3GPP TSG SA WG3 Security — S3#22 **S3-020103**

25 - 28 February 2002

Bristol, UK

Source: **Alcatel**

Title: **Security of terminateAccess() function in OSA**

Document for: **Adoption**

Agenda item: **T.b.d.**

1 Introduction

This contribution identifies various issues in TS 29.198-3 v4.2.0 with regards to the security mechanism used to protect the terminateAccess() function.

2 Issues

As specified in TS 29.198-3, the terminateAccess() function is digitally signed by the framework. To achieve this, two extra parameters are specified as input to the function: signingAlgorithm which identifies the algorithm used by the framework to produce the signature, and digitalSignature which contains the signature value itself.

2.1 Issue#1: no indication of public key/certificate to be used by verifier

The framework does not indicate which public key/certificate the client must use to verify the signature. The assumption to be made in the current specification is that the client and the framework have some a-priori agreement in which the client obtains a copy of the public key (embedded in a certificate or not) used by the framework for signing.

However, such a solution is not scalable and indication of the public key used is particularly important as the framework may have several (private/public) key pairs so that the client knows which one to use. The certificate itself is also important as a basis for signature verification (so as to validate the public key itself first).

2.2 Issue#2: no anti-replay protection

As currently specified, the signature is calculated solely over the terminationText string. Because such a string is more of a constant nature (same string is used on many occasions), no mechanism is defined to prevent re-use of a digital signature by a third-party.

2.3 Issue#3: no negotiation of signature algorithm

The signature algorithm used by the framework is not negotiated as it is a parameter of the terminateAccess(0 function itself. There is therefore no way for the client application to indicate which algorithm(s) it supports and it must consequently merely accept what it receives. If the signing algorithm is not supported, the client cannot verify the signature and an exception will be generated but the effect will most probably be that the association with the framework will be considered closed by the client itself. If the latter is the case, the lack of a priori agreement also opens the door to denial of service: an attacker can issue a terminateAccess() to the client with a

signature algorithm that it knows is not supported by the client. In such a scenario, the signature value does not have to be valid since the client will not try and verify it.

2.4 Issue#4: specification of signature algorithm

The list of signature algorithms is provided in table TpSigningAlgorithm, which lists P_MD5_RSA_512 and P_MD5_RSA_1024 as possible algorithms. Such a reference to the use of MD5 with RSA for signing is not sufficient to determine what the exact mechanism to implement is. Moreover, the use of MD5 as hashing algorithm and especially a modulus of 512 bits for RSA are not advisable and are deprecated.

3 Solution

With regards to issue #1, a solution could be add a new parameter to the terminateAccess function, carrying the public key identifier or its certificate. Another solution is to have the digitalSignature field itself carrying the certificate. This can be achieved by using an appropriate digital signature format such as the one defined in Cryptographic Message Syntax (RFC 2630). CMS indeed defines a data structure to carry a digital signature, the signed data and the signer's certificate.

With regards to issue #2, a fresh value must be generated by the framework for use as input into the signing algorithm. If adopting CMS to carry the signature, CMS already contains a field to contain the signing time. The signing time can be used by both parties to detect replayed signatures, under the condition that the verifier keeps track of the last verified value.

A separate contribution discusses a proposed mechanism for the negotiation of the signature algorithm (issue #3).

With regards to issue #4, the list of algorithms must be more precisely defined and can also be extended to other signing algorithms. Such a list of possible algorithms is given in IETF draft draft-ietf-pkix-ipki-pkalgs-05.txt, which itself refers to RFC 2437 specifying in detail RSA-based signing mechanism, to FIPS-186 for DSA signing mechanism and X9.62 for ECDSA signing mechanism.

4 Required Modifications to TS 29.198-3

The definition of terminateAccess() function in section 6.3.1.2 must be modified to cope with the above issues and solutions, especially the use of CMS for varrying the signature and certificate, and the anti-replay mechanism.

The list of algorithms in section 10.3.11 (TpSigningAlgorithm table) must be modified to cover other signature algorithms: RSA_with_SHA1 (modulus length of 1024 bits), DSA, ECDSA. The definition of each algorithm in the table must refer to a specification that clearly describes how to generate the signature value (RFC 2437, FIPS-186, X9.62 for RSA, DSA, ECDSA respectively).