LURK/TLS

draft-mglt-lurk-tls.txt

D. Migault

01/06/2016- Interim Meeting- (Virtual)

Scope

- Defines exchanges between Edge Serve and Key Server
- Authentication limited to RSA and ECDHE_*
- TLS1.2

LURK Design

- Query - Response Protocol
- LURKTLS messages
  - LURKTLS Header
  - LURKTLS Payload
- LURKTLS Header (Query)
  - QueryBit
  - LURKTLSVersion
  - LURKTLSQueryType
  - ID
- LURK Header (Response)
  - LURKTLS Header (Query)
  - LURKTLSStatus

Design

The LURKTLSQuery Type considered are:

- ping,
- capabilities
- rsa_master
- rsa_extended_master
- echde
- pfs_non_predictable_ecdhe

RSA Master Secret

```
TLS Client                    Edge Server          Key Server

ClientHello
---->                     ServerHello
                              <----

ClientKeyExchange
    EncryptedPremasterSecret

Finished
---->
                          LURKTLSMasterRSAInputPayload
                              key_id, master_prf
                              client_random, edge_server_random
                              client_version, edge_server_version
                              EncryptedPremasterSecret
                          ---->

                                                  1. Computing Master Secret
                                                  master_secret = master_prf(\
                                                  pre_master_secret + "master secret" +\
                                                  client_random + edge_server_random)[0..47];

                                                LURKTLSMasterPayload
                                                    master
                          Finished              <----
                          <----
```

RSA Master Secret

The TLS Client can perform a chosen clear text attack

- Similar to standard TLS
- RSA MUST be resistant to clear text attack.

The Edge Server MUST NOT be able to perform a chosen ciphertext attack

- master_secret generation MUST be performed deterministically
- master_secret generation MUST NOT return error

master_prf prevents the accumulation of (cleartext / ciphertext) both by the Edge Server and passive attackers listening to the Key Server:

- master_prf MUST be non-inversible - especially considering the Edge Server can chose randoms
- bindings are known by the TLS Client

If the private key is disclosed and communications between the TLS Client and Edge Server recorded, the master secret can be computed.

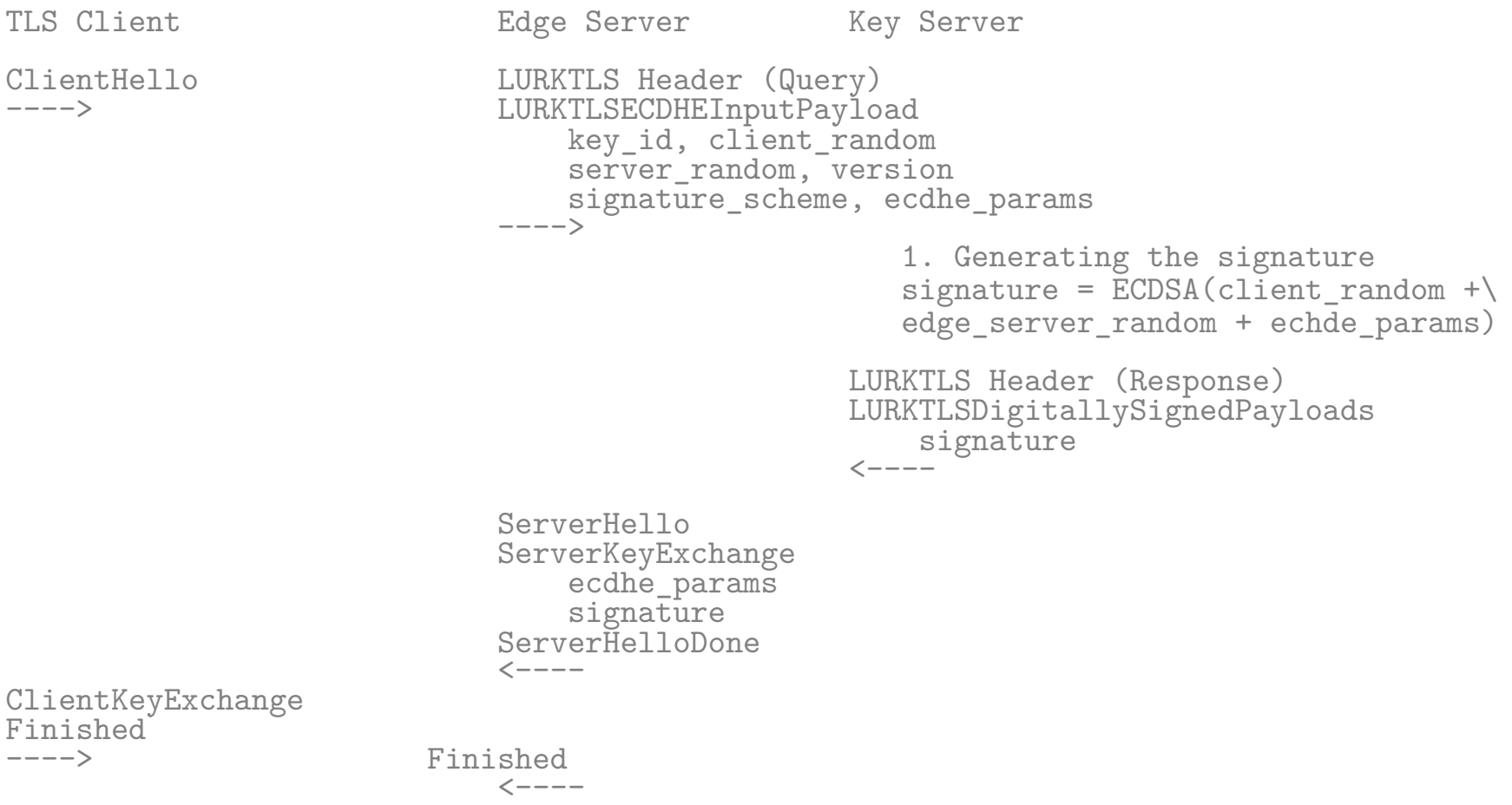
RSA Extended Master Secret

```
TLS Client                    Edge Server          Key Server

ClientHello
---->                     ServerHello
                              <----
ClientKeyExchange
    EncryptedPremasterSecret
Finished
---->
                              LURKTLS Header (Query)
                              LURKTLSExtendedMasterRSAInputPayload
                                  key_id, master_prf, session_prf
                                  client_version, edge_server_version
                                  edge_server_version
                                  EncryptedPreMasterSecret
                                  session_hash
                              ---->
                                                      1. Computing Master Secret
                                                      master_secret = master_prf(
                                                      pre_master_secret + "extended master secret"\
                                                      + session_hash)[0..47]

                                                     LURKTLS Header (Response)
                                                     LURKTLSMasterPayload
                                                         master
                                                     <----
                             Finished
                              <----
```

RSA Extended Master Secret

- Same security considerations apply
- Edge Server controls len(session_hash) instead of 2*len(random)

ECDHE Signature



ECDHE Signature

The Key Server MUST NOT be an open signing oracle:

- hash function must be collision resistant
- ecdhe_params have specific structures
- randoms, ecdhe_params have specific sizes

The Edge Server is able to chose the first 64 bits vs the first 32 bits for a TLS Client

- Edge Servers are expected to be authenticated unlike TLS Clients
- Without LURK a compromises Edge Servers is likely to leak the key

Given the randoms, and echde_params, the Key Server can generate the signature.

ECDHE Unpredictable Signature

```
TLS Client                  Edge Server          Key Server

ClientHello                 LURKTLS Header (Query)
---->                       LURKTLSECDHEInputPayload
                                key_id, client_random
                                edge_server_random
                                version, signature_scheme
                                ecdhe_params, prf
                            ---->
                                                  1. Generates a random nonce
                                                  2. Compute modified edge_server_random
                                                  modified_server_random = prf(padding\
                                                  + context + zero-byte \
                                                  + edge_server_random + once)[32]
                                                  3. Generate the signature
                                                  signature = ECDSA(client_random\
                                                  + modified_server_random + echde_params)

                                                LURKTLS Header (Response)
                                                LURKTLSDigitallySignedPayloads
                                                    signature
                                                    mofified_edge_server_random
                                                <----

                            ServerHello
                                Random modified_edge_server_random
                            ServerKeyExchange
                                ecdhe_params, signature
                            <----
ClientKeyExchange
Finished
---->                   Finished
                            <----
```

ECDHE Unpredictable Signature

The unpredictable signature provides

- More resistance regarding the signing oracle
- Prevents the Key Server to be requested regarding the data exchanged between the Edge Server and the TLS Client.

The Key Server does not sign the input content

- Signed content is hardly predictable

Questions:

- Reference on the vulnerabilities provides by controlling the first 32 / 64 bytes.
- Opinion on using TLS1.3 approach for signing vs SHA-256.

Questions

- Format: Binary format - TLS like.
- What checks can be done on ECDHEParams by the Key Server ?
- Capabilities restricted to supported query type ?
  - Should we have a more complex structure to have - for ex. hash function, ECParameters, EC Points supported per query type ?
- Is using TLS1.3 signature scheme
- Extending for TLS1.3 ?

Thank you for your attention