FDO Support for Multi-Tenancy

Final Document, July 10, 2023

This version:
https://fidoalliance.org/specs/fidoiot/fdo-appnote-5-tenant-id-v1.0-fd-20230710.html

Issue Tracking:
GitHub

Editor:
Geoffrey Cooper (Intel)

Version:
1.0

Abstract

Multi-tenancy is common for device managers. Although FDO does not support multi-tenancy directly in the protocol, it can accommodate a multi-tenancy ID using OVEExtra. This note discusses how this can be done.

Table of Contents

1. Introduction§

A multi-tenant service for FDO is a single FDO Owner that onboards multiple FDO clients. In this case, all the Ownership Vouchers for each tenant are sent to this single FDO Owner. As clients are onboarded using FDO, control of the clients is passed to the tenant. For example, a tenant-specific credential may be onboarded to the client so that it can communicate with a tenant-based server. How to hand off to a tenant is specific to the particular multi-tenant service and may involve special keys or identifiers stored in databases on the service side.

For purposes of the FDO protocol, we generalize the tenant information that is passed in FDO into a tenant identifier (TenantID), associated with the Device. We generalize the supply-chain entity that assigns the TenantID as the Tenant Assigner. Other aspects of multi-tenancy are outside the scope of this document.

The TenantID can be stored inside the Device, such as in the file system. If the TenantID is known when the Device is manufactured, the TenantID may be included as a final step of manufacturing. In the common case that the TenantID is not yet known at manufacturing, the device is boxed without a TenantID. When the TenantID becomes known, the Device may be unboxed and powered on so the TenantID can be stored.

Unboxing and re-boxing the device consumes time and effort, so we desire to encode the TenantID in the FDO Ownership Voucher, which is already passed through the supply chain and modified without requiring the Device

to be unboxed. The TenantID is added to the Ownership Voucher using the OVEExtra entry at the next Ownership Voucher extension after the TenantID becomes known. If necessary, the Ownership Voucher may be extended without a change of key, just to add the TenantID.

As a side note, an alternative technique is to encode the TenantID into an FDO Owner key pair for each tenant. Then the Ownership Voucher is extended to the key pair that serves to select the tenant. While this is a workable technique, we feel that the OVEExtra mechanism gives a more flexible result.

2. Definitions§

- **Manufacturer**: Same meaning as for FDO
- **Owner**: Same meaning as for FDO, except that the Owner is also implementing multi-tenancy.
- **Tenant Assigner**: An entity in the supply chain that learns the TenantID for a given Device, and applies it to the Ownership Voucher for that Device.
- **Tenant**: A client of the Owner who has its own identity and service parameters
- **TenantID**: an identifier used to identify the tenant in the Ownership Voucher
- **TenantID Key**: an encryption key known to the Tenant Assigner and the Owner, used to encrypt the TenantID.

3. Encoding the Tenant ID in the Ownership Voucher§

This document provides a standard place to encode a plaintext or encrypted TenantID. The details of the arrangement must be worked out between the Tenant Assigner and the Owner. However, we provide some default parameters to make a standard implementation easier.

The following CDDL definitions are used by the Tenant Assigner to encode the TenantID into the OVEExtra field of the Ownership Voucher:

```
OVEExtraTypeTenantID = 3
OVEExtraInfoTypeTenantid = bstr .cbor OVETenantID
OVETenantID = [
    TIDTenantID: bstr
    TIDSalt: bstr
    TIDKeyIndex: uint
]
```

The TenantID is encoded as a byte string. If another type is desired, the Owner and Tenant Assigner can agree on an encoding of this value into a byte string (e.g., an integer type is encoded byte-wise, a string type is encoded in CBOR, and so on).

If OVETenantID.TIDKeyIndex = 0, the TenantID is not encrypted. In this case, TIDSalt MUST be an empty byte string.

When OVETenantID.TIDKeyIndex != 0, the Owner and Tenant Assigner must agree in advance on a sequence of keys with which to encrypt TenantIDs and encode them into a bstr. Then OVETenantID.TIDKeyIndex gives the index of the key used, and TIDSalt gives salt for the encryption (see below).

Any mutually-agreed encryption mechanism is permitted for TenantID, including no encryption at all. We specify a default encryption mechanism to increase interoperability, below. Note that the Ownership Voucher already preserves data integrity to all elements, so an encryption mechanism alone may be used.

3.1. Tenant Key Sharing (default mechanism)§

- One or more AES-256 keys are shared between Tenant Assigner and Owner. The keys are arranged in a sequence.
- Key in use starts at entry 1 in the sequence, and the entry number is incremented when a change of key is desired, such as on a mutually agreed use count, date interval or time interval. The sequence may have new keys added to it over time.
- TenantID values are encrypted/decrypted if OVETenantID.TIDKeyIndex != 0.

The number of keys to be shared, when to switch keys, and the key sharing/distribution mechanism are out-of-scope for this document. The encryption algorithm selected is AES-256, chosen to meet NIST recommendations for cryptographic strength at the year 2030. The required cryptographic strength for any given deployment is out-of-scope for this document.

3.2. Encryption / Decryption Steps§

These steps apply only if OVETenantID.TIDKeyIndex != 0.

To encrypt OVETenantID.TIDTenantID, first select the shared key based on OVETenantID.TIDKeyIndex. Then:

- TIDSalt is initialized with cryptographically random numbers.
- Plaintext is salted as: TIDTenantID || TIDSalt || TIDKeyIndex || 0xff
- Plaintext is padded with integer-valued bytes so as to be a multiple of 128 bits. For example, if 15 padding values are needed, these are 1,2,3,...,15; if 7 padding values are needed, these are 1,2,3,4,5,6,7; if no zero padding values are needed (i.e., the plaintext was already a multiple of 128 bits) then no padding bytes are added. The 0xff byte at the end of the plaintext makes detecting this latter condition easier.
- Each 128-bit block is encrypted using AES-256 with a key indexed by OVETenantID.TIDKeyIndex.
- Successive AES encryption blocks are placed in the TIDTenantID bstr.

To decrypt OVETenantID.TIDTenantID, select the shared key based on OVETenantID.TIDKeyIndex. Then:

- OVETenantID.TIDTenantID is decrypted using the selected key to form the plaintext
- If the last byte is in the range [1,15], strip that many bytes off the end of the plaintext. The last byte indicates how many bytes to strip.
- Strip off the last byte, which contains 0xff.
- Strip off TIDKeyIndex, verifying that it matches the TIDKeyIndex used to select the shared key.
- Strip TIDSalt bytes off the plaintext, matching them from the end of the salt backwards.

If the bytes in the range [1,15] are not in sequence or the 0xff byte contains another value, or the TIDTenantID does not match, or the decrypted salt does not match, an error is declared.

Otherwise, the remaining plaintext bytes are assigned to TIDTenantID.

4. Example§

- A small but growing website has 100+ tenants.
- The manufacturer is the Tenant Assigner. Orders for each machine include a tenant ID expressed as a positive 32-bit integer.
- The website sends 24 AES-256 keys to the manufacturer to start the process, and an additional 12 keys every December. The next key in the sequence is used on the 1st of each month. Keys are transmitted in a PGP encrypted file. PGP keys are set up in person.
- Orders include the TenantID, expressed as a number. All ownership vouchers are extended to the website key and transmitted via email. The TenantID is placed in this extension of the Ownership Voucher, using the

OVETenantID structure. The TIDTenantID is encrypted as described above. TIDKeyIndex contains the index of the key used, equivalent to the number of months since the service started, plus one.

- When Ownership Vouchers are received, the TenantID is looked up and decrypted using the key corresponding to TIDKeyIndex and the procedure above. The Ownership Voucher is stored in a database. A table in the database maintains the relationship between the TenantID and the GUID from the Ownership Voucher.

  When a Device is onboarded, its TenantID is looked up. The scripting for the onboarding is selected for the appropriate tenant. After onboarding, the device is assigned to the tenant, and the Ownership Voucher and related tables are deleted from the database.

↥
→