Proposal for a new encryption scheme

Analysing of the existing schemes

In the existing S-63 and S-100 encryption scheme the data will be encrypted with symmetric block cypher algorithms. S-63 uses the Blowfish algorithm with the ECB operation mode (Electronic code book) and S-100 uses the AES algorithm with a modified CBC operation mode (Cipher Block Chaining). For the fast encryption and decryption such symmetric key algorithms are a good choice. Especially the AES algorithm is currently a widely used and accepted algorithm.

Symmetric encryption algorithm uses the same key for the encryption and the decryption. The problem here is how decoder and encoder can exchange these keys in a secure way.

In the existing schemes the data are encrypted with so called "**Cell Keys**". Such "**Cell Keys**" will be provided by a data server in an encrypted form called "**Cell Permits**". The key for that encryption process is based on the "**Hardware Identifier**". This "**Hardware Identifier**" identifies a system such as an ECDIS. It is provided to the data server in form of an "**User Permit**". This "**User Permit**" contains the "**Hardware Identifier**" again in an encrypted form. The key for this encryption is the so called "**Manufacturer Key**". This "**Manufacturer Key**" must be kept secret, with other words it is the private key of the OEM. The "**User Permit**" contains also a "**Manufacturer identifier**", a public available identifier that tells the data server who has created the "**User Permit**". Then the data server can decrypt the "**User Permit**" with the corresponding "**Manufacturer Key**". To do so each data server must have the "**Manufacturer Keys**" of any OEM that is a member of the encryption scheme.

Though the data server must keep this list secret the risk that the private keys become known increases with any new data server. The entire security of the existing scheme is because no possible attacker has the knowledge of such keys.

A private key should only be known to its owner and only the owner should be responsible for this information. In the current scheme this rule is broken twice:

- The "Manufacturer Key" is generated by the scheme administrator and then send to the OEM.
- A list of all "Manufacturer Keys" is distributed to each data server.

Though the S-100 scheme uses more modern algorithms for the encryption with state-of-the-art key length the general problem exists exactly in the same way as described above.

The problem to be solved is: How the OEM can encrypt the "Hardware identifier" so that the data server can decrypt it without the knowledge of the OEMs private key.

Two solutions are possible:

1. Using asymmetric encryption algorithms or
2. Using an asymmetric key exchange algorithm

The latter combines the convenience of public-key crypto system with the efficiency of a symmetric-key crypto system. Therefor the proposal made in this document is based on an asymmetric key exchange.

Another problem that the current security schemes have is the missing of CRLs (Certificate Revocation List). This is necessary to indicate which certificates should not be used though their

validity period is not expired. Usually, certificates are revoked if their corresponding private keys have been compromised. The introduction of CRLs should be made as soon as possible but is not in the scope of this paper.

Background

Elliptic Curve Diffie Hellman – ECDH

Elliptic Curves

For the purpose of this proposal the theory of elliptic curves should be limited to its most interesting property with regards to cryptography. The curves used for cryptography are plane curves over a finite field which consists of the points that satisfying the formula of the curve.

For a public key crypto system, it is important that a mathematical problem exists that can be easily solved but the 'reverse' operation is not solvable in polynomial time. For elliptical curves this problem is known as "elliptic curve discrete logarithm problem" ECDLP. The security depends on the ability to calculate a point multiplication and the inability to compute the multiplicand ***a*** given the original point ***P*** and product point ***P'***.

$$\boldsymbol{P' = a \cdot P}$$

The size of the elliptic curve, measured by the total number of discrete integer pairs satisfying the curve equation, determines the difficulty of the problem.

The main advantage of elliptic curves over other public key algorithms is the shorter key length, reducing storage, transmission, and performance requirements.

That makes elliptic curves the perfect tool for a secure key exchange.

The Key Exchange

Alice and Bob want to exchange a secret key. To be precise in this method they do not exchange the key but rather establish a shared secret and derive the key from that secret.

Alice and Bob must agree on the same elliptic curve. Several curves have been developed and one very popular is the curve 25519. For each curve exists a 'base point' or generator from which all other points can be generated. Let's denote that point ***G***. Both Alice and Bob choose randomly a number ***p*** in the interval ***[1, n-1]*** where ***n*** is the size of the finite field. This number ***p*** is their private key. Then both calculate a point ***Q*** on the curve by multiplying p with the generator point $\boldsymbol{Q = p \cdot G}$. This point ***Q*** is their public key.

Alice now has a (private, public) key pair $\boldsymbol{(p_a, Q_a)}$ and Bob has $\boldsymbol{(p_b, Q_b)}$.

The public keys of both parties must be known to each other.

Now Alice calculates $\boldsymbol{d_a \cdot Q_b}$ and Bob calculates $\boldsymbol{d_b \cdot Q_a}$. Both getting the same result because:

$$\boldsymbol{d_a \cdot Q_b = d_a \cdot d_b \cdot G = d_b \cdot d_a \cdot G = d_b \cdot Q_a}$$

The calculated point, let's denote it ***X***, is the shared secret. Nobody can calculate it without the knowledge of either Alice's or Bob's private key. The result is safe because of the unsolvable ECDLP.

From the coordinate of the shared secret point now a key can be generated by a key generating function, usually a secure hash algorithm. Here the key is made of the lowest 128 bits of the output of the SHA256 hash method.

$$sharedKey = SHA256(X.x)\ Mod\ 2^{128}$$

This key is then used for a symmetric-key encryption algorithm like AES.

It is worth to be noted that the exchange of the public keys must use a secure authentication method (digital signature) to prevent the protocol from man-in-the-middle attacks.

Static versus Ephemeral Keys

The keys that are used from Alice and Bob can be long-time keys also called static keys or short-time keys called ephemeral keys.

If static keys are used the public keys are often authenticated by means of certificates. This prevents man-in-the-middle attacks. Short time keys on the other hand provides forward secrecy, thus if one key is compromised, keys that have been used in the past are still safe. The disadvantage is that the keys are only valid for one session. Additionally, the exchange of the public ephemeral key must be secured by an authentication mechanism that is based on a long-time key (digital signatures).

Since forward secrecy is not required (the plain message is constant) this proposal will be based on the use of static keys. This avoids the additional session management and leads to less changes to the existing security scheme.

Sketch of the scheme

As the critical and weak part of the existing scheme is the generation and decryption of the '**User Permit**' the new scheme will use a safe mechanism to improve this.

The creation of '**Data Permits**' and the encryption of the data does not need to be changed, though the format will slightly change.

The encryption key for a data file will be sent from the data server to the data client in an encrypted form. This is called the **DATA_PERMIT**. The key for that encryption process is the system key - **SYS_KEY**. This key is provided by the data client. The **SYS_KEY** must be unique for one system several installations of the same system (e.g. all ECDIS on board of a vessel) may share the same **SYS_KEY**. The OEM is responsible that the **SYS_KEY** is unique and well protected.

The **SYS_KEY** has the analogy **HW_ID** in the existing schemes, but the term '*Hardware Identifier*' is somehow misleading because it can be used for multiple entities of hardware.

This proposal describes how the **SYS_KEY** can be transferred from the data client to the data server in a secure way overcoming the drawbacks of the existing schemes.

The algorithm is:

1. Preparation done by the **OEM**
    a. The **OEM** creates a valid private key $\boldsymbol{sk_O}$ for the curve **x25519**.
    b. The **OEM** creates the corresponding public key $\boldsymbol{pk_O}$ by calculating $\boldsymbol{pk_O = sk_O \cdot G}$ where ***G*** is the generator of the used elliptic curve.
    c. The **OEM** create a certificate signing request for the public key $\boldsymbol{pk_O}$ and send this to the certificate authority (**CA**). This may be the scheme administrator or an organisation to which this task is assigned by the scheme administrator.
    d. The **CA** issues a certificate that contains $\boldsymbol{pk_O}$ and is signed with the private key of the **CA**. The certificate is sent back to the **OEM**.
2. Preparation done by the data server (**DS**)

   a. The **DS** creates a valid private key sk_S for the curve **x25519**.
   b. The **DS** creates the corresponding public key pk_S by calculating $pk_S = sk_S \cdot G$ where ***G*** is the generator of the used elliptic curve.
   c. The **DS** create a certificate signing request for the public key pk_S and send this to the certificate authority (**CA**). This may be the scheme administrator or an organisation to which this task is assigned by the scheme administrator.
   d. The **CA** issues a certificate that contains pk_S and is signed with the private key of the **CA**. The certificate is sent back to the **DS**. Note, that this certificate is only for the use with the ECDH key exchange and should not be used for the generation of digital signatures. For that purpose, a DS needs to have second certificate (and corresponding private key) for the ECDSA algorithm. This is already described in the current schemes.
3. The certificates that have been issued by the **CA** must be rolled out such that:
   a. The **OEM**s get all certificates issued for data servers.
   b. The **DS**s get all certificates issued for **OEM**s.
4. The technical implementation of the rollout is not described in this proposal but is worth to be mentioned the rollout should contain a 'Certificate Revocation List' (CRL) as well.
5. The data client (**DC**) generates a ***user permit*** which contains the encrypted **SYS_KEY**.
   a. Note that the ***user permit*** is no longer unique for a system. For each **DS** the ***user permit*** will be different but constant.
   b. The **DC** calculates $sharedSecret = sk_O \cdot pk_S$ and generates a key for the AES encryption algorithm by:
   $$AESKey = SHA256(sharedSecret.x)\ Mod\ 2^{128}$$
   c. The ***AESKey*** is used to create the ***user permit*** as described in S-100 Part 15 Ed5.2.
   d. The ***user permit*** is than used if the **DC** orders data from a **DS**.
6. The data server (**DS**) decrypts the ***user permit***:
   a. The **DS** identifies the **OEM** by using the **M_ID** which is part of the ***user permit*** and find the corresponding public key for that **OEM** pk_O.
   b. The **DS** calculates $sharedSecret = sk_S \cdot pk_O$ and generates a key for the AES encryption algorithm by:
   $$AESKey = SHA256(sharedSecret.x)\ Mod\ 2^{128}$$
   c. The ***AESKey*** is used to decrypt the ***user permit*** to obtain the **SYS_KEY**
   d. The **SYS_KEY** is then used to generate the ***data permits*** exactly as described in the current schema.

Impact on S-100 Part 15

Though the proposed change only affects how the **SYS_KEY** (formerly known as **HW_ID**) is transported from the data client to the data server, there are several changes in the Part 15 of S-100.

The most important are:

- Remove **M_KEY** completely from the scheme. (**M_ID** will be kept)
- Change the description of the ***user permit***. This includes the algorithm how the ***user permit*** is created and can be decrypted. Note that a ***user permit*** still can be used to identify a system but for each data server the ***user permit*** will be different. In order to identify for which DS the ***user permit*** has been created a data server identifier (**DS_ID**) should be appended to the ***user permit***. This would made the **user permit** also distinguishable from user permits of the existing scheme.

- **OEM**s and **DS**s need to create key-pairs for the Elliptic Curve Diffie-Hellman ECDH algorithm.
- **OEM**s and **DS**s need to create Certificates Signing Requests.
- The scheme administrator or a certificate authority (CA) that is assigned by the scheme administrator needs to sign (issue) ECDH certificates for both **OEM**s and **DS**s

A working example

The following section contains a working example of the creation and use of the ***user permit***.

The OEM creates a key pair for curve **x25519**:

```
-----BEGIN PRIVATE KEY-----
MC4CAQAwBQYDK2VuBCIEIGC5pHUIDpn/X61A8Vuw+d4YC8222L1AoRP0C9IoPyte
-----END PRIVATE KEY-----

SEQUENCE (3 elem)
  INTEGER 0
  SEQUENCE (1 elem)
    OBJECT IDENTIFIER 1.3.101.110 curveX25519 (ECDH 25519 key agreement algorithm)
  OCTET STRING (34 byte) 042060B9A475080E99FF5FAD40F15BB0F9DE180BCDB6D8BD40A113F40BD2283F2B5E
    OCTET STRING (32 byte) 60B9A475080E99FF5FAD40F15BB0F9DE180BCDB6D8BD40A113F40BD2283F2B5E
```

Figure 1 OEMs private key as PEM and ASN1 dump

The private key is the byte string (little-endian encoded number):

$\mathbf{sk_o}$ =

```
{60, B9, A4, 75, 08, 0E, 99, FF, 5F, AD, 40, F1, 5B, B0, F9, DE,
 18, 0B, CD, B6, D8, BD, 40, A1, 13, F4, 0B, D2, 28, 3F, 2B, 5E}
```

```
-----BEGIN PUBLIC KEY-----
MCowBQYDK2VuAyEAfzpy/4JOoEPVCQZ4bCnzxDbsi5xEIAKZ0jjo6nfIhjg=
-----END PUBLIC KEY-----


SEQUENCE (2 elem)
  SEQUENCE (1 elem)
    OBJECT IDENTIFIER 1.3.101.110 curveX25519 (ECDH 25519 key agreement algorithm)
  BIT STRING (256 bit)
011111110011101001110010111111111100000100100111010100000010000111110101…
```

Figure 2 OEMs public key PEM and ASN1 dump

The public key is the byte string (little-endian encoded number):

$\mathbf{pk_o}$ =

```
{7F, 3A, 72, FF, 82, 4E, A0, 43, D5, 09, 06, 78, 6C, 29, F3, C4,
 36, EC, 8B, 9C, 44, 20, 02, 99, D2, 38, E8, EA, 77, C8, 86, 38}
```

The data server also creates a key pair:

Private key:

```
-----BEGIN PRIVATE KEY-----
MC4CAQAwBQYDK2VuBCIEICjualqCi6+dU/5Q/8VWpNKcmCrMDQb/z8kFSZXSE8Zq
-----END PRIVATE KEY-----
```

$\mathbf{sk_s}$ =

```
{28, EE, 6A, 5A, 82, 8B, AF, 9D, 53, FE, 50, FF, C5, 56, A4, D2,
 9C, 98, 2A, CC, 0D, 06, FF, CF, C9, 05, 49, 95, D2, 13, C6, 6A}
```

Public key:

```
-----BEGIN PUBLIC KEY-----
MCowBQYDK2VuAyEARCVdYAY8RkqdGetAWQiczeQ5+DqDaYqhpDzQNO0ogHg=
-----END PUBLIC KEY-----
```

pk_S =

```
{44, 25, 5D, 60, 06, 3C, 46, 4A, 9D, 19, EB, 40, 59, 08, 9C, CD,
 E4, 39, F8, 3A, 83, 69, 8A, A1, A4, 3C, D0, 34, ED, 28, 80, 78}
```

The public keys ***pk_O*** is known to the **DS** and the ***pk_S*** is known to the **DC**. Both keys are available as certificates issued by the scheme administrator. The details are left out here for the sake of simplicity. Commands how to create certificate signing request and for issuing certificates will be described later.

The **DC** checks the validity of the data servers' certificate and obtains the public key ***pk_S*** from it. Then the shared secret will be calculated.

sharedSecret = $sk_O \cdot pk_S$ =

```
{DD, 39, 53, F2, CF, AB, BA, D8, BE, 45, 53, 0C, 7D, 78, E6, 05,
 32, E1, 68, 6B, DB, E1, 42, E1, 4B, 5F, 2B, CD, 58, 86, C9, 64}
```

The output of the SHA256 hash method is:

SHA256(sharedSecret) =

```
{29, C5, C7, 0A, 85, 22, D6, 2E, 2D, BC, 95, 7F, 4E, 9F, D1, 9A,
 BB, BF, 74, DA, 4E, 10, CA, CB, 81, 18, 00, 76, E1, 98, 52, 12}
```

The key for the encryption of the **SYS_KEY** are the 128 lower bits of the hash output.

AESKey = SHA256(sharedSecret) Mod 128 =

```
{BB, BF, 74, DA, 4E, 10, CA, CB, 81, 18, 00, 76, E1, 98, 52, 12}
```

We assume that the **SYS_KEY** is:

SYS_KEY =

```
{40, 38, 4B, 45, B5, 45, 96, 20, 11, 14, FE, 99, 04, 22, 01, 01}
```

The encrypted **SYS_KEY'** will be (using **AES** to encrypt the **SYS_KEY**):

SYS_KEY' = AES(SYS_KEY) =

```
{AF, B9, AA, 9A, C4, 81, DC, A8, 12, 3A, 73, 0F, 3F, D6, D7, 8F}
```

Assuming the M_ID is:
M_ID = **"859868"**

The ***user permit*** will be: **AFB9AA9AC481DCA8123A730F3FD6D78F6AA0DE5E859868**

Note, that this is the old format a data server identifier should be added to the user permit.

e.g. when DS_IS = **"XY"**

```
----- SYS_KEY (encrypted) -------- CRC --M_ID-DS_ID
AFB9AA9AC481DCA8123A730F3FD6D78F6AA0DE5E859868XY
```

The data server identifies the **OEM** from the **M_ID** and get the public key of the **OEM:** ***pk_O***.

Note that the public key is available in a certificate and the validity of the certificate must be checked before the public key can be used.

Then the **DS** calculates the shared secret:

$$\mathbf{sharedSecret} = sk_S \cdot pk_O =$$

```
{DD, 39, 53, F2, CF, AB, BA, D8, BE, 45, 53, 0C, 7D, 78, E6, 05,
 32, E1, 68, 6B, DB, E1, 42, E1, 4B, 5F, 2B, CD, 58, 86, C9, 64}
```

And derives the same ***AESKey*** as above. With this key the **DS** can decrypt the user permit and as the result gets the **SYS_KEY**.

Certificate creation with openssl

The following commands can be used to create the key material and the certificates by using the program *openssl*. It has been tested with version 3.0.

It is worth to be mentioned that *openssl* does not allow to create a certificate signing request (CSR) for an **x25519** key. The reason is that the CSR is self-signed and therefore the key must be valid for digital signatures. To solve this problem, the signing request must be created for another 'dummy' key that is valid for digital signatures. When the certificate is issued the issuer can replace the dummy key with the x25519 key. This is a limitation of the *openssl* command line tool, other software may support the requested functionality without the 'hack'. Nevertheless, the commands will create valid certificates for the use in EcDH key exchange.

Note that the certificate authority (**CA**) in this example is the scheme administrator. It is possible that the scheme administrator delegates the task to another organisation. In this case the **CA** does not have a self-signed certificate but one that is in a chain of certificates with the self-signed root certificate at the top of that list.

Either the **OEM** or the data server will:

Task	Command
Create a private key for the curve **x25519**	`openssl genpkey -algorithm x25519 -out x25519_priv_key.pem`
Create a corresponding public key	`openssl pkey -in x25519_priv_key.pem -pubout -out x25519_pub_key.pem`
Create a dummy key	`openssl ecparam -name secp256r1 -genkey -out dummy_key.pem`
Create a certificate signing request	`openssl req -new -key dummy_key.pem -out signed.csr`

The certificate signing request and the **x25519** public key are sent to the scheme administrator.

The **CA** issues the certificate:

Task	Command
Issue the certificate and replace the dummy key with the x25519 key	`openssl x509 -req -in signed.csr -CAkey sa-priv.pem -CA sa.crt -force_pubkey x25519-pub_key.pem -days 365 -out x25519.crt`

Note: The file sa-priv.pem contains the private key for of the scheme administrator and the file sa.crt is the self-signed certificate of the scheme administrator.

The certificate is then sent back to the requester and/or will be published. The certificate should also be stored in the scheme administrator's database.

The **OEM** or the data server can now validate their certificate. This operation is required before any information can be used that is stored in the certificate. This includes the public key.

Task	Command
Validate the certificate against the CAs self-signed certificate	`openssl verify -verbose -CAfile sa.crt x25519.crt`

It is possible to store the M_ID or the DS_ID in the certificate in one field of the Distinguished Name (DN). If this is requested the actual fields must be specified in the standard.