TEEP Protocol

draft-tschofenig-teep-protocol-01

Hannes Tschofenig, Ming Pei, Dave Wheeler, Dave Thaler

History

- draft-tschofenig-teep-protocol-00 created to reflect WG decisions to offer COSE/JOSE encoding, re-use EAT for attestation (RATS WG), and SUIT manifest (SUIT WG), and to align with TEEP architecture draft.
  - Reduces need to re-design already existing functionality
  - Lowers protocol complexity
  - Allows code re-use
- draft-tschofenig-teep-protocol-01 adds
  - Dave Thaler as co-author
  - Fixes several editorial bugs

Protocol Abbreviation

- Is TEEP-P good or should we just use TEEP Protocol ?
- Is calling it “teepee” as a nickname OK?

JSON/COSE Messaging

- Structure

type : 1 // QueryRequest
token: "a2"
request information:
    1: attestation
    2: trusted_apps
    3: extensions
        4: suit commands
Ciphersuite : [0,1]

- QueryRequest (JSON)

{"1": 1, "2": "a2", "3": [1, 2, 3], "3“:[1,2]}
Or
{
    "type" : 1,
    "token" : "a2",
    "request" : [1,2,3],
    "cipher_suite" : [0,1]
}

Ciphersuite

A ciphersuite consists of an AEAD algorithm, a HMAC algorithm, and a signature algorithm. Each ciphersuite is identified with an integer value, which corresponds to an IANA registered ciphersuite. This document specifies two ciphersuites.

- What algorithms should be in the list?
- What should be mandatory to implement?

(Details about the algorithm combination depends on other factors.)

Value	Ciphersuite
0	AES-CCM-16-64-128, HMAC 256/256, X25519, EdDSA
1	AES-CCM-16-64-128, HMAC 256/256, P-256, ES256

Message Protection: Encryption?

Message	Content
QueryRequest	Request field, ciphersuite list, nonce, version list, OCSP data
QueryResponse	EAT*, TA list, supported extensions, selected ciphersuite + version
TrustedAppInstall	Manifest** (or list of manifests)
TrustedAppDelete	TA List
Success	Success Code
Failure	Error code,

*: EAT token is signed by low level software on device.
**: The manifest is e2e protected and signed by author. The manifest may have an encrypted content attached or may reference encrypted content.

Message Protection: Symmetric Key?

Message	Signed or MACed
QueryRequest	Signed by TAM
QueryResponse	Signed with device key
TrustedAppInstall	MACed
TrustedAppDelete	MACed
Success	MACed
Failure	MACed

Where could the symmetric key come from? Key transport - as defined in OTrP

But is it useful? Symmetric keys are typically used for performance improvements. Not many messages are exchanged to begin with

Certificates and Chains

- Certificate and certificate chain (up to but excluding trust anchor) is communicated as part of COSE/JOSE header.
- At least needed for QueryRequest, which conveys the TAM certificate/certificate chain to the TEEP Agent)
- Not always needed (assuming the TEEP Agent caches data).

- Assumption is that there is no specific work that needs to be done at the TEEP Protocol level.

SUIT Commands

The SUIT manifest supports optional parameters and optional commands.

Commands:

- Set Current Dependency (setd)
- Set Parameters (setp)
- Process Dependency (pdep)
- Run (run)
- Fetch (getc)
- Use Before (ubf)
- Check Component Offset (cco)
- Check Device Identifier (cdid)
- Check Image Not Match (nimg)
- Check Minimum Battery (minb)
- Check Update Authorised (auth)
- Check Version (cver)
- Abort (abrt)
- Try Each (try)
- Copy (copy)
- Swap (swap)
- Wait For Event (wfe)
- Run Sequence (srun) mandatory component set
- Run with Arguments (arun)

Parameters:

- Image Size
- URI
- Strict Order
- Soft Failure
- Device ID
- Encryption Info
- Unpack Info
- Source Component
- URI List

Additionally, compression algorithms.

The SUIT manifest already assigns numerical values to all the parameters (Section 7.6) & commands (=conditions and directives).

Next Steps

- Update draft based on discussion
- Add examples in JSON (based on Hackathon feedback) and COSE
- Prototyping
- Should we plan a virtual interim meeting/conference call?