Connected Vehicle Security Infrastructure

Key Packaging Specification

Version 0.72

UNCONTROLLED COPY IF PRINTED

FORD CONFIDENTIAL

Revision History:

Version	Revision Date	Description of Change	Affected Sections	Author
0.1	5/18/17	Initial Draft		Dmitriy Ansolis [dansolis]
0.2	6/7/17			Dmitriy Ansolis [dansolis]
0.31	7/12/17			Dmitriy Ansolis [dansolis]
0.32	8/23/17			Dmitriy Ansolis [dansolis]
0.4	10/18/17			Dmitriy Ansolis [dansolis]
0.5	11/17/17			Dmitriy Ansolis [dansolis]
0.6	11/21/17			Dmitriy Ansolis [dansolis] Mustafa Tambawala [mtambawa]
0.7	1/21/18	Added IVSS web service details Added high level flow diagram Updated supplier bundle storage/management/ details Updated data flow “Happy Path” diagram	6, 9, 10	Dmitriy Ansolis [dansolis] Mustafa Tambawala [mtambawa]
0.71	1/22/18	Fixed diagram	6.4	Dmitriy Ansolis [dansolis] Mustafa Tambawala [mtambawa]
0.72	2/22/18	Updated REST API details to include FESN prefix, and bundle hash to be hex encoded. Added error code for missing parameters.	6, 6.3, 6.4	Mustafa Tambawala [mtambawa]

Table of Contents:

1 Introduction

1.1 Overview

Various ongoing and upcoming vehicle programs and use cases require creation of security objects, primarily keys and certificates, injection of these objects into vehicle modules at supplier and/or Ford assembly lines, and secure transfer of the objects between Ford and the supplier site. For programs following this specification, security objects will (in most cases) be generated by Ford and transmitted to suppliers for injection into modules.

This document defines the schema and processes for packaging of security materials and their delivery to module suppliers and endpoint systems (GIVIS/SDN).

1.2 Historical Background

The introduction of the SYNC infotainment system in 2006 necessitated development of security infrastructure to manage the complexities involved with software installation, code signing, telematics messages and other features/requirements. The flow of security objects involved creation of keys at the supplier site that were sent to Ford in a secure data feed. This data flow is described in the A51 Supplier Feed Specifications.

Due to requirements of new vehicle programs, beginning with Fully Networked Vehicle (FNV2), security objects (e.g. keys) will no longer will be generated by module suppliers. Instead, they will be generated by Ford and transmitted to suppliers and other systems (GIVIS/SDN). The normal supplier feed process (A51) will continue; however, the feed will not contain the actual security objects but instead a reference to those objects. Immediate use cases include next-gen TCU, Ethernet communication between ECUs, secure CAN communication, V2V features, etc…

1.3 Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and “OPTIONAL" in this document are to be interpreted as described in RFC 2119.

In this document, the term “security object” indicates an object that is used together with cryptographic methods to provide some level of data security. Security objects include, but are not limited to, symmetric keys, public/private asymmetric keys, certificates, and passwords.

1.4 Assumptions

- TBD

1.5 References

This section contains references to documents which affect the requirements presented in this requirement specification.

Ref #	Document Name
1	A51 Supplier Feed Specification
2	FESN Generation Specification
3	RFC 3370: Cryptographic Message Syntax (CMS) Algorithms
4	RFC 4056: Use of the RSASSA-PSS Signature Algorithm in Cryptographic Message Syntax (CMS)

1.6 Terminology and Abbreviation

Term	Description
AES	Advanced Encryption Standard (symmetric key encryption)
CMAC	Cipher-Based Message Authentication Code
CMS	Cryptographic Message Syntax
DER	Distinguished Encoding Rule
ECDSA	Elliptic Curve Digital Signature Algorithm
ECIES	Elliptic Curve Integrated Encryption Scheme
ECU	Electronic Control Unit
EOL	End of Line
FESN	Ford Electronic Serial Number
FNV2	Fully Networked Vehicle 2
GCM	Galois/Counter Mode
GIVIS	Global In Vehicle Information System
IV	Initialization Vector
IVSS	In-Vehicle Security Services
KDF	Key Derivation Function
PBKDF2	Password-Based Key Derivation Function
PEM	Privacy Enhanced Mail
RSA	Asymmetric public-private key cryptosystem
SHA	Secure Hash Algorithm
SPID	Security Package ID
S/MIME	Secure/Multipurpose Internet Mail Extensions
TCU	Telematics Control Unit

2 High Level Flow

The high-level flow of the key packaging process is as follows (refer to flow diagram in Section 10):

- Ford will create security objects based on specific module requirements
  - Ford Electronic Serial Numbers (FESN) will be generated by Ford (in most cases)
  - Security objects will be packaged together for each individual module FESN
  - The Security Package will be encrypted using a method agreed upon with the supplier
  - The Security Package will include a XML manifest file which describes the security objects in the package and includes any data necessary to decrypt/unpack the package.
  - A unique Security Package Identifier (SPID) will be generated and included in the Security Package.
  - Ford will store all security materials, with complete Security Package archived and some specific security objects stored in GIVIS/SDN for online access.
- GIVIS will host a web service to receive security objects from IVSS for each FESN.
  - Security objects will be sent to GIVIS in XML format.
- Security Packages will be bundled and shared with the supplier in a secure location accessible only by the appropriate module supplier.
- Module supplier will download a bundle of Security Packages.
- The supplier will verify the signature on each Security Package.
- The supplier will utilize data in the XML manifest, and pre-shared keys (if any), to decrypt the Security Package content into its component security objects.
- Security objects will be flashed onto modules at supplier site.
- At supplier EOL, the <SPID, FESN> pairings will be sent back to Ford (via supplier feed) to confirm that packages were provisioned to appropriate modules. The SPID and FESN will be written to DIDs on the module. For supplier feed details, please refer to module-specific A51 Supplier Feed specifications.
- At Ford EOL, the SPID and FESN DIDs will be read and sent to GIVIS to be associated with a VIN.

3 Key Package Format

3.1 Overview

The schema described in this document supports packaging of security objects for a particular module FESN. The resulting package contains all security material that is to be provisioned onto a module at a supplier site.

The file hierarchy described in subsequent paragraphs is shown below. In general terms, all security objects for a particular FESN that are to be delivered to a supplier are packaged, along with a file containing the FESN of the target module, into a tar file, which is encrypted. In addition, an XML manifest file is created, containing general information about the package, data needed to decrypt the package and verify signatures, and descriptions of the security objects. The tar file and the manifest are packaged into another tar file, and a detached signature is generated for that tar file. The signature and the new tar file are packaged into yet another tar file, which can be compressed. The final tar file name will consist of the FESN and the SPID.

```
|---<UniquePackageName>.tar
    |---Package.sig
    |---Package.tar
        |---Manifest.xml
        |---Data.tar.enc (encrypted Data.tar)
            |---FESN.txt
            |---SecObj_1.*
            |---SecObj_2.*
            |---SecObj_3.*
                 …
                 …
                 …
            |---SecObj_N.*
```

As seen above, each object or collection of objects (depending on supplier/module requirements) must be stored in a separate file *(<SecObj_X>.**). There is also a file named *FESN.txt,* which simply contains the 8-character FESN. These files must be packaged together into a *Data.tar* file and this file must be encrypted and named *Data.tar.enc*.

In addition to the *Data.tar.enc* file, there must be a *Manifest.xml* file containing a listing of the security objects (referred to by file name), as well as general information about the package (sender, receiver, date, etc…), and any data required for performing decryption and signature verification. See Section 3.6 for details.

The *Data.tar.enc* file and the *Manifest.xml* file must be packaged into a *Package.tar* file. This file must be digitally signed, with the signature stored in a separate *Package.sig* file. The *Package.tar* and *Package.sig* files will then be packaged up into a *<UniquePackageName>.tar* file, which can be compressed into a *<UniquePackageName>.tar.gz* file.

Details on supported encryption/signing methods are listed below.

3.2 File Definitions

File	Description
FESN.txt	File containing the 8-character FESN
SecObj_X.*	File containing security object or collection of security objects
Data.tar	Tar file containing all security objects in the package, including the FESN.txt file
Data.tar.enc	Encrypted Data.tar file See Section 3.5.1 for details on encryption methods.
Manifest.xml	XML file describing the data package (see table below)
Package.tar	Tar file containing the manifest, and encrypted data package (Data.tar.enc)
Package.sig	Detached signature of the Package.tar file. The digest being signed is a SHA-256 hash of Package.tar. See Section 3.5.2 for details on signing methods.
<UniquePackageName>.tar	Tar file containing Package.tar and Package.sig UniquePackageName is the combination of the FESN and SPID (SHA-1 hash of Data.tar, as defined in Section 3.6 below), separated by a hyphen. Example: “ECG00001-18132b4a69c29a4d55e4c963c8fd9588557f37a3.tar”
<UniquePackageName>.tar.gz	Compressed <UniquePackageName>.tar file UniquePackageName is the combination of the FESN and SPID (SHA-1 hash of Data.tar, as defined in Section 3.6 below), separated by a hyphen. Example: “ECG00001-18132b4a69c29a4d55e4c963c8fd9588557f37a3.tar.gz”

3.3 FESN Generation

FESNs will be generated according to the *FESN Generation Specification*.

3.4 Formats

For certificates and keys, the Ford--preferred security object formats are listed below. If other formats are needed, this must be agreed upon between Ford and the supplier.

- Certificates: DER or PEM
- Public/private keys: PKCS #8, PEM or DER
- Symmetric keys: Binary

3.5 Cryptographic Algorithms

This section describes the supported cryptographic algorithms for encryption and signing of provisioning packages. Both symmetric and asymmetric algorithms are supported for signing and encryption. If any algorithm or mode specified in this document (SHA-256 hashes, AES/RSA encryption, GCM auth tags, etc..) is not supportable by the supplier, Ford and the supplier must agree on a different method.

3.5.1 Encryption

The supported encryption methods are listed below.

Asymmetric
- CMS (PKCS #7)
  - RSA-OAEP (2048-bit)
  - ECDH (256-bit) (Curve P-256)

Symmetric
- AES-GCM (128-bit); Key Derivation (PBKDF2)
- AES-CBC (128-bit);

If symmetric methods are used to encrypt the package, then key derivation based on a pre-shared symmetric key is preferred. The preferred Key Derivation Function (KDF) is PBKDF2.
For AES-GCM, a 128-bit KDF salt and a 96-bit Initialization Vector are required. The KDF salt, IV, and the auth tag generated by the algorithm will be included in the package manifest file.

3.5.2 Signing

The following signing methods are supported:

Asymmetric
- CMS (PKCS #7)
  - RSA-PSS (2048-bit)
  - ECDSA (256-bit) (Curve P-256)
- Raw
  - RSA-PSS (2048-bit)
  - ECDSA (256-bit) (Curve P-256)

Symmetric
- CMAC (128-bit)

Asymmetric signatures will be generated using a Ford-issued signing certificate, and the certificate chain will be provided to the supplier for signature validation.

3.6 XML Structure

Please refer to the XML schema in Section 4 as needed when reviewing the table below.

Note: depending on encryption/signing methods used, some fields may be empty or absent from the manifest (will vary by module type).

Field	Type/Size (Chars)	Description/Source/Location
Header		
SPID (Security Package Identifier)	String (40)	SHA-1 hash (Hex ASCII) of unencrypted data package (Data.tar).
FESN (Ford Electronic Serial Number)	String (8)	Generated by sender based on FESN specification defined in this document.
Module Type	String (8)	String representing the module type for which security objects are generated. For example: ECG TCU SYNC4 …
Date/Time of Generation	DateTime	Date/Time when file was generated (Format YYYY-MM-DDThh:mm:ssTZD).
Sender	String (40)	String representing sender of package (usually Ford).
Receiver	String (40)	String representing receiver of package (usually a supplier).
Signature Method	String (20)	String representing type of digital signature applied to manifest and data package. Must be one of the following (unless another method is agreed upon between Ford and supplier): CMS_RSA CMS_ECDSA RAW_RSA RAW_ECDSA CMAC
Encryption Method	String (20)	String representing type of encryption applied to data package. Must be one of the following (unless another method is agreed upon between Ford and supplier): CMS_RSA CMS_ECDH AES_GCM AES_CBC
Key Derivation Function Salt	String (24)	Sender-generated 128-bit salt for AES-128 encryption, if using key derivation method. This value will be represented as a Base-64 encoded value.

Initialization Vector (if using AES encryption)	String (16)	Sender-generated unique initialization vector for AES-128 encryption. This value will be represented as a Base-64 encoded value.
GCM Auth Tag (if using AES encryption)	String (24)	Sender-generated authentication tag for verifying integrity of data in AES encryption. This value will be represented as a Base-64 encoded value.
Wrapped Encryption Key (if not using key derivation)	String	Sender-generated asymmetrically wrapped symmetric encryption key (in cases where the key derivation method cannot be used, and a wrapped key must be transmitted in the manifest).
Security Object Listing		For each security object file contained in the Data.tar file, there will be a record containing the fields below.
Security Object Name	String (80)	Sender-assigned object identifier
Security Object File Name	String (80)	Security object file name (matching name of file in Data.tar package).
Security Object Type	String (20)	String representing security object type (symmetric key, private/public key, certificate, password, etc…). If none of the predefined types apply, (for example, the object is a bundle of keys or certificates), the “Custom” type must be selected. SymmetricKey PrivateKey PublicKey Certificate Password ExpansionFunction Token FESN Custom
Security Object Hash	String (44)	SHA-256 hash of the unencrypted file, Base-64 encoded.
Security Object Format	String (20)	Format of certs/keys, if applicable (ASN.1 DER, Base64 ASCII, etc...) This is an informational field - a descriptive label should be provided.

4 Security Package Manifest XML Schema

Please refer to Section 3.6 for details on the various fields in the schema below.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

	<!-- definition of simple elements -->
	<!-- Header -->
	<xs:element name="SPID" type="xs:base64Binary"/>
	<xs:element name="FESN" type="xs:string"/>
	<xs:element name="ModuleType" type="xs:string"/>
	<xs:element name="CreationDateTime" type="xs:dateTime"/>
	<xs:element name="Sender" type="xs:string"/>
	<xs:element name="Receiver" type="xs:string"/>
	<xs:element name="SignatureMethod" type="xs:string"/>
	<xs:element name="EncryptionMethod" type="xs:string"/>
	<xs:element name="KDFSalt" type="xs:base64Binary"/>
	<xs:element name="IV" type="xs:base64Binary"/>
	<xs:element name="AuthTag" type="xs:base64Binary"/>
	<xs:element name="WrappedEncryptionKey" type="xs:base64Binary"/>


	<!-- Security Object List -->
	<xs:element name="SecurityObjectName" type="xs:string"/>
	<xs:element name="SecurityObjectFileName" type="xs:string"/>
	<xs:element name="SecurityObjectType">
		<xs:simpleType>
			<xs:restriction base="xs:string">
				<xs:enumeration value="SymmetricKey"/>
				<xs:enumeration value="PrivateKey"/>
				<xs:enumeration value="PublicKey"/>
				<xs:enumeration value="Certificate"/>
				<xs:enumeration value="Password"/>
				<xs:enumeration value="ExpansionFunction"/>
				<xs:enumeration value="Token"/>
				<xs:enumeration value="FESN"/>
				<xs:enumeration value="Custom"/>
			</xs:restriction>
		</xs:simpleType>
	</xs:element>
	<xs:element name="SecurityObjectHash" type="xs:base64Binary"/>
	<xs:element name="SecurityObjectFormat" type="xs:string"/>

	<!-- definition of complex elements -->

	<xs:element name="SecurityObjectInfo">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="SecurityObjectName" minOccurs="1" maxOccurs="1"/>
				<xs:element ref="SecurityObjectFileName" minOccurs="1" maxOccurs="1"/>
				<xs:element ref="SecurityObjectType" minOccurs="1" maxOccurs="1"/>
				<xs:element ref="SecurityObjectHash" minOccurs="1" maxOccurs="1"/>
				<xs:element ref="SecurityObjectFormat" minOccurs="1" maxOccurs="1"/>
			</xs:sequence>
			<xs:attribute name="ID" type="xs:nonNegativeInteger" use="required"/>
		</xs:complexType>
	</xs:element>


	<xs:element name="SecurityObjectList">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="SecurityObjectInfo" maxOccurs="unbounded"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>


	<xs:element name="Package">
		<xs:complexType>
			<xs:sequence>
```

```
                        <!-- Header -->
                        <xs:element ref="SPID" minOccurs="1" maxOccurs="1"/>
                        <xs:element ref="FESN" minOccurs="1" maxOccurs="1"/>
                        <xs:element ref="ModuleType" minOccurs="1" maxOccurs="1"/>
                        <xs:element ref="CreationDateTime" minOccurs="1" maxOccurs="1"/>
                        <xs:element ref="Sender" minOccurs="1" maxOccurs="1"/>
                        <xs:element ref="Receiver" minOccurs="1" maxOccurs="1"/>
                        <xs:element ref="SignatureMethod" minOccurs="1" maxOccurs="1"/>
                        <xs:element ref="EncryptionMethod" minOccurs="1" maxOccurs="1"/>
                        <xs:element ref="KDFSalt" minOccurs="0" maxOccurs="1"/>
                        <xs:element ref="IV" minOccurs="0" maxOccurs="1"/>
                        <xs:element ref="AuthTag" minOccurs="0" maxOccurs="1"/>
                        <xs:element ref="WrappedEncryptionKey" minOccurs="0" maxOccurs="1"/>
                        <!-- Security Object List -->
                        <xs:element ref="SecurityObjectList" minOccurs="1" maxOccurs="1"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
</xs:schema>
```

5 Data Format

The following is an example of a package manifest.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Package>
        <SPID>18132b4a69c29a4d55e4c963c8fd9588557f37a3</SPID>
        <FESN>ECG00001</FESN>
        <ModuleType>ECG</ModuleType>
        <CreationDateTime>2017-06-08T09:30:47-05:00</CreationDateTime>
        <Sender>Ford Motor Company</Sender>
        <Receiver>Ford Motor Company</Receiver>
        <SignatureMethod>CMS_RSA</SignatureMethod>
        <EncryptionMethod>AES_GCM</EncryptionMethod>
        <KDFSalt></KDFSalt>
        <IV>ASNFZ4mrze8BI0Vn</IV>
        <AuthTag>OB/oZss0wtYZiIRmCvEhoQ==</AuthTag>

        <SecurityObjectList>
                <SecurityObjectInfo ID="0">
                        <SecurityObjectName>FESN</SecurityObjectName>
                        <SecurityObjectFileName>FESN.txt</SecurityObjectFileName>
                        <SecurityObjectType>FESN</SecurityObjectType>
                        <SecurityObjectHash>W3+jDZsvGQU3uWFmG97PtLffqwJ5rDfQ8wMmjSnBqFQ=</SecurityObjectHash>
                        <SecurityObjectFormat>ASCII</SecurityObjectFormat>
                </SecurityObjectInfo>
                <SecurityObjectInfo ID="1">
                        <SecurityObjectName>MQTT</SecurityObjectName>
                        <SecurityObjectFileName>mqtt_password.key</SecurityObjectFileName>
                        <SecurityObjectType>SymmetricKey</SecurityObjectType>
                        <SecurityObjectHash>cLEi6sOaClCZYyyE3nwxKPtHslv88ppAq1RqncJ/xVM=</SecurityObjectHash>
                        <SecurityObjectFormat>Base64 ASCII</SecurityObjectFormat>
                </SecurityObjectInfo>
                <SecurityObjectInfo ID="2">
                        <SecurityObjectName>SYNCP0</SecurityObjectName>
                        <SecurityObjectFileName>syncp_0.key</SecurityObjectFileName>
                        <SecurityObjectType>SymmetricKey</SecurityObjectType>
                        <SecurityObjectHash>K+ga0hJyNFuyRpyVjSdMplJm6m2URUhWf47CRbgamOY=</SecurityObjectHash>
                        <SecurityObjectFormat>Base64 ASCII</SecurityObjectFormat>
                </SecurityObjectInfo>
                <SecurityObjectInfo ID="3">
                        <SecurityObjectName>SYNCP1</SecurityObjectName>
                        <SecurityObjectFileName>syncp_1.key</SecurityObjectFileName>
                        <SecurityObjectType>SymmetricKey</SecurityObjectType>
                        <SecurityObjectHash>dpn9jm1Q3g8HS0KU/+/86m1Drm/ER+TJMIx6m4GPjuk=</SecurityObjectHash>
                        <SecurityObjectFormat>Base64 ASCII</SecurityObjectFormat>
                </SecurityObjectInfo>
                <SecurityObjectInfo ID="4">
                        <SecurityObjectName>SYNCP2</SecurityObjectName>
                        <SecurityObjectFileName>syncp_2.key</SecurityObjectFileName>
                        <SecurityObjectType>SymmetricKey</SecurityObjectType>
                        <SecurityObjectHash>Ded37QdJIULZu2J3xZ66yEKuLVImQ2B7eKJ1oGEoHzU=</SecurityObjectHash>
                        <SecurityObjectFormat>Base64 ASCII</SecurityObjectFormat>
                </SecurityObjectInfo>
                <SecurityObjectInfo ID="5">
                        <SecurityObjectName>SYNCP3</SecurityObjectName>
                        <SecurityObjectFileName>syncp_3.key</SecurityObjectFileName>
                        <SecurityObjectType>SymmetricKey</SecurityObjectType>
                        <SecurityObjectHash>iqSsp/i07891QO7SwpS59BzOEHHiR0ZmWJ/5yzya6oo=</SecurityObjectHash>
                        <SecurityObjectFormat>Base64 ASCII</SecurityObjectFormat>
                </SecurityObjectInfo>
                <SecurityObjectInfo ID="6">
                        <SecurityObjectName>SYNCP4</SecurityObjectName>
                        <SecurityObjectFileName>syncp_4.key</SecurityObjectFileName>
                        <SecurityObjectType>SymmetricKey</SecurityObjectType>
                        <SecurityObjectHash>tL/1wmZlykWgqm41B5IZaYapRSD0MksQRE8QzrJFUfk=</SecurityObjectHash>
                        <SecurityObjectFormat>Base64 ASCII</SecurityObjectFormat>
                </SecurityObjectInfo>
                <SecurityObjectInfo ID="7">
                        <SecurityObjectName>SYNCP5</SecurityObjectName>
                        <SecurityObjectFileName>syncp_5.key</SecurityObjectFileName>
                        <SecurityObjectType>SymmetricKey</SecurityObjectType>
                        <SecurityObjectHash>qBYhR5fynZahyuf7Am8EjyC+77MH2uNhvZvjbXYgRW4=</SecurityObjectHash>
                        <SecurityObjectFormat>Base64 ASCII</SecurityObjectFormat>
```

```
            </SecurityObjectInfo>
            <SecurityObjectInfo ID="8">
                  <SecurityObjectName>SYNCP6</SecurityObjectName>
                  <SecurityObjectFileName>syncp_6.key</SecurityObjectFileName>
                  <SecurityObjectType>SymmetricKey</SecurityObjectType>
                  <SecurityObjectHash>KEbP6rS522jlAvUA1dA0P+ELODij7XeVDROmJWZCRcE=</SecurityObjectHash>
                  <SecurityObjectFormat>Base64 ASCII</SecurityObjectFormat>
            </SecurityObjectInfo>
            <SecurityObjectInfo ID="9">
                  <SecurityObjectName>SYNCP7</SecurityObjectName>
                  <SecurityObjectFileName>syncp_7.key</SecurityObjectFileName>
                  <SecurityObjectType>SymmetricKey</SecurityObjectType>
                  <SecurityObjectHash>c7NruQ+HnqRcSXKsSuz1aWv8c0piTqJoPDBeXt+cvPQ=</SecurityObjectHash>
                  <SecurityObjectFormat>Base64 ASCII</SecurityObjectFormat>
            </SecurityObjectInfo>
            <SecurityObjectInfo ID="10">
                  <SecurityObjectName>SOAPrivKey</SecurityObjectName>
                  <SecurityObjectFileName>dev-soa-tls-ECG00001-1.key</SecurityObjectFileName>
                  <SecurityObjectType>PrivateKey</SecurityObjectType>
                  <SecurityObjectHash>ZuYdTb8YUm5A1f9zh3brQ2TOLXWOpF4vJMwE8UwYjCo=</SecurityObjectHash>
                  <SecurityObjectFormat>PKCS8-PEM</SecurityObjectFormat>
            </SecurityObjectInfo>
            <SecurityObjectInfo ID="11">
                  <SecurityObjectName>SOAPublicCert</SecurityObjectName>
                  <SecurityObjectFileName>dev-soa-tls-ECG00001-1.pem</SecurityObjectFileName>
                  <SecurityObjectType>Certificate</SecurityObjectType>
                  <SecurityObjectHash>jJ2MfegzmMIOisauYzwqxaJwnV9ij6HZ0xoDm8SU3OM=</SecurityObjectHash>
                  <SecurityObjectFormat>PEM</SecurityObjectFormat>
            </SecurityObjectInfo>
      </SecurityObjectList>
</Package>
```

6 Delivery of Security Packages to Supplier

Ford will host two web services in its Azure cloud environment to deliver Security Package bundles to suppliers. These services will be implemented using Representational State Transfer (REST). Mutual TLS will be used for authentication to these web services. Ford will upload bundles to cloud storage (the upload process will run internally in IVSS). The supplier will be provided with an expiring link to download bundles from the Ford cloud (refer to high level flow diagram in Section 6.4).

Note: Before Security Packages can be shared with the supplier, information relating to the bundles and FESNs for which the bundles have been created should already be in GIVIS. The process for sending information to GIVIS is described in Section 7.

- Security Packages will be created based on details in Section 3 of this document. They will be bundled based on details in Section 6.1 below.
- Bundles (tar files) will be uploaded to cloud storage through an automated process (IVSS internal process). This process will maintain an inventory of a pre-determined number of bundles in cloud storage. In addition, the supplier will maintain a bundle inventory in order to mitigate the risk of failures in the delivery process. Both the Ford cloud and supplier site inventories will be based on supplier bundle consumption requirements (agreed upon between Ford and supplier).
- The *GetSecurityBundleUrl* web service will be hosted in the Ford cloud to be used by the supplier to get the URL of the next available bundle to download. This web service will take a required parameter (FESN prefix) and an optional parameter (Bundle ID).
    - If the web service is called with only FESN prefix, it will return the URL of the next available bundle for the supplier.
    - If the web service is called with FESN prefix and Bundle ID as the parameters, it will return an expiring URL to download the corresponding bundle.
    - Supplier can download bundles using HTTPS GET request.
- The supplier, after downloading the bundle, will generate the SHA-256 hash of the bundle file and call the *KeyPackageBundleAck* web service, sending the Bundle ID, FESN prefix, and bundle hash (hex encoded) to Ford for verification.
- The *KeyPackageBundleAck* web service will verify the bundle hash and confirm file download success or failure by sending a response back to the supplier.
- The previously delivered bundle can be requested and resent by calling the *GetSecurityBundleUrl* web service with FESN prefix and Bundle ID as a parameter. This will return the URL of the requested bundle for download. Delivered bundles will be available for redelivery through this web service for 7 days. After 7 days, the delivered bundle will have to be requested outside this process, if re-delivery is necessary.
- Upon successful bundle download by the supplier, the *KeyPackageBundleAck* web service will notify GIVIS to update Security Package status to “Distributed” for packages contained in the bundle (new service needs to be hosted in GIVIS to receive this status update).
- Supplier access to the web service will be based on mutual TLS authentication.
- Bundle ID will be formatted as described in Section 6.1 below.
- The number of Security Packages contained in a bundle may vary based on supplier requirements.

6.1 Bundle ID Format

Example: 1ABA0001-A0099-20171108080525

Description	Position	Number of Characters	Value	Example
FESN prefix (3 character fixed prefix representing unique Supplier/Site/Major product)	1 to 3	3	A-Z, 0-9	1AB
Last 5 characters of FESN. This is start of module range added to bundle	4 to 8	5	A-Z, 0-9	A0001 start range
Last 5 characters of FESN. This is end of module range added to bundle	10 to 14	5	A-Z, 0-9	A0099 end range. Based on range it can be deduced that bundle contains 99 packages/modules.
Time stamp representing bundle creation date and time	16 to 29	14 (YYYYMMDDHHMISS)	0-9	20171108080525 (November 8, 2017 time 8:05:25 am)

6.2 Mutual TLS Authentication

The supplier will provide Ford with their X.509 TLS cert to be configured in the Ford cloud (Azure) for web service authentication. Certificates will be valid for 3 years and must be renewed by the supplier 6 months prior to expiration. The supplier must provide a new certificate to Ford upon renewal. Supplier will utilize this certificate in their HTTPS requests to the web service. The supplier can provide the certificate to Ford in PEM or DER format. The certificate can be self-signed or can come from a public or private Certification Authority.

The following table describes required attributes for X.509 certificates and how they need to be configured.

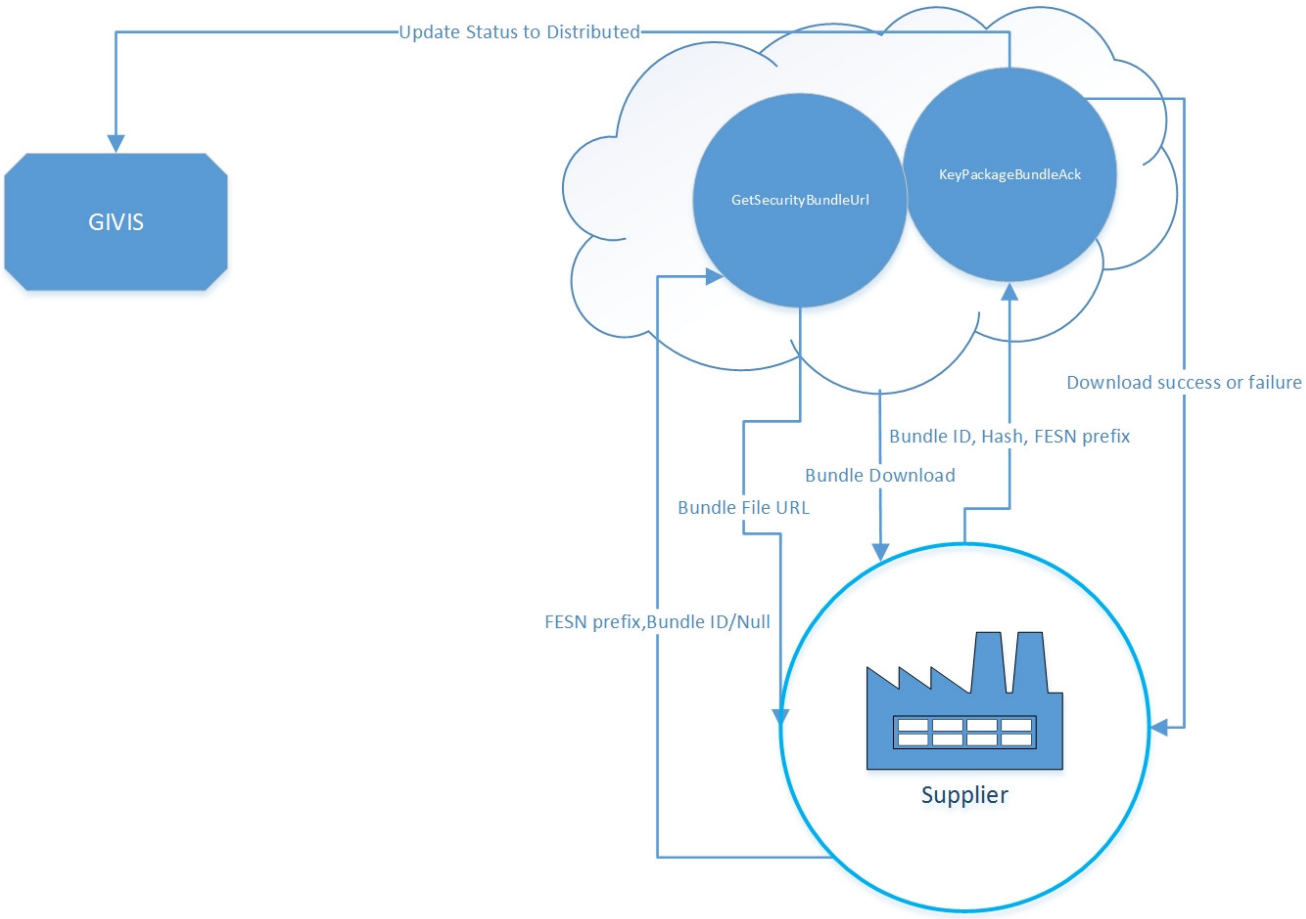
Field	Value	Example
CN	SupplierName_TLS_Environment	ABCco_tls_prod
C	Supplier country code	US
ST	Supplier state/province	MI
L	City	Dearborn
O	Supplier company name	ABCco
Thumbprint	SHA-256 hash	4f9c3633e8859bbe74114c4f82aa23ada90dc9a7b59643fd36451239ee1163ea
Basic Constraint	Subject Type=End Entity Path Length Constraint=None	
Key Usage	Digital Signature, Key Encipherment	
Enhanced Key Usage	Client Authentication	

6.3 Web Service Details

The following web services using REST API will be hosted in the cloud to deliver Security Package bundles to the supplier.

Request	**GetSecurityBundleUrl**
URL	https://(hostname)/GetSecurityBundleUrl/FESN_prefix/[BundleID]
Method	Get
URL Parameters	Required (string) FESN prefix Optional (string) Bundle ID
XML Return Response	Bundle download URL
Success Return Codes	*HTTP* 200 *Application* 000
Error Return Codes	*HTTP* Any code other than 200 *Application* Any code other than 000 (refer to error code table in section 6.5)
Request	**KeyPackageBundleAck**
URL	https://(hostname)/KeyPackageBundleAck/BundleID/BundleFileHash/FESN_prefix
Method	Get
URL Parameters	Required (string) Bundle ID (string) Bundle file SHA-256 hash, hex encoded uppercase with no separator (string) FESN prefix
XML Return Response	Success or Failure
Success Return Codes	HTTP 200 *Application* 000
Error Return Codes	*HTTP* Any code other than 200 *Application* Any code other than 000 (refer to error code table in section 6.5)

6.4 High Level Flow



6.5 Bundle Delivery Error Codes

There are two types of error codes returned by the web services – HTTP errors and application validation errors. For information on error codes for handling of HTTP errors, please refer to https://docs.microsoft.com/en-us/rest/api/storageservices/common-rest-api-error-codes.

The table below covers any errors raised during processing of web service requests. Codes are returned with XML response.

Code	Description	Reference Information / Comment
000	**Success**	**General success message**
100	**Validation Error**	**Errors occurring due to incorrect parameters**
101	Bundle ID format error	Bundle ID sent in the request is not valid
102	Invalid bundle file hash	Hash sent with ACK request is not proper SHA-256 hash
103	Required parameter missing	Requested call is missing required parameter for the call
200	**Error while processing request**	**Errors relating to process for bundle URL request or confirmation of bundle download**
201	Requested bundle not found	Bundle requested with Bundle ID not found
202	No bundle available	There are no bundles available for download - bundle buffer is depleted
203	Bundle hash did not match	Hash of the downloaded bundle sent for verification did not match; Bundle needs to be downloaded again
300	**Authentication Errors**	**Errors relating to client authentication**
301	Not Authorized	Unable to authenticate the caller
302	Certificate not sent in the request.	X.509 certificate is missing in header for Mutual TLS authentication
303	Certificate expired	Certificate sent with the request is expired
400	**Custom defined error range**	**Extension point for additional errors**

7 Delivery of Security Objects to GIVIS

- GIVIS will host a web service to receive security objects (e.g., SyncP keys) from IVSS
- IVSS will provide security data to GIVIS as an XML document
- The XML document will contain data for multiple modules and each module may contain multiple security objects
- The GIVIS web service will parse XML and process the Security Package data according to specific module requirements

7.1 XML Structure

XML Field	Type/Size (Chars)	Description/Source/Location
Application Area		
ModuleType	String (8)	String representing the module type for which security objects are generated. Examples: ECG TCU SYNC4
Supplier	String(40)	Supplier receiving modules sent in this XML. It will be in uppercase (e.g., CLARION)
CreateTimeStamp	DateTime	Timestamp when XML is created
BundleID	String (27)	Unique identifier of the tar bundle sent to supplier
Package List		
SPID	String (40)	SHA-1 hash (Hex ASCII) of unencrypted data package (Data.tar).
FESN	String (8)	Generated by Ford based on FESN specification.
Security Object Listing		
SecurityObjectName	String (80)	Sender-assigned object identifier
SecurityObjectType	String (20)	String representing security object type (symmetric key, private/public key, certificate, password, etc…). If none of the predefined types apply, (for example, the object is a bundle of keys or certificates), the “Custom” type must be selected. SymmetricKey PrivateKey PublicKey Certificate Password ExpansionFunction Token FESN Custom
SecurityObjectApplication	String (20)	String representing application/purpose of object (e.g., SyncP key wrapped for NA region in SDN)

		TBD: Listing of allowed application strings
SecurityObjectHash	String (44)	SHA-256 hash of the unencrypted security object. Base-64 encoded.
SecurityObjectFormat	String (20)	Format of certs/keys, if applicable (ASN.1 DER, Base64 ASCII, etc...) This is an informational field - a descriptive label should be provided.
SecurityObject	Blob	Encrypted security object. Base64 encoded.

7.2 XML Schema

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" attributeFormDefault="unqualified"
elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="ShowModuleList">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="ApplicationArea">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="ModuleType" type="xs:string" minOccurs="1" maxOccurs="1"/>
              <xs:element name="Supplier" type="xs:string" minOccurs="1" maxOccurs="1"/>
              <xs:element name="CreateTimeStamp" type="xs:dateTime" minOccurs="1" maxOccurs="1"/>
              <xs:element name="BundleID" type="xs:string" minOccurs="1" maxOccurs="1" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="PackageList">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Module" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="SPID" type="xs:string" minOccurs="1" maxOccurs="1"/>
                    <xs:element name="FESN" type="xs:string" minOccurs="1" maxOccurs="1"/>
                    <xs:element name="SecurityObjectList">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element maxOccurs="unbounded" name="SecurityObjectInfo">
                            <xs:complexType>
                              <xs:sequence>
                                <xs:element name="SecurityObjectName" type="xs:string" minOccurs="1"
maxOccurs="1"/>
                                <xs:element name="SecurityObjectType" type="xs:string" minOccurs="1"
maxOccurs="1"/>
                                <xs:element name="SecurityObjectApplication" type="xs:string" minOccurs="1"
maxOccurs="1"/>
                                <xs:element name="SecurityObjectHash" type="xs:base64Binary" minOccurs="1"
maxOccurs="1"/>
                                <xs:element name="SecurityObjectFormat" type="xs:string" minOccurs="1"
maxOccurs="1"/>
                                <xs:element name="SecurityObject" type="xs:base64Binary" minOccurs="1"
maxOccurs="1"/>
                              </xs:sequence>
                            </xs:complexType>
                          </xs:element>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute name="count" type="xs:nonNegativeInteger" use="required"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

7.3 Sample XML Feed from IVSS to GIVIS

Note: additional regional SDN key wrappings may be required for some of the security objects in example below.

```
<?xml version="1.0" encoding="UTF-8" ?>
<ShowModuleList xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <ApplicationArea>
    <ModuleType>ECG</ModuleType>
    <Supplier>CLARION</Supplier>
    <CreateTimeStamp>2017-06-08T09:30:47-05:00</CreateTimeStamp>
    <BundleID>1ABA0001A0050110820170805 25</BundleID>
  </ApplicationArea>
  <PackageList count="1">
    <Module>
      <SPID>18132b4a69c29a4d55e4c963c8fd9588557f37a3</SPID>
      <FESN>ECG00001</FESN>
      <SecurityObjectList>
        <SecurityObjectInfo>
          <SecurityObjectName>MQTTPwd</SecurityObjectName>
          <SecurityObjectType>SymmetricKey</SecurityObjectType>
          <SecurityObjectApplication>NAWrappedMQTTPwd</SecurityObjectApplication>
          <SecurityObjectHash>cLEi6sOaClCZYyyE3nwxKPtHs1v88ppAq1RqncJ/xVM= </SecurityObjectHash>
          <SecurityObjectFormat>Base64 ASCII</SecurityObjectFormat>
          <SecurityObject>c1KnGLoVdu0idOUP4KYP9YF2Zt193fYRDeFQ328P6wiXlihz6HT04dEYEyiOOWwnnGC4WX0LVesg
mxDsxiqa9MT6VwgCdFfudS21QU7aGWxJn/KfjUK91iHTs9l8qT5jF23ersxWOQs42uFW1TQ9obJw
uh3Uoi/j+UYg10gme9y8gZ0gLi0DVjcannYlhdDhyQqQXwqdjiOI2rlEuM6744y9MMI146Y60fpV
uZNZkwyd8zO0zxDkmdQBYkDySqo//925C6IKyJkVQ14AZPVMIzKAo9fpTPXdF31VI/TQBXk4Dqas
8alsyLwkzbjLjszVberKUwa+xgNApX7bXk7GVA==</SecurityObject>
        </SecurityObjectInfo>
        <SecurityObjectInfo>
          <SecurityObjectName>SYNCP0</SecurityObjectName>
          <SecurityObjectType>SymmetricKey</SecurityObjectType>
          <SecurityObjectApplication>NAWrappedSyncP</SecurityObjectApplication>
          <SecurityObjectHash>K+ga0hJyNFuyRpyVjSdMplJm6m2URUhWf47CRbgamOY=</SecurityObjectHash>
          <SecurityObjectFormat>Base64 ASCII</SecurityObjectFormat>
          <SecurityObject>N2J0wbvIk3iXqA2ZYXU2AuKKaqBBjfzDCkQGLm9LxpizDoQbwLWJUs/r23Knojuf1zB4kfE6cY1+
He+tc31kBrTfyDvdWbAC+xs6DqsezwA4x0ntxgzwsuhIqeM6eMFQDgw3WBEZorXy039nW0So8bWS
oVumEavV0zIha3ZlS+pE9JiFnneLGM73b2gHdozjl7JS+wp63Uu3KmkPUOI4/SvpShMDaZAek6jj
dkC0VOk6LTgHXuM9Ky5joipXRmK0WBXvDQUx6JDlRgW7TG49T7HPjqrY7jFYBokJlBnrydCAzY2o
+wAmkXLm/Pur8sgg+um2EFoe43rm6h2TXPBFMQ==</SecurityObject>
        </SecurityObjectInfo>
        <SecurityObjectInfo>
          <SecurityObjectName>SYNCP1</SecurityObjectName>
          <SecurityObjectType>SymmetricKey</SecurityObjectType>
          <SecurityObjectApplication>NAWrappedSyncP</SecurityObjectApplication>
          <SecurityObjectHash>dpn9jm1Q3g8HS0KU/+/86m1Drm/ER+TJMIx6m4GPjuk=</SecurityObjectHash>
          <SecurityObjectFormat>Base64 ASCII</SecurityObjectFormat>
          <SecurityObject>N3J0wbvIk3iXqA2ZYXU2AuKKaqBBjfzDCkQGLm9LxpizDoQbwLWJUs/r23Knojuf1zB4kfE6cY1+
He+tc31kBrTfyDvdWbAC+xs6DqsezwA4x0ntxgzwsuhIqeM6eMFQDgw3WBEZorXy039nW0So8bWS
oVumEavV0zIha3ZlS+pE9JiFnneLGM73b2gHdozjl7JS+wp63Uu3KmkPUOI4/SvpShMDaZAek6jj
dkC0VOk6LTgHXuM9Ky5joipXRmK0WBXvDQUx6JDlRgW7TG49T7HPjqrY7jFYBokJlBnrydCAzY2o
+wAmkXLm/Pur8sgg+um2EFoe43rm6h2TXPBFMQ==</SecurityObject>
        </SecurityObjectInfo>
        <SecurityObjectInfo>
          <SecurityObjectName>SYNCP2</SecurityObjectName>
          <SecurityObjectType>SymmetricKey</SecurityObjectType>
          <SecurityObjectApplication>NAWrappedSyncP</SecurityObjectApplication>
          <SecurityObjectHash>Ded37QdJIULZu2J3xZ66yEKuLVImQ2B7eKJ1oGEoHzU=</SecurityObjectHash>
          <SecurityObjectFormat>Base64 ASCII</SecurityObjectFormat>
          <SecurityObject>N4J0wbvIk3iXqA2ZYXU2AuKKaqBBjfzDCkQGLm9LxpizDoQbwLWJUs/r23Knojuf1zB4kfE6cY1+
He+tc31kBrTfyDvdWbAC+xs6DqsezwA4x0ntxgzwsuhIqeM6eMFQDgw3WBEZorXy039nW0So8bWS
oVumEavV0zIha3ZlS+pE9JiFnneLGM73b2gHdozjl7JS+wp63Uu3KmkPUOI4/SvpShMDaZAek6jj
dkC0VOk6LTgHXuM9Ky5joipXRmK0WBXvDQUx6JDlRgW7TG49T7HPjqrY7jFYBokJlBnrydCAzY2o
+wAmkXLm/Pur8sgg+um2EFoe43rm6h2TXPBFMQ==</SecurityObject>
        </SecurityObjectInfo>
        <SecurityObjectInfo>
          <SecurityObjectName>SYNCP3</SecurityObjectName>
          <SecurityObjectType>SymmetricKey</SecurityObjectType>
          <SecurityObjectApplication>NAWrappedSyncP</SecurityObjectApplication>
          <SecurityObjectHash>iqSsp/i07891QO7SwpS59BzOEHHiR0ZmWJ/5yzya6oo=</SecurityObjectHash>
          <SecurityObjectFormat>Base64 ASCII</SecurityObjectFormat>
          <SecurityObject>N5J0wbvIk3iXqA2ZYXU2AuKKaqBBjfzDCkQGLm9LxpizDoQbwLWJUs/r23Knojuf1zB4kfE6cY1+
He+tc31kBrTfyDvdWbAC+xs6DqsezwA4x0ntxgzwsuhIqeM6eMFQDgw3WBEZorXy039nW0So8bWS
```

```
oVumEavV0zIha3ZlS+pE9JiFnneLGM73b2gHdozjl7JS+wp63Uu3KmkPUOI4/SvpShMDaZAek6jj
dkC0VOk6LTgHXuM9Ky5joipXRmK0WBXvDQUx6JDlRgW7TG49T7HPjqrY7jFYBokJlBnrydCAzY2o
+wAmkXLm/Pur8sgg+um2EFoe43rm6h2TXPBFMQ==</SecurityObject>
        </SecurityObjectInfo>
        <SecurityObjectInfo>
          <SecurityObjectName>SYNCP4</SecurityObjectName>
          <SecurityObjectType>SymmetricKey</SecurityObjectType>
          <SecurityObjectApplication>NAWrappedSyncP</SecurityObjectApplication>
          <SecurityObjectHash>tL/1wmZlykWgqm41B5IZaYapRSD0MksQRE8QzrJFUfk=</SecurityObjectHash>
          <SecurityObjectFormat>Base64 ASCII</SecurityObjectFormat>
          <SecurityObject>N6J0wbvIk3iXqA2ZYXU2AuKKaqBBjfzDCkQGLm9LxpizDoQbwLWJUs/r23Knojuf1zB4kfE6cY1+
He+tc31kBrTfyDvdWbAC+xs6DqsezwA4x0ntxgzwsuhIqeM6eMFQDgw3WBEZorXy039nW0So8bWS
oVumEavV0zIha3ZlS+pE9JiFnneLGM73b2gHdozjl7JS+wp63Uu3KmkPUOI4/SvpShMDaZAek6jj
dkC0VOk6LTgHXuM9Ky5joipXRmK0WBXvDQUx6JDlRgW7TG49T7HPjqrY7jFYBokJlBnrydCAzY2o
+wAmkXLm/Pur8sgg+um2EFoe43rm6h2TXPBFMQ==</SecurityObject>
        </SecurityObjectInfo>
        <SecurityObjectInfo>
          <SecurityObjectName>SYNCP5</SecurityObjectName>
          <SecurityObjectType>SymmetricKey</SecurityObjectType>
          <SecurityObjectApplication>NAWrappedSyncP</SecurityObjectApplication>
          <SecurityObjectHash>qBYhR5fynZahyuf7Am8EjyC+77MH2uNhvZvjbXYgRW4=</SecurityObjectHash>
          <SecurityObjectFormat>Base64 ASCII</SecurityObjectFormat>
          <SecurityObject>N7J0wbvIk3iXqA2ZYXU2AuKKaqBBjfzDCkQGLm9LxpizDoQbwLWJUs/r23Knojuf1zB4kfE6cY1+
He+tc31kBrTfyDvdWbAC+xs6DqsezwA4x0ntxgzwsuhIqeM6eMFQDgw3WBEZorXy039nW0So8bWS
oVumEavV0zIha3ZlS+pE9JiFnneLGM73b2gHdozjl7JS+wp63Uu3KmkPUOI4/SvpShMDaZAek6jj
dkC0VOk6LTgHXuM9Ky5joipXRmK0WBXvDQUx6JDlRgW7TG49T7HPjqrY7jFYBokJlBnrydCAzY2o
+wAmkXLm/Pur8sgg+um2EFoe43rm6h2TXPBFMQ==</SecurityObject>
        </SecurityObjectInfo>
        <SecurityObjectInfo>
          <SecurityObjectName>SYNCP6</SecurityObjectName>
          <SecurityObjectType>SymmetricKey</SecurityObjectType>
          <SecurityObjectApplication>NAWrappedSyncP</SecurityObjectApplication>
          <SecurityObjectHash>KEbP6rS522jlAvUA1dA0P+ELODij7XeVDROmJWZCRcE=</SecurityObjectHash>
          <SecurityObjectFormat>Base64 ASCII</SecurityObjectFormat>
          <SecurityObject>N8J0wbvIk3iXqA2ZYXU2AuKKaqBBjfzDCkQGLm9LxpizDoQbwLWJUs/r23Knojuf1zB4kfE6cY1+
He+tc31kBrTfyDvdWbAC+xs6DqsezwA4x0ntxgzwsuhIqeM6eMFQDgw3WBEZorXy039nW0So8bWS
oVumEavV0zIha3ZlS+pE9JiFnneLGM73b2gHdozjl7JS+wp63Uu3KmkPUOI4/SvpShMDaZAek6jj
dkC0VOk6LTgHXuM9Ky5joipXRmK0WBXvDQUx6JDlRgW7TG49T7HPjqrY7jFYBokJlBnrydCAzY2o
+wAmkXLm/Pur8sgg+um2EFoe43rm6h2TXPBFMQ==</SecurityObject>
        </SecurityObjectInfo>
        <SecurityObjectInfo>
          <SecurityObjectName>SYNCP7</SecurityObjectName>
          <SecurityObjectType>SymmetricKey</SecurityObjectType>
          <SecurityObjectApplication>NAWrappedSyncP</SecurityObjectApplication>
          <SecurityObjectHash>c7NruQ+HnqRcSXKsSuz1aWv8c0piTqJoPDBeXt+cvPQ=</SecurityObjectHash>
          <SecurityObjectFormat>Base64 ASCII</SecurityObjectFormat>
          <SecurityObject>N9J0wbvIk3iXqA2ZYXU2AuKKaqBBjfzDCkQGLm9LxpizDoQbwLWJUs/r23Knojuf1zB4kfE6cY1+
He+tc31kBrTfyDvdWbAC+xs6DqsezwA4x0ntxgzwsuhIqeM6eMFQDgw3WBEZorXy039nW0So8bWS
oVumEavV0zIha3ZlS+pE9JiFnneLGM73b2gHdozjl7JS+wp63Uu3KmkPUOI4/SvpShMDaZAek6jj
dkC0VOk6LTgHXuM9Ky5joipXRmK0WBXvDQUx6JDlRgW7TG49T7HPjqrY7jFYBokJlBnrydCAzY2o
+wAmkXLm/Pur8sgg+um2EFoe43rm6h2TXPBFMQ==</SecurityObject>
        </SecurityObjectInfo>
      </SecurityObjectList>
    </Module>
  </PackageList>
</ShowModuleList>
```

8 Ford Local Storage of Security Objects

- IVSS will archive all generated security objects internally
- Some security objects will be stored in other systems (e.g. GIVIS/SDN) as required by various modules and applications

9 Supplier Bundle Storage and Management

Data at rest (unused packages and bundles): Data contained in Security Packages must be treated as secret data. Unused packages from bundles must be stored encrypted on a secured network with restricted access.

Data Retention: Bundles and Security Packages that are provisioned must be purged within 24 hours of usage. Based on the media used for storage of Security Packages, an appropriate method of sanitization must be used such that deleted data is unrecoverable.

10 Data Flow

The state of each individual Security Package will be tracked in GIVIS throughout the package's entire lifecycle. It will be in one of the following states:

Generated
Security Package is generated by IVSS and made available to supplier. In addition, some of the security objects are sent to internal systems (e.g., GIVIS/SDN).

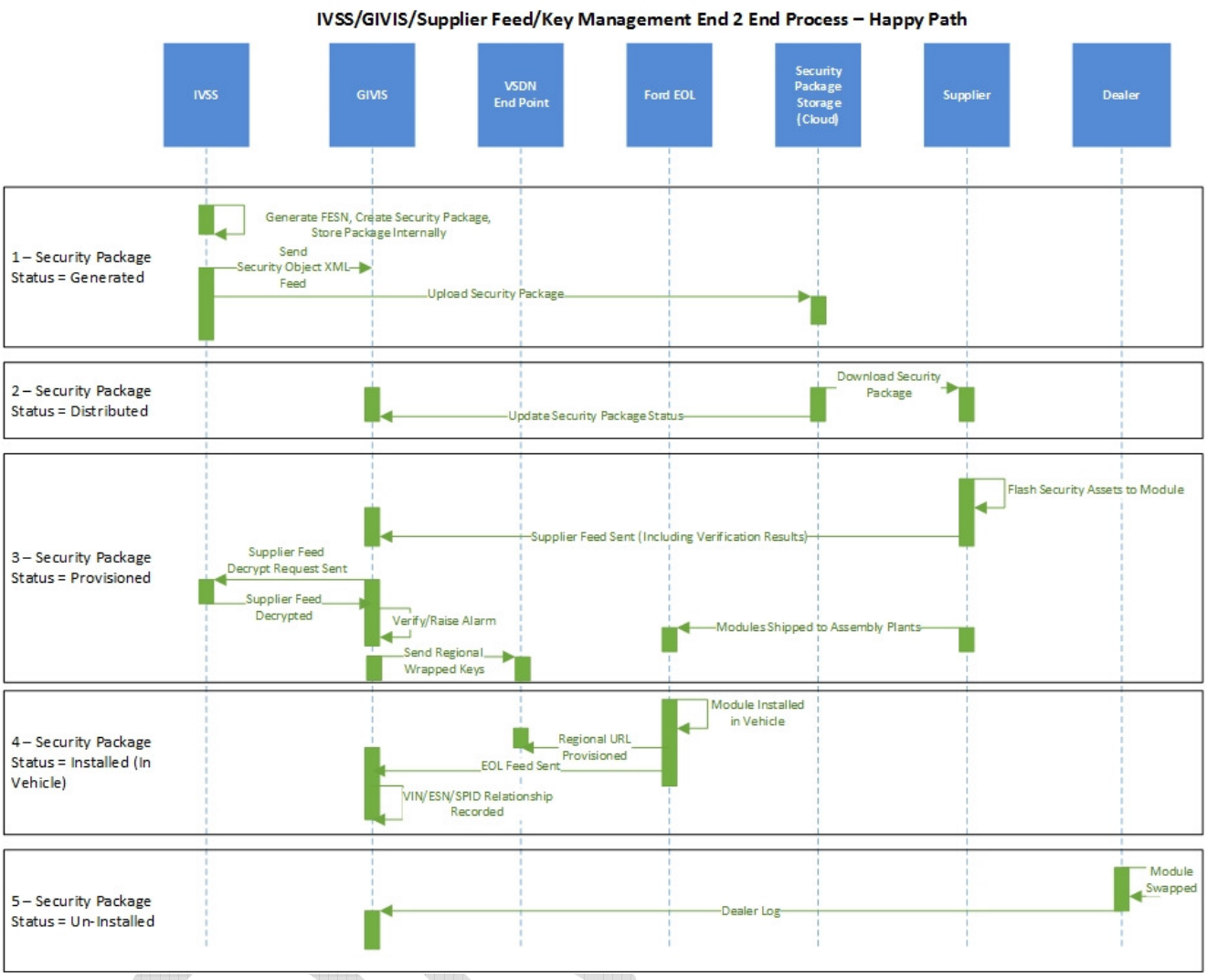
Distributed
Supplier downloads package shared by Ford (as part of a bundle). Ford tracks the FESN and SPID of each package the supplier obtains within a bundle.

Provisioned
Security Package flashed to a specific module. Ford will be notified of this by receiving the FESN and SPID in A51 Supplier Feed. In addition, the FESN and SPID will be written to two module DIDs as follows:

FESN: 0xF17F
SPID: 0xD03D

Installed
Module is installed into a vehicle. The two DID values above will be read at Ford EOL (eCATs) and sent to GIVIS along with the VIN.

Un-installed (retired)
Module removed in Ford service and no longer associated with a VIN.



TBD

- Challenge packets for testing

11 Security Object Package Usage Examples

These are some helpful command line utilities that have been found useful in working with Security Packages. They may or may not apply to any specific module.

Generate Base64 Encoded SHA256 Hashes for each Security Object File in the Package
openssl dgst -sha256 -binary <filename> | openssl enc -A -base64

Tar the Security Objects up into the Data file
*tar -cf Data.tar **

Generate the Hex ASCII SHA1 Hash of the Data file to be used for the SPID
openssl dgst –sha1 Data.tar

Tar the Manifest and Encrypted Data in the Package
tar -cf Package.tar Manifest.xml Data.tar.enc

Generate the Package signature
openssl cms -sign -binary -md sha256 -in Package.tar -outform der -out Package.sig -signer dev-provisioning-packagesign-1.pem -keyopt rsa_padding_mode:pss

Tar and gzip the Package and Package signature into the final Package file (PackageName.tar.gz)
tar -czf 1DC00001-18132b4a69c29a4d55e4c963c8fd9588557f37a3.tar.gz Package.sig Package.tar

Expand the Package File and Package Signature
tar -xzf 1DC00001-18132b4a69c29a4d55e4c963c8fd9588557f37a3.tar.gz

Validate the Package signature
openssl cms -verify -binary -md sha256 -in Package.sig -inform DER -content Package.tar -out Package.dmp -CAfile dev-provisioning-ca-chain.pem

If signature validates, expand the Manifest and Data files from the Package
tar -xf Package.tar

Expand the Data into Security Object files
tar -xf Data.tar