Network Working Group
Request for Comments: 1824
Category: Informational

H. Danisch
E.I.S.S./IAKS

August 1995

The Exponential Security System TESS: An Identity-Based Cryptographic Protocol for Authenticated Key-Exchange (E.I.S.S.-Report 1995/4)

Status of this Memo

Abstract

This informational RFC describes the basic mechanisms and functions of an identity based system for the secure authenticated exchange of cryptographic keys, the generation of signatures, and the authentic distribution of public keys.

Table of Contents

1. Introduction and preliminary remarks

This RFC describes The Exponential Security System TESS [1]. TESS is a toolbox set system of different but cooperating cryptographic mechanisms and functions based on the primitive of discrete exponentiation. TESS is based on asymmetric cryptographical protocols and a structure of self-certified public keys.

The most important mechanisms TESS is based on are the ElGamal signature [2, 3] and the KATHY protocols (KeY exchange with embedded AuTHentication), which were simultaneously discovered by Guenther [4] and Bauspiess and Knobloch [5, 6, 7].

This RFC explains how to create and use the secret and public keys of TESS and shows a method for the secure distribution of the public keys.

It is expected that the reader is familiar with the basics of cryptography, the Discrete Logarithm Problem, and the ElGamal signature mechanism.

Due to the ASCII representation of this RFC the following style is choosen for mathematical purposes:

- a ^ b means the exponentiation of a to the power of b, which is always used within a modulo context.
- a[b] means a with an index or subscription of b.
- a = b means equality or congruency within a modulo context.

1.1. Definition of terms/Terminology

Key pair

A key pair is a set of a public and a secret key which belong together. There are two distinct kinds of key pairs, the SKIA key pair and the User key pair. (As will be shown in the section about hierarchical SKIAs, the two kinds of keys are not really distinct. They are the same thing seen from a different point of view.)

User

Any principal (human or machine) who owns, holds and uses a User key pair and can be uniquely identified by any description (see the Identity Descriptor below).

In this RFC example users are referred to as A, B, C or Alice and Bob.

SKIA

SKIA is an acronym for "Secure Key Issuing Authority". The SKIA is a trusted local authority which generates the public and secret part of a User key pair. It is the SKIA's duty to verify whether the identity encoded in the key pair (see below) belongs to the key holder. It has to check passports, identity cards, driving licenses etc. to investigate the real world identity of the key owner. Since every key has an implicite signature of the SKIA it came from, the SKIA is responsible for the correctness of the encoded identity.

Since the SKIA has to check the real identity of users, it is usually able to work within a small physical range only (like a campus or a city). Therefore, not all users of a wide area or world wide area network can get their keys from the same SKIA with reasonable expense. There is the need for multiple SKIAs which can work locally. This implies the need of a web of trust levels and trust forwards. Communication partners with keys from the same SKIA know the public data of their SKIA because it is part of their own key. Partners with keys from different SKIAs have to make use of the web to learn about the origin, the trust level, and the public key of the SKIA which issued the other key.

Id[A] Identity Descriptor

The Identity Descriptor is a part of the public User key. It is a somehow structured bitstring describing the key owner in a certain way. This description of the key owner should be precise enough to fully identify the owner of a User key. The description depends on the nature of the owner. For a human this could be the name, the address, the phone number, date of birth, size of the feet, color of the eyes, or anything else. For a machine this could be the hostname, the hostid, the internet address etc., for a fax machine or a modem it could be the international phone number.

Furthermore, the description bitstring could contain key management data as the name of the SKIA (see below) which issued the key, the SKIA-specific serial number, the expiry date of the

key, whether the secret part of the key is a software key or hidden in a hardware device (see section Enhancements), etc.

Note that the numerical interpretation (the hash value) of the Identity Descriptor is an essential part of the mathematical mechanism of the TESS protocol. It can not be changed in any way without destroying the key structure. Therefore, knowing the public part of a user key pair always means knowing the Identity Descriptor as composed by the SKIA which issued this key. This is an important security feature of this mechanism.

The contents of the Identity Descriptor have to be verified by the issuing SKIA at key generation time. The trust level of the User Key depends on the trust level of the SKIA. A certain Identity Descriptor must not be used more than once for creating a User Key. There must not exist distinct keys with the same Identity Descriptor. Nevertheless, a user may have several keys with distinct expiration times, key lengths, serial numbers, or security levels, which affect the contents of the Identity Descriptor.

However, it is emphasized that there are no assumptions about the structure of the Identity Descriptor. The SKIA may choose any construction method depending on its purposes.

The Identity Descriptor of a certain user A is referred to as Id[A]. Whereever the Identity Descriptor Id[A] is used in a mathematical context, its cryptographical hash sum H(Id[A]) is used.

Encrypt(Key,Message)
Decrypt(Key,Message)

Encryption and Decryption of the Message with any common cipher.

1.2. Required mechanisms

The protocols described in this RFC require the following submechanisms:

- A random number generator of cryptographic quality
- A prime number generator of cryptographic quality
- A hash mechanism H() of cryptographic quality
- An encryption mechanism (e.g. a common block cipher)

- An arithmetical library for long unsigned integers

- A method for checking network identities against real-world identities (e.g. an authority which checks human identity cards etc.)

2. Setup

This section describes the base method for the creation of the SKIA and the User key pairs. Enhancements and modifications are described in subsequent sections.

The main idea of the protocols described below is to generate an ElGamal signature (r,s) for an Identity Descriptor Id[A] of a user A. Id[A] and r form the user's public key and s is the users secret key. The connection between the secret and the public key is the verification equation for the ElGamal signature (r,s). Instead of checking the signature (r,s), the equation is used in 'reverse mode' to calculate r^s from public data without knowledge of the secret s.

The authority generating those signatures is the SKIA introduced above.

2.1. SKIA Setup

By the following steps the SKIA key pair is created:

- p: choose a large prime p of at least 512 bit length.

- g: choose a primitive root g in GF(p)

- x: choose a random number x in the range 1 < x < p-1

- y:= (g ^ x) mod p

The public part of the SKIA is the triple (p,g,y), the secret part is x.

Since the public triple (p,g,y) is needed within the verification equation for the signatures created by the SKIA, this triple is also an essential part of all user keys generated by this SKIA.

2.2. User Setup

The User Setup is the generation of an ElGamal signature on the user's Identity Descriptor by the SKIA. This can be done more than once for a specific User, but it is done only once for a specific Identity Descriptor.

To create a User key pair for a User A, the SKIA has to perform the following steps:

- Id[A]: Describe the key owner A in any way (name, address, etc.), convert this description into a bit- or byte-oriented representation, and concatenate them to form the Identity Descriptor Id[A].

- k[A]: choose a random number k[A] with gcd(k[A],p-1) = 1. k[A] must not be revealed by the SKIA.

- r[A] := (g ^ k[A]) mod p

- s[A] := (H(Id[A]) - x * r[A]) * (k[A] ^ -1) mod (p-1)

The calculated set of numbers fulfills the equation:

$$x * r[A] + s[A] * k[A] = H(Id[A]) \mod (p-1).$$

The public part of the generated key of A consists of Id[A] and r[A], referenced to as (Id[A],r[A]) in the context of the triple (p,g,y). (Id[A],r[A]) always implicitely refers to the triple (p,g,y) of its parent SKIA.

The secret part of the key is s[A].

k[A] must be destroyed by the SKIA immediately after key generation, because User A could solve the equation and find out the SKIAs secret x if he knew both the s[A] and k[A]. The random number k must not be used twice. s[A] must not be equal to 0.

Since (r[A],s[A]) are the ElGamal signature on Id[A], the connection between the SKIA public key und the User key pair is the ElGamal verification equation:

$$r[A] \hat{} s[A] = (g \hat{} H(Id[A])) * (y \hat{} (-r[A])) \mod p.$$

This equation allows to calculate r[A] ^ s[A] from public data without knowledge of the secret s[A]. Since this equation is used very often, and for reasons of readability, the abbreviation Y[A] is used for this equation.

Y[A] means to calculate the value of r[A] ^ s[A] which is

$$(g \hat{} H(Id[A])) * (y \hat{} (-r[A])) \mod p.$$

Note that a given value of Y[A] is not reliable. It must have been reliably calculated from (p,g,y) and (Id[A],r[A]). Y[A] is to be understood as a macro definition, not as a value.

Obviously both the SKIA and the User know the secret part of the User's key and can reveal it, either accidently or in malice prepense. The enhancements section below shows methods to avoid this.

3. Authentication

This section describes the basic methods of applying the User keys. They refer to online and offline communication between two users A(lice) and B(ob).

The unilateral and the mutual authentications use the KATHY protocol to generate reliable session keys for further use as session encryption keys etc.

3.1. Zero Knowledge Authentication

The "Zero Knowledge Authentication" is used if Alice wants to authenticate herself to Bob without need for a session key.

Assuming that Bob already reliably learned the (p,g,y) of the SKIA Alice got her key from, the steps are:

1. Alice generates a large random number t, 1<t<p-1, where t should have approximately the same length as p-1.

2. a := r[A] ^ t mod p

3. Alice sends her public key (Id[A],r[A]) and the number a to Bob.

4. Bob generates a large random number c, c<p-1, where c should have approximately the same length as p-1, and sends c to Alice.

5. Alice calculates
   c' := (c * s[A] + t) mod (p-1)
   and sends c' to Bob.

6. Bob verifies whether
   r[A] ^ c' = (Y[A] ^ c) * a mod p.

This is the Beth-Zero-Knowledge protocol [8] which is based on self-certified public keys and an improvement of the DLP-Zero-Knowledge identification protocol from Chaum, Evertse, and van de Graaf [9].

3.2. Unilateral Authentication

The "Unilateral Authentication" (or "Half Authentication") can be used in those cases:

- Alice wants to authenticate herself to Bob without Bob authenticating himself to Alice.

- Bob wants to send an encrypted message to Alice readable by her only (offline encryption).

A shared key is generated by the following protocol. This key can be known by Alice and Bob only.

Assuming that Bob already reliably learned the (p,g,y) of the SKIA Alice got her key from, the steps are:

1. Alice sends her public key (Id[A],r[A]) to Bob if he does not already know it.

2. Bob chooses a random number 1 < z[A] < p-1 and calculates v[A] := r[A] ^ z[A] mod p

3. Bob sends v[A] to Alice.

4. Alice and Bob calculate the session key:

   Alice: key[A] := v[A] ^ s[A] mod p
   Bob: key[A] := Y[A] ^ z[A] mod p

Apply the equations of the User Key Setup section to Bob’s equation to see that Alice and Bob get the very same key in step 4:

   key[A] = r[A] ^ (s[A] * z[A]) mod p

A third party cannot calculate key[A], because it has neither s[A] nor z[A]. Therefore, Bob can trust in the fact that only Alice is able to know the key[A] (as long as nobody else knows her secret s[A]).

This protocol is based on the Diffie-Hellman scheme [10], but avoids the weakness of the missing authenticity of the public keys.

In this protocol Bob did not verify whether Alice really knew her s[A] and was able to calculate key[A]. Therefore, a final challenge-response step should be performed in case of online communication (see the subsection below).

In case of sending encrypted messages, Bob can execute step 4 before step 3, use the key[A] to encrypt the message, and send the encrypted message together with v[A] in step 3.

3.3. Mutual Authentication

The "Mutual Authentication" is used for online connections where both Alice and Bob want to authenticate to each other.

Within this protocol description it is assumed that Alice and Bob have keys of the same SKIA and use the same triple (p,g,y). Otherwise in each step the triple has to be used which belongs to the user key it is applied to.

The steps are as follows (where the first four steps are exactly twice the "Unilateral Authentication" and steps 5-9 form a mutual challenge-response step to find out whether the other side really got the key):

1. Alice sends her (Id[A],r[A]) to Bob.
   Bob sends his (Id[B],r[B]) to Alice.

2. Bob chooses a random number z[A] < p-1
   and calculates v[A] := r[A] ^ z[A] mod p

   Alice chooses a random number z[B] < p-1
   and calculates v[B] := r[B] ^ z[B] mod p

3. Bob sends v[A] to Alice.
   Alice sends v[B] to Bob.

4. Alice and Bob calculate the session keys:

   ```
   Alice: key[A] := v[A] ^ s[A] mod p
          key[B] := Y[B] ^ z[B] mod p

   Bob:   key[B] := v[B] ^ s[B] mod p
          key[A] := Y[A] ^ z[A] mod p
   ```

5. Alice chooses a random number R[B]
   Bob chooses a random number R[A]

6. Alice sends Encrypt(key[B],R[B]) to Bob.
   Bob sends Encrypt(key[A],R[A]) to Alice.

7. Alice and Bob decrypt the received messages to R’[A] and R’[B].

8. Alice sends Encrypt(key[A],T(R'[A])) to Bob.
   Bob sends Encrypt(key[B],T(R'[B])) to Alice.

9. Alice and Bob decrypt the received messages to R''[A] and R''[B]

10. Alice verifies whether T(R[B]) = R''[B].
    Bob verifies whether T(R[A]) = R''[A].

T() is a simple bijective transformation function, e.g. increment().

After step 4 Alice can trust in the fact that only Bob and herself can know key[B], but she still does not know whether she is really talking to Bob. Therefore, she forces Bob to make use of his key within steps 5-9. Alice now has checked whether she really talks to Bob. Since the scheme is symmetrical, Bob also knows that he talks to Alice.

3.4. Message Signing

To sign a message m (where H(m) is a cryptographic hash value of the message), the message author A generates an ElGamal signature by using his r[A] as the generator and the s[A] as his secret:

- A generates a random number K with gcd(K,p-1) = 1.

- R := r[A] ^ K mod p

- S := (H(m) - s[A] * R) * (K ^ -1) mod (p-1)

The calculated set of numbers fulfills the equation:

(s[A] * R + K * S) = H(m) mod(p-1)

The signed message consists of (m,Id[A],r[A],R,S).

The receiver of the message checks the authenticity of the message by calculating the hash value H(m) and verifying the equation:

r[A] ^ H(m) = (Y[A] ^ R) * (R ^ S) mod p

4. Enhancements

This section describes several enhancements and modifications of the base protocol as well as other comments.

4.1. Non-Escrowed Key Generation

Within the normal User Setup procedure for a User A, the SKIA gains knowledge about the secret key s[A]. The SKIA could use this key to fake signatures or decrypt messages, or to allow others to do so.

To avoid this situation, a slight modification of the User Setup procedure may be applied. The SKIA Setup is the same as in the base protocol.

Within the User Setup the SKIA does not use its primitive element g, but a generator created by the User instead.

The modified scheme looks like this:

- User A generates a random number a with gcd(a,p-1)=1
- User A calculates g' := g^a mod p and forwards g' to the SKIA.
- The SKIA generates Id[A] and k[A] as in the base protocol
- The SKIA sets r[A] := (g' ^ k[A]) mod p and
  s'[A] := (H(Id[A]) - x * r[A]) * (k[A] ^ -1) mod (p-1)
- The SKIA forwards (Id[A],r[A],s'[A]) to the user A
- The user A calculates his s[A] := s'[A] * (a^-1) mod (p-1)

The SKIA is not able to find out the secret key s[A] of A. This protocol is based on the idea of the 'testimonial' [11].

The SKIA is still able to create a second key with the same Identity Descriptor (identical or at least having same contents), but with different r[A] and s[A]. If such a second key was successfully used for authentication or message signing, the real key owner can use his own key to proof the existence of two different keys with identical (equivalent) Descriptors. The existence of such two keys shows that the SKIA cannot be trusted any longer.

If the key is generated by this method, it should be mentioned in the Identity Descriptor. This allows any communication partners to look up in the public part of a key whether the secret part is known to the SKIA.

4.2. Hardware Protected Key

The protocol of the previous subsection guaranteed that the SKIA does not know the user's secret key.

On the other hand, the SKIA may wish that the user himself does not know his own secret key. This may be necessary because the user could otherwise reveal his secret key accidently or intentionally. Especially if untrusted hard- or software or an environment without trusted process protection is used, the secret key can be spied out. For high-level security applications this might not be acceptable. The key owner must be able to use his key without being able to read this key. This contradiction can be solved by hiding the secret part of the User Key within a protected hardware device.

Within the SELANE project, the protocols described in this RFC were implemented for SmartCards. The User Key is created using the non-escrowed key generation procedure described in the previous section, modified such that the random number is generated inside the card. The secret s[A] exists only inside the card and does not get outside. The SmartCard is able to execute all parts of the algorithms which need access to the secret key. To make use of the SmartCard an additional password is required.

If the key is hidden in such a hardware device, it should be mentioned in the Identity Descriptor. This allows any communication partners to look up in the public part of a key whether the key is hardware protected.

4.3. Key Regeneration

If both methods of the previous subsections are used to protect the key, neither the SKIA nor the User himself knows the secret key. This could be harmful for the User if the hardware device is lost or damaged, because the User could become unable to decrypt messages encrypted with the public key.

To prevent such a denial of service, there are two methods:

- If the protection factor ’a’ was choosen by the User, the User can deposit the factor ’a’ in a secure way, e.g. give it as a shared secret to his friends. The SKIA can do the same and deposit s’[A] somewhere else. If the SKIA and the User cooperate, they are able to create a second hardware device equivalent to the first.

- If the protection factor a was generated inside of the hardware device, the device itself may give out the s[A] or the a in a secure way (e.g. as a shared secret).

Since the recreation of a User key defeats the property of such a key to exist only once, the SKIA should restrict this to special cases only. Furthermore it should be done only after the end of the

lifetime of the key, if its lifetime was limited.

4.4. r ^ r

A slight modification of the base protocol allows some speedup in the key exchange:

- The SKIA is created as in the base protocol

- For the User Setup the SKIA solves the equation
  x * s[A] + r[A] * k[A] = H(Id[A]) mod (p-1)
  which differs from the base protocol in that r and s were swapped.

- The public key allows to calculate
  y ^ s[A] = (g ^ H(Id[A])) * (r[A] ^ -r[A]) mod p
  without knowing s[A]. Here the term (r[A] ^ -r[A]) can be precalculated for speedup.

- Bob calculates key[A] := (g ^ H(Id[A]) * r[A] ^ -r[A]) ^ z[A]
  and v[A] := y ^ z[A] mod p
  Alice gets key[A] := v[A] ^ s[A] mod p
  where key[A] = y ^ (s[A] * z[A])

This protocol is similar to the AMV modification by Agnew et al. [12].

4.5. Implicit Key Exchange

If the r ^ r protocol of the previous section is used, an implicit shared key can be calculated for Alice and Bob by using the Diffie-Hellman scheme:

- Alice: key[A,B] = (g ^ H(Id[B]) * r[B] ^ -r[B]) ^ s[A] mod p

- Bob: key[B,A] = (g ^ H(Id[A]) * r[A] ^ -r[A]) ^ s[B] mod p

where key[A,B] = key[B,A] = y ^ (s[A] * s[B]).

This can not be used with Non-escrowed keys.

4.6. Law Enforcement

This will be subject of a separate RFC.

4.7. Usage of other Algebraic Groups

Within this RFC calculations were based on a specific algebraic group, the multiplicative group of integers modulo a prime number p (which is the multiplicative group of a finite field GF(p)). However, any cyclic finite group with a strong discrete logarithm problem can be used, e.g., a subgroup of the multiplicative group or elliptic curves.

As an example the subgroup used by the DSA (Digital Signature Algorithm) of length N can be used instead of the full multiplicative group of GF(p) for speedup (in this case the Secure Hash Algorithm SHA is recommended as the hash algorithm). See [13, 14] for a description of DSA and SHA.

4.7.1. DSA subgroup SKIA Setup

- Generate large primes p and q such that p is at least 512 bit long, q is 160 bit long, and q is a factor of (p-1).

- choose a primitive root h in GF(p)

- g:= h^((p-1)/q)
  Note that g generates a subgroup G with |G|=q

- x: a random number of about 160 bit.

- y:= (g ^ x) mod p

The public key of the SKIA is (p,g,y,q). (q is required for speedup only.)

The secret key of the SKIA is x.

4.7.2. Escrowed DSA subgroup User Setup

- k[A]: a random number of 160 bit length with gcd(k[A],q)=1

- r[A]:= (g ^ k[A]) mod p

- s[A]:= (H(Id[A]) + x * r[A]) * (k[A] ^ -1) mod q

Again, (Id[A],r[A]) is the public key and s[A] is the secret key. Note that r[A] has the length of p and s[A] has the length of q (160 bit).

4.7.3. Non-Escrowed DSA subgroup User Setup

- User A generates a random number h of 160 bit length.
- User A calculates a := g^h mod p and sends a to the SKIA.
- The SKIA generates the user key with the secret key s'[A].
- User A calculates s[A]:= s'[a] * (h^-1) mod q

4.7.4. DSA subgroup Authentication

The protocols for authentication are the same as described above, except that wherever the modulus (p-1) was used the smaller modulus q is used instead, and DSA is used for message signing.

The abbreviation Y[A] still stands for r[A] ^ s[A], which is now (the sign of r[A] was changed for speedup)

```
( g ^ H(Id[A])) * ( y ^ r[A] ) mod p
```

and can be calculated in a faster way as

```
u1 * u2 mod p
```

where

```
u1 := g ^ ( H(Id[A])  mod q )  mod p
u2 := y ^ ( r[A] mod q ) mod p.
```

5. Multiple SKIAs

In the preceding sections it was assumed that everybody learned the (p,g,y) triple of a SKIA reliably.

By default, a User reliably learns only the (p,g,y) of the SKIA which generated his own key, because he gets the triple with his key and can verify the triple with the signature verification equation.

If the User wants to communicate with someone whose key was generated by a different SKIA, a method for authenticating the (p,g,y) of the other SKIA is needed.

5.1. Unstructured SKIAs

This will be subject of a separate RFC.

5.2. Hierarchical SKIAs

If there is a hierarchy between the SKIAs, their keys can be generated hierarchically:

- Every SKIA and every User has a level (expressed as a cardinal number). The root SKIA has level 0. All Users and all other SKIAs have levels greater than 0.

- Each SKIA except the root SKIA is also a User, and each User can be a SKIA.

  A SKIA of level n generates keys for Users of level n+1.

  A User of level n is also a SKIA of level n.

- Since every SKIA (except the root SKIA) is also a User, each SKIA has an Identity Descriptor describing its Identity and perhaps its level and its parent SKIA. There is a function parent(A) which finds the parent SKIA for every user A. This function may use informations stored in the Identity Descriptor.

  Thus, the parent() function allows to find the path to the root SKIA for every node of the tree forming the hierarchy.

  The root SKIA may also have an Identity Descriptor.

- The root SKIA creates itself as in the base protocol.

- The key for a User A of level n (n>0) is generated by the parent SKIA of level n-1. The public part is (Id[A],r[A]), the secret part is (s[A]).

  User A is automatically SKIA A:

  ```
  p[A] := p[parent(A)]  = p of the root SKIA
  g[A] := r[A]
  x[A] := s[A]
  y[A] := g[A] ^ x[A] = r[A] ^ s[A] = Y[A] =
         ( g[parent(A)] ^ H(Id[A]) ) * ( y[parent(A)] ^ -r[A]) mod p
  ```

  Therefore, the public data (p,g[A],y[A]) of the SKIA A can be calculated by everyone from the public data of the User A and the public data of its parent SKIA. The SKIA A itself may use the faster method to get y[A] by calculating r[A] ^ s[A], while everybody else has to use the slower but public method as in the lower equation. The secret of the "SKIA A" is identical to the secret of the "User A".

Since a User A uses the very same data to act as either a user or as a SKIA, and since message signing (subsection 3.4.) is the very same procedure as generating a User key (in fact it is the same thing), a user should not sign a message which could be misunderstood as an Identity Descriptor. An attacker could intercept the message and its signature and abuse it as a User key. This can be avoided by the use of tags which preceed every set of data being signed and show whether it is a message or an Identity Descriptor.

This scheme allows any two users (even users of distinct hierarchies) to communicate reliably. They need to know the public data (p,g,y) of each other's root SKIA only. There is no need for online key servers.

The communication is the same as in the base protocols but with an extension to the method of finding Y[A] (again with Alice and Bob):

- Bob reliably learned the (p,g,y) of Alice's root SKIA S(0).

- Where Alice presented (Id[A],r[A]) only in the first step, she now presents (Id[S],r[S]) for each SKIA/User node S in her path to her root SKIA S(0). Since this information does not need to be reliable or signed, it can be provided by any simple server mechanism.

- Bob iteratively calculates the public data (p,g,y) of each SKIA in the path, starting with Alice's root SKIA, until he gets the (p,g,y) of Alice where y is Y[Alice].

Note that Bob did not have to verify anything within the iteration. After the iteration he has a set of public SKIA data (p,g,y) to be used with Alice public key, but he still does not know whether he was spoofed with wrong data of Alice or her parent SKIAs.

Since the iteration Bob calculated is a chain of nested signatures, the correctness of the (p,g,y) he gets depends on every single step. If there is at least one step with a bad Id[S] or r[S], Bob will get a wrong Y[S] in this step and all following steps, and the chain doesn't work.

If the chain calculated by Bob was not completely correct for any reason, Alice cannot make use of her key: her signatures do not verify, she cannot decrypt encrypted messages and she cannot answer to the challenge response step in case of mutual authentication.

5.3. Example: A DNS-based public key structure

Here is a simple example of the usage of the hierarchical SKIA scheme within the DNS name space:

Let every domain also be a SKIA, and let the root domain be a root SKIA. Let the Identity Descriptor of any object within the name space be its name: the domain name for domains, the host name for machines, the mail address for humans and services.

Consequently, a user with the mail address "danisch@ira.uka.de" got his key from the SKIA of the domain "ira.uka.de". This SKIA was authorized by the SKIA of "uka.de", which was authorized by the SKIA of "de", which is the root SKIA of Germany. It is assumed that everybody reliably learned the public key of the german root domain "de".

The public key of danisch@ira.uka.de would look like:

```
(  "danisch@ira.uka.de", r[danisch@ira.uka.de] ,
   "ira.uka.de"        , r[ira.uka.de]        ,
   "uka.de"            , r[uka.de]
)
```

For the reasons described in the previous subsection, this key is self-certified and does not need any further signature.

The key can be presented by danisch@ira.uka.de within online communications, be appended to signed messages, or simply be retrieved by the domain name server of ira.uka.de.

Someone who reliably learned the (p,g,y) of the root domain .de (Germany) can now build the chain:

```
"de"                      (p,g,y)[de]
"uka.de"                  (p,g,y)[uka.de]
"ira.uka.de"              (p,g,y)[ira.uka.de]
"danisch@ira.uka.de"      (p,g,y)[danisch@ira.uka.de]
```

Thus it is possible to reliably obtain the Y[danisch@ira.uka.de].

To communicate with the whole world, knowledge of the public keys of all root domain SKIAs only is needed. These keys can be stored within some tens of KBytes. No third party is needed for doing an authenticated key exchange.

The whole world could also be based on a single root SKIA; in this case a single (p,g,y) is needed only.

In a more realistic example the Id[danisch@ira.uka.de] could contain:

```
creator=      ira.uka.de
created=      1-Jun-1995
expiry=       31-Dec-1999
protection=   non-escrowed, smartcard
type=         human
name=         Hadmut Danisch
email=        danisch@ira.uka.de
phone=        +49 721 9640018
fax=          +49 721 696893
photo=        <digitized compressed portrait>
```

Security Considerations

- The strength of TESS depends on the strength of the discrete logarith problem, the strength of the ElGamal signature, and the confidentiality of the SKIAs.

- Attention should be paid to the security considerations of the underlying mechanisms (ElGamal, DSA, Diffie-Hellman, etc.).

- Since the SKIA creates itself under normal circumstances, an attacker could create his own SKIA and use it to create a User Key with an arbitrary Identity Descriptor. This shows that the Identity Descriptor is as reliable as the origin of the triple (p,g,y) of the SKIA it came from. The User Key creation process is a signature process for the Identity Descriptor and strongly depends on the trustworthyness of the signing SKIA.

- It is the SKIA’s duty to give the s[A] only to the user the Identity Descriptor belongs to.

- Since the very same procedure is used for signing messages and generating user keys, it is important to distinguish between messages and keys.

- The authentication protocols work without an online authority. Therefore, there is no simple way for revoking keys. For this reason keys should have an expiration date mentioned in the Identity Descriptor. In case of the hierarchical scheme a key expires if any key in the path to the root SKIA expires.

References

1. Th. Beth, F. Bauspiess, H.-J. Knobloch, S. Stempel, "TESS - A Security System based on Discrete Exponentation," Computer Communcations Journal, Vol. 17, Special Issue, No. 7, pp. 466-475 (1994).

2. T. ElGamal, "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithm," IEEE-Trans. Information Theory, IT-31, pp. 469-472 (July 1985).

3. B. Klein, H.-J. Knobloch, "ElGamal-Signatur" in Sicherheitsmechanismen, ed. Fries, Fritsch, Kessler, Klein, pp. 171-176, Oldenburg, Muenchen (1993).

4. C. G. Guenther, "An Identity-Based Key-Exchange Protocol" in Advances in Cryptology, Proceedings of Eurocrypt ’89, pp. 29-37, Springer (1990).

5. B. Klein, H.-J. Knobloch, "KATHY" in Sicherheitsmechanismen, ed. Fries, Fritsch, Kessler, Klein, pp. 252-259, Oldenburg, Muenchen (1993).

6. F. Bauspiess, H.-J. Knobloch, "How to keep authenticity alive in a computer network" in Advances in Cryptology, Proceedings of Eurocrypt ’89, pp. 38-46, Springer (1990).

7. F. Bauspiess, "SELANE - An Approach to Secure Networks" in Abstracts of SECURICOM ’90, pp. 159-164, Paris (1990).

8. Th. Beth, "Efficient zero-knowledge identification scheme for smart cards" in Advances in Cryptology, Proceedings of Eurocrypt ’88, pp. 77-84, Springer (1988).

9. D. Chaum, J. H. Evertse, J. van de Graaf, "An improved protocol for demonstrating possesion of discrete logarithms and some generalizations" in Advances in Cryptology, Proceedings of Eurocrypt ’87, pp. 127-141, Springer (1988).

10. W. Diffie, M. Hellman, "New directions in cryptography," IEEE-Trans. Information Theory, 22, pp. 644-654 (1976).

11. Th. Beth, H.-J. Knobloch, "Open network authentication without an online server" in Proc. Symposium on Comput. Security ’90, pp. 160-165, Rome, Italy (1990).

12. G. B. Agnew, R. C. Mullin, S. A. Vanstone, "Improved digital signature scheme based on discrete exponentation," Electron. Lett., 26, pp. 1024-1025 (1990).

13. "The Digital Signature Standard," Communications of the ACM, Vol. 35, pp. 36-40 (July 1992).

14. Bruce Schneier, Applied Cryptography, John Wiley & Sons (1994).

Author's Address

Dipl.-Inform. Hadmut Danisch
European Institute for System Security (E.I.S.S.)
Institut fuer Algorithmen und Kognitive Systeme (IAKS)

University of Karlsruhe
D-76128 Karlsruhe
Germany

Phone: ++49 721 96400-18
Fax: ++49 721 696893
EMail: danisch@ira.uka.de
WWW: http://avalon.ira.uka.de/personal/danisch.html