Session Key Establishment between a Security Element and a Server

For on-line (remote) provisioning of keys to SEs (Security Elements), like Smart Cards, there is a wish by issuers to be able to securely verify that the public key part they receive for inclusion in a certificate, indeed was (together with its private counterpart), generated *inside* of a “genuine” SE. In fact, *for compliance with security standards like FIPS 140-2, such a facility would be a prerequisite*. This document shows how key-attestations performed by an SE equipped with an embedded private key and certificate for “credential bootstrapping”, also can be used for establishing a secure, shared session-key between an SE and an issuer server. The primary purpose of such a session-key is for encrypting and integrity-checking secret data like preset PINs (Personal Information Numbers) for download (import) into SEs, as well as binding multiple protocol steps together. Ultimately, secure session-keys enable the creation of attested reports for completed multi-phase processes, a.k.a. “air-tight provisioning”.

1 Existing Solutions

1.1 Public Key Encryption

An established way to perform a secure download is that the client sends a public key (or public key certificate), to the server where the key is subsequently used by the server to encrypt either the secret data itself, or for encrypting a freshly generated symmetric key (which is in turn is used for bulk encryption of the secret payload). The client then uses its corresponding private key to decrypt the downloaded data. To verify the integrity of downloaded data, MACs (Message Authenticate Codes) are usually added as well.

Although the above “works” in the sense that *a provider that recognizes the public key* does not have to worry about sending sensitive or secret data to the wrong party, unfortunately nothing prevents a MITM (Man In The Middle) from replacing the original encrypted data with something else including generating technically correct MACs since the whole scheme relies on a single more or less public key, effectively making data substitution attacks “go under the radar”!

Is this a serious limitation? For authentication keys this is probably a minor snag since a fraudulent key presumably doesn’t give you access anywhere, while fake PINs and PUKs (Personal Unlock Keys) are slightly more worrying. In addition to imagined or real threats, there are probably also market acceptance issues with security systems having questionable data integrity protection. *Replaced encryption keys could in fact be disastrous* since they typically would reveal secret data to the attacker while hiding it from the true recipient!

1.2 Pre-Shared Secrets

Pre-shared secrets exhibit many desirable qualities from a cryptographic point of view but also introduce major difficulties: 1) *The SE unit must be known in advance*. 2) Secure distribution of shared secrets is a critical process since a leak allows an attacker to “emulate” a fake SE in software. Due to these drawbacks, pre-shared secrets have a much more limited scope compared to device certificates because the latter only requires issuers to recognize the SE *type*, not the actual *unit* since the identity (brand, serial number, etc.) of the SE is handled automatically (and securely) through the SE device certificate.

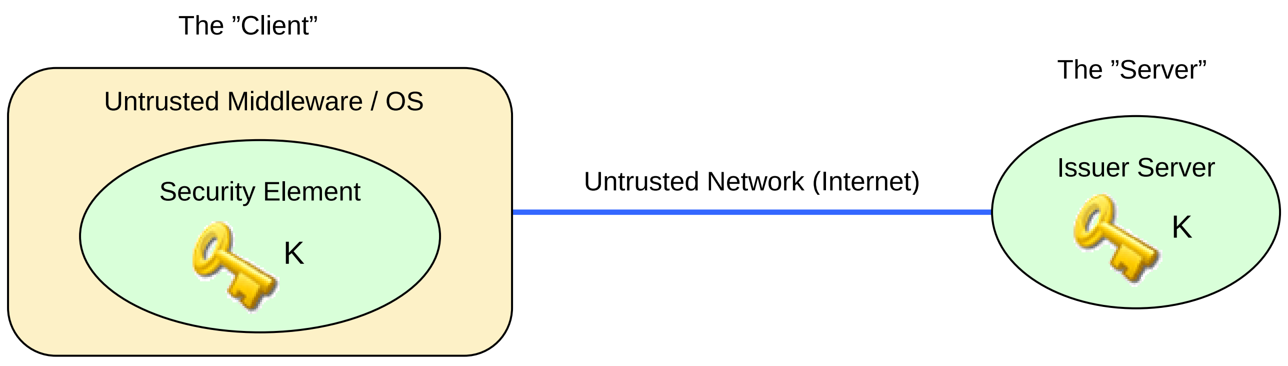
1.3 Server-Signed Data

One remedy to the problem described in 1.1 is that the server signs (using PKI), downloadable data which would reduce the attack space. Signatures though create new issues like dependency on *trust anchors residing in the SE*. It is also quite expensive making SEs that can parse big chunks of XML or ASN.1 DER code as well as validating XML signatures or CMS packages. Performing signature parsing and validation in the SE middleware (software driver) would not thwart “malware” attacks since an attacker may simply ignore the signature altogether.

In addition to these objections, signed arbitrary data to the SE introduces a trust-model which is incompatible with an “open” keystore system where it is actually *issuers* that *technically* trust an SE, while the trust and interest in an issuer’s services is rather coming from the *user* based on non-cryptographic criterions such as being an employee, bank-customer, or citizen needing to interact with the company, bank, or government respectively.

2 Detailed Operation

Below is a picture illustrating what the described method is supposed to accomplish:



The depicted key (K) is a random, SE-generated, symmetric key that after successful protocol execution should be known by the two collaborating components but not by any of the potentially malicious systems facilitating the actual key exchange.

In order to carry out the process described in this document, the parties must be equipped with the following attributes:

Security Element:	A private RSA or ECDSA key [`SE_PrivK`], and a matching public key certificate [`SE_Cert`] used for identifying an SE attestation key *which must be reserved for SE-internal signatures*.
Issuer Server:	A public RSA key [`ServerPubK`] which does not have to identify the issuer, only making it possible for the issuer to receive encrypted data which it can decrypt with a matching private key [`ServerPrivK`].

The process is as follows.

1. The "Client" invokes the "Server". This part is outside of the core security protocol
2. The "Server" responds with its `ServerPubK`
3. The "Client" asks the SE to in an *atomic* operation:
   - Generate K as well as a local handle to K [HK]
   - Encrypt K using `ServerPubK` giving EK [Encrypted K]
   - Sign (KDF_{MAC} (K) || `ServerPubK`) using `SE_PrivK` giving AK [Attested K]
   - Return HK, EK, and AK
4. The "Client" sends EK, AK, and `SE_Cert` to the "Server" which should now perform the following:
   - Decrypt EK using `ServerPrivK` giving K
   - *Finally*, verify the attestation signature [AK] using `SE_Cert`, KDF_{MAC} (K), and `ServerPubK`

Note: KDF_{MAC} is a Key Derivation Function for MAC-operations using K. It is typically relying on something like an HMAC (K, *"a static string literal"*) to further shield K from brute-force attacks.

An earlier version of this document is published at: **http://priorartdatabase.com/IPCOM/000182286D**

3 Security Analysis

If `ServerPubK` is replaced by a MITM somewhere before it reaches the SE, the SE will not be able detecting this and thus returns an EK in step #3 allowing an attacker to get hold of K. This may be considered as a major security hole, but there is no way an attacker can recreate a matching AK since *genuine* AK signatures are only created inside of SEs. That is, a "Server" will notice in step #4 if the wrong `ServerPubK` has been used and refuse to process the rest.

This method depends on that K has high entropy otherwise various attacks may indeed succeed.

Authentication of the SE is performed through operations using `SE_PrivK` and `SE_Cert`. If the PKI behind `SE_Cert` is not recognized by the "Server" (relying party), AK signatures represent no value since the whole point with device certificates for SEs is vouching for a secure container.

Although not shown in this bare-bones protocol description, session nonce values would preferably be added in step #2 and #3 and included in the AK signature in order to protect against replay attacks.

The security of the described scheme is independent on the usage of an authenticated or encrypted channel between the "Client" and the "Server".

4 Implementation Notes

HK may be also be represented by an object constituting of K wrapped by a symmetric SE-internal "master" key depending on the SE design.

Depending on the actual use-case, the "Client" may perform additional operations involving K (referred through HK) already in step #3 since the "Server" should simply reject these if the verification in step #4 fails.

Step #2 is typically combined with a request of some kind.

Rather than using a specific attestation key, the process could be implemented using the DIAS scheme. Reference: **http://priorartdatabase.com/IPCOM/000178924D**

It should be technically feasible replacing the RSA operations in step #3 and step #4 with an attested (authenticated) ECDH sequence without additional protocol rounds assuming that a known ECC curve identifier is sent by the client in step #1.

A protocol should preferably not use K "as-is" but rather derive a key for each usage such as MACing, signing, and encryption. The KDFs (Key Derivation Functions) must be known by both the SE and the "Server" and should be an integral part of the SE to eliminate K exposure to the SE middleware.

5 Other Benefits

A side-effect of using attested shared secrets is the elimination of static SE encryption keys, making the described method ideal for ECDSA-based SE device keys as an alternative to RSA since ECC is slightly cumbersome to use for encryption purposes.

By using a derived K for signing data created outside of the SE, there is no apparent need for having anything but a pre-configured `SE_PrivK` + `SE_Cert` pair in an SE, at least not for credential bootstrapping.

Version: 0.20

Author:
Anders Rundgren
Storbolsäng 50
74010 Almunge
SWEDEN

anders.rundgren@telia.com
+46 70 54 96 535