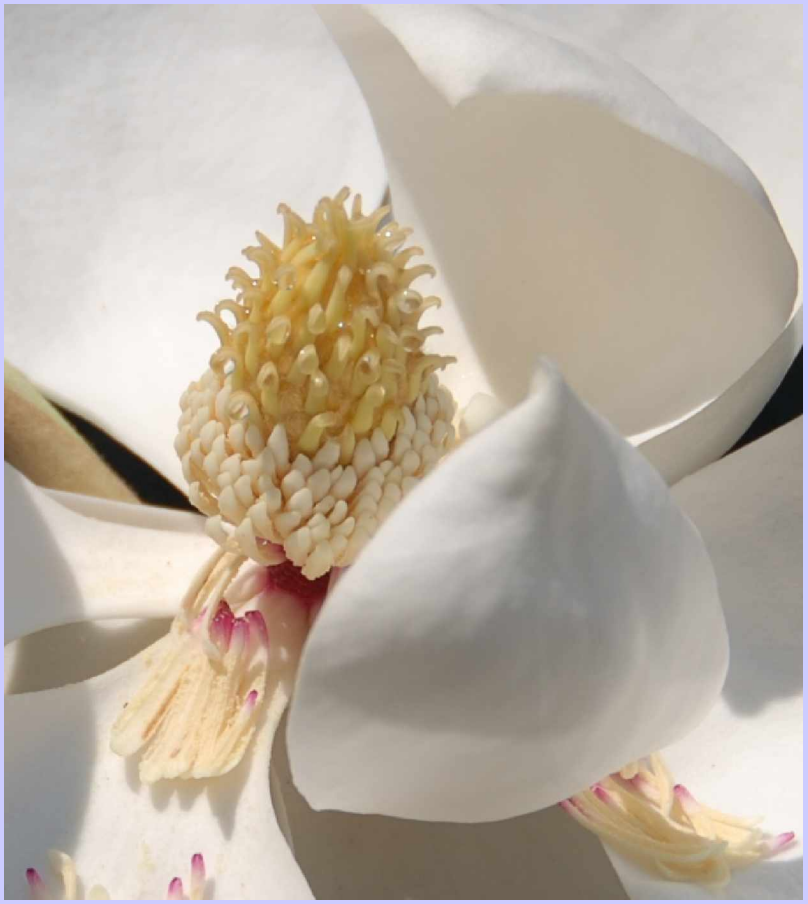
# SSL
# in Detail

Jerry Scott

2007



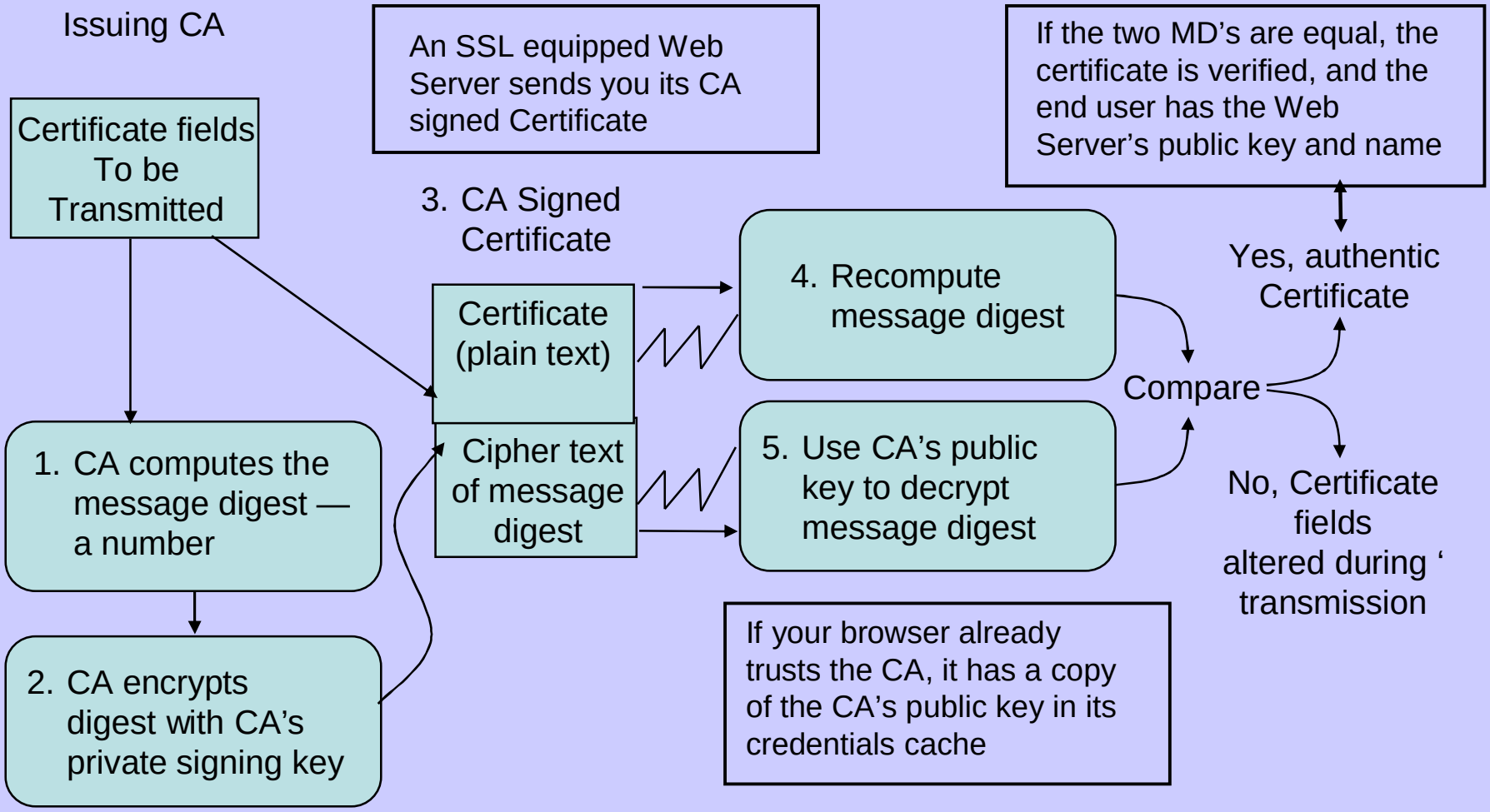
# Verifying a Digital Certificate

- Web Server Certificates must be verified to be useful in E-Commerce
  - Once the certificate is verified, the user has a copy of the server's public key and has authenticated the server name
- How can any user simply verify a digital certificate?
- The digital signature is a Message Digest encrypted with the CA's private key!

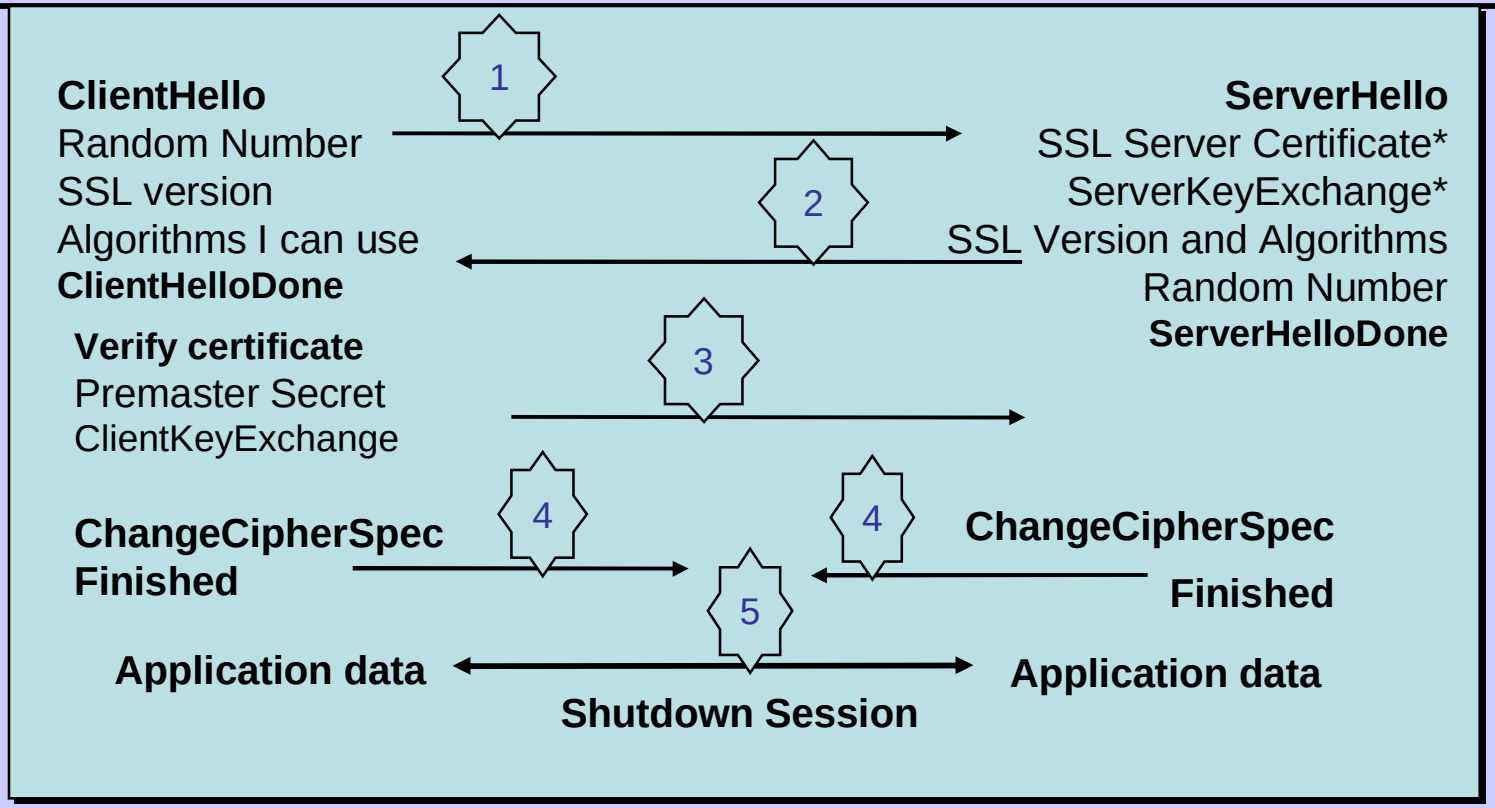
# SSL Certificate Verification

Your browser comes with public keys for many issuing CA's
  - A browser user can add more, if desired, or remove them
- Using the CA's public key in the browser, you can decrypt the digital signature and produce a MD.
  - Hopefully, this is the same MD the CA determined from the certificate data
- Using the known algorithms from the certificate in your browser cache
  - Browser calculates its own MD from the certificate fields
  - If the two MD's are equal, the digital certificate is verified
- Certificate verification produces the public key of the web server and authenticates the server name

# Verifying a Server Digital Certificate



# Handshake Protocol: Sequence




- The ClientKeyExchange contains the PreMaster secret encrypted with the server's public key
  - Everything else, before Step 3 above, is sent unencrypted
- At the end of the SSL/TLS handshake, application data can be sent

Steps marked * are optional.

# Handshake Protocol: Continued

- **ClientHello message consists of**
  - **SSL version number**
    - Highest SSL version that is supported by the client
  - **Cipher suites**
    - List of cryptographic algorithms supported by the client
  - Compression algorithm supported by client
    - Often not used, as the only one defined in SSL is the NULL algorithm
  - 32 Byte Random number (used as input to the master secret)
- **ServerHello message consists of**
  - SSL version number - highest version supported by client and server
  - Cipher suite -- chosen from the list sent by the client
  - Server's digital certificate
  - Compression algorithm -- chosen from the list sent by the client
  - 32 Byte Random number -- used as input to the master secret

# Certification Verification  Step 3

- **Browser verifies the server's digital certificate by**
  - **Using the signing CA's public key gotten from the browser cache**
    - Decrypting the cert's digital signature with this public key, using the algorithms in the server's certificate
      - This produces first message digest  MD1
  - **Using the CA's hashing algorithm, calculates a second Message Digest, MD2, from the certificate fields**
  - **Compares MD1 and MD2.  If they are equal, cert is verified**
- **Once the server's cert is verified, the client now**
  - **Has authenticated the server's name, as the browser's status line and the distinguished name in the cert should be the same**
  - **Has a copy of the server's public key, gotten from the cert**

# Client Key Exchange Step 3

- **The client generates a 48 byte premaster secret**
  - 46 bytes of randomly generated data and a 2 byte version number
- The client encrypts the premaster secret with the server's public key
  - And sends it to the server
- **Server uses its private key to decrypt the premaster**
  - Server still does not know for sure if the premaster is integral and valid
  - The testing and proof of the accuracy of the keys is coming next

# ChangeCipherSpec  Step 4

- Using the premaster secret data and the two 32-byte random numbers exchanged in the “Hello” messages
  - Independently, the client and server use a secure random number generator to create the master secret, and the encryption and signing keys
- Next, the client and server send a ChangeCipherSpec message
  - This message is the message digest of the items sent back and forth

# Application Data Flow: Step 5

- If the ChangeCipherSpec messages are successfully received,
  - This confirms that the keys are successfully generated and identical
- Only after this point do both sides know they can securely communicate and now application data can flow!