# Network Services Interface

## Secure Communications with Self Signed Certificates

John MacAuley, ESnet

18th July 2014

# The Statement

"TLS provides message integrity, confidentiality and authentication via the X.509 certificates, and protects against replay attacks. Authorization is done at the NSAs application level." – Transport Layer Security, NSI CS v2, Section 9.1, page 96

# Definitions

- Authentication is the process of confirming the identity of an entity.
- Authorization is the function of specifying access rights to resources for an entity.

Authentication is the process of verifying that "you are who you say you are", authorization is the process of verifying that "you are permitted to do what you are trying to do”.

# TLS authentication

- TLS uses X.509 certificates and asymmetric cryptography to authenticate communication between two parties.
- Certificate authorities and a public key infrastructure are necessary to verify the relationship between a certificate and its owner, as well as to generate, sign, and administer the validity of certificates.
- Each end of a TLS session authenticates its peer using its trusted certificate authorities via a public key infrastructure.

It is important to note that a successful TLS session can be established to a peer if that peer's certificate can be validated using the PKI. Effectively, the TLS handshake will establish a session if it can determine "you are who you say you are", and does not consider if you have access to the application on the endpoint of the TLS session.

# Example signing hierarchy

- A CA-signed certificate has a chain of Subject and Issuer DNs leading from the leaf certificate to the root certificate of the CA (self-signed).

**{**

**Subject:** /OU=Domain Control Validated/CN=nsi-aggr-west.es.net

**Issuer:** /C=US/ST=Arizona/L=Scottsdale/O=GoDaddy.com, Inc./OU=http://certs.godaddy.com/repository//CN=Go Daddy Secure Certificate Authority - G2

**}**

**{**

**Subject:** /C=US/ST=Arizona/L=Scottsdale/O=GoDaddy.com, Inc./OU=http://certs.godaddy.com/repository//CN=Go Daddy Secure Certificate Authority - G2

**Issuer:** /C=US/ST=Arizona/L=Scottsdale/O=GoDaddy.com, Inc./CN=Go Daddy Root Certificate Authority – G2

**}**

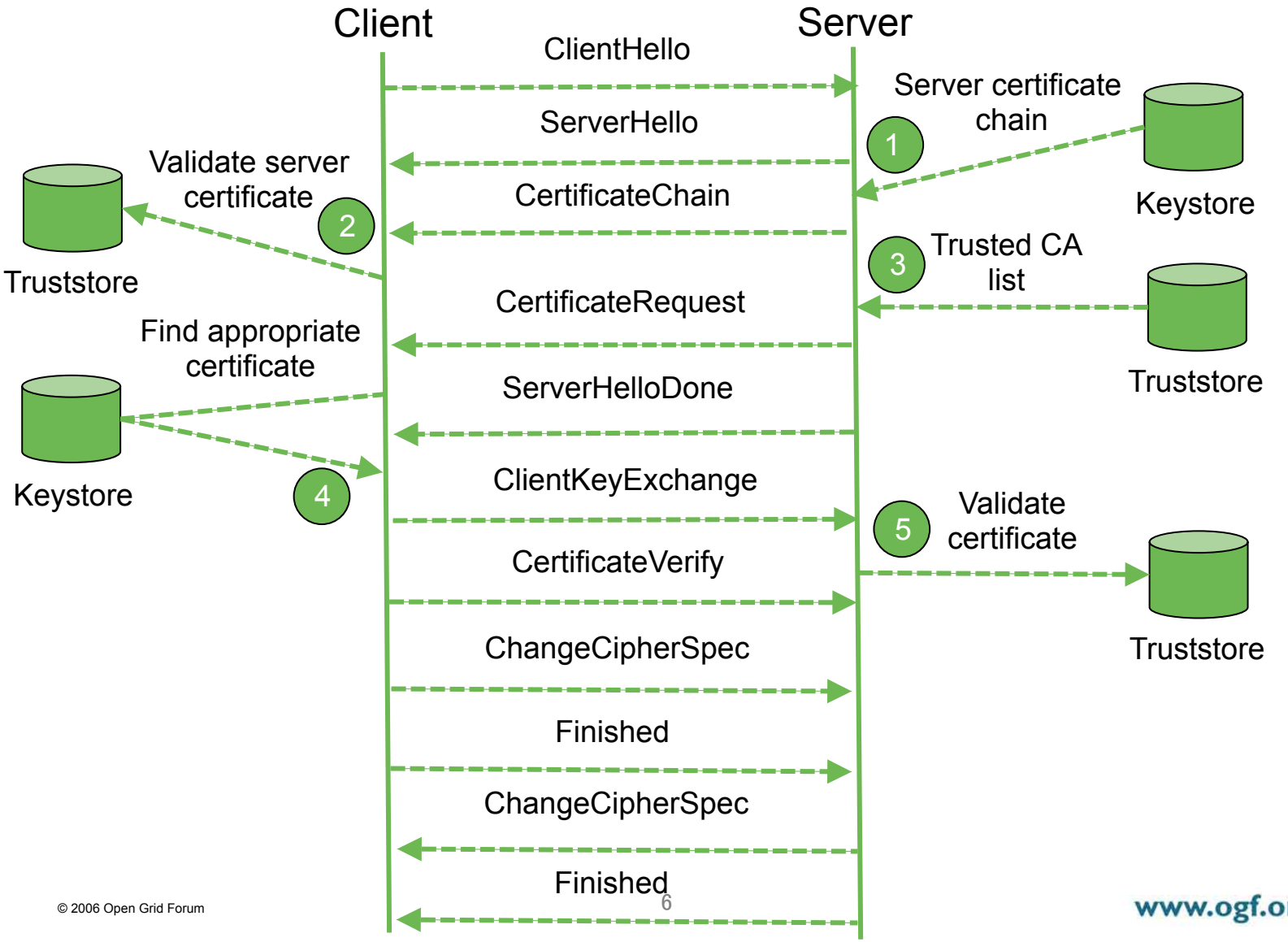
**{**

**Subject:** /C=US/ST=Arizona/L=Scottsdale/O=GoDaddy.com, Inc./CN=Go Daddy Root Certificate Authority - G2

**Issuer:** /C=US/ST=Arizona/L=Scottsdale/O=GoDaddy.com, Inc./CN=Go Daddy Root Certificate Authority - G2

**}**

# TLS negotiation sequence



# Steps: TLS handshake protocol

- The client sends a "Client hello" message to the server, along with the client's random value and supported cipher suites.
- The server responds by sending a "Server hello" message to the client, along with the server's random value.
- The server sends its certificate to the client for authentication.
- The server may request a certificate from the client (client authentication), and if so, sends a list of certificate authorities to the client, one of which must have signed the client's certificate.
- The server sends the "Server hello done" message.
- If the server has requested a certificate from the client, the client must send one signed by a listed certificate authority, otherwise the client aborts the handshake sequence.
- The client creates a random Pre-Master Secret and encrypts it with the public key from the server's certificate, sending the encrypted Pre-Master Secret to the server.

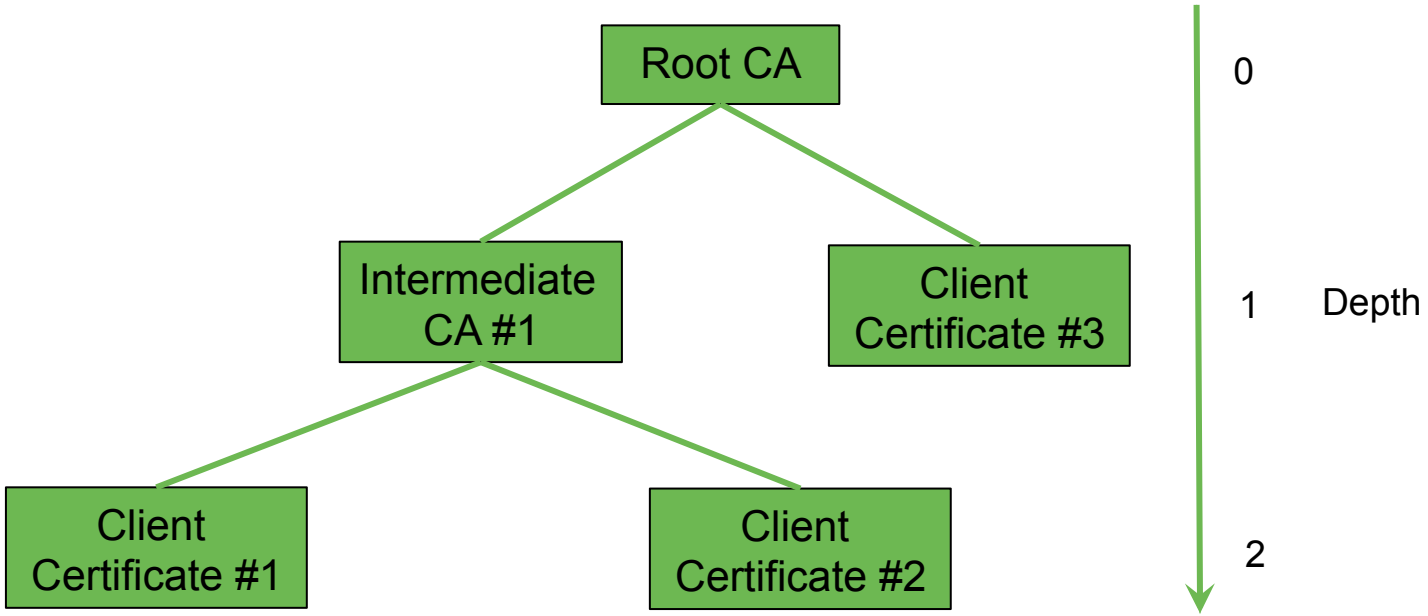
# Steps: TLS handshake protocol

- The server receives the Pre-Master Secret. The server and client each generate the Master Secret and session keys based on the Pre-Master Secret.
- The client sends "Change cipher spec" notification to server to indicate that the client will start using the new session keys for hashing and encrypting messages. Client also sends "Client finished" message.
- Server receives "Change cipher spec" and switches its record layer security state to symmetric encryption using the session keys. Server sends "Server finished" message to the client.
- Client and server can now exchange application data over the secured channel they have established. All messages sent from client to server and from server to client are encrypted using session key.

# Notes

1. The server sends its certificate and the list of certificate authorities that have signed the certificate from leaf to root.
2. The client validates the the server certificate and checks its local trust store to determine if the provided certificate can be trusted. It is important to note that the client does not require a list of valid CA provisioned locally, but only the server certificate itself.
3. The server sends a list of DNs for all certificates within the local trust store (CA certificates), that the server will accept as valid signing CA. The client must respond with a certificate signed by one of the DNs. The implication of this requirement is that the server's trust store must contain the issuing certificate of every client certificate used to connect, and depending on the validation policy configured, could require signing certificates up to the root CA.
4. The client returns an appropriate certificate signed by one of the CA certificates listed by the server. If an appropriate certificate cannot be found the handshake is aborted.
5. The server validates the supplied client certificate against the local trust store. If the certificate has a valid CA chain then it is accepted.

# The leaky world of trust




If we want to allow client certificate #1 to access our NSA, then we must have the Root CA and Intermediate CA #1 as trusted CA within our server. Unfortunately, this also means we will be able to verify the identity of Client #2 and #3, allowing for a TLS session to be established.

Use of certificate depth:

- A verify depth of “0” means only self-signed certificates are allowed.
- A verify depth of “1” means only certificates directly under a self-signed certificate are allowed.
- A verify depth of “2” means at most one intermediate signing certificate is allowed.

# Issues

- TLS is based on a trust relationship - I will trust a certificate signed by a Certificate Authority I also trust.
- The server side is not provisioned with individual client certificates, but requires the root signing CA certificate and any intermediate CA so that any presented client certificate can be validated.
- This opens up successful TLS negotiation for any clients signed by CA certificates provisioned in the server.

Unfortunately, after the TLS layer has verified the client certificate, further access control is required within our NSA applications to access control clients, granting access to only those certificate DNs that truly have access.

# What are self-signed certificates?

- A self-signed certificate is a certificate signed with its own private key.
- If the parties know each other, trust each other to protect their private keys, and can securely transfer public keys (e.g. compare the hash out of band), then self-signed certificates may decrease overall risk when compared to a certificate authority (CA) signed certificate.
- Self-signed certificate transactions may also present a far smaller attack surface.

# Advantages of self-signed certificates

- **Cost**
  - Self-signed certificates can be created for free using a wide variety of tools.
  - Certificates bought from major CAs can cost from tens to hundreds dollars per year depending on the level of trust.
- **Speed to Deploy**
  - Self-signed certificates require the two parties to interact (e.g. to securely trade public keys).
  - Using a CA requires only the CA and the certificate holder to interact; the holder of the public key can validate its authenticity with the CA's root certificate.
- **Customization**
  - Self-signed certificates are easier to customize, for example a larger key size, contained data, metadata, etc.

# Example signing hierarchy

- A self-signed certificate has the same Subject and Issuer DNs.

  **{**

  **Subject:** /C=CA/ST=ON/L=Ottawa/O=US Department of Energy/OU=Energy Sciences Network/CN=John MacAuley/emailAddress=macauley@es.net

  **Issuer:** /C=CA/ST=ON/L=Ottawa/O=US Department of Energy/OU=Energy Sciences Network/CN=John MacAuley/emailAddress=macauley@es.net

  }

- This indicates the Certificate Authority used to verify the authenticity of this certificate is the certificate itself (it is a root certificate).

# Private certificate authorities

- Self-signed certificates require each certificate to be added to the trust store
  - Although manageable for NSA peering into external domains, it may not be appropriate for all locally deployed applications requiring access to an NSA.
- Creating your own private “root” certificate authority allows for the signing of your own certificates
  - Having the ability to provision a private root CA, and not worry about installation of individually generated certificates signed by this CA can provide flexibility.
- This also allows for a very restricted and controlled certificate pool to allow only approved clients to establish a TLS session.

# Recommendation

- If people are open to making our NSA peering more secure via TLS then…
- Switch to the use of self-signed certificates allowing for TLS verification of only allowed NSA.
- We already have ad hoc procedures in place to exchange NSA certificates so this provides a relatively secure solution for the turn-up of NSA peering.
- Additional NSA access control mechanisms can be added on top of TLS as needed by the individual NSA implementations, however, this base TLS policy will guarantee only the desired peers will be allowed to establish.
- Sites can use private root certificate authorities to manage groups of certificates if needed for larger local client deployments.