A Remote Key Implementation

Phillip Hallam-Baker

Use Cases Considered:

- Mine
- Other people’s

Mine

- People are careless with private keys
  - Especially on Linux and OSX
    - Saved unencrypted to file by default
    - Leak through backups, hard drives lost, etc.
  - Windows offers obfuscation, binding to TPM
    - These get in the way of business processes

- Use Scenarios
  - Software signing
  - TLS Server
  - HSM Interface

CDN Example

- Special case of TLS Server?

- Is the real solution short lived certs (1,3 day) + ACME?

Approach

- LURK is natural compliment to ACME
  - Building a toolbox of JSON security services
  - JSON should be the default encoding
    - Current spec -01 supports JSON-B, JSON-C, TLS, CBOR

- Use fingerprints to identify public keys
  - Uniform Data Fingerprint
    - Base32 (Version + SHA-x-512 (<content-type> “:” SHA-x-512 (<PKIX-keyinfo>)))

Step 0: Hello Transaction (optional)

- Specify
  - LURK protocol version
  - Service instances
    - Location (URI)
    - Protocol bindings (HTTP)
    - Instance version(s)
    - Encoding options
      - JSON, JSON-B, JSON-C, CBOR, TLS-Schema
    - Features supported
    - Authentication mechanisms

Request

```
POST /.well-known/acme/HTTP/1.1
Host: example.com
Content-Length: 23

{
  "HelloRequest": {}}
```

Response

```
HTTP/1.1 200 OK
Date: Tue 22 Mar 2016 02:02:33
Content-Length: 403

 {
   "HelloResponse": {
     "Status": 200,
     "StatusDescription": "OK",
     "Version": {
       "Major": 0,
       "Minor": 1,
       "Encodings": [{
           "ID": "application/json"},
         {
           "ID": "application/tls-schema"}]}}}
```

Step 1: Create (or import) a key pair

- Specify
  - Algorithm
  - Cryptographic features to support
  - Restrictions on operations
    - E.g. Only do TLS key exchange, Only TLS/1.2, Only specific TLS/1.2
    - Restrictions specified by constrained vocabulary of IANA registered terms

- Allow for advanced crypto
  - Co-operative key generation (ECDH, Hallam-Baker/Stradling ‘15)
  - Key splitting (RSA, ECDH, TBS)

Step 2: Request a key use

- Require
  - Authentication, [Encryption] of request
  - Authentication, Encryption of response
- Achieved via
  - TLS security (duh)
  - JOSE encryption of HTTP payload
    - [Different approach to ACME, mine is better]

Step 3: Dispose of Key

- When no longer needed.

Current Status

- Developed and documented specification and implementation
  - 3 Elapsed days

- Reference implementation is in C# [Github, MIT License]
  - Can be retargeted to C as additional

- Current restrictions
  - Only RSA + DH (need CFRG implementations)
  - Only JSON (Compiler supports TLSSchema, XML, ASN1, JSON-B, JSON-C)
  - Keys deleted when service shuts down
  - Missing HTTP payload auth and encryption