# TLS 1.3 Option for Negotiation of Visibility in the Datacenter

# <draft-rhrd-tls-tls13-visibility-01>

March 2018

Russ Housley and Ralph Droms

# The Need

- The need for the TLS Visibility Extension was discussed in Seoul and Prague
- Two Internet-Drafts capture the need:
  - **draft-camwinget-tls-use-cases**
    *TLS 1.3 Impact on Network-Based Security*
  - **draft-fenter-tls-decryption**
    *Why Enterprises Need Out-of-Band TLS Decryption*

- Mail list discussion indicated that many people are more comfortable with a solution that requires opt-in by the client

# Goals

- The TLS Visibility Extension addresses one of the impacts of (EC)DH in the datacenter environment
- The extension provides an opt-in mechanism that allows a TLS client and server to explicitly grant access to the TLS session plaintext to other parties
  - The enterprise key manager decides which other parties
- A third party can detect whether this extension is present by observing the ClientHello and ServerHello messages
- No other parties get the TLS server's digital signature private key, so no other party can masquerade as the server in other TLS handshakes

# Prerequisites

- The enterprise key manager:
  1. Generates an ECDH key pair, called SSWrapDH1
  2. Distributes the public key to the TLS server
  3. Distributes the private key to the other parties in the datacenter that are authorized to access the TLS session plaintext
  4. Distributes the AEAD algorithm that will be used to encrypt the TLS session secrets to the TLS server and the other parties

- SSWrapDH1 is identified by its "fingerprint”
  - The leftmost 20 octets of the SHA-256 hash of the public key

# TLS Visibility Extension (1 of 2)

- Client includes an Empty structure in the ClientHello message
- Server encrypts the session secrets and includes them in the ServerHello message
- Other parties that have the SSWrapDH1 private key can decrypt the session secrets and then decrypt the session itself

# TLS Visibility Extension (2 of 2)

```
struct {
    select (Handshake.msg_type) {
        case client_hello:  Empty;
        case server_hello:  WrappedSessionSecrets visibility_data;
    };
} TLSVisibilityExtension;

struct {
    opaque early_secret<1..255>;
    opaque hs_secret<1..
} SessionSecrets;

struct {
    opaque fingerprint<20>;
    opaque key_exchange<1..2^16-1
    opaque wrapped_secrets<1..2
} WrappedSessionSecrets;
```

**The fingerprint of SSWrapDH1**

**The ephemeral public key generated by the server on the same curve as SSWrapDH1, called SSWrapDH2**

**The encrypted session secrets**

# Session Secret Encryption

- Server uses SSWrapDH1 public key and SSWrapDH2 private key to compute a shared secret, called Z
- Other parties compute Z from SSWrapDH1 private key and SSWrapDH2 public key (from the TLS Visibility Extension)
- Session secrets are encrypted with Ke (and nonce if the AEAD needs one):

```
PRK = HKDF-Extract(0x00, Z)
Ke = HKDF-Expand(PRK, "tls_vis_key", AEAD_key_size)
nonce = HKDF-Expand(PRK, "tls_vis_nonce", AEAD_nonce_size)
```

# Questions?