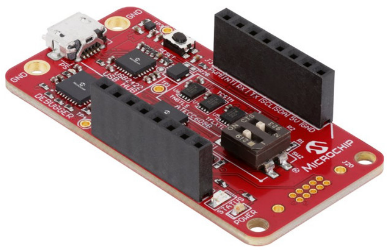


MICROCHIP Trust Platform Manifest File Format User Guide

Home » MICROCHIP » MICROCHIP Trust Platform Manifest File Format User Guide

Trust Platform Manifest File Format
User Guide

Contents

Overview

The manifest file format is designed to convey the unique information about a group of secure subsystems, including unique ID (e.g., serial number), public keys and certificates. This was primarily developed for Crpy to Authentication™ (currently ATECC508A, ATECC608A and ATECC608B) secure elements. However, it is structured to work for other secure subsystems as well.

Manifest files provide a way to link an actual Microchip Trust security device to the infrastructure environment that it needs to connect to. These files are a critical aspect of the Microchip Trust&GO, Trust FLEX and, optionally, Trust CUSTOM development environments. Whether you connect to an IoT cloud, a LoRaWAN® network or, potentially, any other infrastructure or environment, the manifest file uniquely ties a given device to that environment.

When working with Microchip Trustor, Trust FLEX or Trust CUSTOM products, a manifest file will be generated for a group of devices that are provisioned through the Microchip Just-In-Time provisioning services. Each object entry in the manifest file is known as a signed secure element and is signed by a Microchip Elliptic Curve Cryptography (ECC) private key to validate its authenticity. The overall manifest is made of multiple signed secure elements.

Specific information associated with the manufacturer, the secure product device and specific individual device information are all part of the information associated with a given signed secure element.

The manifest file is available in a secure fashion only to the customer that orders the group of devices. Accessing these manifest files is part of the development and provisioning flow provided through Microchip. Once provisioning is completed for a group of products, the manifest file is available for download.

Manifest Generation

The manifest of the Trust FLEX and Trust&GO devices can be generated in two scenarios. One is through the Microchip Just-In-Time provisioning services (Microchip-generated) and the second one is a custom generation using the scripts provided (self-generated).

In both cases, the Trust&GO, Trust FLEX and Trust CUSTOM devices will have different information due to differences in their configuration.

The following sections provide manifest file differences between:

1. Microchip and self-generated files
   – Manifest signature
2. Trust&GO and Trust FLEX files
3. Prototype and production device files

1.1 Microchip vs. Self-Generated Files

The manifest file format and generation procedures are public information; hence, they can be generated by users.

Due to this nature and when the procedures are followed, there will still be minor differences between Microchip and self-generated files.

Manifest Signature

In the manifest file, each element is signed to ensure the integrity of the content. For a Microchip-generated manifest file, the signing operation is performed by Microchip using its Certificate Authority (CA). The corresponding CA certificate can be downloaded from the Microchip website. This certificate can be used to validate the authenticity of the Microchip-generated files.

Tip:

- MCHP Manifest Signer Certificates (under Documentation tab)
- Direct link to Download

For a self-generated manifest file, it is not possible to get each element signed by Microchip CA, as users do not have access to a CA private key. It is required to generate/use a local CA to perform the signature operations. In this case, the users must share the validation certificate along with the manifest file to others. This enables them to validate the content before using it further.

The other differences include:

1. Trust&GO – Content remains the same, as the device data are immutable, but signature and verification

certificates are different, as self-generated scripts use their own CA.

2. Trust FLEX

   a. Device and signer certificates can be different if custom PKI is selected during resource generation.

   b. Slots 1-4, 13-15 vary based on additional key generations as part of resource generation at the user's location.

   c. Signature and verification certificates are different, as self-generated scripts use their own CA.

1.2 Trust&GO vs. Trust FLEX vs. Trust CUSTOM Files

The manifest files only contain public information of the device, such as its serial number, certificates and slots' public information. Depending on the configuration differences, the information in Trust&GO, Trust FLEX and Trust CUSTOM files varies as follows:

Trust&GO	Trust FLEX	Trust CUSTOM
• Slot 0 public key information (immutable) • Device and signer certificates signed by Microchip CA (immutable)	• Slot 0 public key information (immutable) • Device and signer certificates signed by Microchip or customer CA based on custom PKI selection • Slot 1-4 public key information • Slot 13-15 public key information	• Custom information due to unique configuration

Certificate Slots in Trust FLEX Devices

When the user opts to create a custom certificate chain on the Trust FLEX device, the factory provisioned certificates will be overwritten. Trust Platform Design Suite scripts/notebook allow the user to back up default certificates into a local folder before overwriting custom certificates on the device. However, if the board changes hands after provisioning, the new user will not have the back-up certificates and will not be able to revert to factory default.

1.3 Prototype vs. Production Device Files

Prototype devices are meant to be used in-house for R&D; therefore, these devices do not come with a manifest file generated in the factory. However, these devices will have the Slot 0 key generated along with the device and signer certificates generated during factory provisioning. It is required to self-generate the manifest files for prototype Trust&GO and Trust FLEX devices.

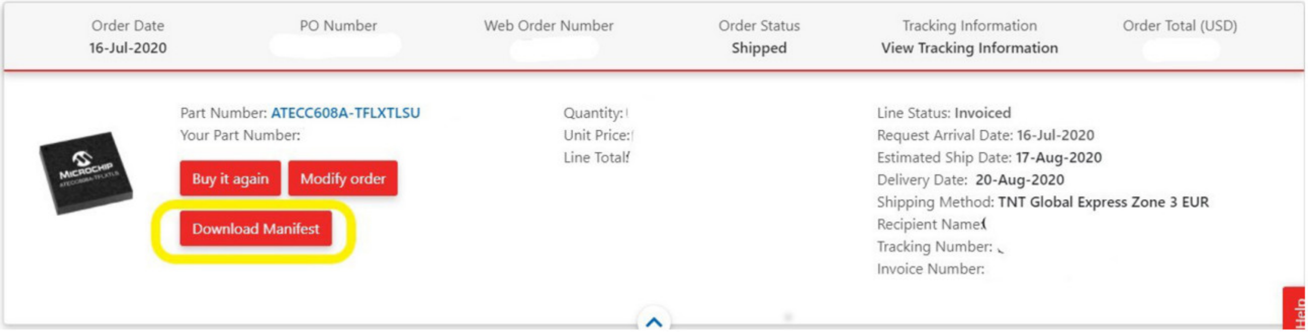
The Trust Platform Design Suite provides the required scripts/tools to self-generate the manifest files.

Tip:

- Trust&GO manifest generation scripts
- Trust FLEX manifest generation scripts (with dev key generation)

For production devices, users can always download the manifest file from the microchip DIRECT portal under their personal login. These files are available only after devices are provisioned and shipped to the customer.

Figure 1-1. Microchip DIRECT Manifest Portal



Order Date	PO Number	Web Order Number	Order Status	Tracking Information	Order Total (USD)
16-Jul-2020	[redacted]	[redacted]	Shipped	View Tracking Information	[redacted]

Part Number: ATECC608A-TFLXTLSU
Your Part Number:

Quantity:
Unit Price:
Line Total:

Line Status: Invoiced
Request Arrival Date: 16-Jul-2020
Estimated Ship Date: 17-Aug-2020
Delivery Date: 20-Aug-2020
Shipping Method: TNT Global Express Zone 3 EUR
Recipient Name:
Tracking Number:
Invoice Number:

Structure and Format of a Manifest File

2.1 Introduction

The base format is an array of JavaScript Object Notation (JSON) objects. Each object represents a single secure element and is signed to allow cryptographic verification of its origins. The format is intentionally "flattened" with common information repeated for each secure element. This is to facilitate parallel processing of manifests and to allow splitting of entries into smaller manifests, where appropriate.

This format makes use of the JavaScript Object Signing and Encryption (JOSE) set of standards to represent keys (JSON Web Key – JWK), certificates (x5c member in a JWK) and provide signing (JSON Web Signature – JWS). In the object definitions, member values may be the name of another JSON object or just an example value.

2.2 Binary Encoding

JSON has no native binary data format, so a number of string encodings are used to represent binary data depending on context.

BASE64URL

This is a base64 encoding using a URL-safe alphabet, as described in RFC 4648 section 5, with the trailing padding characters ("=") stripped.

This is the encoding used by the JOSE standards and will be found in the JWS, JWK and JWE objects used. This is documented in RFC 7515 section 2.

This encoding is also used in a few other non-JOSE members to maintain consistency.

BASE64

This is the standard base64 encoding, as described in RFC 4648 section 4, and includes the trailing padding characters ("=").

This is used for encoding certificates (JOSE x5c members), presumably to more closely match the common PEM encoding that certificates are often found in.

HEX

In some cases, short binary values are expressed as lowercase hex strings. This is to match convention with how these values are typically seen and worked with.

2.3 Secure Element Manifest Object

At the top level, the secure element manifest format is a JSON array of Signed Secure Element objects where each element represents a single secure element.

```
[
SignedSecureElement ,
SignedSecureElement ,
…
]
```

2.4 Signed Secure Element Object

The Signed Secure Element object is a JWS (RFC 7515) object using the Flattened JSON Serialization Syntax (section 7.2.2).

```
{
"payload" : BASE64URL(UTF 8(SecureElement)) ,
"protected" : BASE64 URL(UTF8(SignedSecureElementProtectedHeader)),
"header" : {
"uniqueId" : "0123f1822c38dd7a01"
},
"signature" : BASE 64URL(JWS Signature)
}
```

RFC 7515 section 7.2.1 provides definitions for the encoding and contents of the JWS members being used in this object. Below are some quick summaries and additional details about these members and the specific features being used.

payload

An encoded SecureElement object, which is the primary content being signed. All information about the secure element is contained here.

protected

An encoded SignedSecureElementProtectedHeader object, which describes how to verify the signature.

header

JWS unprotected header. This object contains the unique ID member repeated from the SecureElement object in the payload. The unprotected header is not part of the signed data in the JWS; therefore, it does not need to be encoded and is included to facilitate plain-text searches of the manifest without needing to decode the payload.

signature

The encoded JWS signature of the payload and protected members.

2.4.1 SignedSecureElementProtectedHeader Object

The SignedSecureElementProtectedHeader object is a JWS protected header that describes how to verify the signature. While RFC 7515 section 4.1 lists out the available header members for a JWS, only the ones listed here will be used.

```
{
“alg”: “ES256”,
“kid”: BASE64URL(Subject Key Identfier) ,
“x5t#S256” : BASE64 URL(SHA-256 Certificate Thumbprint)
}
```

alg

Describes the key type used to sign the payload. See RFC 7518 section 3.1. Only public key algorithms will be used.

kid

Encoded Subject Key Identifier (RFC 5280 section 4.2.1.2) of the key used to sign the payload. This is the BASE64URL encoding of the subject key identifier value, not the full extension. Used to help identify the key for verification. kid is a free-form field in the JWS standard (see RFC 7515 section 4.1.4), so this definition applies only to the SignedSecureElement object.

x5t#S256

SHA-256 thumbprint (a.k.a. fingerprint) of the certificate for the public key required to validate the signature. Like kid, it can also be used to help identify the key for verification. See RFC 7515 section 4.1.8.

2.5 SecureElement Object

The SecureElement object contains all the information about the secure element.

```
{
“version” : 1 ,
“model” : “ATECC608A” ,
“partNumber” : “ATECC608A-MAHDA-T” ,
“manufacturer” : EntityName ,
“provisioner” : EntityName ,
“distributer” : EntityName ,
“groupId” : “359SCE55NV38H3CB” ,
“provisioningTimestamp” : “2018-01-15T17:22:45.000Z” ,
“uniqueId” : “0123f1822c38dd7a01” ,
“publicKeySet” : {
“keys” : [ PublicJWK , … ]
},
“encryptedSecretKeySet” : {
“keys” : [ EncryptedSecretJWK , … ]
}
“modelInfo” : ModelInfo
}
```

version

SecureElement object version as an integer. The current version is 1. Subsequent versions will strive to maintain backwards compatibility with previous versions, where possible.

model

Name of the base secure element model. The current options are ATECC508A, ATECC608A and ATECC608B from the Crypto Authentication family.

partNumber

Complete part number of the provisioned secure element.

manufacturer

An EntityName object that identifies the manufacturer of the secure element.

provisioner

An EntityName object that identifies who performed the provisioning/programming of the secure element.

distributer

An EntityName object that identifies who distributed the provisioned secure elements.

In many cases, this will be the same entity that generates the manifest data being described here.

groupId

Secure elements may be organized into groups identified by a single ID. If the secure element is part of a group, this is the unique ID of that set. Group IDs should be globally unique.

provisioningTimestamp

Date and time the secure element was provisioned in UTC. Formatting is per RFC 3339.

uniqueId

Unique identifier for the secure element. For Crypto Authentication devices, this is the 9-byte device serial number as a lowercase hex string.

publicKeySet

An object representing all the public keys (and certificate chains, if available) corresponding to private keys held by the secure element. This object is a JSON Web Key Set (JWK Set) per RFC 7517 section 5, where keys are an array of Public JWK objects.

encryptedSecretKeySet

An object representing all the secret keys (symmetric keys) and data held by the secure element that are marked for export. The keys member is an array of EncryptedSecretJWK objects. Note that an encrypted JWK Set is not used so the metadata about the individual keys (number and key IDs) can be read without decrypting.

modelInfo

If additional non-cryptographic information about the secure element needs to be conveyed, this Modulino object may be present with model-specific information.

2.6 EntityName Object

The EntityName object is used to identify an entity responsible for some part of the secure element. The members in this object are variable and must be the same as the attributes defined in sections 6.4.1 Organization Name and 6.4.2 Organizational Unit Name of ITU-T X.509 (ISO/IEC 9594-6). While none of the members are required, there must be at least one.

```
{
“organizationName” : “Microchip Technology Inc” ,
“organizationalUnitName” : “Secure Products Group” ,
}
```

organizationName

Name of the entity organization (e.g., company name).

organizationalUnitName Optional name of a unit within the organization that the entity applies to specifically.

2.7 Public JWK Object

This object represents an asymmetric public key and any certificates associated with it. This is a JWK object, as defined by RFC 7517. Some JWK member specifications are repeated below for easy reference along with expectations for specific models of secure elements.

The following definition is for elliptic curve public keys, supported by the Crypto Authentication family of secure elements.

```
{
“kid” : “0” ,
“kty” : “EC” ,
“crv” : “P-256” ,
“x” : BASE64URL(X) ,
“y” : BASE64URL(Y) ,
“x5c” : [ BASE 64(cert) , … ]
}
```

The following JWK fields required for elliptic curve public keys are defined in RFC 7518 section 6.2.1:

kid

Key ID string. It uniquely identifies this key on the secure element. For Crypto Authentication secure elements, this will be the slot number of the corresponding private key.

kty

Key type. CryptoAuthentication secure elements only support EC public keys, as defined in RFC 7518 section 6.1.

crv

For elliptic curve keys, this is the curve name. CryptoAuthentication secure elements only support the P-256 curve, as defined in RFC 7518 section 6.2.1.1.

x

For elliptic curve keys, this is the encoded public key X integer, as defined in RFC 7518 section 6.2.1.2.

y

For elliptic curve keys, this is the encoded public key Y integer, as defined in RFC 7518 section 6.2.1.3.

x5c

If the public key has a certificate associated with it, that certificate will be found at the first position in this array. Subsequent certificates in the array will be the CA certificates used to validate the previous one. Certificates will be BASE64 encoded (not BASE64URL) strings of the DER certificate. This is defined in RFC 7517 section 4.7.

2.8 EncryptedSecretJWK Object

This object represents a secret key (symmetric key) or secret data in a secure element that is encrypted for the recipient of the manifest.

It is a JSON Web Encryption (JWE) object, as defined by RFC 7516. The JWE payload will be the JSON serialization (not compact serialization) of a JWK object, as defined by RFC 7517, with a key type of octet (“kty”:”oct”). See RFC 7518 section 6.4 for details on the symmetric key JWK. This technique is described in RFC 7517 section 7.

2.9 ModelInfo Object

This object holds additional model-specific information about a secure element that is not captured by the other cryptographic members. It has no specific members, but depends on the model of the secure element.

Currently, only the CryptoAuthentication models (ATECC508A and ATECC608A) have a ModelInfo object defined.

2.9.1 CryptoAuthentication ModelInfo Object

ModelInfo members defined for CryptoAuthentication models (ATECC508A or ATECC608A):

```
{
“deviceRevision” : “00006002” ,
“publicData” : [ CryptoAuthPublicDataElement , … ]
}
```

deviceRevision

The 4-byte device revision number as returned by the Info (Mode = 0x00) command. Encoded as a lowercase hex string.

publicData

An array of CryptoAuthPublicDataElement objects that defines a location and the public data at that location.

2.9.1.1 CryptoAuthPublicDataElement Object

This object defines the location and contents of a public data element in CryptoAuthentication secure elements.

```
{
“zone” : “data” ,
“slot” : 14 ,
“offset” : 0 ,
“data” : BASE64URL(data)
}
```

zone

CryptoAuthentication zone where the data are found. The options are “data” for one of the slots, “otp” for the OTP zone or “config” for the configuration zone.

slot

If the zone is “data”, this is the slot index (0-15) where the data can be found.

offset

Byte offset into the zone/slot that the data can be found at.

data

Actual data at the location specified by the other members. This data will be BASE64URL encoded (with padding characters (“=”) stripped).

Manifest File Example and Decoding

The following subsections provide examples of a manifest file entry, manifest CA certificate and a Python code example that can be used to decode the manifest file. These files can be downloaded from the Microchip website at Manifest Example Files. The content of the download file is shown below.

Manifest Files Example

ExampleManifest.json	A single element manifest file in json format.
ExampleManifestMCHP_CA.crt	An example of a manufacturing CA certificate produced by Microchip.
ExampleManifestDecode.py	A Python script that will read the example Manifest json file and decode it into its respective elements.

3.1 Manifest Example

This is an example of a Secure Element Manifest object with a single SignedSecureElement entry:

[
{
“payload” :
“eyJ2ZXJzaW9uIjoxLCJtb2RlbCI6IkFURUNDNjA4QSIsInBhcnROdW1iZXIiOiJBVEVDQzYwOEEtTUFIMjIiLCJtYW5
1Z
mFjdHVyZXIiOnsib3JnYW5pemF0aW9uTmFtZSI6Ik1pY3JvY2hpcCBUZWNobm9sb2d5IEluYyIsIm9yZ2FuaXphdGlv
bmF
sVW5pdE5hbWUiOiJTZWN1cmUgUHJvZHVjdHMgR3JvdXAifSwicHJvdmlzaW9uZXIiOnsib3JnYW5pemF0aW9uTm
FtZSI6I
k1pY3JvY2hpcCBUZWNobm9sb2d5IEluYyIsIm9yZ2FuaXphdGlvbmFsVW5pdE5hbWUiOiJTZWN1cmUgUHJvZHVjd
HMgR3J
vdXAifSwiZGlzdHJpYnV0b3IiOnsib3JnYW5pemF0aW9uTmFtZSI6Ik1pY3JvY2hpcCBUZWNobm9sb2d5IEluYyIsIm9
yZ
2FuaXphdGlvbmFsVW5pdE5hbWUiOiJNaWNyb2NoaXAgRGlyZWN0In0sImdyb3VwSWQiOiIzNTlTQ0U1NU5WMzh
IM0NCIiw
icHJvdmlzaW9uaW5nVGltZXN0YW1wIjoiMjAxOS0wMS0yNFQxNjozNToyMy40NzNaIiwidW5pcXVlSWQiOiIwMTIz
ZjE4M
jJjMzhkZDdhMDEiLCJwdWJsaWNLZXlTZXQiOnsia2V5cyI6W3sia2lkIjoiMCIsImt0eSI6IkVDIiwiY3J2IjoiUC0yNTY
iLCJ4IjoieDhUUFFrN2g1T3ctY2IxNXAtVEU2SVJxSFFTRVRwUk5OYnU3bmwwRm93TSIsInkiOiJ1eDN1UDhBbG9V
bThRb
k5ueUZMNlIwS0taWXhGQ0I0VV9RTGdzdWhYb29zIiwieDVjIjpbIk1JSUI5VENDQVp1Z0F3SUJBZ0lRVkN1OGZzdk
FwM3l
kc25uU2FYd2dnVEFLQmdncWhrak9QUVFEQWpCUE1TRXdId1lEVlFRS0RCaE5hV055YjJOb2FYQWdWR1ZqYUc
1dmJHOW5lU
0JKYm1NeEtqQW9CZ05WQkFNTUlVTnlIWEIwYnlCQmRYUm9aVzUwYVdOaGRHbHZiaUJUYVdkdVpYSWdSall3
TURBZ0Z3MHh
PVEF4TWpReE5qQXdNREJhR0E4eU1EUTNNREV5TkRFMk1EQXdNRm93UmpFaE1COEdBMVVFQ2d3WVRXbG
pjbTlqYUdsd0lGU
mxZMmh1YjJ4dloza2dTVzVqTVNFd0h3WURWUVFEREJnd01USXpSakU0TWpKRE16aEVSRGRCTURFZ1FWUkZ
RME13V1RBVEJ
nY3Foa2pPUFFJQkJnZ3Foa2pPUFFNQkJ3TkNBQVRIeE05Q1R1SGs3RDV4dlhtbjVNVG9oR29kQklST2xFMDF1N3
VlWFFXa
kE3c2Q3ai9BSmFGSnZFSnpaOGhTK2tkQ2ltV01SUWlMVlAwQzRMTG9WNktMbzJBd1hqQU1CZ05WSFJNQkFm
OEVBakFBTUE
0R0ExVWREd0VCL3dRRUF3SURpREFkQmdOVkhRNEVGZ1FVcy9HcVpRNk1BYjd6SC9yMVFvNThPY0VGdVpJd
0h3WURWUjBqQ
kJnd0ZvQVUrOXIxRW9yNndiV1NqODJyRWRzSlBzOU52dlI3Q2dZSUtvWkl6ajBFQXdJRFNBQXdSUUlnTkxxUeks1N
mI1VVI
FSGU5WXdxSXM2dVRhbm14Mk9yQjZoL1FZRHNJT1dzTUNJUUNMMURzbHhnVXU4OHhveXlnTVNnTDlYOGxjS
DVCejlSQURKY
W1JZi91UUtnPT0iLCJNSUlDQlRDQ0FhcWdBd0lCQWdJUWVRcW4xWDF6M09sdFpkdG1pM2F5WGpBS0JnZ3Fo
a2pPUFFRREF
qQlBNU0V3SHdZRFZRUUtEQmhOYVdOeWIyTm9hWEFnVkdWamFHNXZiRzluZVNCSmJtTXhLakFvQmdOVkJBT
U1JVU55ZVhCM
GJ5QkJkWFJvWlc1MGFXTmhkR2x2YmlCU2IyOTBJRU5CSURBd01qQWdGdzB4T0RFeU1UUXhPVEF3TURCYUd
BOHlNRFE1TVR

JeE5ERTVNREF3TUZvd1R6RWhNQjhHQTFVRUNnd1lUV2xqY205amFHbHdJRlJsWTJodWIyeHZaM2tnU1c1ak1T
b3dLQVlEV
lFRRERDRkRjbmx3ZEc4Z1FYVjBhR1Z1ZEdsallYUnBiMjRnVTJsbmJtVnlJRVkyTURBd1dUQVRCZ2NxaGtqT1BRS
UJCZ2d
xaGtqT1BRTUJCd05DQUFSMlIwRndzbVBubVZTOGhic1M2ZjV3REZ1TjFOYVRSWmpDS2Fkb0FnNU9DMjFJZGR
EdG9lNzJYN
UZmeHJFV1JzV2h5bU1mWWxWb2RFZHB4ZDZEdFlscW8yWXdaREFPQmdOVkhROEJBZjhFQkFNQ0FZWXdFZ1
lEVlIwVEFRSC9
CQWd3QmdFQi93SUJBREFkQmdOVkhRNEVGZ1FVKzl5cUVvcjZ3YldTajgyckVkc0pQczlOdnZZd0h3WURWUjBqQ
kJnd0ZvQ
VVldTE5YmNhM2VKMnlPQUdsNkVxTXNLUU9Lb3d3Q2dZSUtvWkl6ajBFQXdJRFNRQXdSZ0loQU1Zd01IbXBpekJ
PYUg0R3h
UbDVLc1Y2WEFGTk1CZmUzTko5MVIzTmhqZi9BaUVBeHFJc2JyR3VYNFdSU2N0ZDUzZUxvL01MNlQyYmdHK1V
2ejJRcFlSN
Flkdz0iXX0seyJraWQiOiIxIiwia3R5IjoiRUMiLCJjcnYiOiJQLTI1NiIsIngiOiIyT2huZTl2MGFUU0NkclpObVh2dE9
XaXI1RVRnUmhudmVjSkRYUEh6RnBnIiwieSI6ImhjUDkxQ01UQUt2amR6Nl9pTldPNDZnNXVQalJ2Smt1dVFfNlRI
Y2tGL
UEifSx7ImtpZCI6IjIiLCJrdHkiOiJFQyIsImNydiI6IlAtMjU2IiwieCI6IkVFRXhpUmYwVEJYd1BrTGloSlZSdGVTWTN
oVS1JR1RMbFVPLUZSTUpaRmciLCJ5IjoiTnVib2F3NFdfYTNLd2kwbFZlRzlwNGg0Mkk0bTd2bUs1UDQ5U1BlYkZ2
TSJ9L
Hsia2lkIjoiMyIsImt0eSI6IkVDIiwiY3J2IjoiUC0yNTYiLCJ4IjoiaktCOERrY2k1RXhSemcwcXREZEFqcFJJSFNoeFl
PTjgyWVoyLWhhamVuWSIsInkiOiJOWU1KOUR0YkN0Nk9wbmoyZzQzQWhrMnB4UXU5S1JkTXkzbTBmLUpfclJFI
n0seyJra
WQiOiI0Iiwia3R5IjoiRUMiLCJjcnYiOiJQLTI1NiIsIngiOiJMVFUwSUdoM3ltQXpXbFdtWjg0ZmhYN1lrQjRaQ21tbFY
tWU9ORHREYURVIiwieSI6ImN2TnIyVEpEV1hmNFhPNlB6eWJSV29FY1FMVDRGM05WUDhZajltWDhxYncifV19f
Q” ,
“protected” :
“eyJ0eXAiOiJKV1QiLCJhbGciOiJFUzI1NiIsImtpZCI6IjdjQ0lMbEFPd1lvMS1QQ2hHdW95VUlTTUszZyIsIng1dCNTM
jU2IjoiVEVjNDZTVDJSREZfQU92QnRvQ1lhODM4VldJUGZOVl8yalRxTmE0ajVSNCJ9” ,
“header” : {
“uniqueId” : “0123f1822c38dd7a01”
},
“signature” :
“7btSLIbS3Yoc6yMckm7Moceis_PNsFbNJ6iktVKl86IuxZ6cU_yVZuLSgLCstMs4_EBFpvsyFy7Ij5rM9oMDw”
}
]

Decoding the protected member gives the following SignedSecureElementProtectedHeader:

{
“typ” : “JWT” ,
“alg” : “ES256” ,
“kid” : “7cCILlAOwYo1-PChGuoyUISMK3g” ,
“x5t#S256” : “TEc46ST2RDF_AOvBtoCYa838VWIPfNV_2jTqNa4j5R4”
}

Decoding the payload member gives the following SecureElement:

{
“version” : 1 ,
“model” : “ATECC608A” ,
“partNumber” : “ATECC608A-MAH22” ,
“manufacturer” : {
“organizationName” : “Microchip Technology Inc” ,
“organizationalUnitName” : “Secure Products Group”
} ,
“provisioner” : {
“organizationName” : “Microchip Technology Inc” ,
“organizationalUnitName” : “Secure Products Group”
} ,
“distributor” : {
“organizationName” : “Microchip Technology Inc” ,
“organizationalUnitName” : “Microchip Direct”

} ,
“groupId” : “359SCE55NV38H3CB” ,
“provisioningTimestamp” : “2019-01-24T16:35:23.473Z” ,
“uniqueId” : “0123f1822c38dd7a01” ,
“publicKeySet” : {
“keys”: [
{
“kid”: “0” ,
“kty”: “EC” ,
“crv”: “P-256” ,
“x”: “x8TPQk7h5Ow-cb15p-TE6IRqHQSETpRNNbu7nl0FowM” ,
“y”: “ux3uP8AloUm8QnNnyFL6R0KKZYxFCItU_QLgsuhXoos” ,
“x5c”: [
“MIIB9TCCAZugAwIBAgIQVCu8fsvAp3ydsnnSaXwggTAKBggqhkjOPQQDAjBPMSEwHwYDVQQKDBhNaWNyb2N
oaXAgVGVja
G5vbG9neSBJbmMxKjAoBgNVBAMMIUNyeXB0byBBdXRoZW50aWNhdGlvbiBTaWduZXIgRjYwMDAgFw0xOTAx
MjQxNjAwMDB
aGA8yMDQ3MDEyNDE2MDAwMFowRjEhMB8GA1UECgwYTWljcm9jaGlwIFRlY2hub2xvZ3kgSW5jMSEwHwYDV
QQDDBgwMTIzR
jE4MjJDMzhERDdBMDEgQVRFQ0MwWTATBgcqhkjOPQIBBggqhkjOPQMBBwNCAATHxM9CTuHk7D5xvXmn5M
TohGodBIROlE0
1u7ueXQWjA7sd7j/AJaFJvEJzZ8hS+kdCimWMRQiLVP0C4LLoV6KLo2AwXjAMBgNVHRMBAf8EAjAAMA4GA1Ud
DwEB/
wQEAwIDiDAdBgNVHQ4EFgQUs/GqZQ6MAb7zH/
r1Qo58OcEFuZIwHwYDVR0jBBgwFoAU+9yqEor6wbWSj82rEdsJPs9NvvYwCgYIKoZIzj0EAwIDSAAwRQIgNLTzK5
6b5UYE
He9YwqIs6uTanmx2OrB6h/QYDsIOWsMCIQCL1DslxgUu88xoyygMSgL9X8lcH5Bz9RADJamIf/uQKg==” ,
“MIICBTCCAaqgAwIBAgIQeQqn1X1z3OltZdtmi3ayXjAKBggqhkjOPQQDAjBPMSEwHwYDVQQKDBhNaWNyb2No
aXAgVGVja
G5vbG9neSBJbmMxKjAoBgNVBAMMIUNyeXB0byBBdXRoZW50aWNhdGlvbiBSb290IENBIDAwMjAgFw0xODEyM
TQxOTAwMDB
aGA8yMDQ5MTIxNDE5MDAwMFowTzEhMB8GA1UECgwYTWljcm9jaGlwIFRlY2hub2xvZ3kgSW5jMSowKAYDVQ
QDDCFDcnlwd
G8gQXV0aGVudGljYXRpb24gU2lnbmVyIEY2MDAwWTATBgcqhkjOPQIBBggqhkjOPQMBBwNCAAR2R0FwsmPn
mVS8hbsS6f5
wDFuN1NaTRZjCKadoAg5OC21IddDtoe72X5FfxrEWRsWhymMfYlVodEdpxd6DtYlqo2YwZDAOBgNVHQ8BAf8EB
AMCAYYwE
gYDVR0TAQH/BAgwBgEB/
wIBADAdBgNVHQ4EFgQU+9yqEor6wbWSj82rEdsJPs9NvvYwHwYDVR0jBBgwFoAUeu19bca3eJ2yOAGl6EqMsK
QOKowwCgY
IKoZIzj0EAwIDSQAwRgIhAMYwMempizBOaH4GxTl5KsV6XAFNMBfe3NJ91R3Nhjf/AiEAxqIsbrGuX4WRSctd53eL
o/
ML6T2bgG+Uvz2QpYR4Ydw=”
]
},
{
“kid”: “1” ,
“kty”: “EC” ,
“crv”: “P-256” ,
“x”: “2Ohne9v0aTSCdrZNmXvtOWir5ETgRhnvecJDXPHzFpg” ,
“y”: “hcP91CMTAKvjdz6_iNWO46g5uPjRvJkuuQ_6THckF-A”
},
{
“kid”: “2” ,
“kty”: “EC” ,
“crv”: “P-256” ,
“x”: “EEExiRf0TBXwPkLihJVRteSY3hU-IGTLlUO-FRMJZFg” ,
“y”: “Nuboaw4W_a3Kwi0lVeG9p4h42I4m7vmK5P49SPebFvM”
},

{
"kid": "3" ,
"kty": "EC" ,
"crv": "P-256" ,
"x": "jKB8Dkci5ExRzg0qtDdAjpRIHShxYON82YZ2-hajenY" ,
"y": "NYMJ9DtbCt6Opnj2g43Ahk2pxQu9KRdMy3m0f-J_rRE"
},
{
"kid": "4" ,
"kty": "EC" ,
"crv": "P-256" ,
"x": "LTU0IGh3ymAzWlWmZ84fhX7YkB4ZCmmlV-YONDtDaDU" ,
"y": "cvNr2TJDWXf4XO6PzybRWoEcQLT4F3NVP8Yj2-X8qbw"
}
]
}
}

The SignedSecureElement example above can be verified with the following certificate:

—–BEGIN CERTIFICATE—-
MIIBxjCCAWygAwIBAgIQZGIWyMZI9cMcBZipXxTOWDAKBggqhkjOPQQDAjA8MSEw
HwYDVQQKDBhNaWNyb2NoaXAgVGVjaG5vbG9neSBJbmMxFzAVBgNVBAMMDkxvZyBT
aWduZXIgMDAxMB4XDTE5MDEyMjAwMjc0MloXDTE5MDcyMjAwMjc0MlowPDEhMB8G
A1UECgwYTWljcm9jaGlwIFRlY2hub2xvZ3kgSW5jMRcwFQYDVQQDDA5Mb2cgU2ln
bmVyIDAwMTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABEu8/ZyRdTu4N0kuu76C
R1JR5vz04EuRqL4TQxMinRiUc3Htqy38O6HrXo2qmNoyrO0xd2I2pfQhXWYuLT35
MGWjUDBOMB0GA1UdDgQWBBTtwIguUA7BijX48KEa6jJQhIwreDAfBgNVHSMEGDAW
gBTtwIguUA7BijX48KEa6jJQhIwreDAMBgNVHRMBAf8EAjAAMAoGCCqGSM49BAMC
A0gAMEUCIQD9/x9zxmHkeWGwjEq67QsQqBVmoY8k6PvFVr4Bz1tYOwIgYfck+fv/
pno8+2vVTkQDhcinNrgoPLQORzV5/l/b4z4=
—–END
CERTIFICATE—–

3.2 Decode Python Example

This is a Python script example for verifying the signed entries and decoding the contents. The script is tested on Python 2.7 and Python 3.7. Required packages can be installed with the Python package manager pip:

```
pip install python-jose[cryptography]
```

```
# (c) 2019 Microchip Technology Inc. and its subsidiaries.
#
# Subject to your compliance with these terms, you may use Microchip software
# and any derivatives exclusively with Microchip products. It is your
# responsibility to comply with third party license terms applicable to your
# use of third party software (including open source software) that may
# accompany Microchip software.
#
# THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER
# EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED
# WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A
# PARTICULAR PURPOSE. IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT,
# SPECIAL, PUNITIVE, INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE
# OF ANY KIND WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF
# MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE
# FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL
# LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED
# THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR
# THIS SOFTWARE.
import json
from base64 import b64decode , b16encode
from argparse import ArgumentParser
import jose . jws
from jose . utils import base64url_decode , base64url_encode
from cryptography import x509
from cryptography . hazmat . backends import default_backend
from cryptography . hazmat . primitives import hashes , serialization
from cryptography . hazmat . primitives . asymmetric import ec
parser = ArgumentParser (
description = ‘Verify and decode secure element manifest’
)
parser . add_argument (
‘–manifest’ ,
help = ‘Manifest file to process’ ,
nargs =1 ,
type = str ,
required =True ,
metavar =’file’
)
parser . add_argument (
‘–cert’ ,
help = ‘Verification certificate file in PEM format’ ,
nargs =1 ,
type = str ,
required =True ,
metavar =’file’
)
args = parser . parse_args ()
# List out allowed verification algorithms for the JWS. Only allows
# public-key based ones.
verification_algorithms = [
‘RS256’ , ‘RS384’ , ‘RS512’ , ‘ES256’ , ‘ES384’ , ‘ES512’
]
# Load manifest as JSON
with open ( args . manifest [ 0 ] , ‘rb’ ) as f :
manifest = json . load ( f )
# Load verification certificate in PEM format
with open ( args . cert [ 0 ], ‘rb’ ) as f :
verification_cert = x509 . load_pem_x509_certificate (
data =f . read (),
backend = default_backend ()
)
# Convert verification certificate public key to PEM format
verification_public_key_pem = verification_cert . public_key ().public_bytes (
encoding =serialization . Encoding. PEM ,
format =serialization . PublicFormat . SubjectPublicKeyInfo
). decode ( ‘ascii’ )
# Get the base64url encoded subject key identifier for the verification cert
ski_ext = verification_cert . extensions . get_extension_for_class (
extclass =x509 . SubjectKeyIdentifier
)
verification_cert_kid_b64 = base64url_encode (
ski_ext . value . digest
). decode ( ‘ascii’ )
# Get the base64url encoded sha-256 thumbprint for the verification cert
verification_cert_x5t_s256_b64 = base64url_encode (
verification_cert . fingerprint ( hashes . SHA256 ())
). decode ( ‘ascii’ )
# Process all the entries in the manifest
for i , signed_se in enumerate ( manifest ):
print ( ” )
print ( ‘Processing entry {} of {}:’ . format ( i +1 , len(manifest )))
print ( ‘uniqueId: {}’ . format (
signed_se [ ‘header’ ][ ‘uniqueId’ ]
))
# Decode the protected header
protected = json . loads (
base64url_decode (
signed_se [ ‘protected’ ]. encode ( ‘ascii’ )
)
)
if protected [ ‘kid’ ] != verification_cert_kid_b64 :
raise ValueError ( ‘kid does not match certificate value’ )
if protected [ ‘x5t#S256’ ] != verification_cert_x5t_s256_b64 :
raise ValueError ( ‘x5t#S256 does not match certificate value’ )
# Convert JWS to compact form as required by python-jose
jws_compact = ‘.’ . join ([
signed_se [ ‘protected’ ],
signed_se [ ‘payload’ ],
signed_se [ ‘signature’ ]
])
# Verify and decode the payload. If verification fails an exception will
# be raised.
se = json. loads (
jose . jws . verify (
token =jws_compact ,
key = verification_public_key_pem ,
algorithms =verification_algorithms
)
)
if se [ ‘uniqueId’ ] != signed_se [ ‘header’ ][ ‘uniqueId’ ]:
raise ValueError (
(
‘uniqueId in header “{}” does not match version in’ +
‘ payload “{}”‘
). format (
signed_se [ ‘header’][ ‘uniqueId’ ] ,
se [ ‘uniqueId’ ]
)
)
print ( ‘Verified’ )
print ( ‘SecureElement = ‘ )
print ( json . dumps ( se , indent =2 ))
# Decode public keys and certificates
try :
public_keys = se [ ‘publicKeySet’ ][ ‘keys’ ]
except KeyError :
public_keys = []
for jwk in public_keys :
print ( ‘Public key in slot {}:’ . format ( int ( jwk[‘kid’ ])))
if jwk [ ‘kty’ ] != ‘EC’ :
raise ValueError (
‘Unsupported {}’ . format ( json . dumps ({ ‘kty’ : jwk[‘kty’ ]}))
)
if jwk [ ‘crv’ ] != ‘P-256’ :
raise ValueError (
‘Unsupported {}’ . format ( json . dumps ({ ‘crv’ : jwk[‘crv’ ]}))
)
# Decode x and y integers
# Using int.from_bytes() would be more efficient in python 3
x = int (
b16encode ( base64url_decode ( jwk[ ‘x’ ]. encode (‘utf8’ ))),
16
)
y = int (
b16encode ( base64url_decode ( jwk[ ‘y’ ]. encode (‘utf8’ ))),
16
)
public_key = ec . EllipticCurvePublicNumbers (
curve =ec. SECP256R1 (),
x =x ,
y =y
). public_key ( default_backend ())
print ( public_key . public_bytes (
encoding =serialization . Encoding . PEM ,
format =serialization . PublicFormat . SubjectPublicKeyInfo
). decode ( ‘ascii’ ))
# Decode any available certificates
for cert_b64 in jwk . get( ‘x5c’ , []):
cert = x509. load_der_x509_certificate (
data =b64decode ( cert_b64 ),
backend =default_backend ()
)
print ( cert . public_bytes (
encoding =serialization . Encoding . PEM
). decode ( ‘ascii’ ))
```

Revision History

Doc Rev.	Date	Description
A	02/2022	Initial release of this document

The Microchip Website

Microchip provides online support via our website at www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- Product Support – Data sheets and errata, application notes and sample programs, design resources, user’s guides and hardware support documents, latest software releases and archived software
- General Technical Support – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip design partner program member listing
- Business of Microchip – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Product Change Notification Service

Microchip’s product change notification service helps keep customers current on Microchip products. Subscribers will receive email notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, go to www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office

- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.
Technical support is available through the website at: **www.microchip.com/support**

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip products:

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner, within operating specifications, and under normal conditions.
- Microchip values and aggressively protects its intellectual property rights. Attempts to breach the code protection features of Microchip product is strictly prohibited and may violate the Digital Millennium Copyright Act.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is “unbreakable”. Code protection is constantly evolving. Microchip is committed to continuously improving the code protection features of our products.

Legal Notice

This publication and the information herein may be used only with Microchip products, including to design, test, and integrate Microchip products with your application. Use of this information in any other manner violates these terms. Information regarding device applications is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. Contact your local Microchip sales office for additional support or, obtain additional support at **www.microchip.com/en-us/support/design-help/client-support-services**.
THIS INFORMATION IS PROVIDED BY MICROCHIP “AS IS”. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE, OR WARRANTIES RELATED TO ITS CONDITION, QUALITY, OR PERFORMANCE.
IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL, OR CONSEQUENTIAL LOSS, DAMAGE, COST, OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE INFORMATION OR ITS USE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THE INFORMATION OR ITS USE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THE INFORMATION.
Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Trademarks

The Microchip name and logo, the Microchip logo, Adaptec, Any Rate, AVR, AVR logo, AVR Freaks, Bestie, Bit Cloud, Crypto Memory, CryptoRF, dsPIC, flexPWR, HELDO, IGLOO, JukeBlox, KeeLoq, Kleer, LANCheck, LinkMD, maXStylus, maXTouch, MediaLB, megaAVR, Microsemi, Microsemi logo, MOST, MOST logo, MPLAB,

OptoLyzer, PIC, picoPower, PICSTART, PIC32 logo, PolarFire, Prochip Designer, QTouch, SAM-BA, SenGenuity, SpyNIC, SST, SST Logo, SuperFlash, Symmetricom, Sync Server, Tachyon, TimeSource, tinyAVR, UNI/O, Vectron, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

AgileSwitch, APT, ClockWorks, The Embedded Control Solutions Company, EtherSynch, Flashtec, Hyper Speed Control, Hyper Light Load, IntelliMOS, Libero, motorBench, mTouch, Powermite 3, Precision Edge, ProASIC, ProASIC Plus, ProASIC Plus logo, Quiet- Wire, Smart Fusion, Sync World, Temux, TimeCesium, TimeHub, TimePictra, TimeProvider, TrueTime, WinPath, and ZL are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, Any In, Any Out, Augmented Switching, Blue Sky, Body Com, Code Guard, CryptoAuthentication, Crypto Automotive, Crypto Companion, Crypto Controller, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, Espresso T1S, EtherGREEN, Grid Time, Ideal Bridge, In-Circuit Serial Programming, ICSP, INICnet, Intelligent Paralleling, Inter-Chip Connectivity, JitterBlocker, Knob-on-Display, max Crypto, max View, mem Brain, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, NVM Express, NVMe, Omniscient Code Generation, PICDEM, PICDEM.net, PI Ckit, PI Ctail, Power Smart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, RTAX, RTG4, SAMICE, Serial Quad I/O, simpleMAP, SimpliPHY, SmartBuffer, Smart HLS, SMART-I.S., storClad, SQI, SuperSwitcher, SuperSwitcher II, Switchtec, SynchroPHY, Total Endurance, TSHARC, USB Check, VariSense, Vector Blox, VeriPHY, ViewSpan, WiperLock, XpressConnect, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

The Adaptec logo, Frequency on Demand, Silicon Storage Technology, Symmcom, and Trusted Time are registered trademarks of Microchip Technology Inc. in other countries.

GestIC is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

All other trademarks mentioned herein are property of their respective companies.

©2022, Microchip Technology Incorporated and its subsidiaries. All Rights Reserved.

ISBN: 978-1-5224-9757-8

Quality Management System

For information regarding Microchip’s Quality Management Systems, please visit www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support: www.microchip.com/support
Web Address: www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Austin, TX
Tel: 512-257-3370

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453
Tel: 317-536-2380

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608
Tel: 951-273-7800

Raleigh, NC
Tel: 919-844-7510

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110
Tel: 408-436-4270

Canada – Toronto
Tel: 905-695-1980
Fax: 905-695-2078

Australia – Sydney
Tel: 61-2-9868-6733

China – Beijing
Tel: 86-10-8569-7000

China – Chengdu
Tel: 86-28-8665-5511

China – Chongqing
Tel: 86-23-8980-9588

China – Dongguan
Tel: 86-769-8702-9880

China – Guangzhou
Tel: 86-20-8755-8029

China – Hangzhou
Tel: 86-571-8792-8115

China – Hong Kong SAR
Tel: 852-2943-5100

China – Nanjing
Tel: 86-25-8473-2460

China – Qingdao
Tel: 86-532-8502-7355

China – Shanghai
Tel: 86-21-3326-8000

China – Shenyang
Tel: 86-24-2334-2829

China – Shenzhen
Tel: 86-755-8864-2200

China – Suzhou
Tel: 86-186-6233-1526

China – Wuhan
Tel: 86-27-5980-5300

China – Xian
Tel: 86-29-8833-7252

China – Xiamen
Tel: 86-592-2388138

China – Zhuhai
Tel: 86-756-3210040

India – Bangalore
Tel: 91-80-3090-4444

India – New Delhi
Tel: 91-11-4160-8631

India – Pune
Tel: 91-20-4121-0141

Japan – Osaka
Tel: 81-6-6152-7160

Japan – Tokyo
Tel: 81-3-6880- 3770

Korea – Daegu
Tel: 82-53-744-4301

Korea – Seoul
Tel: 82-2-554-7200

Malaysia – Kuala Lumpur
Tel: 60-3-7651-7906

Malaysia – Penang
Tel: 60-4-227-8870

Philippines – Manila
Tel: 63-2-634-9065

Singapore
Tel: 65-6334-8870

Taiwan – Hsin Chu
Tel: 886-3-577-8366

Taiwan – Kaohsiung
Tel: 886-7-213-7830

Taiwan – Taipei
Tel: 886-2-2508-8600

Thailand – Bangkok
Tel: 66-2-694-1351

Vietnam – Ho Chi Minh
Tel: 84-28-5448-2100

Austria – Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark – Copenhagen
Tel: 45-4485-5910
Fax: 45-4485-2829

Finland – Espoo
Tel: 358-9-4520-820

France – Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany – Garching
Tel: 49-8931-9700

Germany – Haan
Tel: 49-2129-3766400

Germany – Heilbronn
Tel: 49-7131-72400

Germany – Karlsruhe
Tel: 49-721-625370

Germany – Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Germany – Rosenheim
Tel: 49-8031-354-560

Israel – Ra'anana
Tel: 972-9-744-7705

Italy – Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Italy – Padova
Tel: 39-049-7625286

Netherlands – Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Norway – Trondheim
Tel: 47-72884388

Poland – Warsaw
Tel: 48-22-3325737

Romania – Bucharest
Tel: 40-21-407-87-50

Spain – Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

Sweden – Gothenberg
Tel: 46-31-704-60-40

Sweden – Stockholm
Tel: 46-8-5090-4654

UK – Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820

Documents / Resources

	MICROCHIP Trust Platform Manifest File Format [pdf] User Guide Trust Platform Manifest File Format, Manifest File Format

References

- _ _ _
- Empowering Innovation | Microchip Technology
- Empowering Innovation | Microchip Technology
- Support | Microchip Technology
- Product Change Notification | Microchip Technology
- Quality | Microchip Technology
- Microchip Lightning Support
- cryptoauth_trustplatform_designsuite/TrustFLEX/00_resource_generation at master · MicrochipTech/cryptoauth_trustplatform_designsuite · GitHub
- cryptoauth_trustplatform_designsuite/TrustnGO/00_resource_generation at master · MicrochipTech/cryptoauth_trustplatform_designsuite · GitHub
- RFC 3339 - Date and Time on the Internet: Timestamps
- RFC 4648: The Base16, Base32, and Base64 Data Encodings
- RFC 4648: The Base16, Base32, and Base64 Data Encodings
- RFC 5280 - Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile
- RFC 7515: JSON Web Signature (JWS)
- RFC 7515: JSON Web Signature (JWS)
- RFC 7515: JSON Web Signature (JWS)
- RFC 7515: JSON Web Signature (JWS)
- RFC 7515: JSON Web Signature (JWS)
- RFC 7515: JSON Web Signature (JWS)
- RFC 7515: JSON Web Signature (JWS)
- RFC 7516: JSON Web Encryption (JWE)
- RFC 7517 - JSON Web Key (JWK)
- RFC 7517: JSON Web Key (JWK)
- RFC 7517: JSON Web Key (JWK)
- RFC 7517: JSON Web Key (JWK)
- RFC 7518: JSON Web Algorithms (JWA)
- RFC 7518: JSON Web Algorithms (JWA)
- RFC 7518: JSON Web Algorithms (JWA)
- RFC 7518: JSON Web Algorithms (JWA)
- RFC 7518: JSON Web Algorithms (JWA)

- RFC 7518: JSON Web Algorithms (JWA)
- RFC 7518: JSON Web Algorithms (JWA)
- X.520 : Information technology - Open Systems Interconnection - The Directory: Selected attribute types
- microchip.com/en-us/product/ATECC608B-TNGTLS#document-table
- Client Support Services | Microchip Technology