SSL/TLS Explained

Brent Denny

SSL/TLS Encryption

- SSL/TLS Uses a hybrid combination of encryption techniques
  - Asymmetric Encryption (also known as Public/Private encryption)
    - When one of the key pair encrypts (Public), only the other can decrypt (Private).
  - Symmetric or Bulk Encryption (also known as Same key and Secret Key)
    - The same key is used for both encryption and decryption.
  - Hash Encryption (Digest Encryption)
    - This is a one-way encryption algorithm, it is used to compare data integrity.

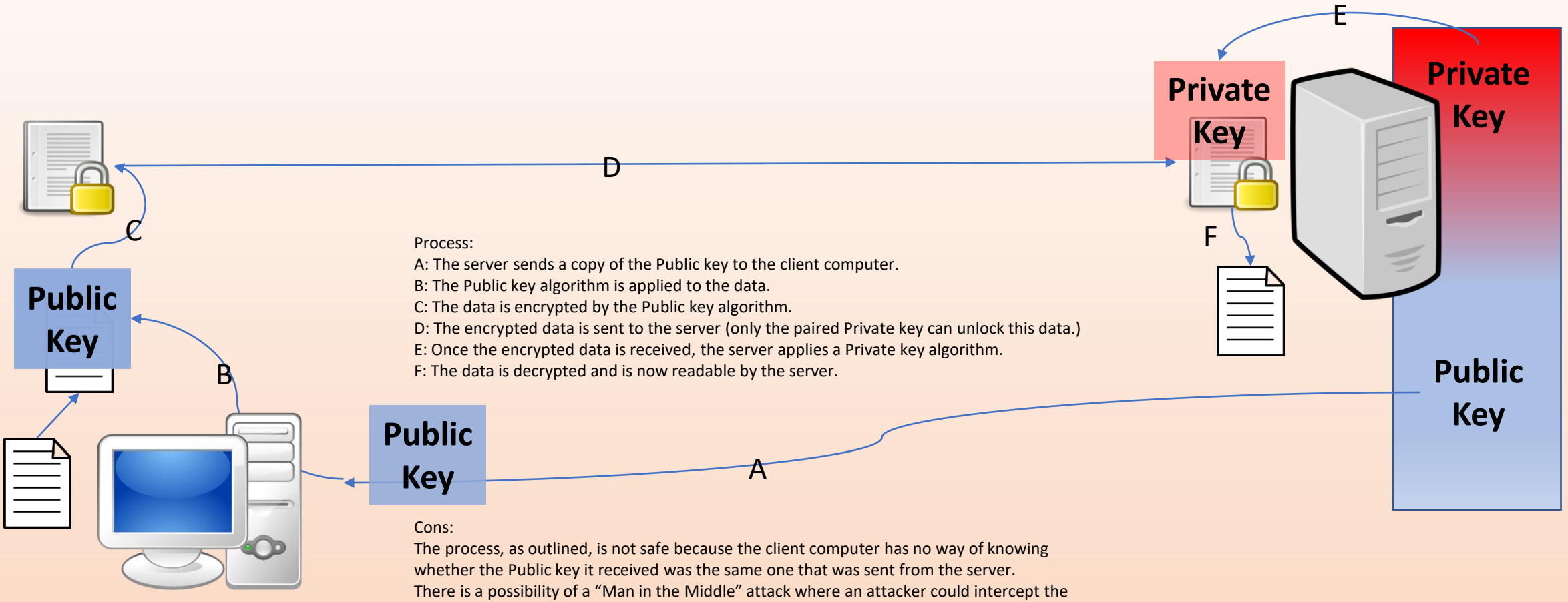
Pros and Cons of each cipher

- Individual ciphers have good and bad issues
- The next few slides highlights the good/bad points of each cipher
- Later slides show how the hybrid of these ciphers makes SSL/TLS strong and secure

Asymmetric Encryption

- Encryption
  - Key Size: 2048 or 4096 bits (RSA)
  - Cons: Very CPU intensive for large data (only used for small data blocks)
  - Pros: Uses a split key system that fosters security (Public and Private)
  - Use: The server holds a Public Key and a Private Key, any data encrypted by the Public key can only be decrypted by the paired Private key. The Public key owner, gives others a copy of the Public key, so that they can encrypt information with that public key and then send the encrypted data back to the server. The server uses their Private key to decrypt this data. In this example, the Public key encrypts the data, it cannot be then used to decrypt that encrypted data, only the paired Private key can do this.

Asymmetric Encryption – Example



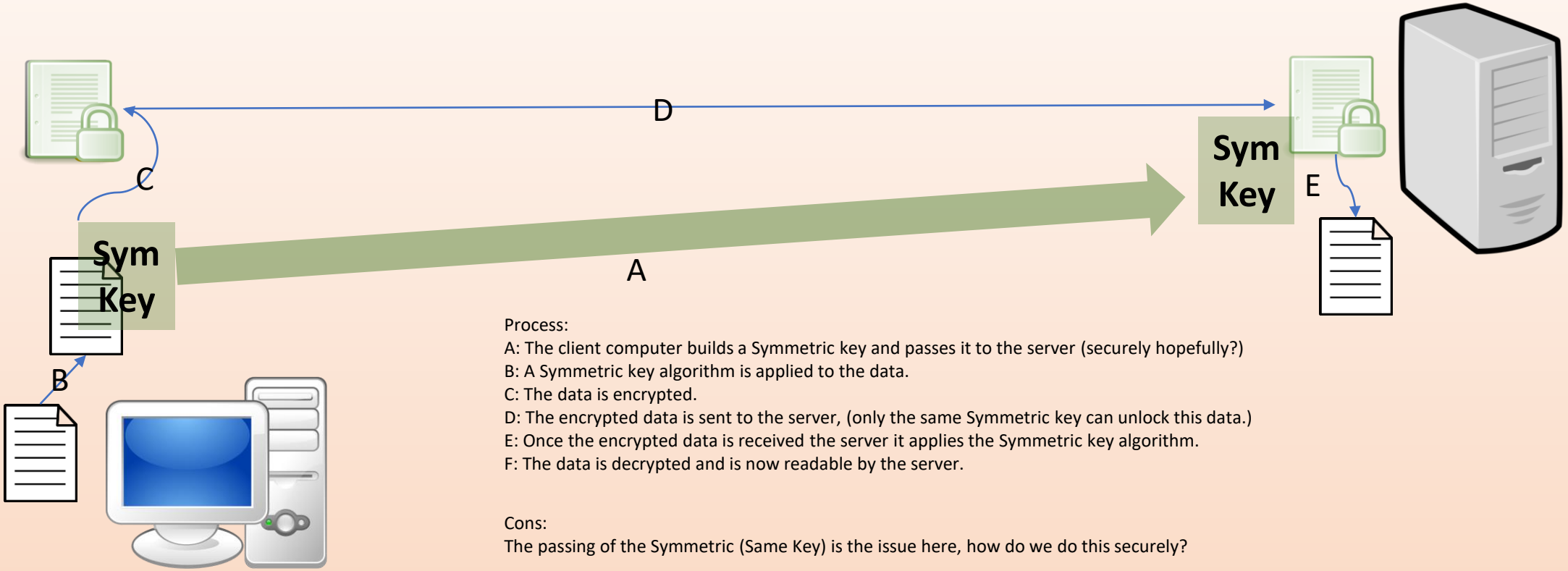
Process:
A: The server sends a copy of the Public key to the client computer.
B: The Public key algorithm is applied to the data.
C: The data is encrypted by the Public key algorithm.
D: The encrypted data is sent to the server (only the paired Private key can unlock this data.)
E: Once the encrypted data is received, the server applies a Private key algorithm.
F: The data is decrypted and is now readable by the server.

Cons:
The process, as outlined, is not safe because the client computer has no way of knowing whether the Public key it received was the same one that was sent from the server.
There is a possibility of a “Man in the Middle” attack where an attacker could intercept the Public key transmission and replace it with another Public key.

Symmetric Encryption

- Key Size: 40, 56, 128, 168, 256bits (CAST, DES, RC4, 3DES, AES, etc.)
- Cons: Same key required by both parties, (how to pass the key from one party to the other securely, is the issue.)
- Pros: Extremely fast encryption algorithm
- Use: The same key is used to encrypt and decrypt the data, the encryption happens very quickly even for large amounts of data. The major issue with this process is how to first pass the same key to both parties securely so that the encryption is not compromised. Sometimes called: Same/Secret Key Encryption or Bulk Encryption.

Symmetric Encryption – Example



Process:
A: The client computer builds a Symmetric key and passes it to the server (securely hopefully?)
B: A Symmetric key algorithm is applied to the data.
C: The data is encrypted.
D: The encrypted data is sent to the server, (only the same Symmetric key can unlock this data.)
E: Once the encrypted data is received the server it applies the Symmetric key algorithm.
F: The data is decrypted and is now readable by the server.

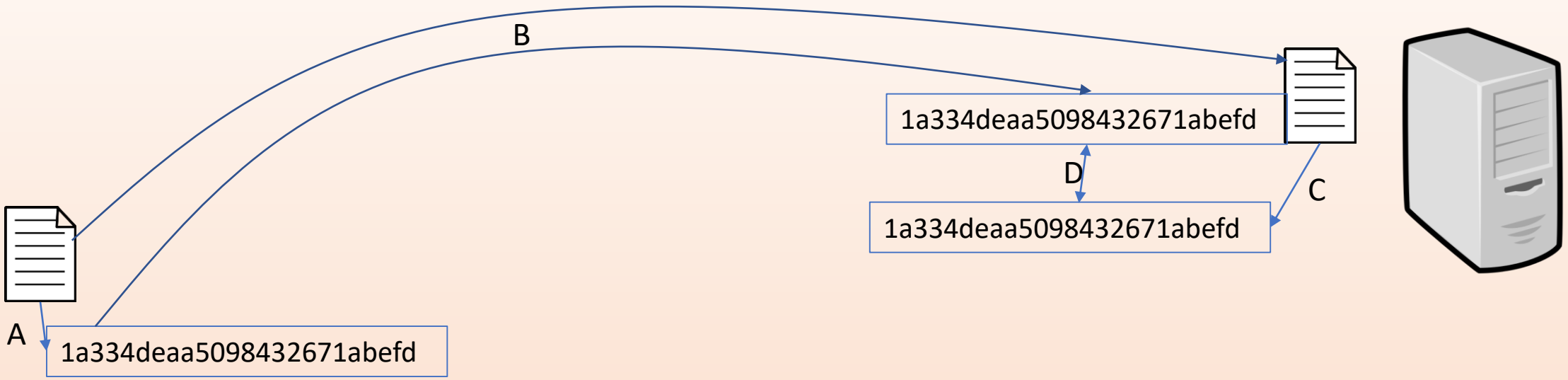
Cons:
The passing of the Symmetric (Same Key) is the issue here, how do we do this securely?

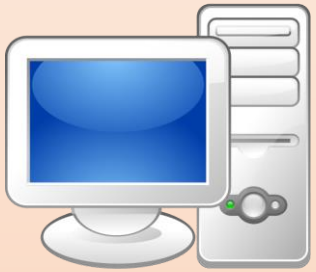
Hash Encryption (Digest Encryption)

- Common Algorithms: MD5, SHA256
- Cons: In extremely rare cases two different data blocks can produce the same hash results, this is called a hash collision.
- Pros: This is useful in determining the integrity of the data.
- Use: Hashing is used by taking the unencrypted data and applying a hash algorithm, this produces a resulting value. Hashing the data again with the same algorithm will produce the same result as the original hash, proving the data has not been modified. However if the data has been tampered with the two hash results will be different*.

  * Except in the extremely rare case of a collision.

Hash Encryption – Example





Process:
A: The client computer applies a Hash algorithm to the data and produces a result
B: The data is transmitted to the server (The Hash result is normally sent as well, as a Digital Signature*, discussed later*.)
C: The server applies the same Hash algorithm and produces its own result.
D: The server compare the two Hash results, if they are the same the data has not been modified.

Cons:
In extremely rare cases a hash collision can occur, meaning two different blocks of data have produced the same hash result.

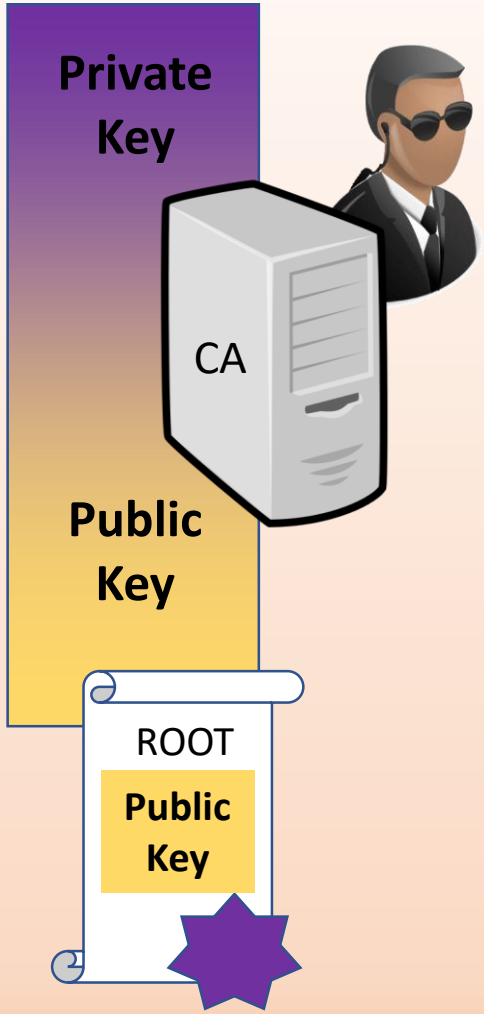
SSL/TLS and PKI

- To overcome the issues discussed earlier
  - Asymmetric – only used for very small data size encryption.
  - Symmetric – difficulty in passing the key so both parties have the same key.
  - Hashing – How to send the hash as a Digital Signature.
- A Certificate Authority produces a certificate to verify
  - Authenticity of the public key.
  - Declare the Dates that the certificate is valid (From and To dates),
  - Declare the FQDN the clients use, to access the server holding this certificate.
  - Declare which Certificate Authority has certified the certificate.
  - Certify that the certificate has not been tampered with.
  - Certify that the certificate has not been revoked.

Requesting a Certificate

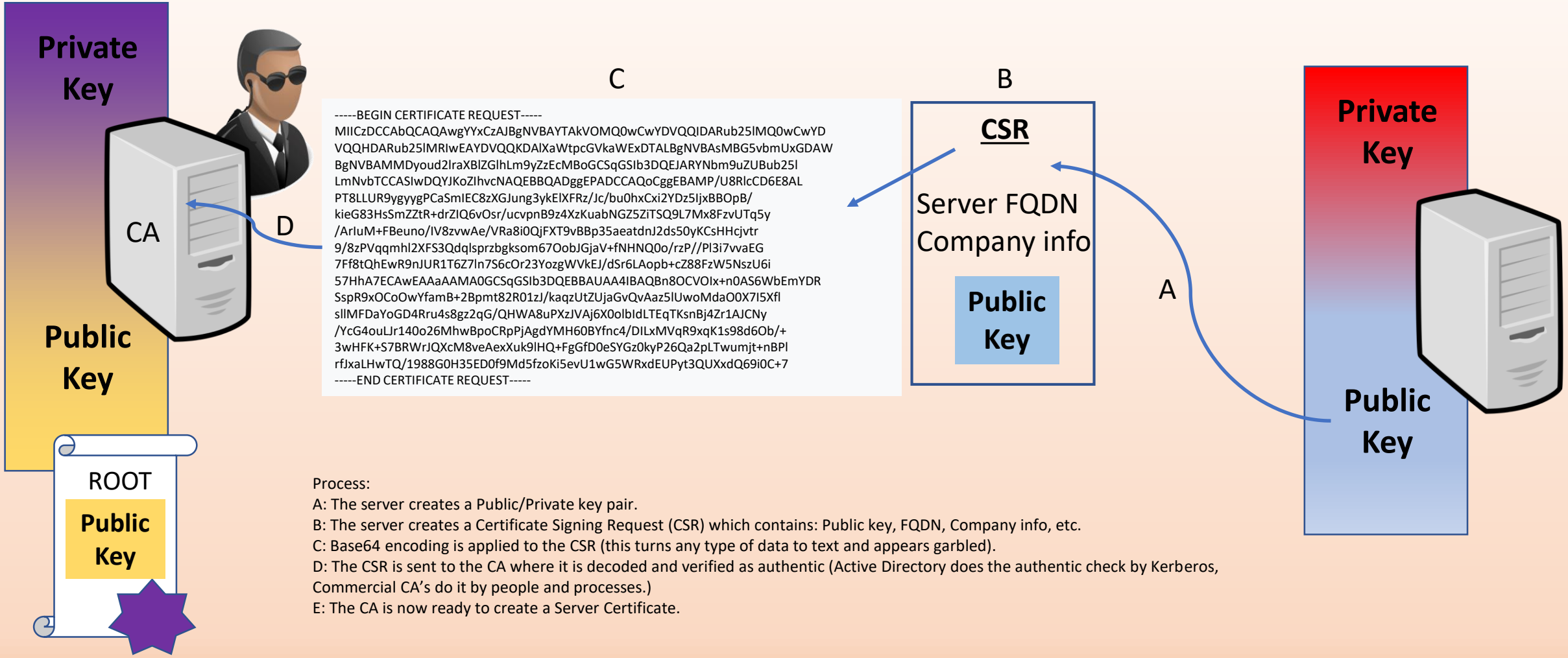
- The server (wishing to host an SSL service) creates
  - Public / Private Key pair
  - Certificate Signing Request (obscured by base64 encoding)
    - FQDN of server (Common Name)
    - Public Key
    - Company Name
    - Company Address
    - Contact details
- The server then
  - Sends the CSR to the Certificate Authority (CA)
  - Will retrieve the Certificate from the CA certifying the server’s public key

Certificate Authorities (CA)



- Have their own Public and Private key pairs.
- Have a certificate that holds the CA's Public key.
- Has processes in place to verify incoming CSR requests.
- Regularly publishes a list of compromised certificates in a CRL.
- Can have subordinate CA's called intermediate and issuing CA's.

Requesting a Certificate - Example



Process:
A: The server creates a Public/Private key pair.
B: The server creates a Certificate Signing Request (CSR) which contains: Public key, FQDN, Company info, etc.
C: Base64 encoding is applied to the CSR (this turns any type of data to text and appears garbled).
D: The CSR is sent to the CA where it is decoded and verified as authentic (Active Directory does the authentic check by Kerberos, Commercial CA's do it by people and processes.)
E: The CA is now ready to create a Server Certificate.

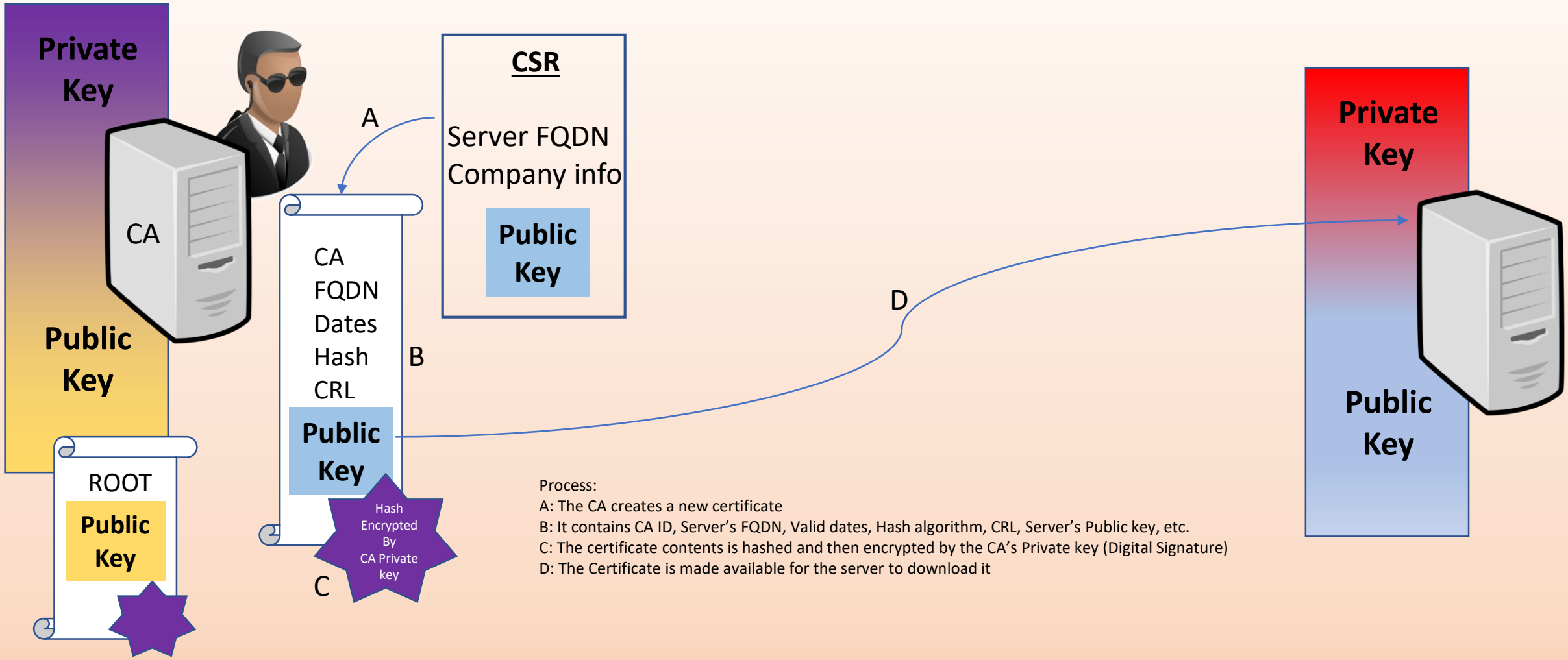
Certificate Authority - Certificate Creation

- The CA receives the CSR from the server.
- The CA verifies that the CSR came from the server.
- The CA starts the certificate creation.
- The Certificate contains: (this is some of the contents)
  - Name of CA (Issuer)
  - FQDN of server (Subject)
  - Public key of server
  - Dates of validity (From and To)
  - Certificate Revocation Location (CRL)
  - Hash algorithm
  - Other info (Serial number, certificate role, …)

Certificate Creation cont…

- The CA then produces a hash value (Hashing the certificate contents)
- Encrypts the hash value with the CA's Private key (Digital Signature)
- Attaches the Digital Signature to the Certificate
  - The Digital Signature is encrypted (It can be decrypted by the CA Public key)
  - The other certificate contents mentioned earlier are not encrypted
- The CA makes this new certificate available for download, by the server

Certificate Creation - Example

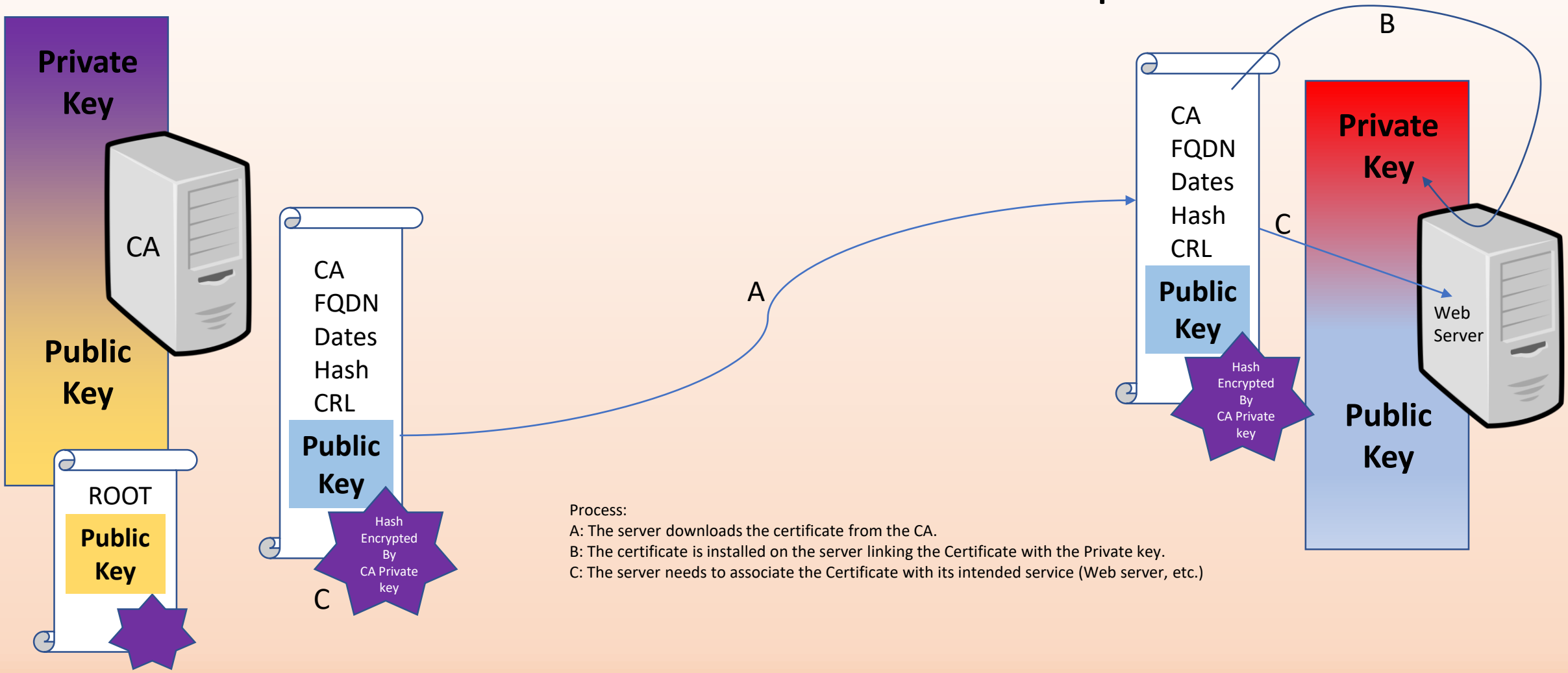


Process:
A: The CA creates a new certificate
B: It contains CA ID, Server's FQDN, Valid dates, Hash algorithm, CRL, Server's Public key, etc.
C: The certificate contents is hashed and then encrypted by the CA's Private key (Digital Signature)
D: The Certificate is made available for the server to download it

Installing the Certificate

- The Server
  - Downloads the Certificate from the CA
  - Install the certificate which links the Server’s private key with the certificate
  - Binds the certificate to a https service (Web Server, etc.)

Install Certificate - Example

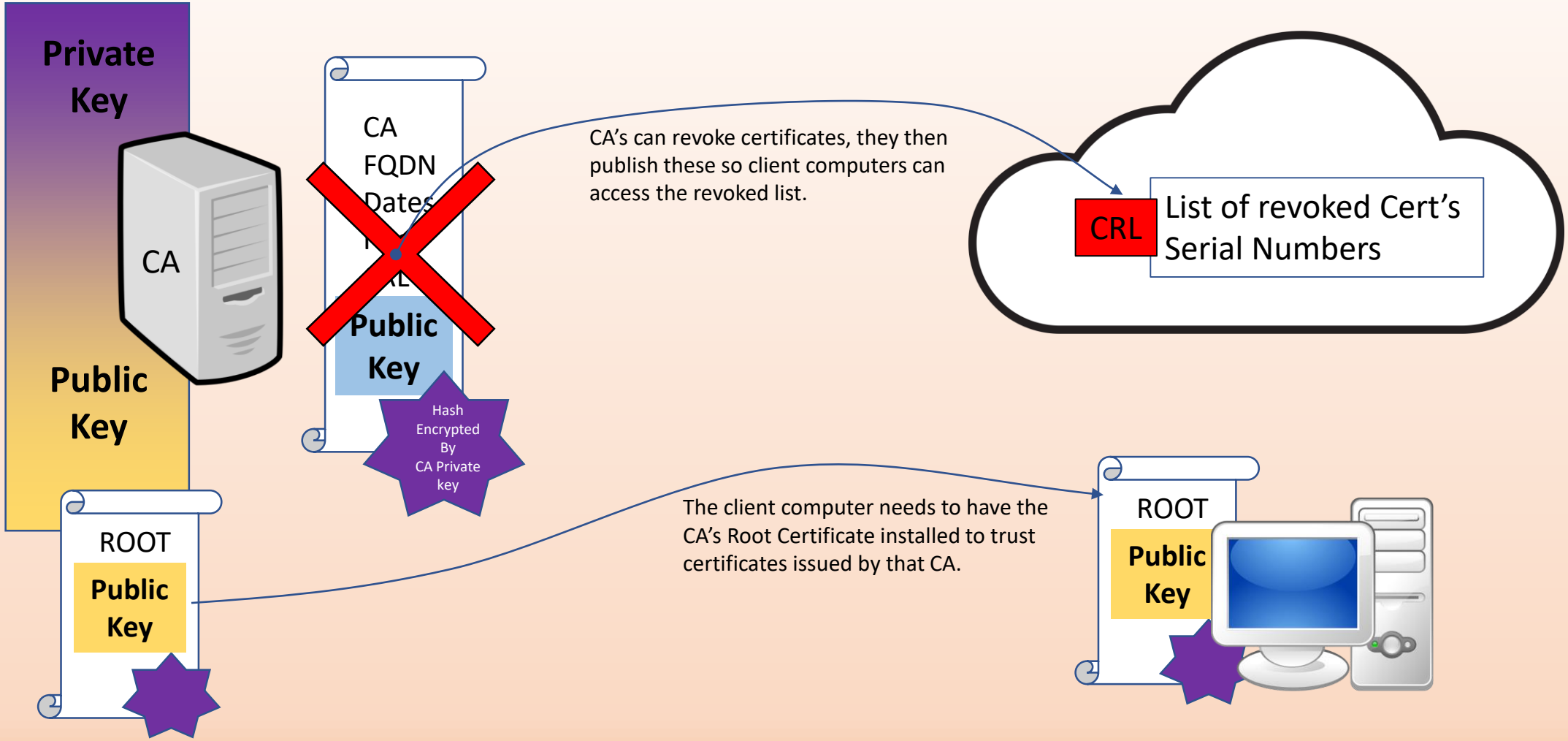


Process:
A: The server downloads the certificate from the CA.
B: The certificate is installed on the server linking the Certificate with the Private key.
C: The server needs to associate the Certificate with its intended service (Web server, etc.)

Other important information

- Certificate security breaches
  - The CA can revoke any certificate it has issued (for a number of reasons)
  - The revoked certificates are published and are available for download
  - The client computer views the CRL to see if the Server Certificate is valid
- Certificate Trusts
  - The client must be able to trust the Server Certificate
    - To do this the client must have the CA's Root Certificate installed on the client

Other important information

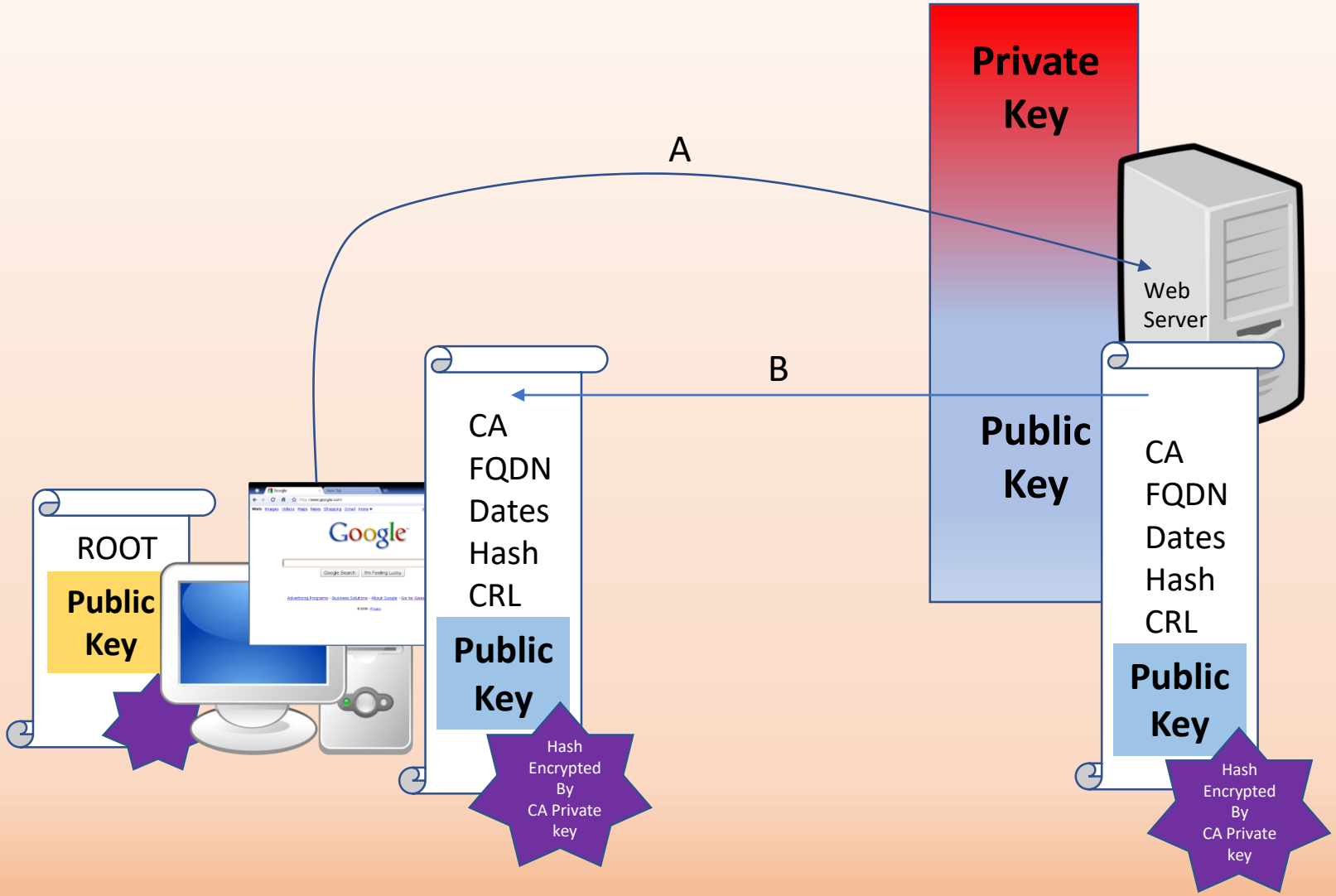


Creating the secure connection

- Client uses https://FQDN to access the SSL service (e.g. web server)
- SSL/TLS handshake begins
  - Client downloads Server Certificate from server (e.g. Web Server)
  - Finds an installed Root Certificate that matches the CA ID in Server Certificate
  - Hashes the contents of the Server Certificate (Browser hash result)
  - Extracts the CA Public key from Root Certificate
  - Decrypts the signature with CA Public key (Original hash result)
  - Compares the two hash results (if equal => Certificate has not been modified)
  - Compares system clock with validity dates in the Server Certificate
  - Compares FQDN in Server Certificate with FQDN in the URL

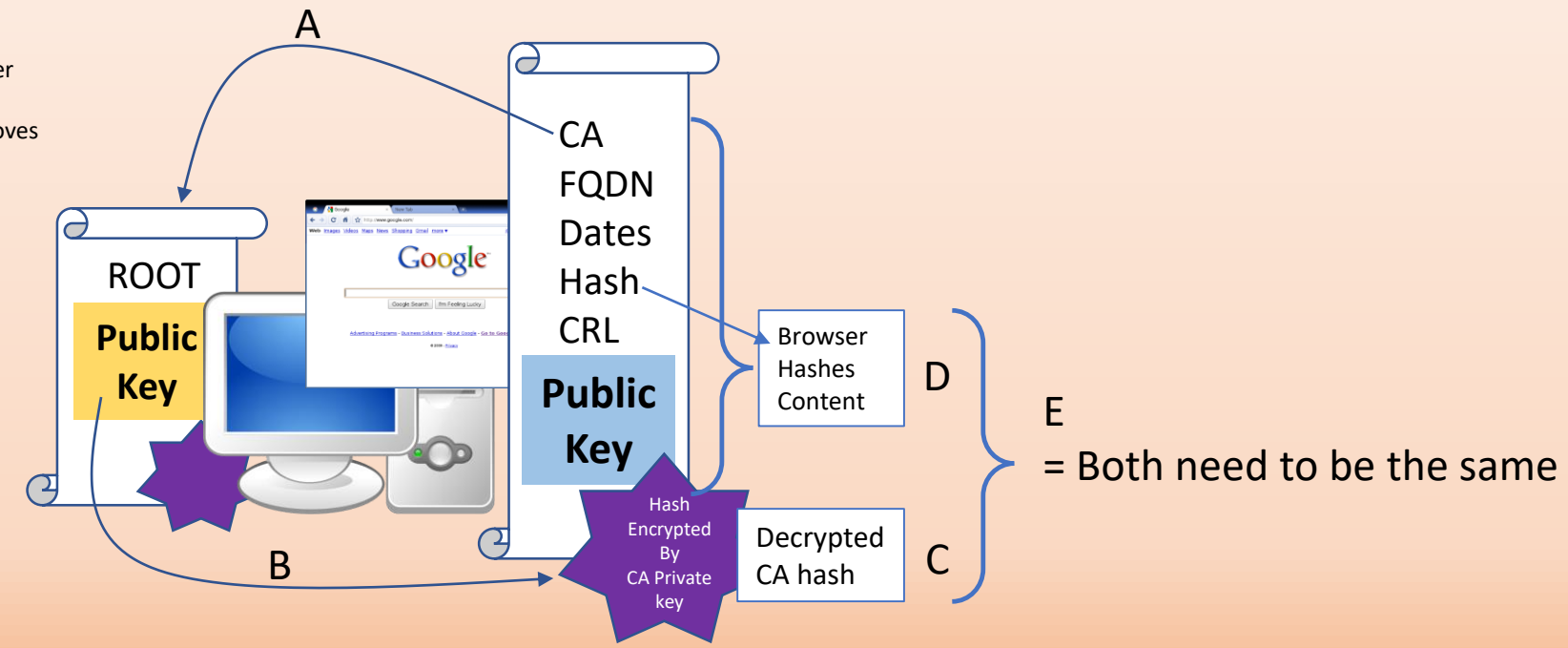
SSL/TLS Session – Certificate download

Process:
A: The client issues a URL (https://FQDN) to the web server
B: The Server Certificate associated with that web site is downloaded to the client

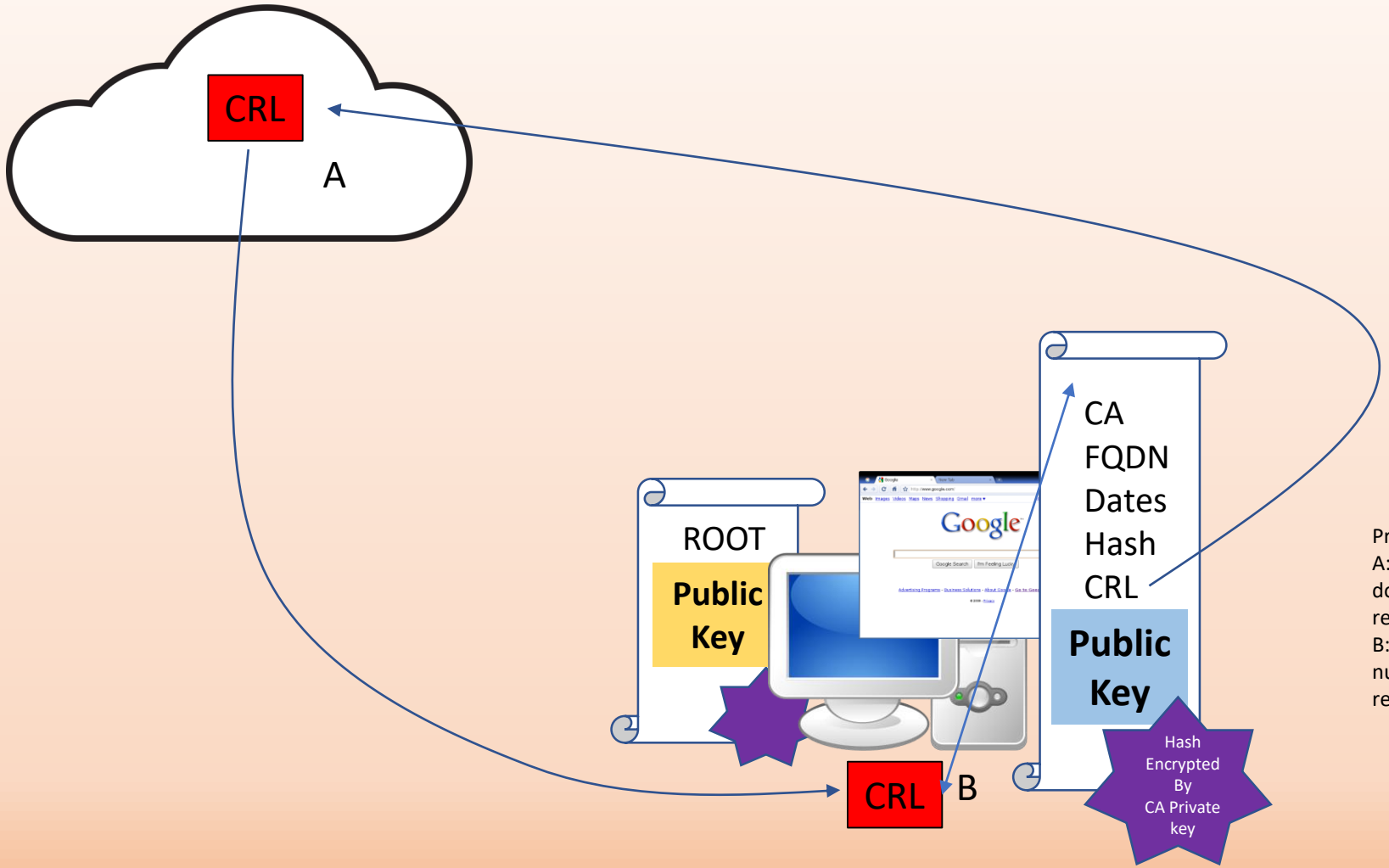


SSL/TLS Session – Certificate integrity

Process:
A: Client finds a locally installed Root Certificate that matches the CA name in the downloaded Server Certificate.
B: The Public Key from the Root Certificate is used to decrypt the signature.
C: The original CA Hash is exposed to the Client.
D: The Client finds the Hash algorithm method in the Server Certificate and uses this type to create its own Hash.
E: The two hash results are compared, if the same, this proves the certificate has not been modified by a third party.



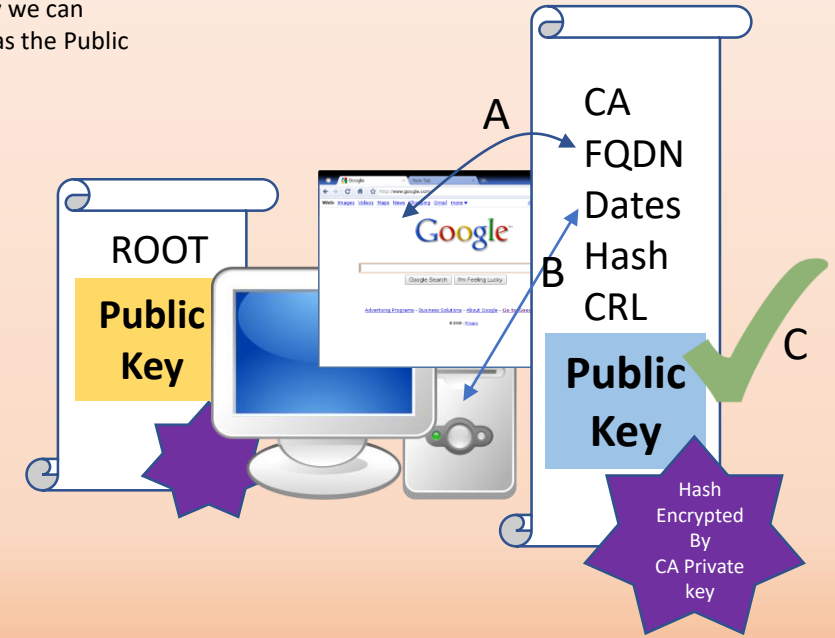
SSL/TLS Session – CRL Checking



Process:
A: Client uses the CRL in the Server Certificate to Locate and download the CRL which is a list of revoked certificates referenced by serial number.
B: The Client checks to see that the Server Certificate serial number is NOT listed in the CRL. This means it has not been revoked by the CA.

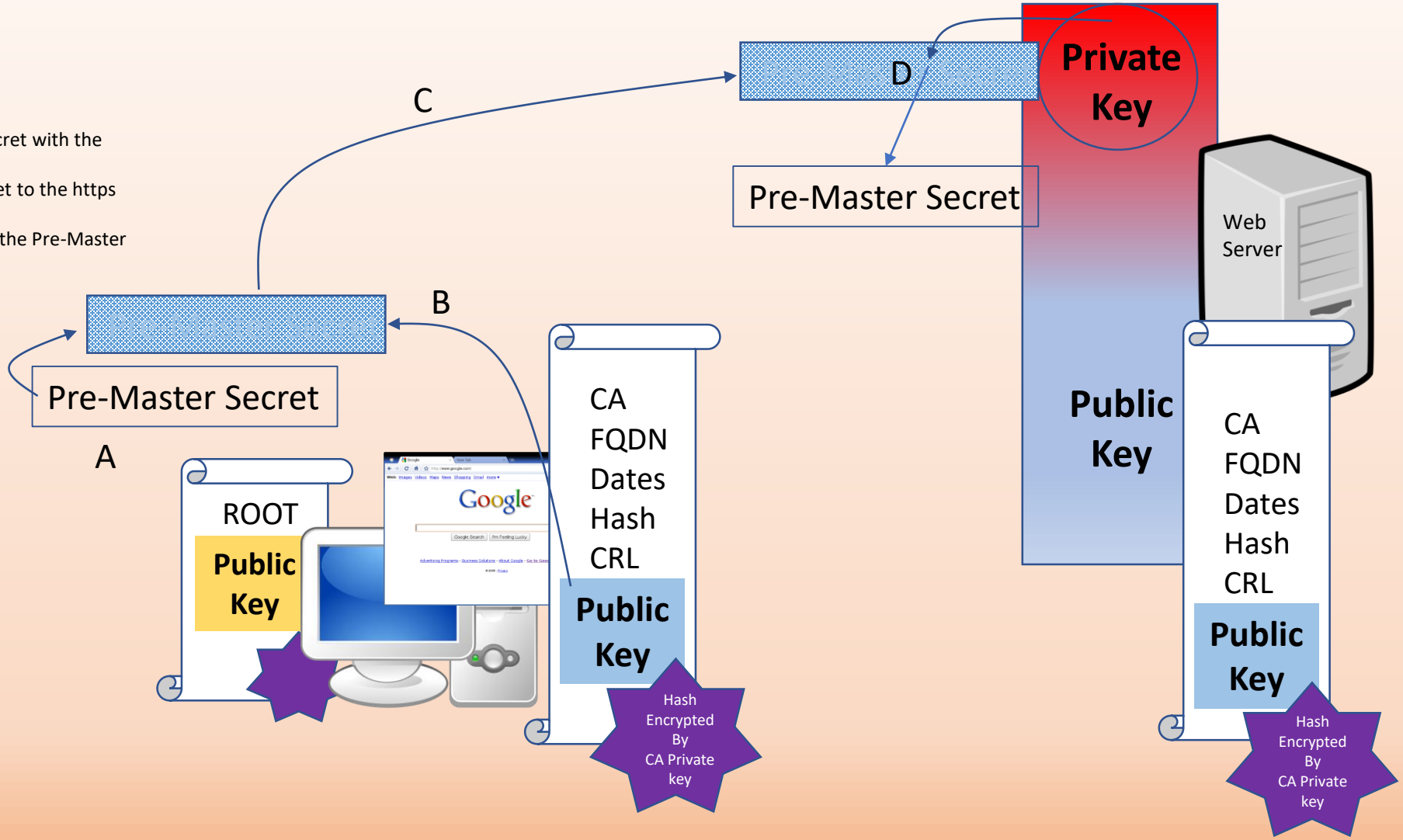
SSL/TLS Session – Check Certificate Validity

Process:
A: The Client checks to see is the FQDN typed in the URL matches the FQDN in the Server Certificate.
B: The Client checks to see if the From and To dates found in the Server Certificate are valid when comparing these to the current system date.
C: After proving the Certificate integrity, trust and validity we can assume that the Public key in this Certificate is the same as the Public key on the service (Web Server) we engaged via https.



SSL/TLS Session – Pre Master Secret

Process:
A: The Client creates a Pre-Master secret value.
B: The Client encrypts a copy of the Pre-Master Secret with the Server's Public key found in the Server Certificate.
C: The Client sends the encrypted Pre-Master Secret to the https service (Web Server).
D: The https service uses its Private key to decrypt the Pre-Master Secret value.



SSL/TLS Session – The Final Part

Process:
A: Using the Pre-Master Secret the Client creates (Seeds) a Symmetric Encryption key.
B: Using the Pre-Master Secret the Server creates (Seeds) a Symmetric Encryption key.
C: Because both keys have been created from the same master, they are the same as each other and this is what is required for Symmetric key encryption, both parties having the same key.
D: The client now encrypts the data with the Symmetric Key,
E: The Client sends the encrypted data to the web server.
F: The web server used its Symmetric key to decrypt the data.
G: The client's data can now be read by the web server

This process is now done in reverse for data from the webserver that needs to be sent back to the client, hence all data sent between client and server are being encrypted by the same symmetric key.