Internet Draft D. Huehnlein (secunet GmbH)
H. Schupp (GMD GmbH)
expires in six months March 1998

Credential Management for SPKM

<draft-huehnlein-credman-spkm-00.txt>

STATUS OF THIS MEMO

Comments on this document should be sent to "cat-ietf@mit.edu", the IETF Common Authentication Technology WG discussion list or to the authors "huehnlein@secunet.de" and "schupp@darmstadt.gmd.de".

ABSTRACT

The GSS-API [GSS-API1,2] offers security services independent of underlying mechanisms. A possible GSS-mechanism is the Simple Public Key Mechanism [SPKM]. This paper complements [SPKM] by providing concrete rules for the Credential Management. Our proposal allows beside the standard Credential Management based on X.509v3 [X509v3] and PKIX [PKIX] the self certification of temporary public keys, which may be used to implement a Secure Single Login variant, which works with temporary keys instead of the sensitive long term keys. The benefits of this approach are discussed in [SSLogin] more detailed. Since DL-based signature- and encryption algorithms are very well suited for the efficient generation of the temporary keys we propose two new RECOMMENDED algorithms for SPKM.

1 INTRODUCTION

The GSS-API [GSS-API1,2] "offers security services to callers in a

generic fashion, supportable with a range of underlying mechanisms and technologies and hence allowing source-level portabiltiy of applications to different environments".

Possible GSS mechanisms are e.g. the well known Kerberos V5 [KERBEROS-V5] based on symmetric cryptography or the Simple Public Key Mechanism [SPKM].

During context establishment the communication partners verify the peer's identity and authorization and agree on a common session key, which may be used for confidentiality and integrity purposes during the actual communication. To proof the identity one has to acquire credentials. In Kerberos these credentials are so called tickets with LIMITED LIFETIME.

In public-key based mechanisms like SPKM the credentials are the secret keys and certificates for the public keys. The secret keys are stored in a personal security environment (PSE), which is 'opened', i.e. made accessible, by entering a password. In practice the PSE usually is a smartcard or a PKCS#5 encrypted file. These credentials have to be available whenever a new GSS-connection is requested, i.e. a GSS-context is to be established (which can be quite often, since a client usually requests multiple GSS-connections to different servers at different times).

This means, that either the PSE has to be open for a long time or the user has to enter the password everytime a new connection is set up. If this is no problem, this is the most obvious way to provide access to the secret keys and, while not specified in SPKM, X.509 v3 / PKIX certification for the related public keys.

However if keeping the PSE open for a long time bears security problems or multiple entering of the password is not possible for usability reasons the proposed SSLogin variant, as discussed in [SSLogin], is preferable. To implement this SSLogin we need to specify a slightly different credential management, which allows the end user to CERTIFY ITS OWN TEMPORARY KEYS.

To keep the generation of the temporary keys efficient, we propose to use Discrete Logarithm based algorithms for the context establishment rather than RSA. Therefore we will propose two new RECOMMENDED algorithms for SPKM in section 3. For convenience we will include the comparison between the different variants in the appendix. For details we refer to [SSLogin].

2. CREDENTIAL MANAGEMENT FOR SPKM

In this section we will specify the credential management for SPKM. In [SPKM] there is not very much said about this problem.

> "The key management employed in SPKM is intended to be as compatible

as possible with both X.509 and PEM, since these represent large communities of interest and show relative maturity in standards."

2.1 STARTING POINT - X.509 v3 / PKIX

Before we treat the new credential management in detail, we will point out why X.509 v3 / PKIX certification only "almost" fit our needs.

The 'Basic Constraints' extension is specified by X.509 v3 and required by PKIX. It allows to distinguish between an end user- and a CA-certificate. It contains a boolean flag 'cA' which determines (when FALSE), that the certificate belongs to an end user. Since this extension is recommended to be critical an end user cannot act as a CA without notice. If an end user signs a certificate it will not be valid.

Another extension is 'Key Usage', which allows to restrict the usage of the key contained in the certificate and is recommended to be critical. The information is provided by a combination of distinguished bits, each standing for a special purpose. The most interesting bits for us are digitalSignature(0), nonRepudiation(1), keyCertSign(5), and cRLSign(6). As one can see, the signing of certificates is not covered by the simple 'digitalSignature' bit, but requires the 'keyCertSign' bit, which is only set for CA-certificates.

Up to here it seems to be impossible to reach our goal of self-certification of temporary keys, if there were not the 'Extended Key Usage' extension.
This extension field was not present in the X.509 recommendations [DAM-X.509] and the PKIX profile [PKIX] until recently.
Experiences in deploying the base standard showed, that there are situations, in which it is necessary to specify the key usage somewhat more concrete.

We will use this extension (together with a somewhat relaxed evaluation of the Basic Constraints and KeyUsage extensions) to specify a scheme for certificate verification that is as close to PKIX as possible but allows self-certification of temporary keys to implement our proposed Secure Single Login.

It is secure in that it allows circumvention of PKIX in only one, well defined manner and for our special purpose. Furthermore it introduces only one small exception to the PKIX verification procedure, so it will be easy to implement in an existing implementation of the SPKM GSS-API mechanism.

Another extension to mention is the 'Subject Alternative Name'. It provides the possibility to specify the subject's identity in another form as the Distinguished Name in the X.509 v1 field 'subject'. If the latter one is left empty, this extension should be critical.

The same extension exists for the 'issuer' field ('Issuer Alternative Name').

Last but not least, the extension 'Authority Key Identifier' provides a way to refer to a specific key of the issuer, that shall be used for the verification of a certificate. The issuing certificate can be identified by the name of its issuing authority (the "grandparent" of the certificate we look at right now) and its serial number or by a unique octet string (KeyIdentifier), which should appear in the issuing certificates 'Subject Key Identifier' extension.

2.2 SECURE SINGLE LOGIN

During our proposed secure single login, the user will

1. open the PSE to be able to access the long term signature key.

Furthermore the user's software will

2. generate a new temporary (assymetric) key(s) for use in SPKM.

There is no specific order for these two operations, they even could be performed at the same time in a multi-threaded environment.
Then the temporary public key(s) are

3. self-certified with the long term signature key.

After that, the PSE is closed again and the valuable long term keys are not exposed any more.

The philosophy of SSLogin is equal to Kerberos' [KERBEROS-V5], where we use temporary keys (tickets) with a limited lifetime to authenticate. If an attacker manages to access the memory or the harddrive he will only get the temporary secret keys instead of the valuable long term keys. If this disclosure is recognized the temporary certificate may be revoked.

Note, that the self-certification in step 3. violates the PKIX-profile, because an end user is not allowed to sign certificates. In the proposed SSLogin-SPKM however it is essential, that this self-certification is allowed. It would take too long for a user to contact a trusted certification authority to get the temporary public key(s) certified. Since the conservative evaluation of the basic constraints and the key-usage extension is not necessary in this context, we can change the proposed SPKM-profile accordingly, like discussed in the next section.

2.3 SSLOGIN-SPKM PROFILE

We propose to use the PKIX profile with the following incremental changes to allow the self-certification of the temporary public key(s).

We define two new key purposes for the 'Extended Key Usage' field:

* SignTempCert

  If this key purpose is present, the contained subject public key is allowed to verify a certificate for a temporary key, even if the cA-flag in the Basic constraints extension is set to FALSE. The Key Usage bit 'digitalSignature' has to be set and 'nonRepudiation' may be set.

  id-kp-SignTempCert OBJECT IDENTIFIER ::= { id-kp 1 }

* Temporary

  This key purpose indicates, that it is a temporary key and that the next certificate in the verification chain is a user certificate in which id-kp-SignTempCert should be present.

  id-kp-Temporary OBJECT IDENTIFIER ::= { id-kp 2 }

To use already issued certificates without id-kp-SignTempCert being present, it is valid that this key purpose is not present. For new certificates however we recommend to include this key purpose.

Since the proposed SSLogin mechanism is tailormade for SPKM we recommend to subordinate these two object identifiers to SPKM behind the two mechanism-variants ({id-spkm 1} for 3 way authentication and {id-spkm 2} for two-way authentication with secure time stamps) specified in [SPKM]. Thus we propose

  id-kp OBJECT IDENTIFIER ::= { id-spkm 3 },

where

  id-spkm OBJECT IDENTIFIER ::= { iso(1) org(3) dod(6) internet(1)
                                  security(5) mechanisms(5) 1 }

We will assume, that a user already got X.509 v3 / PKIX certified public keys. Since we want to use the key for signing a certificate, we only need the signature key. The user's corresponding certificate will contain these fields (informally):

v1 fields:
  * U-serialNo
  * issuer = CA
  * validity = U-validity
  * subject = User
v3 extensions:
  * subjectAltName = UserAlt (optional, if subject not empty)

```
  * issuerAltName = CAAlt (optional, if issuer not empty)
  * Key usage
    - digitalSignature = TRUE
  * Basic constraints
    - cA = FALSE

To be in conformance to our proposed profile, these additional
restrictions are necessary:

+ the 'Key Usage' extension is made critical and contains at least the
  bit digital signature(0)
+ the 'Basic Contraints' extension is made critical
+ the 'Extended Key Usage' extension is present and contains
    id-kp-SignTempCerts
  This extension should be marked non-critical to allow usage of this
  certificate in applications which are not in conformance with this
  profile.

In summary, the users long term certificate of the public signature key
is:

v1 fields:
  * U-serialNo
  * issuer = CA
  * validity = U-validity
  * subject = User
v3 extensions:
  * subjectAltName = UserAlt
  * issuerAltName = CAAlt
  * Key usage (critical = TRUE)
    - digitalSignature = TRUE
  * Extended key usage (critical = FALSE)
    - id-kp-SignTempCert
  * Basic constraints (critical = TRUE)
    - cA = FALSE

It is assumed, that this certificate is issued by a trusted CA,
conforming to PKIX.

The certificate for the temporary public signature key will
(informally) look like this:

v1 fields:
  * T-serialNo
  * issuer = User
  * validity = T-validity
  * subject = User
v3 extensions:
  * subjectAltName = UserAlt
  * issuerAltName = UserAlt
  * Key usage (critical = TRUE)
```

```
    - digitalSignature = TRUE
      (all others = FALSE)
  * Extended key usage (critical = TRUE)
    - id-kp-Temporary
  * Basic constraints (critical = TRUE)
    - cA = FALSE
  * Authority key identifier (critical = FALSE)
    - authorityCertIssuer = CA
    - authorityCertSerialNumber = U-serialNo
```

The certificate for the temporary public encryption key looks similar, except for the 'Key Usage' extension, which should have set the bits keyEncipherment(2) - or keyAgreement(4), whichever is appropriate, instead of digitalSignature.

The use of the 'Authority Key Identifier' extension is recommended (also by PKIX), but not required. It facilitates the search for the users long-term certificate and makes absolutely clear, that this is a self-signed certificate and no prototype certificate like used by some certification request mechanisms.

Alternatively or additionally to the issuer/serialNumber pair, the key identifier method can be used for this extension. This is closer to PKIX recommendations.

The serial number of the temporary certificates should ideally be consecutively numbered, so that the temporary certificates can be identified uniquely by issuer (User) and serial number, but this is no critical requirement.

Clearly, a certificate verification procedure conform to the PKIX profile will reject this temporary certificate, because it is certified (signed) with an end user's signature key. The (long term) user certificate has not set the 'keyCertSign' bit and the 'cA' flag in the Basic constraints extension is FALSE. Therefore we propose a somewhat relaxed certificate verification:

A certificate will be valid, if

  a) all certificates in the verification chain are PKIX conform

or if

  b) all of the following requirements are fulfilled:

```
  1. The first (temporary) certificate has the following properties:
    1.1 issuer == subject,
    1.2 issuerAltName == subjectAltName,
    1.3 T-validity is subrange of U-validity, i.e.
      1.3.1 T-validity.notBefore > U-validity.notBefore,
      1.3.2 T-validity.notAfter < U-validity.notAfter,
    1.4 the Key Usage extension is critical,
```

1.5 nonRepudiation == FALSE,
1.6 keyCertSign == FALSE,
1.7 cRLSign == FALSE,
1.8 the Extended Key Usage extension is critical,
1.9 'id-kp-Temporary' is present in Extended Key Usage,
1.10 the Basic constraints extension is critical,
1.11 cA == FALSE,

2. all other certificates in the verification chain are PKIX conform,

3. and the second (User's long term) Certificate has the following, properties:
3.1 the Key Usage extension is critical,
3.2 digitalSignature=TRUE,
3.3 'id-kp-SignTempKey' is present in Extended Key Usage,
3.4 subject == issuer-of-temp-cert, see 1.1

It is obvious, that these requirements restrict the presented temporary certificate concept to the first certificate in the verification chain. Since the Basic constraints and Key Usage extension is set appropriate, it will not be possible to act as CA, by maliciously presenting the temporary certificate to an ordinary (PKIX conform) verification procedure. Furthermore the validity period of the temporary certificate is smaller than the validity period of the long term user certificate and it is not possible to issue such temporary certificates for other subjects. Typically the validity period of the temporary certificate will not exceed one day. Through the requirements given in 3, it is ensured, that only the long term signature key can be used to produce temporary certificates and that this self-certification is allowed at all. Note, that the verification has to fail, if the subject of the temporary certificate is different from the subject in the long term certificate, which is implicit in our description above and enforced by step 3.4 of the verification procedure.

Note, that other extensions, like Policy constraints are not affected by these changes, because we self-certify only user keys. Furthermore it should be mentioned, that the absence of the extended key purpose 'id-kp-SignTempkey' could be treated more relaxed to allow the use of already issued certificates without 'id-kp-SignTempkey' for our scheme.

It should be mentioned, that it is not possible to (mis-)use the Authorization Data field (included in the first SPKM-token, see [SPKM] page 12) to implement the full functionality proposed, because this Authorization Data field is ONLY present in the FIRST token. Thus the target system can not use temporary keys to authenticate.

3. NEW ALGORITHMS FOR SPKM

Since Discrete Logarithm based encryption- and signature algorithms are very well suited for the efficient generation of temporary key pairs, we propose two new algorithms to be used for SPKM, like mentioned

in [SSLogin].

We propose the RECOMMENDED algorithms, specified in [OSI/OIW]:

elGamal ALGORITHM PARAMETER NULL ::= { 1 3 14 7 2 1 }
and
dsaWithSHA1 ALGORITHM PARAMETER DSAParameters ::= { 1 3 14 7 3 27 }

It is intended to propose analogous algorithms based on Elliptic Curves as OPTIONAL algorithms for SPKM as soon as the standardization in e.g. [IEEEP1363] is finished.

4. ACKNOWLEDGEMENT

We would like to thank C. Adams, P. Eisenacher and T. Surkau for fruitful comments on earlier versions of this draft.

APPENDIX A (Security - Usability - Efficiency)

In this section we will briefly recall the different variants for the SPKM credential management introduced in [SSLogin], compare them in terms of Security, Usability, Time-Efficiency and Space-Efficiency and give more concrete recommendations for implementation.

To compare the time efficiency, we group the operations to be performed in Once, Every GSS-session and Every GSS-context establishment and estimate the workload in terms of modular (1024 bit) multiplications. The times for generation of temporary keys are crude estimates, based on practical measurements with SECUDE [SECUDE].

To compare the space efficiency we estimate the number of bytes, which have to be stored permanently in Secure Storage, i.e. inside the PSE and Insecure Storage, i.e. on the harddisk. We may assume, that a user already got certified long term (1024 bit) RSA key pairs for signatures and encryption and that the SPKM credential management like presented in this specification is applied to all variants. We will only focus on additional time and space requirements. I.e. we neither consider the time needed to generate the long term keys, nor the space required to store them in the PSE. Since we neglect some operations, e.g. computing hash values, generation of random numbers etc. and have to 'convert' the time for some operations to our 'unit' (1024 bit modular multiplication), these estimates are very rough in nature. We may assume, that the secret RSA keys are stored in the RSAPrivateKey-format [PKCS1], so that the decryption and signature operations can be speeded up by application of the Chinese Remainder Theorem.

In the following table, these variants are compared:

1. **Single Login: enter password once and leave PSE open for the whole GSS-session.**
2. **Multiple Login: enter password everytime a GSS-context is established. Close the PSE right afterwards.**
3. **Secure Single Login with RSA:**
   usage of one temporary RSA key pair, used both for signatures and

encryption.

4. **Secure Single Login with a Discrete Logarithm based crypto system** (naive, without precomputation):
   usage of one temporary DSA key pair for signatures and one temporary ElGamal key pair for encryption
5. **Secure Single Login with a Discrete Logarithm based crypto system** (with precomputation):
   same as 4. but using a precomputed table of group elements

Note, that variants 4. and 5. assume a GSS-API implementation, that supports the algorithms ElGamal and DSA (see section 3).

Variant	1.	2.	3.	4.	5.
Security	low	high	high	high	high
Usability	good	bad	good	good	good
Time-Efficiency (in 1024bit mod. multiplications)					
Once	/	/	/	116000	116517
Session	/	/	108315	1267	675
Context					
- e = Fermat4	698	698	321	2408	1540
- general	5526	5526	3758	7236	6368
Space-Efficiency (in bytes)					
Secure	/	/	/	20	20
Insecure	/	/	740	1348	19780

Table 1. Variants for the SPKM credential management

>From this table we can see, that we can combine security and usability at only slightly higher expenses at Login-time and context establishment. The application of DL based algorithms turns out to be very well suited to implement the the SSLogin variant, because the time for the actual key generation is negligible small and may be well performed at Login-time. Furthermore it is possible to speed up the context establishment by applying exponentiation variants with precomputation, which is not possible in RSA-type cryptosystems. In this case only slightly more memory is needed to store the precomputed values. Since the additional storage of about 20000 byte should be possible in every implementation, we recommend the application of this exponentiation technique.

APPENDIX B (References)

[GSS-API1] J. Linn: Generic Security Service Application Program Interface, RFC 1508, Sep. 1993

[GSS-API2] J. Linn: Generic Security Service Application Program Interface Version 2, RFC 2078, Jan. 1997

[IEEEP1363] IEEE Working Group P1363: Standard for Public Key Cryptography, draft, via http://www.ieee.org

[KERBEROS-V5] J. Kohl, C. Neumann: The Kerberos Network Authentication Service (V5), RFC 1510, Sep. 1993

[OSI/OIW] OSI/OIW: Part 12 - OSSecurity (Stable), June 1995, via http://www.nemo.ncsl.nist.gov/oiw/agreements/stable/OSI/12s_9506.txt

[PKCS1] RSA Lab.: PKCS#1 - RSA Encryption Standard, Version 1.5, 1993

[PKIX] R. Housley, W. Ford, S. Farrel, D. Solo: Internet Public Key Infrastructure, Part I: X.509 Certificate and CRL Profile, Internet-Draft: draft-ietf-pkix-ipki-part1-06.txt, 15th October 1997

[SECUDE] GMD/TKT-SIT: SECUrity Development Environment for Open Networks - Online Documentation, http://www.darmstadt.gmd.de/secude/Doc/index.htm

[SPKM] C. Adams: The Simple Public-Key GSS-API Mechanism (SPKM), RFC 2025, Okt. 1996

[SSLogin] D. Huehnlein: Credential Management and Secure Single Login for SPKM, to appear in the Proceedings of NDSS '98, San Diego, March 11-13, 1998

[X509v3] ISO/IEC 9594-8: Information Technology - Open Systems Interconnection - The Directory: Authentication Framework, CCITT/ITU Recommendation X.509, 1993.

[DAM-X.509] ISO/IEC JTC 1/SC 21/WG 4 and ITU-T Q15/7: Final Text of Draft Amendment 1 to ISO/IEC 9594-8 on Certificate Extensions, June 1997