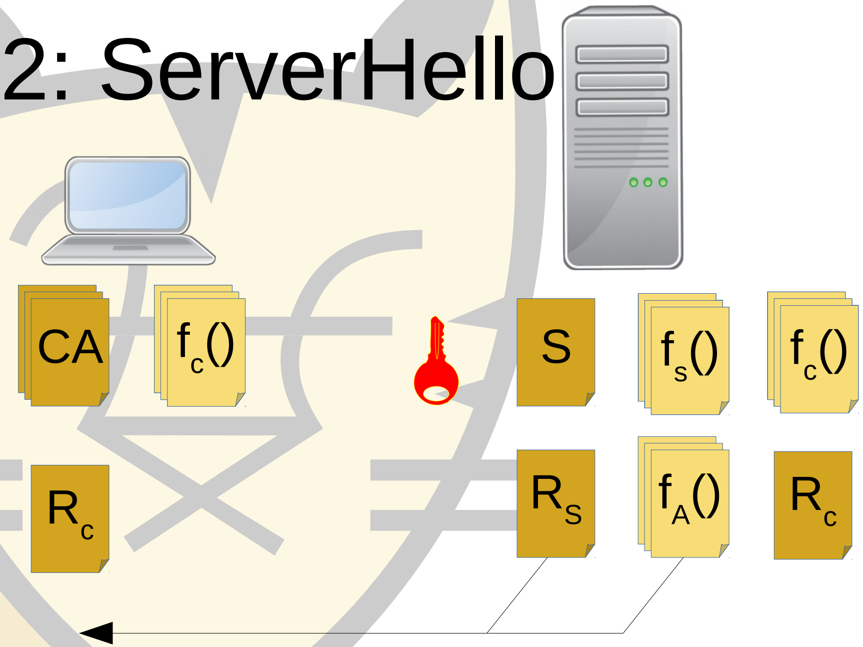
TLS: Handshake Step 1: ClientHello

- Client generates random number
- Client sends message to server
  - Client’s random number
  - Client’s supported algorithms



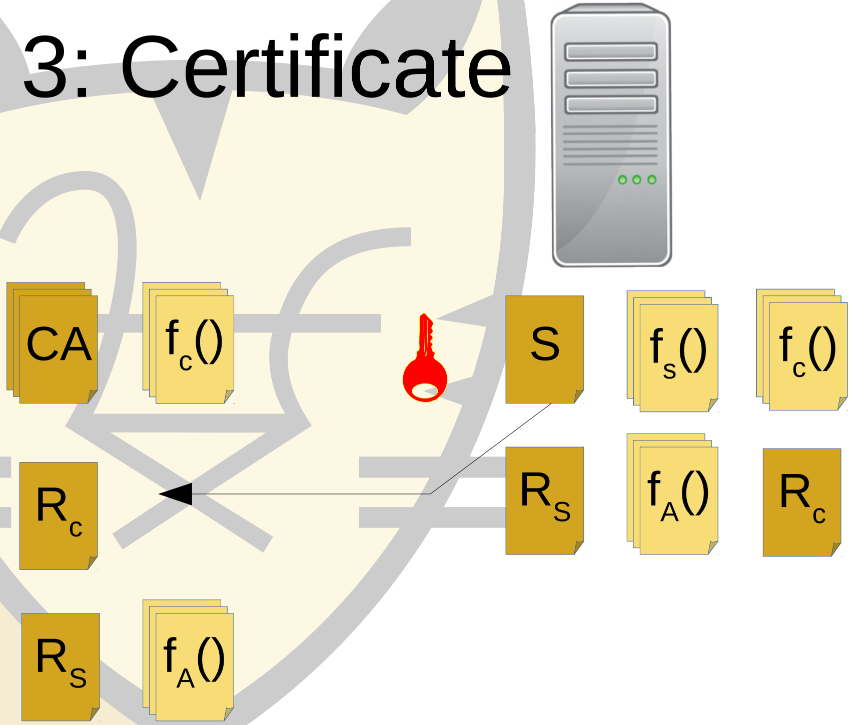
TLS: Handshake Step 2: ServerHello

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



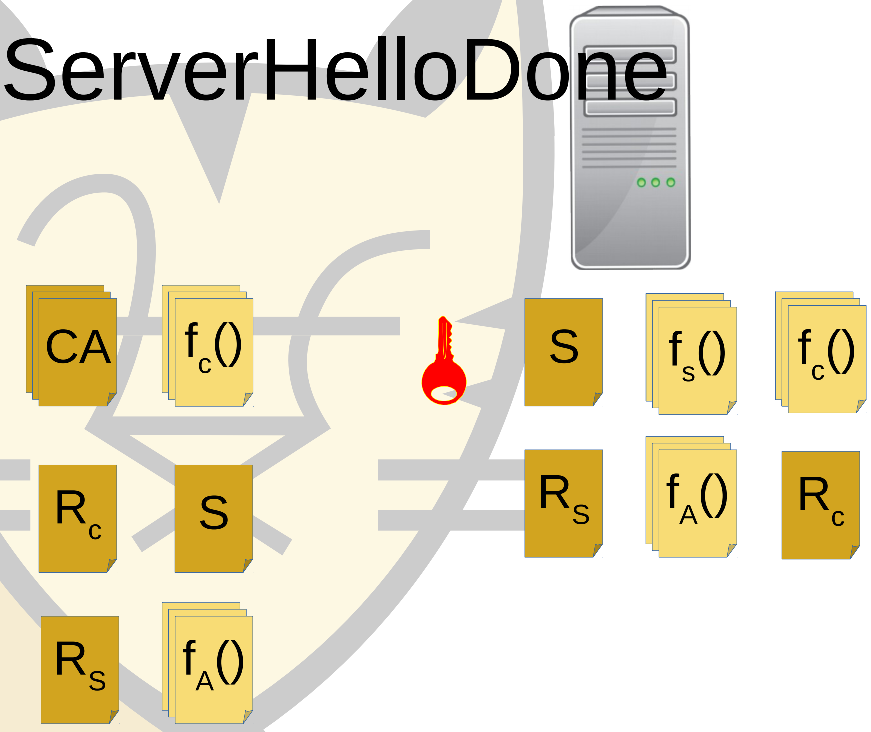
TLS: Handshake Step 3: Certificate

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



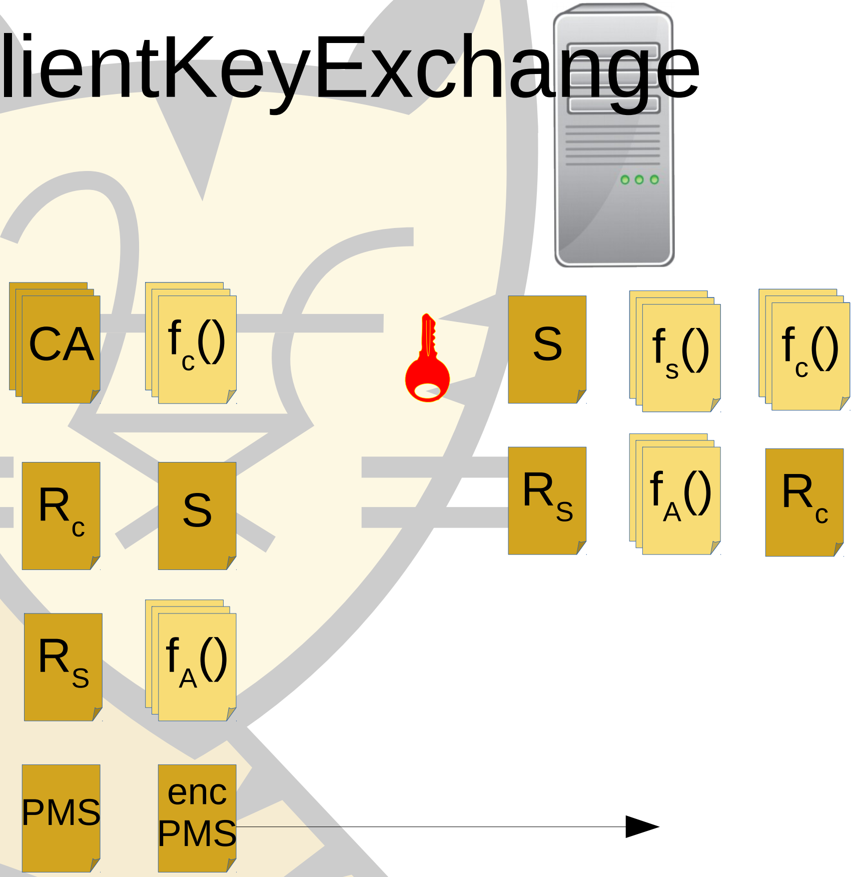
TLS: Handshake Step 6: ServerHelloDone

- Server sends message to client
  - No content



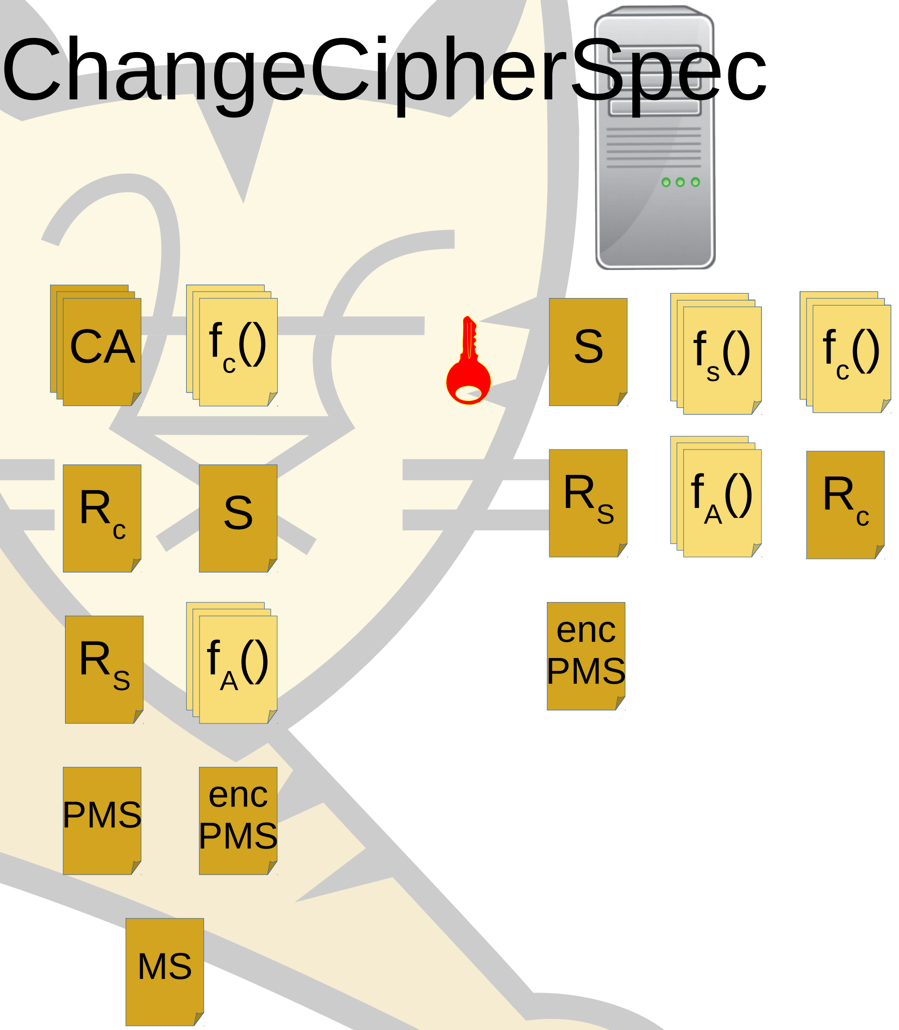
TLS: Handshake Step 8: ClientKeyExchange

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



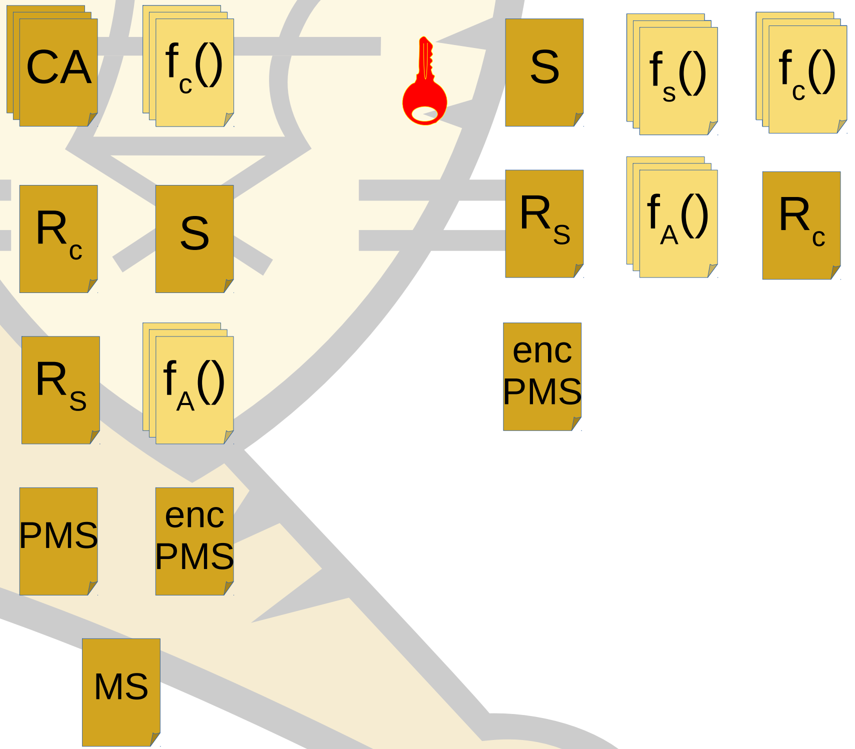
TLS: Handshake Step 10: ChangeCipherSpec

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



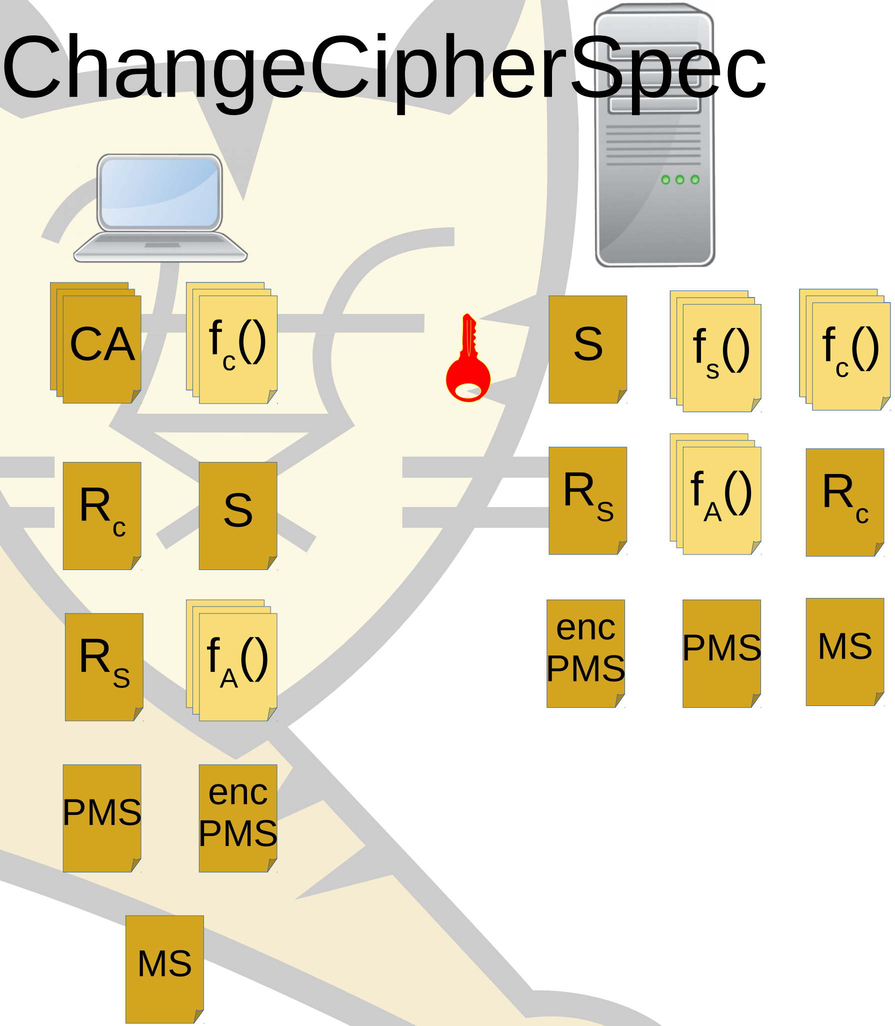
TLS: Handshake Step 11: Finished

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



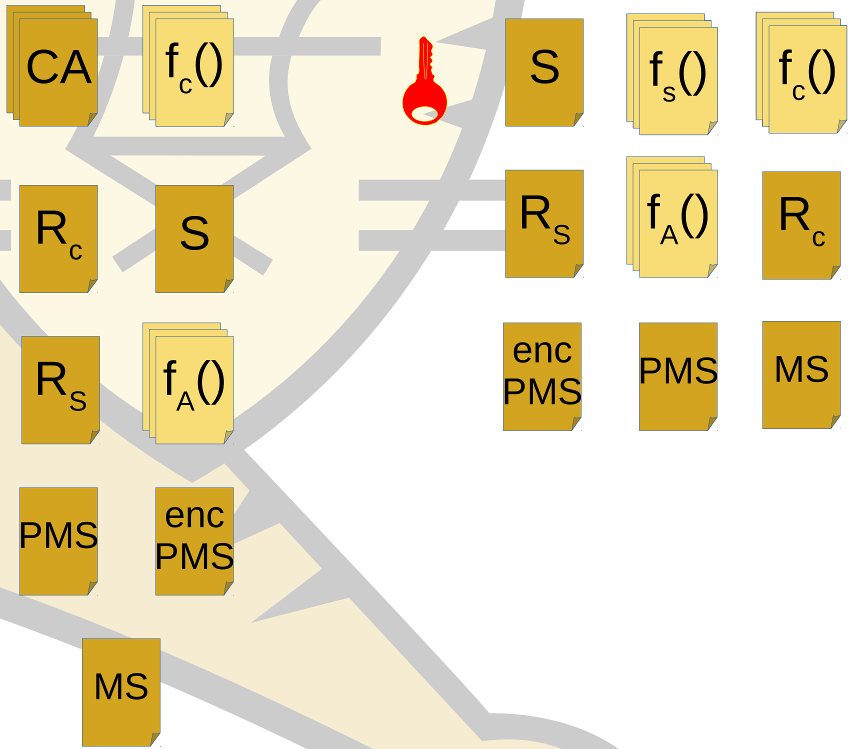
TLS: Handshake Step 12: ChangeCipherSpec

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



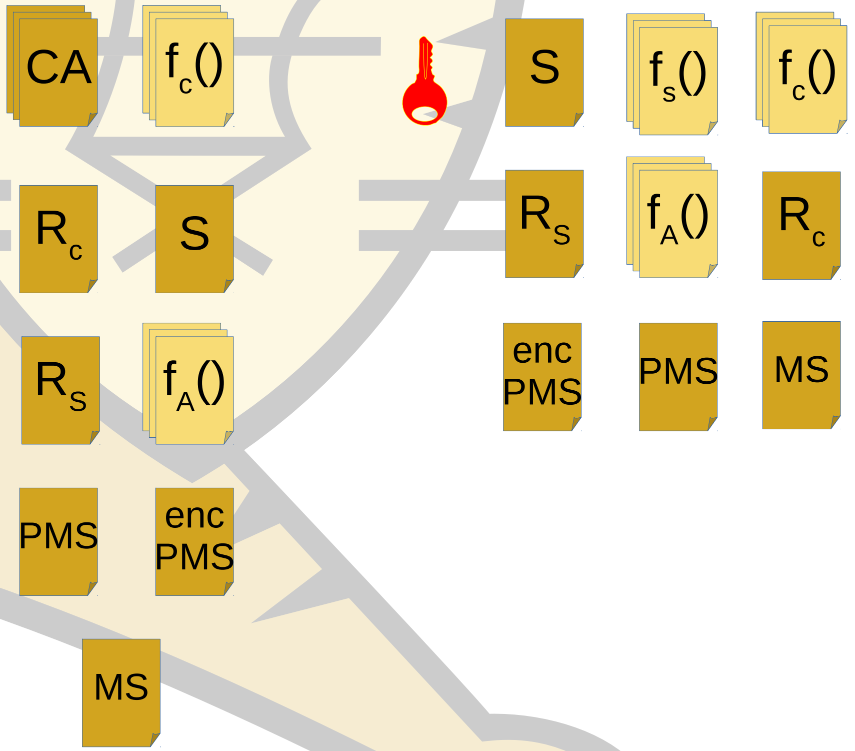
TLS: Handshake Step 13: Finished

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



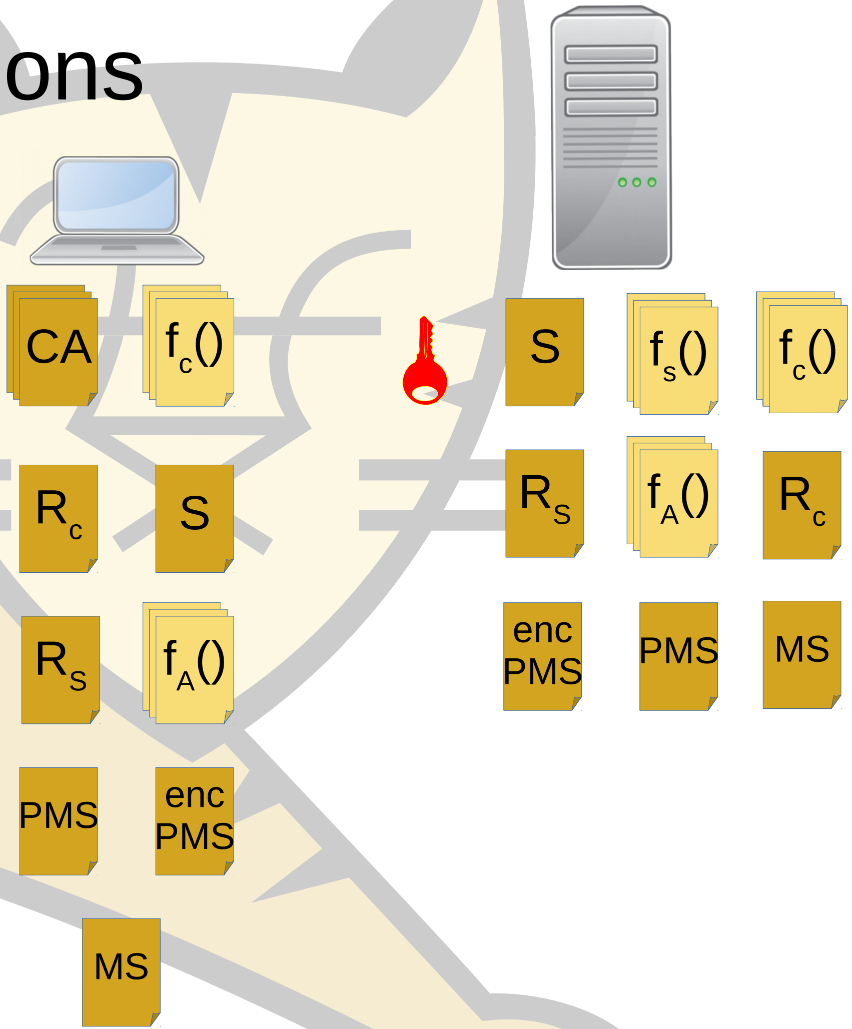
TLS: Encrypted Communication

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



TLS: Extensions

- Client certificate authentication
  - Client authenticates to server with a certificate
- Server Name Indication
  - Client tells server which host is wants to connect to and server sends appropriate certificate (virtual hosting)
- Application Layer Protocol Negotiation
  - Client and server agree protocol to for encrypted communication during handshake



Configuring Tomcat for TLS

Requirements

- Private key
- Server certificate
- Certificate chain

- Configuration in server.xml

File Formats

- .pem / .crt / .cer / .key
  - ASCII
  - Key, certificate or chain
- .der
  - Binary form of .pem
- .p7b (PKCS7)
  - ASCII
  - Cert and chain only

- .p12 (PKCS12)
  - Binary
  - Key, cert or chain
- .jks / .keystore
  - Binary
  - Java specific
  - Key, cert or chain

Which Format Do I Need?

- It depends…
- Tomcat 7 or 8, BIO or NIO
  - JSSE implementation, JSSE configuration
  - Keystore
  - PKCS12 with Java 7+
- Tomcat 7 or 8 APR/native
  - OpenSSL implementation, OpenSSL configuration
  - PEM

Which Format Do I Need?

- Tomcat 8.5 and 9, NIO and NIO2
  - KeyStore, PKCS12 or PEM
  - JSSE or OpenSSL for configuration
  - JSSE or OpenSSL for implementation
  - Can’t mix JSSE and OpenSSL attributes in a single configuration
- Tomcat 8.5 and 9, APR/native
  - PEM
  - OpenSSL implementation and OpenSSL configuration

Tomcat 7 or 8: BIO or NIO

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

Tomcat 7 or 8: APR/native

```
<Connector
    protocol="org.apache.coyote.http11.Http11AprProtocol"
    port="8443" maxThreads="200"
    SSLEnabled="true" scheme="https" secure="true"
    SSLProtocol="TLSv1+TLSv1.1+TLSv1.2"
    SSLCertificateFile="/usr/local/ssl/server.crt"
    SSLCertificateKeyFile="/usr/local/ssl/server.pem"
    SSLVerifyClient="optional"
    />
```

Changes in Tomcat 8.5

- Tomcat 7 / Tomcat 8
  - 1 Connector, 1 Hostname, 1 certificate
- Tomcat 8.5 / Tomcat 9
  - 1 Connector, 1 or more Hostnames
  - 1 Hostname, 1 or more certificates (different types)
- Tomcat 8 style configuration is supported but deprecated
  - Connector level attributes are equivalent to the default TLS Host

Tomcat 8.5 onwards: APR/Native

```
<Connector
    protocol="org.apache.coyote.http11.Http11AprProtocol"
    port="8443" maxThreads="150" SSLEnabled="true">
  <SSLHostConfig>
    <Certificate
        certificateKeystoreFile="conf/localhost-rsa.jks"
        type="RSA" />
  </SSLHostConfig>
</Connector>
```

Tomcat 8.5 onwards: NIO or NIO2

```
<Connector
    protocol="org.apache.coyote.http11.Http11NioProtocol"
    port="8443" maxThreads="150" SSLEnabled="true">
  <SSLHostConfig>
    <Certificate certificateKeyFile="conf/localhost-rsa-key.pem"
                 certificateFile="conf/localhost-rsa-cert.pem"
                 certificateChainFile="conf/localhost-rsa-chain.pem"
                 type="RSA" />
  </SSLHostConfig>
</Connector>
```

Tomcat 8.5 onwards: APR/native

```
<Connector
    protocol="org.apache.coyote.http11.Http11AprProtocol"
    port="8443" maxThreads="150" SSLEnabled="true">
  <SSLHostConfig>
    <Certificate certificateKeyFile="conf/localhost-rsa-key.pem"
                 certificateFile="conf/localhost-rsa-cert.pem"
                 certificateChainFile="conf/localhost-rsa-chain.pem"
                 type="RSA" />
  </SSLHostConfig>
</Connector>
```

Questions